

Trabalho 2 de Organização e Arquitetura de Computadores

1/2018

Cristiano Silva Júnior: 13/0070629

16 de Abril de 2018

1 Introdução

Este relatório explica como foi implementado um simulador de programas em linguagem de máquina *MIPS*. O objetivo deste projeto foi fazer um programa que fosse capaz de executar um mínimo de instruções que bastasse para alguns casos de teste, já que a implementação de um simulador completo é muito grande para o escopo da disciplina.

Em termos mais abstratos, um processador *MIPS* funciona executando um *loop* básico, onde a máquina carrega instruções e dados de suas memórias correspondentes; decodifica as instruções em variáveis; e as executa, podendo ou não salvar dados em seu banco de registradores ou na própria memória.

Na linguagem *MIPS*, as instruções podem ser divididas em três tipos:

- Tipo R
- Tipo I
- Tipo J

2 Metodologia

Para melhor simular um processador *MIPS*, desenvolvemos um programa em linguagem *GNU C* compilados com *GCC* e *Make* tanto em Microsoft Windows 8 e Arch Linux. O foco principal neste programa encontra-se em uma entidade doravante denominada *processador*, cujo objetivo é executar três procedimentos:

- Carregar a próxima instrução em memória;
- Decodificar a instrução, carregando os parâmetros *RT*, *RS*, *RD*, *opcode*, *funct*, *shamt* e *imm* na memória;
- Executar a instrução atual baseado nos parâmetros definidos naquele momento.

A implementação do processador encontra-se em 2 arquivos separados:

- Uma biblioteca *mips.h* que contém diversas funções cujos objetivos são auxiliar o processador em tarefas relacionadas à linguagem *MIPS*;

- O simulador em si, que consiste de uma *struct* do C com banco de registradores, memória de instruções, memória de dados e algumas variáveis relevantes para a execução do programa. Com exceção de um interruptor para indicar o fim da execução de um programa, todas as propriedades foram implementadas utilizando inteiros de 32 bits como implementados na biblioteca padrão *stdint.h*.

O processador deverá executar arquivos binários de programas *MIPS* compilados na plataforma *MARS Simulator v4.5* como indicados em requerimentos descritos no arquivo *makefile*.

Com isto em mente, o programa pode ser compilado e executado por meio da seguinte chamada no diretório raiz:

Listing 1: Compilação do simulador MIPS

```
$ make
```

Ela gerará os binários MIPS relevantes; executará uma rotina de testes unitários nas bibliotecas do processador; compilará o programa do processador; e o executará.

3 Resultados

A fim de validar a aplicação desenvolvida, o programa foi testado em dois programas de exemplo em linguagem *MIPS*.

3.1 Programa de Exemplo 1

Listing 2: Primeiro teste

```
.data
CONST: .word 5

.text
# read a
la $t0, CONST
lw $t0, 0($t0)

# read b
li $v0, 5
syscall
add $t1, $v0, $zero

# sum a + b
add $a0, $t1, $t0
li $v0, 1
syscall

# exit
li $v0, 10
syscall
```

Este programa deverá ler um número inteiro do teclado e apresentar a sua soma com 5. Os testes em ambos sistemas operacionais funcionaram.

3.2 Programa de Exemplo 2

O segundo programa a ser testado foi um teste enviado pelo professor como exemplo e já contém algumas operações mais complexas.

Listing 3: Segundo teste

```
.data
primos: .word 1, 3, 5, 7, 11, 13, 17, 19
size: .word 8
msg: .asciiz "Os oito primeiros numeros primos sao : "
space: .ascii " "
.text
la $t0, primos #carrega endereco inicial do array
la $t1, size #carrega endereco de size
lw $t1, 0($t1) #carrega size em t1
li $v0, 4 #imprime mensagem inicial
la $a0, msg
syscall
loop: beq $t1, $zero, exit #se processou todo o array, encerra
li $v0, 1 #servico de impressao de inteiros
lw $a0, 0($t0) #inteiro a ser exibido
syscall
li $v0, 4 #imprime separador
la $a0, space
syscall
addi $t0, $t0, 4 #incrementa indice array
addi $t1, $t1, -1 #decrementa contador
j loop #novo loop
exit: li $v0, 10
syscall
```

Este programa deverá escrever na saída padrão o seguinte texto:

```
os oitos primeiros numeros primos sao : 1 3 5 7 11 13 17 19
```

Em testes em ambos os sistemas operacionais, não foi possível escrever os caracteres codificados em números na memória *MIPS* por erros não compreendidos.

4 Conclusão

5 Referência Bibliográfica

1. "MIPS32® Architecture For Programmers". Volume II.