Universidade de Brasília – UnB

Departamento de Ciência da Computação – CIC

Simulador MIPS

Organização e Arquitetura de Computadores

Professor: Marcelo Grandi Mandelli

Autores:

Nome: Alon Mota	Matrícula: 130005002
Nome: André Souto	Matrícula: 140016261
Nome: Pedro Henrique S. Perruci	Matrícula: 140158596

Turma: C

Data de entrega: 07/05/2017

1 Resumo

O objetivo deste trabalho é desenvolver um simulador da arquitetura MIPS em linguagem de alto nível. O simulador foi desenvolvido em C++ e se mostrou funcional para os código MIPS exemplo e testes realizados. O simulador prima pela interatividade e clareza ao executar os comandos traduzidos da linguagem de máquina MIPS.

2 Requisitos do Simulador

Nesta seção, estão dispostos alguns dos principais requisitos para o desenvolvimento do simulador MIPS. Eles também podem ser encontrados no roteiro para o Trabalho 2 da disciplina.

2.1 Arquivos de Entrada

Para o funcionamento do simulador, são necessários dois arquivos em formato binário que devem ser provenientes de algum programa já implementado em assembly. Um deles deve conter todo o segmento de código do programa, ou seja, todas as instruções e o outro deve conter o segmento de dados. Tais segmentos devem ser obtidos através do simulador Mars como explicitado no roteiro.

2.2 Memória e Registradores

A memória do simulador deve ser declarada como um vetor de inteiros de 32 bits com tamanho de 4096 (#define MEM_SIZE 4096) e deverá ser endereçada byte a byte. O segmento de código deve ser lido para a memória e começar na posição 0x00000000 enquanto o segmento de dados também deve ser lido para a memória mas endereçado a partir da posição 0x00002000.

Quanto aos registradores, o simulador deve conter os 32 registradores do MIPS e os registradores HI e LO declarados como inteiros de 32 bits (int32_t) e os registradores PC e RI declarados como inteiros de 32 bits sem sinal (uint32_t). Todos devem ser declarados como variáveis globais. No trabalho, um vetor regs[32] foi utilizado para representar os 32 registradores.

3 Metodologia

O código foi produzido utilizando-se uma metodologia modular de programação, ou seja, as funções utilizadas foram divididas em diferentes códigos fonte sendo integradas através de arquivos do tipo **.hpp** (headers). Dentro do diretório trabalho2, os recursos necessários para produção foram divididos em diretórios distintos. Em bin, há os códigos **.bin** dos programas fibonacci.asm e primos.asm. Em include, há todos os arquivos do tipo header utilizados, incluindo os que foram criados pelo grupo. Em src há todos os códigos **.cpp** utilizados no projeto junto com seu executável, nomeado por **main**. E no diretório References existem todos os documentos que serviram de base para a produção do simulador.

O código foi feito utilizando-se o paradigma orientado a objetos da linguagem C++, ou seja, foi criado um objeto **simulador** do tipo **MIPS_simulador,** o que contribuiu para garantir a modularidade do projeto.

3.1 Códigos Fonte e Funções Desenvolvidas

No arquivo ***main.cpp*** consta a função ***main*** do projeto, onde é criado o objeto **MIPS_simulador simulador (argv[1], argv[2]).** É essa função que recebe os parâmetros iniciais (arquivos binários e opção "p" ou "f") e dependendo deles executa o restante do programa.

No arquivo ***binaryIO.cpp*** constam as funções ***printAddressValues***, ***allocateValueOnMemory*** e ***loadBinFile*.** O funcionamento delas consiste basicamente em garantir a leitura dos arquivos **.bin** passados como parâmetros e da sua devida alocação na memória do simulador.

Já no arquivo ***MIPS_simulador.cpp*** constam as funções:

[1] ***MIPS_simulador*:** cria um objeto **mips_** do tipo **MIPS_core** e chama **loadMemory**;

[2] ***loadMemory*:** chama a função **loadBinFile** para carregar os dados dos arquivos **.bin** na memória;

[3] ***menu*:** imprime na tela as opções de escolha do usuário (sendo elas, step, run, dump_mem, dump_reg e exit);

[4] ***limpaTela*:** executa a limpeza da tela no terminal;

[5] ***interfaceUsuario*:** de acordo com a opção escolhida pelo usuário, chama a função correspondente (step(), run(), dump_mem(), dump_reg()) ou termina a execução do programa;

[6] ***step*****:** executa o programa passo a passo chamando a função **step** de **MIPS_core**;

[7] ***run*****:** executa o programa continuamente chamando a função **run** de **MIPS_core**;

[8] ***dump_mem*****:** imprime o valores atuais de cada campo da memória do simulador chamando a função **dump_mem** de **MIPS_core**;

[9] ***dump_reg*****:** imprime os valores atuais de cada um dos registradores do simulador chamando a função **dump_reg** de **MIPS_core**. Permite a opção "h" para valores serem impressos no formato hexadecimal ou a opção "d" para valores serem impressos no formato decimal.

E no arquivo ***MIPS_core.cpp*** constam as funções:

[1] ***MIPS_core*****:** apenas chama a função **inicializaRegs**. Serve para inicializar a execução do programa como um todo;

[2] ***inicializaRegs*****:** inicializa todos os registradores do simulador com o valor 0x00000000;

[3] ***dump_mem*****:** efetivamente imprime na tela os valores de cada endereço da memória no instante em que é chamada;

[4] ***dump_reg*****:** efetivamente imprime na tela os valores de cada um dos registradores. Recebe o parâmetro ("h" ou "d") da função **dump_reg** de **MIPS_simulador**;

[5] ***fetch*****:** função que busca uma instrução, coloca-a no registrador RI e incrementa o registrador PC para o mesmo apontar para a próxima instrução;

[6] ***decode*****:** função que extrai o opcode da instrução que se encontra em RI por meio de um shift right e então, dependendo do opcode, extrai o resto dos campos da instrução;

[7] ***execute*****:** dado uma determinada instrução, **execute** por meio de switch cases encontra e efetua a operação determinada pela instrução. Nela, constam todas as instruções pedidas no roteiro;

[8] ***step*****:** executa um ciclo (funções fetch(), decode() e execute()) de cada vez;

[9] ***run*****:** executa o programa até o fim das instruções.

4 Como Utilizar?

O código fonte C++ do Simulador Mips encontra-se dividido nas pastas include/ e src/. Esta seção contém as instruções básicas para compilá-lo e executá-lo.

4.1 Requisitos de Sistema

O simulador foi desenvolvido em ambiente Unix-like. Foi projetado e testado em nos sistemas Linux Ubuntu e Mac OSX 10.12.4. Recomenda-se, também, a instalação do simulador MARS [1] para gerar arquivos binários de entrada para o simulador.

4.2 Instruções de Compilação

O código fonte para o simulador acompanha um arquivo src/makefile que contém as diretivas básicas para a compilação. Para compilar basta entrar no diretório src pelo terminal e digitar o comando “make run”.

4.3 Chamada do Simulador

Conforme solicitado pelo roteiro do trabalho, a chamada do simulador deve ser realizada da seguinte maneira:

```
./simuladorMIPS <arquivo .text> <arquivo .data> <parâmetro>
```

cujo o parâmetro pode assumir os valores p, ou f. Tal chamada deve ser realizada via terminal, dentro da pasta src do projeto. No caso deste projeto, simuladorMips corresponde ao executável main (dentro da pasta src). Todas as informações, ditas aqui, necessárias para compilação e execução do simulador constam dentro de um arquivo README.txt dentro da pasta src.

5 Testes Realizados

Os testes realizados foram baseados nos códigos fornecidos pelo professor e em um código assembly feito pelos integrantes do grupo.

5.1 Códigos exemplo MIPS

Foram fornecidos como auxílio na produção do trabalho dois códigos exemplo, fibonacci e números primos, cujos binários se encontram dentro da pasta bin e foram

utilizados para verificar o funcionamento do programa. O programa feito funcionou bem com os dois códigos fornecidos.

5.2 Testes Individuais das Funções

Para complementar os testes com os programas fornecidos, foi feito em assembly, um código onde se testa cada uma das instruções feitas no trabalho. Tal código serviu como teste a partir de seus arquivos em formato **.bin**. Cada teste é relatado com a palavra teste + nome da instrução, por exemplo, no caso da instrução ADD o teste feito para testá-la tem o nome de “testeADD” e assim por diante. O mesmo encontra-se dentro da pasta src/asm, nomeado por “teste.asm”. O simulador feito funcionou bem para todos os testes feitos neste código assembly.

Referências Bibliográficas

[1] Vollmar, Kenneth, and Pete Sanderson. "MARS: an education-oriented MIPS assembly language simulator." ACM SIGCSE Bulletin. Vol. 38. No. 1. ACM, 2006.

[2] Roteiro “TRABALHO 2 - SIMULADOR MIPS”, fornecido pelo professor Marcelo Grandi Mandelli.