Trabalho 2 de Organização e Arquitetura de Computadores

Bruno Justino Garcia Praciano (13/0006912)

19 de setembro de 2017

Resumo

Esse relatório é referente ao trabalho 2 da disciplina OAC, que tem como a finalidade demonstrar como foi construido o simulador Mips em liguagem `C`.

1 INTRODUÇÃO

O trabalho consiste na elaboração de um simulador da arquitetura MIPS, como o MARS, em uma linguagem de alto nivel. O simulador deve implementar as seguintes funções: fetch(), busca da instrução; decode(), decodificação da instrução; execute(), execução da instrução. O programa binário a ser executado deve ser gerado a partir do MIPS. O simulador implementado deve ler os arquivos binarios contendo as instruções e os dados para sua memória e executá-lo. As instruções deste trabalho começam no endereço 0x00000000 e se encerram no endereço 0x00000044, enquanto os dados no endereço 0x00002000 e acabam no endereço 0x0000204c. A memória simulada é definida com o tamanho de 4KWords, ou seja, 16KBytes. Tomando como exemplo, o código dos números primos

2 Instruções Para Execução do Código

A cerca dos detalhes do código fonte, o programa foi compilado e executado através de um sistema ArchLinux x64, sem o uso de IDEs, apenas com o auxilio do editor de texto Atom. Ademais, o código foi escrito em linguagem `C` e compilado com o gcc version 7.2.0 (GCC).

- Abrindo um terminal e indo para a pasta onde encontra-se os arquivos.
- Efetuar o seguinte comando no Terminal:
  `gcc` trabalho2.c
  É necessário que os arquivos: funcoes.c, funcoes.h, data.bin e text.bin estejam no mesmo diretório.
- para executar o arquivo compilado, deve-se usar o seguinte comando: ./a.out

- Em relação aos testes das funções, estes ocorrem em base as instruções passadas pelo binário e, por causa disso, o usuário deve utilizar-se dos arquivos "text"e "data"que já vem inclusos na pasta compactada, podendo também modificar o código ASM ou criar um novo para ser executado pelo simulador - gerando os binários necessários.

3 Metodologia

Foram aproveitadas as funções utilizadas no trabalho 1 para realizar esse simulador, portanto será descrito apenas as novas funções implementadas para essa tarefa.

3.1 Registradores

Os registradores pc, ri foram declarados como unsigned, pois o pc armazena um endereço, e o ri, uma instrução. Já os registradores hi e lo foram declarados como signed, pois estes podem possuir contéudos númericos dentro de si. O banco de registradores reg é um vetor de 32 posições, representando os 32 registradores do MIPS, declarado com sinal, pois os registradores contém conteúdos númericos. Os campos da instrução: opcode, rs, rt, rd, shamt, funct foram declarados todos como unsigned. Para intruções do tipo I, um campo k16 também foi declarado como signed e para instruções do tipo J, outro campo, k26, foi declarado como unsigned.

3.2 Função Fetch()

A função fetch() le uma instrução da memória e a coloca em ri, atualizando o pc para apontar para a próxima instrução.

3.3 Decode()

A função decode() decodifica a instrução armazenada em ri em campos, sendo estes campos opcode, rs, rt, rd, shamt, funct, k16 e k26. Esta decodificação é realizada por meio de deslocamentos lógicos de bits, por exemplo, para extrair o campo opcode de uma instrução, sabendo que este campo possui 6 bits e a instrução inteira tem 32 bits, ri é deslocado 26 bits para a direita, resultando no campo opcode.

3.4 Execute()

A função execute() executa a instrução lida pela fetch() e decodificada pela decode(). De acordo com os campos da instrução opcode e funct, a instrução é identificada e executada, por exemplo, se o opcode obtido de ri for 0x23, em hexadecimal, a green sheet (ilustrada em parte na figura 3) do MIPS diz que a função a ser executada é um load word.

3.5 Step()

A função step() executa uma instrução do MIPS : fetch(pc), decode(), execute().

3.6 Run()

A função run() executa o programa até encontrar um syscall de encerramento, ou até o pc atingir o fim da memória.

3.7 dump mem(uint32 t start, uint32 t end, char format)

Imprime o conteúdo da memória a partir do endereço start até o endereço end, ambos em indice do vetor de memória. O formato pode ser em hexadecimal (padrão), ou decimal 'd'.

3.8 dump reg(char format)

Imprime o conteúdo do banco de registradores e dos registradores pc, hi, lo do simulador MIPS. O formato pode ser em hexadecimal (padrão), ou decimal 'd'.

4 Testes e Resultados

Os testes foram desenvolvidos no arquivo de testes, os resultados obtidos foram iguais aos resultados obtidos pelo MARS, portanto o programa desenvolvido foi considerado satisfatório.