CCF 252 - Organização de Computadores I
Trabalho Prático 03 - Caminho de Dados do RISC-V

Gabriel Miranda(3857)[1], Felipe Dias(3888)[2], Mariana de Souza(3898)[3]

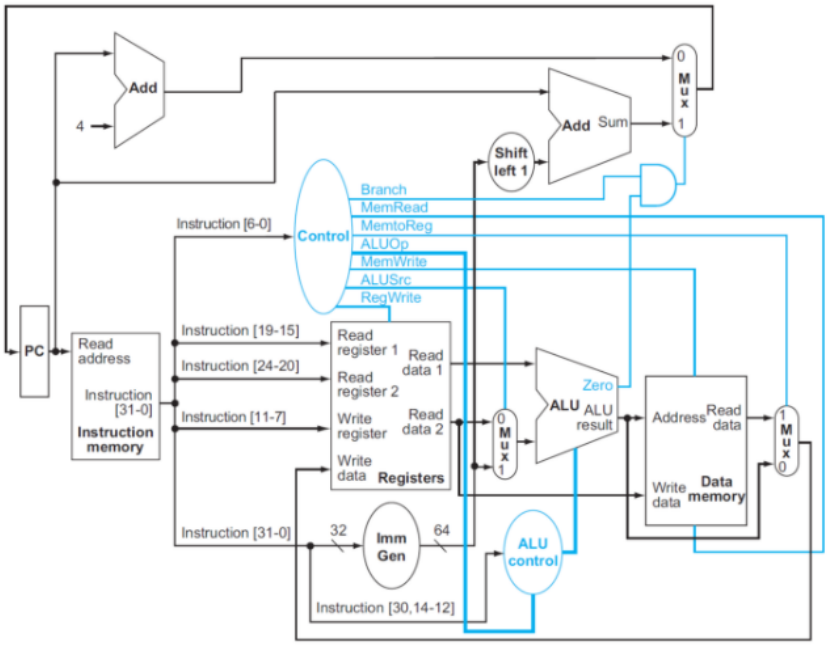
[1]Instituto de Ciências Exatas e Tecnológicas,
Universidade Federal de Viçosa, Florestal, MG, Brasil

1. Descrição

Neste trabalho, buscamos aplicar os conceitos do RISC-V apresentados nas aulas da disciplina de Organização de Computadores I, tendo como objetivo a implementação de um caminho de dados em verilog, nesse sentido, iremos usar a linguagem de descrição de hardware para descrever o caminho de dados, para assim gerar o gráfico de ondas e a saída em binário.

2. Caminho de dados RISC-V

Esse é o Caminho de dados que vai ser descrito pelo grupo em Verilog(Descrição de hardware). Esse trabalho prático baseia-se na implementação de uma versão simpificada do caminho de dados do RISC-V.



3. Desenvolvimento do Trabalho Prático

Buscando a implementação da lingaugem de descrição de hardware do Caminho de Dados, obtivemos algumas adversidades na implementação do mesmo, começamos elaborando os módulos de acordo com a **figura 1.**, com nossas respectivas pesquisas, e a ajuda dos monitores. Porém, ao decorrer do trabalho, fazendo os commits e ao executar

utilizando o arquivo ”make.py”encontrado no GitHub dos monitores, tivemos problemas com a aplicação no ModelSim, apresentando o erro como demonstra a **Figura 2.**

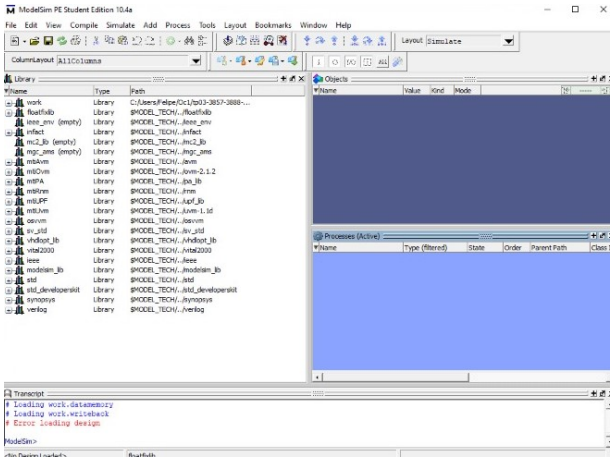


Figura 1.

Sendo assim, optamos pela a implementação de um novo código que diferentemente do posterior seria trabalhado com um ciclo único para a realização do Caminho de Dados RISC-V, e essa execução é a qual será apresentada nas seções abaixo.

4. Caminho de dados e seus módulos

Para um melhor entendimento do trabalho prático, foram criados arquivos para cada módulo. Nesse sentido, fica mais fácil de compreender como funciona o caminho de dados e também interpretar como funcionam os módulos. Em cada seção serão apresentados os nomes dos respectivos arquivos em verilog e suas descrições.

4.1. Arquivo entrada do caminho de dados

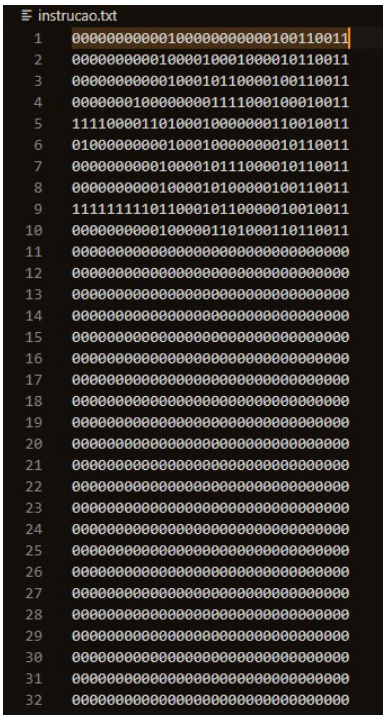


Figura 2.

A **figura 2.** é o arquivo saída do ultimo trabalho prático, esse arquivo vai ser a entrada de instruções contendo 32 bits, sendo lido linha por linha do arquivo pelo PC, para assim serem executadas a instruções do caminho de dados.

4.2. riscv.v

riscv.v é o modulo principal do trabalho prático, pois nele contém todos os outros módulos, que serão de suma importância para a compilação do caminho de dados, neste módulo foram declaradas algumas variáveis(fios), como menread, mentoreg, memtoreg, alusrc, regwrite, branch, aluop1, aluop0 e entre outras variáveis(fios) que são muito importantes para o caminho de dados, contendo também todas as inclusões dos módulos.

4.3. Control.v

A unidade de controle, que tem a instrução como entrada, é usada para determinar como definir as linhas de controle para as unidades funcionais e os dois dos multiplexadores.

4.4. mux.v

Um multiplexador que informa se o estágio deve pular para a próxima instrução ou realizar o desvio condicional. Se receber sinal de controle para realizar um BEQ (desvie se forem iguais), avalia se isAluOutputZero é igual a 1, ou seja, se a subtração dos dois parâmetros a serem avaliados resultou em zero, o que significa que eles são os mesmos. Utilizando 64 bits.

4.5. ALUcontrol.v

O módulo ALU control é o controlador usado para decidir qual operação deve ser realizada pela ALU. Para decidir isso, este controlador avalia o sinal de um fio do controle e os bits menos significativos da instrução.

4.6. ALU.v

Este módulo é usado para executar todas as operações reconhecidas pelo controle aritimético, sendo ADD, SUB, AND, OR, LD, SD e BEQ, que foram as instruções implementadas pelo grupo, todavia, também ele executa o processamento do caminho de dados de forma correta e precisa.

4.7. dataMemory.v

O módulo data memory tem como entrada a saída da ALU que entra como um endereço e a saída de um multiplexador, este módulo está lendo o arquivo intrucao.txt com 32 bits.

4.8. adder.v

adder é o módulo Add, que tem como saída a soma, sendo assim, esse bloco soma de 64 bits das duas entradas.

4.9. testbench.v

O módulo testbench tem como objetivo atualizar os clocks que serão apresentados posteriormente na ondas.

4.10. PC.v

O módulo PC contém todas as instruções para o Program Counter, contendo no total três módulos em um arquivo.v, PCIn, clk, reset e PCOut, fazem parte do primeiro módulo dentro do arquivo PC.v, esse primeiro módulo faz uma ligação com o fio do input e a saída do multiplexador, já o segundo módulo ”Soma4”tem como objetivo a implementação do add, somando quatro para a próxima instrução ser colocada, também contendo o módulo PCNext que irá somar as entradas.

4.11. Registers.v

Registers contém todos os registradores declarando clk, reset, RegWrite, WriteData, RS1, RS2, WriteRegister, ReadData1 e ReadData2, sendo declarados como fios como input ou output, inicializaado com um vetor Register recebendo a entrada zero(reset) e posteriormente atualizando com as bordas de clock.

4.12. branch.v

Módulo branch contém Branch, Zero e branchout, declara a quantidade de bits necessários para realizar a operação com a porta lógica and, recebendo a quantidade de bits, onde, caso o bit seja zero a saída será um e caso não seja ele irá receber um, sendo assim, formando a lógica da porta and.

5. Compilação caminho de dados riscv

A execução feita pelo terminal utilizando comandos do Icarus Verilog e também o Gtkwave, para gerar as ondas e saídas. Com os comandos, buscar o local onde se encontra os arquivos em verilog para serem executados, posteriomente inserir o comando `iverilog -o teste testbench.v`, para a compilação, após, `vvp teste` para o print no terminal e em sequência `gtkwave riscv.vcd` para serem gerada as ondas no Gtkwave. Como o apresentado na **Figura 3.**, **Figura 4.** e **Figura 5.**

```
C:\Users\MARIANA>cd C:\Users\MARIANA\Desktop\Riscv

C:\Users\MARIANA\Desktop\Riscv>iverilog -o teste testbench.v

C:\Users\MARIANA\Desktop\Riscv>vvp teste
```

Figura 3.

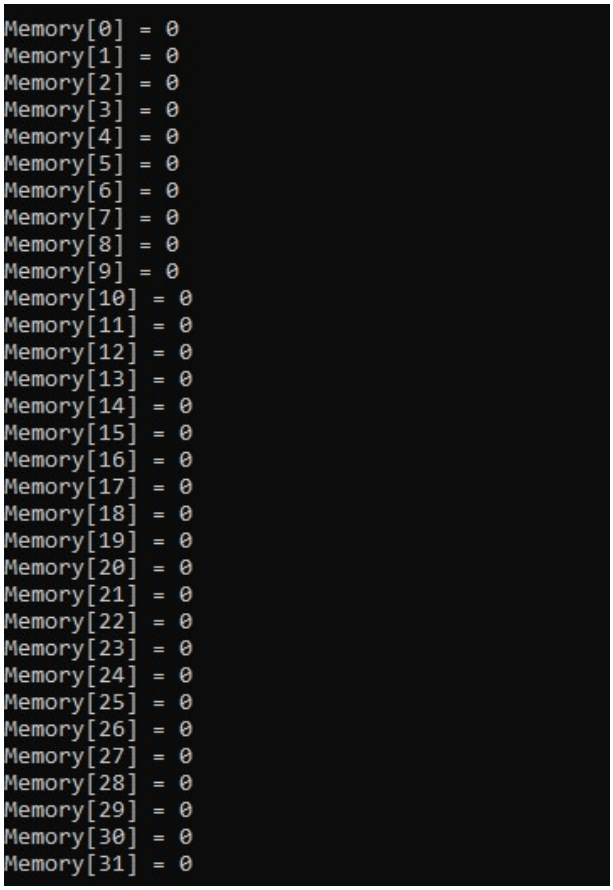


Figura 4.

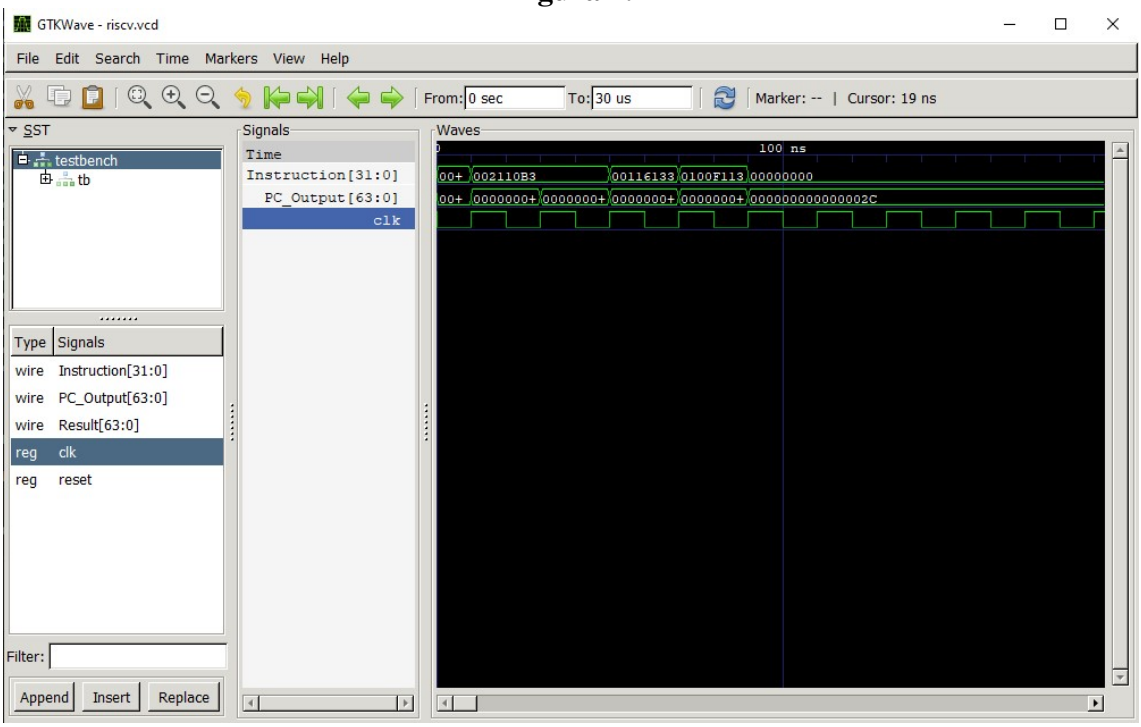


Figura 5.

6. Conclusão

Neste trabalho prático, conseguimos ter um maior parecer de como é o funcionamento no interior de um computador, todavia também tivemos mais entedimento de como é o andamento de um processador. No início da matéria achavamos que seria um pouco complicado essa parte de operações com hardware, entretanto esse nível que achavamos mais abstrato foi se decaindo aos poucos, com os passar da matéria, isso se deu pelos trabalhos práticos e execícios feitos durante as aulas de Organização de Computadores 1.

Um ponto importante a ser citado são as operações computacionais, pois no fim do trabalho prático foi perceptível que tudo que opera em um computador são apenas contas matemáticas, sendo elas, subtração, multiplicação, adição e operções lógicas computacionais. Nesse sentido, é possível ressaltar a linguagem que obtivemos um maior entendimento durante o trabalho prático, e foi necessária para sua execução, o verilog, com ele foi possível descrever o caminho de dados. Nele descrevemos cada parte do hardaware do caminho de dados e seus módulos.

Logo, é importante ressaltar o Visual Studio Code, pois foi nele que baixamos as extensões do verilog e obtivemos uma melhor compreensão do trabalho prático, no mais achamos a aplicação do RISC-V de extrema importância, pois com as aulas foi possível compreender e aprender mais sobre essa tecnólogia.

7. Referências:

- ModelSim
- Repositório GitHub
- Instalação de Python no linux
- Gtkwave
- Icarus Verilog
- Livro Computer Organization and Design RISC-V Edition: The Hardware Software Interface (The Morgan Kaufmann Series)