Caminho de Dados Simplificado do RISC-V
Trabalho Prático 02 - CCF 252

Gabriel Sanches Vinhal de Queiroz - 3748[1], **Vinícius Augusto Assis Ferreira - 4242**[2]

[1]Instituto de Ciências Exatas e Tecnológicas – Universidade Federal de Viçosa(UFV) - Florestal
Rodovia LMG 818, km 06 – 35.690-000 – Florestal – MG – Brazil

[2]Ciência da Computação – Universidade Federal de Viçosa - UFV
Florestal, M.G.

gabriel.sanches@ufv.br, vinicius.assis@ufv.br

Abstract. *This work consists of the implementation in synthesizable verilog language of a simplified version of the RISC-V data path that can execute the ADD, SUB, AND, OR, LD, SD, BEQ, LW, SW, XOR, ADDI, SLL, and BNE instructions. This code will also be used to implement on a DE2-115 FPGA (Cyclone IV, Altera).*

Resumo. *Este trabalho consiste na implementação em linguagem verilog sintetizável de uma versão simplificada do caminho de dados do RISC-V que consiga executar as instruções ADD, SUB, AND, OR, LD, SD, BEQ, LW, SW, XOR, ADDI, SLL e BNE. Esse código também será usado para implementar em uma FPGA DE2-115 (Cyclone IV, Altera).*

1. Introdução

A arquitetura RISC-V é uma abordagem moderna e aberta no campo da ciência da computação. Como uma Arquitetura de Conjunto de Instruções (ISA) flexível e simplificada, ela tem ganhado popularidade crescente. Ao seguir os princípios da Computação de Conjunto de Instruções Reduzido (RISC), o RISC-V utiliza um conjunto de instruções simples e um design eficiente, facilitando o desenvolvimento de processadores. Uma das características mais notáveis do RISC-V é sua natureza de código aberto. Com especificações e documentação disponíveis gratuitamente, essa arquitetura promove a colaboração e a inovação.

Em RISC-V, o caminho de dados é uma estrutura composta por unidades lógico-aritméticas responsáveis por executar diversas instruções. Essas unidades realizam operações como adição, subtração, desvios, leitura e escrita de dados, entre outras. Essas funções são essenciais para o processamento e execução de instruções dentro de um sistema computacional. Neste trabalho, foi pedido para utilizar o caminho de dados simplificado, como podemos ver logo abaixo.

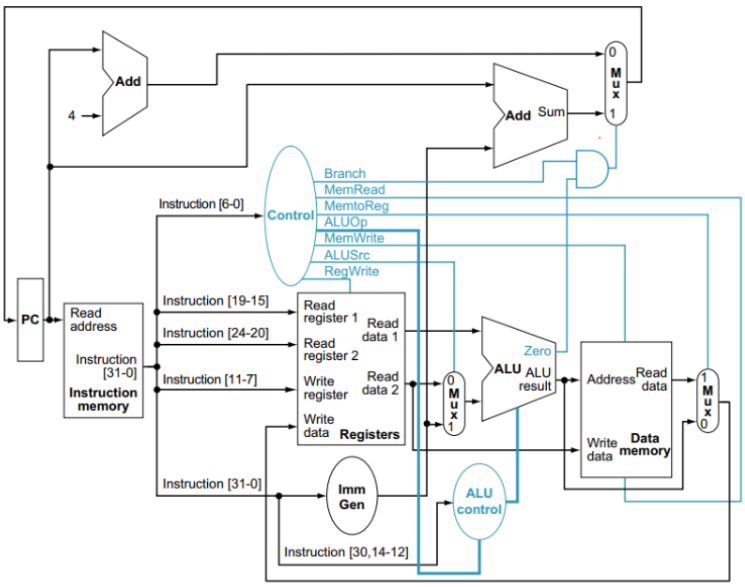


Figure 1. Caminho de dados simplificado

2. Desenvolvimento

2.1. Organização

Os arquivos ficaram organizados da seguinte forma:

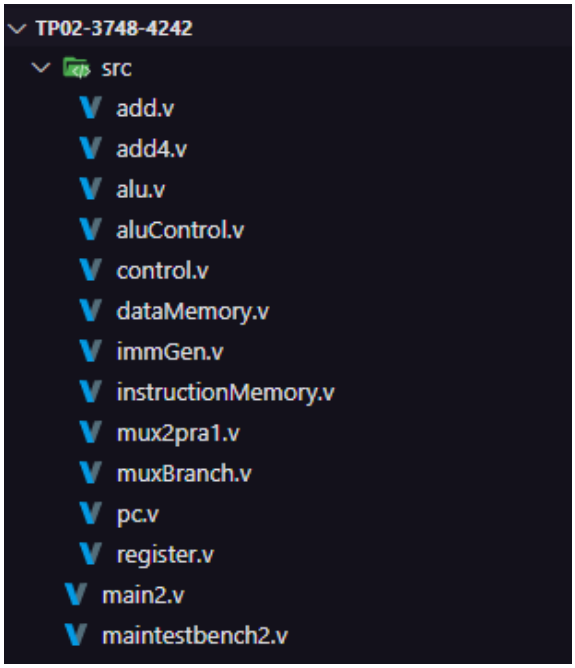


Figure 2. Estrutura dos arquivos

Módulos do código:

2.1.1. Program Counter (PC)

O program counter é um registro que contém o endereço da instrução atual que está sendo executada. Após a execução de uma instrução, o PC é atualizado para apontar para o próximo endereço de memória onde a próxima instrução está armazenada.

src > pc.v

You, há 23 horas | 1 author (You)

```
module pc (
    input wire clock,
    input wire reset,
    input wire [31:0] enderecoEntrada,
    output reg [31:0] enderecoSaida);

    always @(posedge clock) begin

        if (reset) begin
            enderecoSaida = 31'd0;
        end
        else begin
            enderecoSaida = enderecoEntrada;
        end
    end

endmodule
```

Figure 3. Módulo PC

2.1.2. add4

O ”add4” executa operações aritméticas para incrementar o PC para buscar a próxima instrução.

src > add4.v

You, há 33 minutos | 1 author (You)

```
module add4(
    input wire clock,
    input wire reset,
    input wire [31:0] saidaPC,
    output reg [31:0] saida
);

    always @(posedge clock) begin
        if (reset) begin
            saida = 32'b0;
        end
        else begin
            saida = saidaPC + 32'b1;
        end
    end
endmodule
```

Figure 4. Módulo add4

2.1.3. add

O ”add” executa operações aritméticas somando dois operandos, o primeiro é endereço do PC e o outro é o imediato gerado.

src > add.v

You, há 2 minutos | 1 author (You)

```v
module add(
    input wire clock,
    input reset,
    input wire [31:0] saidaPC,
    input wire [31:0] saidaImmGen,
    output reg [31:0] saida
);
    wire [31:0] mudancaIMM;

    assign mudancaIMM = (saidaImmGen <<< 1);

    always @(posedge clock) begin
        if (reset) begin
            saida = 32'b0;
        end
        else begin
            saida = saidaPC + (mudancaIMM);
        end
    end
endmodule
```

Figure 5. Módulo add

2.1.4. Tipos de Multiplexadores implementados

O ”mux2pra1” e o ”muxBranch” representa um multiplexador que seleciona uma das várias entradas com base em sinais de controle. No caminho de dados, um MUX é usado para selecionar a entrada apropriada para diferentes estágios do caminho de dados, como a seleção entre o PC e o endereço de destino da ramificação.

src > mux2pra1.v

You, há 23 horas | 1 author (You)

```v
module mux2pra1(
    input wire clock,
    input wire [31:0] entrada1,
    input wire [31:0] entrada2,
    input wire controle,
    output reg [31:0] saida);

    always @(posedge clock) begin

        if (controle == 0) begin
            saida <= entrada1;
        end

        if (controle == 1) begin
            saida <= entrada2;
        end       You, ontem • refator

    end

endmodule
```

Figure 6. Módulo mux2pra1

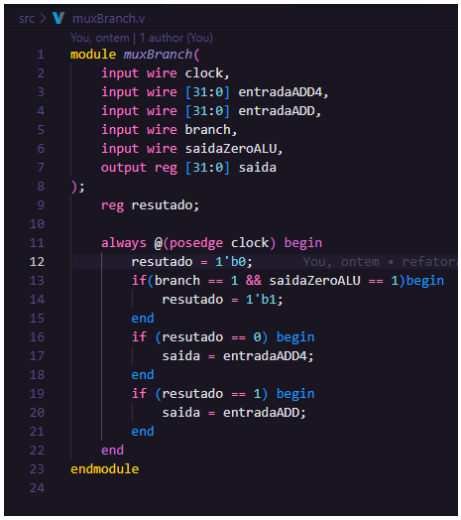


Figure 7. Módulo muxBranch

2.1.5. intructionMemory

O ”instructionMemory” é responsável por armazena as instruções do programa. Normalmente, ela é quem fornece as instruções ao processador com base no valor do PC.

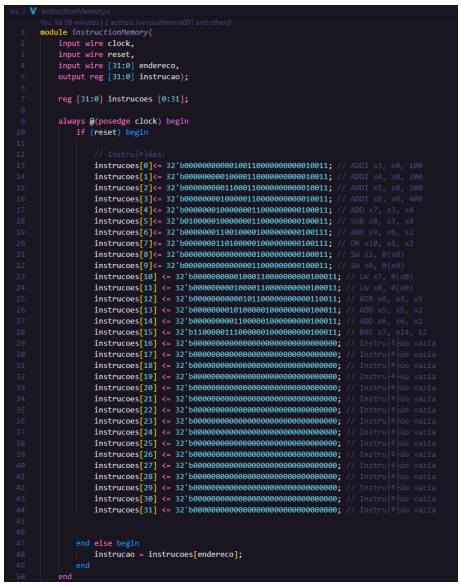


Figure 8. Módulo instructionMemory

2.1.6. ImmGen

O modulo ”immGen” extrai o valor imediato da instrução. Normalmente, ele é usado em instruções como ramificação, carregamento e armazenamento.

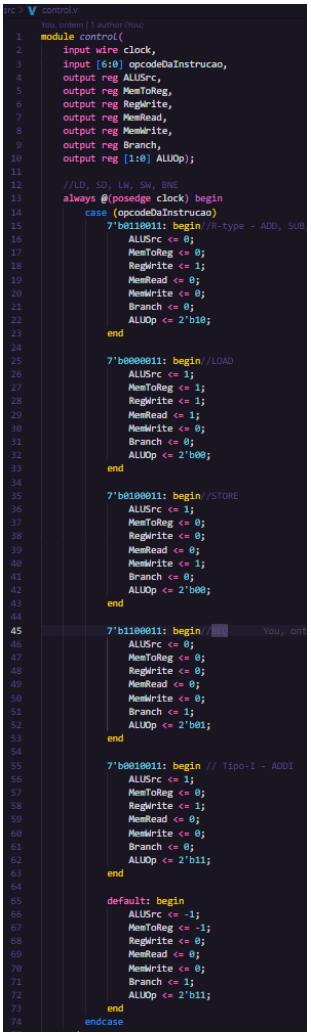
Figure 9. Módulo ImmGen

2.1.7. control

O módulo ”control” é responsável por gerar sinais que controlam o funcionamento dos componentes do caminho de dados de um processador. Ele recebe como entrada o código de operação (opcode) da instrução atual e produz sinais de controle que ativam ou desativam componentes específicos no caminho de dados.

Esses sinais de controle são essenciais para coordenar as operações dentro do processador. Eles indicam quais componentes devem estar ativos ou inativos em cada ciclo de clock, garantindo que a instrução correta seja executada e que os recursos necessários sejam utilizados adequadamente.

O módulo ”control” desempenha um papel fundamental na sincronização e coordenação do processador, permitindo que as instruções sejam executadas corretamente e controlando o fluxo de dados através dos diferentes componentes do caminho de dados.



```
src > V control.v
    You, ontem | 1 author (You)
1   module control(
2       input wire clock,
3       input [6:0] opcodeDaInstrucao,
4       output reg ALUSrc,
5       output reg MemToReg,
6       output reg RegWrite,
7       output reg MemRead,
8       output reg MemWrite,
9       output reg Branch,
10      output reg [1:0] ALUOp);
11
12      //LD, SD, LW, SW, BNE
13      always @(posedge clock) begin
14          case (opcodeDaInstrucao)
15              7'b0110011: begin//R-type - ADD, SUB
16                  ALUSrc <= 0;
17                  MemToReg <= 0;
18                  RegWrite <= 1;
19                  MemRead <= 0;
20                  MemWrite <= 0;
21                  Branch <= 0;
22                  ALUOp <= 2'b10;
23              end
24
25              7'b0000011: begin//LOAD
26                  ALUSrc <= 1;
27                  MemToReg <= 1;
28                  RegWrite <= 1;
29                  MemRead <= 1;
30                  MemWrite <= 0;
31                  Branch <= 0;
32                  ALUOp <= 2'b00;
33              end
34
35              7'b0100011: begin//STORE
36                  ALUSrc <= 1;
37                  MemToReg <= 0;
38                  RegWrite <= 0;
39                  MemRead <= 0;
40                  MemWrite <= 1;
41                  Branch <= 0;
42                  ALUOp <= 2'b00;
43              end
44
45              7'b1100011: begin//[illegible]      You, ont
46                  ALUSrc <= 0;
47                  MemToReg <= 0;
48                  RegWrite <= 0;
49                  MemRead <= 0;
50                  MemWrite <= 0;
51                  Branch <= 1;
52                  ALUOp <= 2'b01;
53              end
54
55              7'b0010011: begin // Tipo-I - ADDI
56                  ALUSrc <= 1;
57                  MemToReg <= 0;
58                  RegWrite <= 1;
59                  MemRead <= 0;
60                  MemWrite <= 0;
61                  Branch <= 0;
62                  ALUOp <= 2'b11;
63              end
64
65              default: begin
66                  ALUSrc <= -1;
67                  MemToReg <= -1;
68                  RegWrite <= 0;
69                  MemRead <= 0;
70                  MemWrite <= 0;
71                  Branch <= 1;
72                  ALUOp <= 2'b11;
73              end
74          endcase
```

Figure 10. Módulo control

2.1.8. Register

O módulo register vai armazenar os dados durante a execução das instruções destro dos registradores. Eles fornecem acesso rápido aos operandos para operações aritméticas e lógicas. No caminho de dados, os registradores são normalmente implementados como uma matriz e inicializada com zeros.

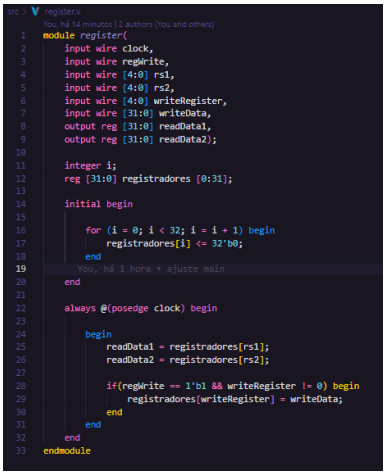


Figure 11. Módulo register

2.1.9. aluControl

O módulo ”aluControl” é responsável por gerar sinais de controle que determinam a operação a ser realizada pela Unidade de Lógica Aritmética (ALU). Ele recebe como entrada o código de função da instrução e produz sinais de controle que especificam qual operação aritmética ou lógica deve ser executada, como adição, subtração, operações lógicas E (AND), OU (OR), entre outras.

Esses sinais de controle são cruciais para configurar corretamente a ALU e direcionar suas operações de acordo com a instrução atual. Eles indicam à ALU qual operação específica deve ser realizada, garantindo que a instrução seja executada corretamente e produza o resultado desejado.

O módulo ”aluControl” desempenha um papel importante na coordenação e no controle das operações aritméticas e lógicas dentro do processador, permitindo que diferentes instruções sejam executadas de acordo com suas respectivas operações específicas.

Figure 12. Módulo aluControl

2.1.10. ALU

A ALU é responsável pela execução de operações aritméticas e lógicas nos operandos. Ela recebe a entrada do arquivo de register ou do ImmGen e executa a operação especificada com base nos sinais de controle gerados pelo controle da ALU.

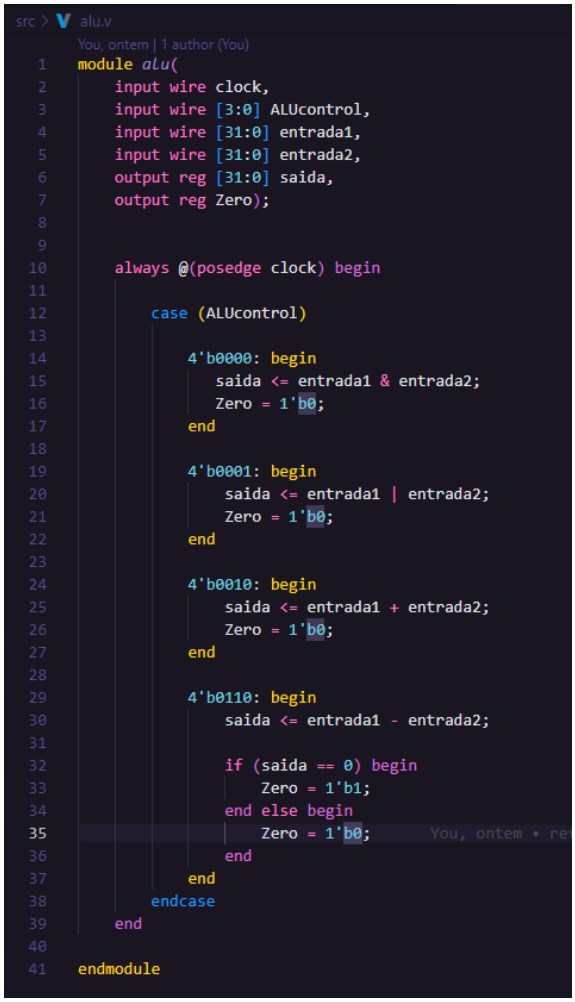


Figure 13. Módulo alu

2.1.11. DataMemory

O módulo ”dataMemory” é responsável por armazenar os dados que são acessados por instruções de leitura e escrita. Ele oferece operações de leitura e escrita para transferir dados entre o processador e a memória. O endereço utilizado para acessar a memória de dados geralmente é fornecido pela ALU (Unidade Lógica Aritmética) ou pelo gerador imediato.

Esse módulo atua como uma interface entre o processador e a memória, permitindo que o processador leia dados da memória ou escreva dados na memória conforme necessário. Quando um endereço de memória e um sinal de controle são recebidos,

o módulo ”dataMemory” realiza as operações adequadas para recuperar ou armazenar os dados na memória.

Figure 14. Módulo dataMemory

Esses componentes trabalham em conjunto no caminho de dados de um processador RISC-V para executar instruções. O PC (Program Counter) mantém o endereço da próxima instrução a ser buscada na memória de instruções. A Unidade de Controle decodifica a instrução buscada, gerando sinais de controle que são enviados para os outros componentes. O gerador imediato extrai o valor imediato presente na instrução, enquanto a ALU (Unidade Lógica Aritmética) executa operações aritméticas ou lógicas com base nos sinais de controle fornecidos pela Unidade de Controle da ALU. Os registradores armazenam operandos e resultados intermediários, e a memória de dados é utilizada para armazenar e recuperar dados conforme necessário.

De forma geral, o caminho de dados forma o núcleo central para a execução de instruções em um processador RISC-V, fornecendo os componentes essenciais para buscar, decodificar, executar e armazenar os resultados das instruções. Cada componente desempenha um papel específico, contribuindo para o funcionamento geral do processador.

3. Instruções de execução

Para a execução do programa, foi criado um makefile para facilitar, sendo importante ressaltar que é pré-requisito ter o ”Icarus Verilog” instalado na máquina. Para rodar o programa, basta digitar o comando no terminal: *”make all”* e, logo em seguida digitar o comando *”make run”*.

4. Conclusão

Em conclusão, a arquitetura RISC-V é uma arquitetura de conjunto de instruções moderna e aberta que oferece simplicidade, flexibilidade e personalização. Sua natureza de código aberto e escalabilidade a tornam uma opção atraente para estudantes e profissionais interessados em arquitetura de computadores. Esse trabalho também permitiu conhecimento prático dos estudos dados em aula.

5. Referências

Nesse trabalho prático foi usado como suporte bibliográfic0 o livro base da disciplina de Organização de Computadores I, o [Patterson and Hennessy 2021].

References

Patterson, D. A. and Hennessy, J. L. (2021). *Computer Organization and Design - RISC-V edition*. Patterson-Hennessy, 2ª edição edition.