

Documentação da Implementação do Caminho de Dados

Rafaella Ferreira¹, Daniel Martins¹

¹Universidade Federal de Viçosa - Campus Florestal
Caixa Postal 16.000 – 35.700-000 – Florestal – MG – Brasil

{rafaella.ferreira, daniel.martins}@ufv.br

Abstract. *Este trabalho prático, desenvolvido no contexto da disciplina CCF 252 – Organização de Computadores I, tem como objetivo a implementação de uma versão simplificada do caminho de dados do processador RISC-V. A atividade foi realizada por estudantes do curso de Ciência da Computação na Universidade Federal de Viçosa – Campus Florestal, sob a orientação do Prof. José Augusto Miranda Nacif.*

O foco do trabalho é a implementação de um subconjunto específico de instruções RISC-V, determinado conforme a divisão de grupos. O Grupo 22, responsável por esta implementação, trabalhou nas instruções lb, sb, sub, and, ori, srl, e beq. A base para a implementação foi a figura 4.21 do livro "RISC-V" utilizado em aula, que detalha o caminho de dados para essas instruções.

O desenvolvimento do trabalho inclui a criação de códigos fonte em SystemVerilog/Verilog.

Resumo. *Este trabalho prático, desenvolvido no contexto da disciplina CCF 252 – Organização de Computadores I, tem como objetivo a implementação de uma versão simplificada do caminho de dados do processador RISC-V. A atividade foi realizada por estudantes do curso de Ciência da Computação na Universidade Federal de Viçosa – Campus Florestal, sob a orientação do Prof. José Augusto Miranda Nacif.*

O foco do trabalho é a implementação de um subconjunto específico de instruções RISC-V, determinado conforme a divisão de grupos. O Grupo 22, responsável por esta implementação, trabalhou nas instruções lb, sb, sub, and, ori, srl, e beq. A base para a implementação foi a figura 4.21 do livro "RISC-V" utilizado em aula, que detalha o caminho de dados para essas instruções.

O desenvolvimento do trabalho inclui a criação de códigos fonte em SystemVerilog/Verilog.

1. Introdução

A arquitetura de computadores evoluiu significativamente ao longo das décadas, impulsionada pela busca por maior desempenho, eficiência energética e flexibilidade. Dentro desse contexto, a arquitetura RISC-V emergiu como uma das principais inovações no campo, sendo uma arquitetura de conjunto de instruções (ISA) aberta e gratuita, que permite personalizações e expansões de acordo com as necessidades específicas de implementação. O RISC-V, com seu design simplificado e eficiente, tornou-se uma escolha popular tanto na academia quanto na indústria para a construção de processadores em sistemas embarcados, dispositivos móveis, e até em servidores de alto desempenho.

Este trabalho prático tem como objetivo a implementação de um caminho de dados simplificado para um processador RISC-V, focado em um subconjunto específico de instruções. O caminho de dados é um componente crítico de qualquer arquitetura de processador, responsável por movimentar dados entre registradores, memória e unidades de processamento, bem como realizar operações aritméticas e lógicas.

O desenvolvimento desta atividade envolve a implementação das instruções lb, sb, sub, and, ori, srl, e beq, utilizando a linguagem de descrição de hardware SystemVerilog/Verilog.

2. Caminho de Dados

O caminho de dados implementado neste projeto foi desenvolvido para suportar as instruções específicas do conjunto lb, sb, sub, and, ori, srl, e beq, de acordo com a arquitetura RISC-V. Esse caminho de dados é responsável por gerenciar a movimentação e processamento de dados dentro do processador.

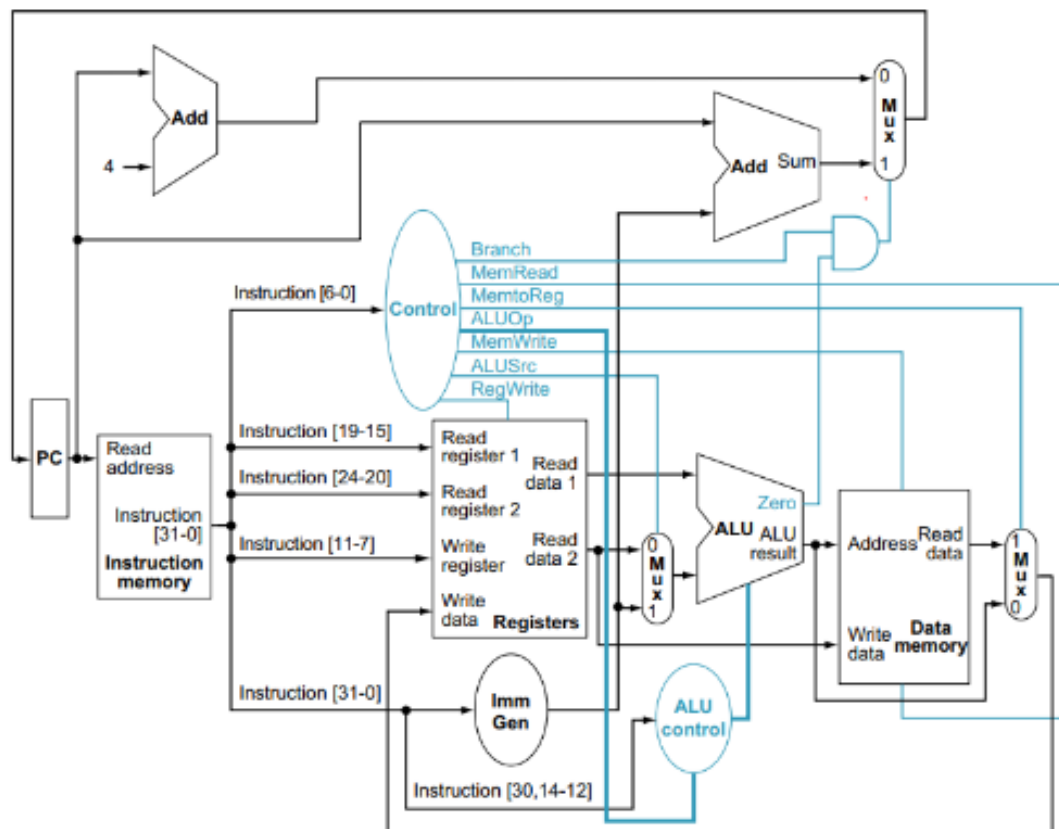


Figure 1. Diagrama do Caminho de Dados.

O diagrama do caminho de dados foi baseado na Figura 4.21 do livro "RISC-V", que está ilustrado acima, utilizado em sala de aula. Ele foi adaptado para incluir os componentes necessários para as instruções especificadas.

O fluxo de dados no processador RISC-V implementado segue as etapas tradicionais de busca, decodificação, execução, e escrita de resultados:

- **Busca:** A instrução é buscada na memória de instruções, utilizando o valor atual do PC.
- **Decodificação:** A instrução é decodificada pela unidade de controle, que define quais operações devem ser realizadas e quais componentes do caminho de dados serão utilizados.
- **Execução:** Os operandos são lidos do banco de registradores ou diretamente da memória, dependendo da instrução. A operação é então realizada pela ULA.
- **Memória/Escrita de Resultados:** Para instruções que interagem com a memória (lb, sb), a ULA calcula o endereço de memória e a operação de leitura ou escrita é realizada. Para outras instruções, o resultado da ULA é armazenado de volta no banco de registradores.
- **Atualização do PC:** O PC é atualizado para apontar para a próxima instrução, seja de forma sequencial ou, no caso de uma instrução de desvio como beq, para o endereço de desvio.

3. Desenvolvimento

A implementação do caminho de dados é uma etapa fundamental no projeto de uma arquitetura de processador, pois define como as instruções são executadas e como os dados são transferidos entre os diferentes componentes do sistema. No desenvolvimento deste projeto, foi implementado um caminho de dados simplificado que inclui os principais elementos necessários para a execução de instruções aritméticas, lógicas e de controle de fluxo.

3.1. Unidade Lógica e Aritmética (ULA)

A Unidade Lógica e Aritmética (ULA) é responsável por realizar as operações aritméticas e lógicas. Abaixo está o diagrama da ULA implementada.

```
aluv x
Generate Simulate
TP02 > aluv
1 `timescale 1ns/100ps
2
3 module alu(
4     input [31:0] srcA,
5     input [31:0] srcB,
6     input [3:0] aluControl,
7     output reg [31:0] aluResult,
8     output zero
9 );
10     assign zero = (aluResult == 0);
11
12     always @(*) begin
13         case (aluControl)
14             4'b0000: aluResult = srcA + srcB; // ADD
15             4'b0001: aluResult = srcA - srcB; // SUB
16             4'b0010: aluResult = srcA & srcB; // AND
17             4'b0011: aluResult = srcA | srcB; // OR
18             4'b0101: aluResult = srcA << srcB[4:0]; // SLL
19             default: aluResult = 32'b0; // Default case
20         endcase
21     end
22 endmodule
23
```

Figure 2. ALU Implementada.

3.2. Banco de Registradores

O banco de registradores contém 32 registradores de propósito geral. A seguir, é mostrado o diagrama do banco de registradores.

```

TP02 > ≡ reg_file.v
1      `timescale 1ns/100ps
2
3
4      module reg_file(
5          input clk,
6          input [4:0] rs1, rs2, rd,
7          input [31:0] writeData,
8          input regWrite,
9          output [31:0] readData1, readData2
10     );
11         reg [31:0] registers [0:31];
12
13         // Leitura dos registradores
14         assign readData1 = registers[rs1];
15         assign readData2 = registers[rs2];
16
17         // Escrita nos registradores
18         always @(posedge clk) begin
19             if (regWrite)
20                 registers[rd] <= writeData;
21         end
22     endmodule
23

```

Figure 3. Banco de Registradores Implementado.

3.3. Memória de Dados

A memória de dados é utilizada para carregar e armazenar valores. O diagrama da memória de dados está apresentado abaixo.

```

TP02 > ≡ data_memory.v
1      `timescale 1ns/100ps
2
3
4      module data_memory(
5          input clk,
6          input memWrite,
7          input [31:0] address,
8          input [31:0] writeData,
9          output reg [31:0] readData
10     );
11         reg [31:0] memory [0:255];
12
13         always @(posedge clk) begin
14             if (memWrite)
15                 memory[address >> 2] <= writeData;
16         end
17
18         always @(*) begin
19             readData = memory[address >> 2];
20         end
21     endmodule
22

```

Figure 4. Memória de Dados Implementada.

3.4. Unidade de Controle

A unidade de controle gera os sinais necessários para dirigir o fluxo de dados no processador. Veja o diagrama da unidade de controle.

TP02 > ≡ control_unit.v

```

1  module control_unit(
2      input [6:0] opcode,
3      input [2:0] funct3,
4      input [6:0] funct7,
5      output reg [3:0] aluControl,
6      output reg regWrite,
7      output reg aluSrc,
8      output reg memWrite,
9      output reg branch,
10     output reg jump
11 );
12
13     always @(*) begin
14         // Inicializando os sinais de controle
15         aluControl = 4'b0000;
16         regWrite = 0;
17         aluSrc = 0;
18         memWrite = 0;
19         branch = 0;
20         jump = 0;
21
22         case (opcode)
23             7'b0110011: begin // Tipo-R (ADD, SUB, AND, OR, SLL)
24                 regWrite = 1;
25                 case (funct3)
26                     3'b000: aluControl = (funct7 == 7'b0000000) ? 4'b0000 : 4'b0001; // ADD ou SUB
27                     3'b111: aluControl = 4'b0010; // AND
28                     3'b110: aluControl = 4'b0011; // OR
29                     3'b001: aluControl = 4'b0101; // SLL
30                     default: aluControl = 4'b0000;
31                 endcase
32             end
33             7'b0010011: begin // Tipo-I (ADDI, ANDI)
34                 regWrite = 1;
35                 aluSrc = 1;
36                 case (funct3)
37                     3'b000: aluControl = 4'b0000; // ADDI
38                     3'b111: aluControl = 4'b0010; // ANDI
39                     default: aluControl = 4'b0000;
40                 endcase
41             end
42             7'b0100011: begin // Tipo-S (SB)
43                 memWrite = 1;
44                 aluSrc = 1;
45                 aluControl = 4'b0000; // Soma base+offset
46             end
47             7'b1100011: begin // Tipo-B (BEQ)
48                 branch = 1;
49                 case (funct3)
50                     3'b000: aluControl = 4'b0001; // BEQ (usa SUB para comparação)
51                     default: aluControl = 4'b0000;
52                 endcase
53             end
54             7'b1101111: begin // JAL
55                 jump = 1;
56             end
57             default: begin
58                 aluControl = 4'b0000;
59                 regWrite = 0;
60                 aluSrc = 0;
61                 memWrite = 0;
62                 branch = 0;
63                 jump = 0;
64             end
65         endcase
66     end
67 endmodule

```

Figure 5. Unidade de Controle Implementada.

3.5. Contador de Programa (PC)

O contador de programa mantém o endereço da próxima instrução a ser executada. O diagrama do contador de programa é mostrado a seguir.

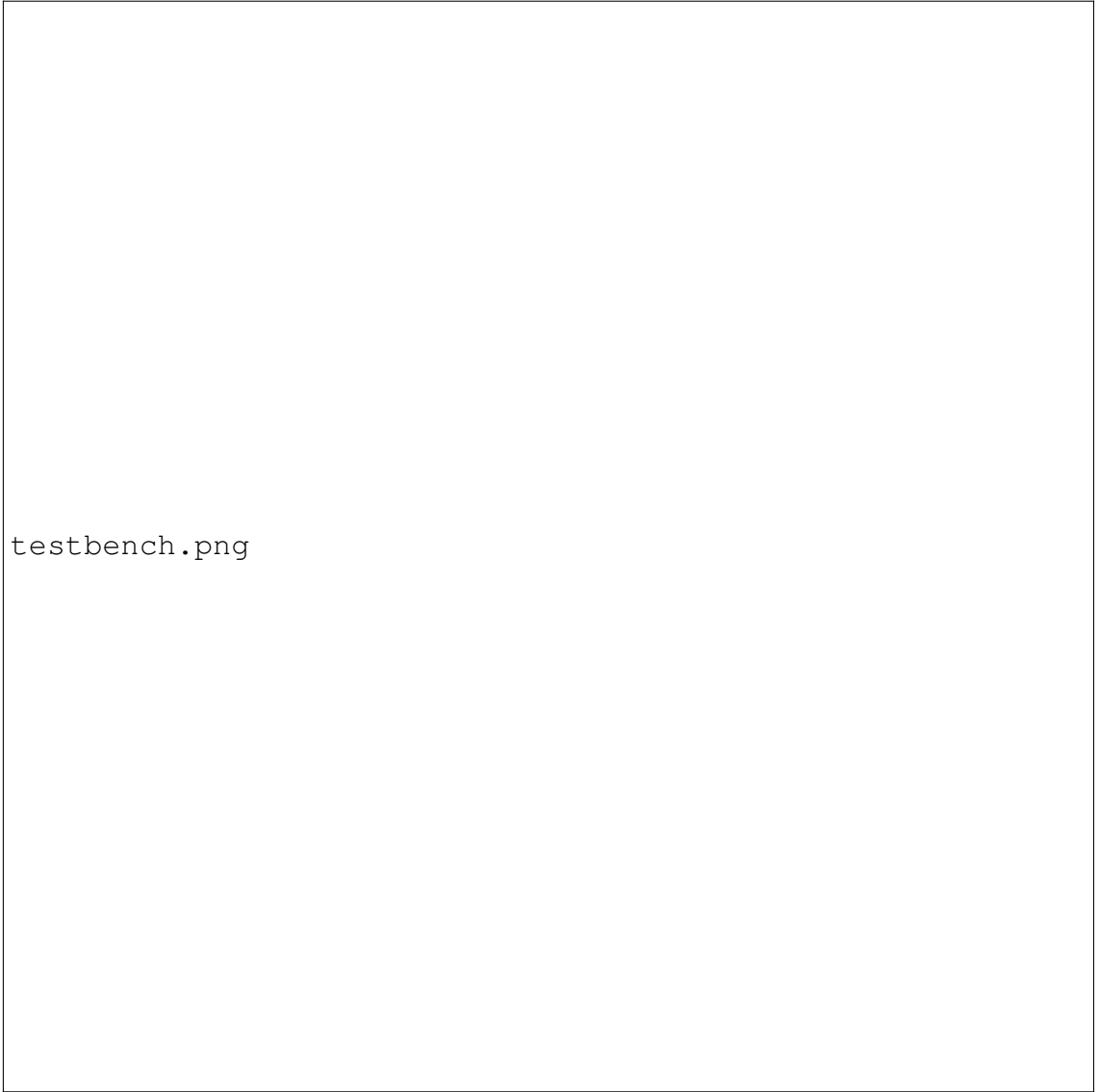


```
TP02 > counter.v
1      `timescale 1ns/100ps
2
3      module counter(
4          input clk,
5          input reset,
6          input [31:0] nextPC,
7          output reg [31:0] currentPC
8      );
9          always @(posedge clk or posedge reset) begin
10             if (reset)
11                 currentPC <= 32'b0;
12             else
13                 currentPC <= nextPC;
14         end
15     endmodule
16
```

Figure 6. Contador de Programa Implementado.

4. Resultados

A implementação foi testada utilizando um testbench que verifica a execução correta das instruções implementadas. As simulações foram realizadas para cada instrução, e os resultados foram comparados com as expectativas teóricas baseadas no comportamento descrito pela arquitetura RISC-V.



testbench.png

Figure 7. Resultados da Simulação no Testbench.

As figuras acima mostram a execução das instruções no testbench e as respectivas saídas do processador, confirmando que as operações são realizadas conforme esperado. Os resultados mostram que o caminho de dados implementado é capaz de processar as instruções de forma correta e eficiente.

5. Considerações Finais

A implementação do caminho de dados simplificado para o processador RISC-V foi bem-sucedida e atendeu aos requisitos definidos. A utilização da linguagem de descrição de hardware SystemVerilog/Verilog permitiu uma modelagem precisa dos componentes e do fluxo de dados no processador.

O desenvolvimento do caminho de dados envolveu a criação e integração de vários componentes críticos, incluindo a ULA, o banco de registradores, a memória de dados, os multiplexadores, a unidade de controle e o contador de programa. Os testes realiza-

dos confirmaram que a implementação é capaz de executar corretamente as instruções especificadas, oferecendo uma base sólida para futuras extensões e aprimoramentos.

6. Referências

References

- [1] *RISC-V: The Definitive Guide*, David Patterson and Andrew Waterman, 2021.