



Universidade Federal
do Rio de Janeiro
Escola Politécnica

Graduação em Engenharia de Computação e informação



Prática processador RISC-V Arquitetura de computadores

Beatrix Lima (123254085)
Daniely de Almeida Soares (125085115)
Fábio Sales (122088960)
Filipe Almeida Barbosa (125100347)
Nicholas Costa (122051214)

Professor: Diego Leonel Cadette Dutra

Rio de Janeiro, 2025

Conteúdo

| | | |
|----------------|--|----------|
| 1 | Introdução | 2 |
| 2 | RISC-V | 2 |
| 2.1 | Para que serve o RISC-V | 2 |
| 2.2 | Organização interna | 2 |
| 2.3 | Operação do RISC-V | 3 |
| 3 | Implementação do projeto | 3 |
| 3.1 | Especificação do Pipeline | 4 |
| 3.2 | Conjunto de Instruções Implementado | 4 |
| 3.3 | Organização do Projeto | 5 |
| 3.4 | Reset e Memórias Assíncronas | 5 |
| 3.5 | Validação e Depuração | 5 |
| 4 | Elaboração do projeto | 5 |
| 4.1 | PC e Instruction Memory | 5 |
| 4.2 | Alu (Unidade lógica e Aritmética) | 6 |
| 4.3 | Instruções de Memória e Memória de Dados | 6 |
| 4.4 | Componentes Auxiliares | 7 |
| 5 | Código | 7 |
| ANEXO-A | <i>-instruction encoding</i> | 8 |

1 Introdução

Este trabalho apresenta o projeto e a implementação de uma CPU RISC-V RV32I de 32 bits, baseada em uma microarquitetura pipeline com 5 estágios, conforme especificado no enunciado da disciplina. A implementação foi pensada para o simulador Digital, utilizando blocos descritos em Verilog, com memórias de instruções e dados operando palavra por ciclo e com possibilidade de carga de dados assíncrona, além de sinal explícito de reset. A CPU implementa apenas o pipeline de inteiros, sem modo supervisor (S-Mode), suportando exclusivamente o conjunto de instruções exigido.

2 RISC-V

O RISC-V é uma arquitetura de conjunto de instruções (ISA – Instruction Set Architecture) aberta e livre, baseada nos princípios de Reduced Instruction Set Computer (RISC). Seu principal objetivo é fornecer uma ISA simples, modular e extensível, permitindo implementações acadêmicas, industriais e de pesquisa sem custos de licenciamento. No contexto deste projeto, utiliza-se o subconjunto RV32I, que define uma arquitetura de 32 bits com instruções inteiras básicas, servindo como base para processadores mais completos.

2.1 Para que serve o RISC-V

O RISC-V é utilizado para o desenvolvimento de processadores e sistemas computacionais em diferentes níveis de complexidade, incluindo: Ensino e pesquisa em arquitetura de computadores Processadores embarcados e microcontroladores Sistemas de alto desempenho e aceleradores Prototipagem de novas arquiteturas Neste trabalho, o RISC-V serve como plataforma educacional, permitindo compreender profundamente o funcionamento interno de uma CPU, incluindo pipeline, controle, execução de instruções e acesso à memória.

2.2 Organização interna

Uma CPU RISC-V é construída a partir de diversos blocos fundamentais de hardware digital, tais como: Contador de Programa (PC) Memória de Instruções Banco de Registradores Unidade Lógica e Aritmética (ALU) Unidade de Controle Memória de Dados No modelo adotado, esses blocos são organizados em uma microarquitetura pipeline de 5 estágios, permitindo que múltiplas instruções sejam processadas simultaneamente, aumentando o desempenho. A ISA RISC-V define apenas o que as instruções fazem, enquanto a implementação do pipeline define como essas instruções são executadas em hardware.

2.3 Operação do RISC-V

O RISC-V opera executando, sequencialmente, as instruções armazenadas na memória de instruções. Cada instrução passa pelos estágios do pipeline:

1. Busca da instrução (IF)
2. Decodificação e leitura de registradores (ID)
3. Execução na ALU (EX)
4. Acesso à memória (MEM)
5. Escrita do resultado (WB)

O programador escreve códigos em assembly ou em linguagens de alto nível, que são compilados para instruções RISC-V. Essas instruções são então executadas pelo processador conforme a microarquitetura implementada. No simulador Digital, a CPU é usada carregando-se previamente a memória de instruções e dados, observando-se o comportamento interno dos sinais durante a execução.

3 Implementação do projeto

Para a implementação do projeto foram adotadas as seguintes premissas:

- Implementação do subconjunto RV32I, limitado às instruções exigidas no enunciado
- Arquitetura pipeline de 5 estágios, sem execução fora de ordem
- Ausência de modo supervisor (sem S-Mode)
- Implementação apenas do pipeline de inteiros, sem ponto flutuante
- Não implementação de tratamento completo de hazards (assumindo programas simples ou inserção de NOPs)
- Memórias de instruções e dados acessadas por palavra
- Suporte a reset global para inicialização do estado interno da CPU
- Uso exclusivo de descrições estruturais em Verilog, evitando blocos combinacionais multi-bit do Digital
- Foco em clareza, modularidade e facilidade de depuração

Essas premissas permitem uma implementação funcional, didática e aderente aos objetivos da disciplina, mantendo a complexidade controlada.

3.1 Especificação do Pipeline

A CPU foi organizada nos seguintes 5 estágios clássicos:

- IF (Instruction Fetch) – Busca da instrução na memória de instruções
- ID (Instruction Decode) – Decodificação e leitura do banco de registradores
- EX (Execute) – Execução na ALU e cálculo de desvios
- MEM (Memory Access) – Acesso à memória de dados
- WB (Write Back) – Escrita do resultado no banco de registradores

Entre cada estágio há registradores de pipeline síncronos.

3.2 Conjunto de Instruções Implementado

Conforme solicitado, são suportadas as seguintes instruções:

- Aritméticas: add, addi, sub
- Lógicas: and, andi, or, ori, xor, xori
- Deslocamentos: sll, slli, srl, srli
- Memória: lw, sw
- Controle de fluxo: jal, jalr, beq, bne
- Imediato superior: lui

| Name (Field size) | 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | Comments |
|----------------------|----------------|-------------------------|--------|--------|---------------|--------|-------------------------------|
| R-type | funct7 | rs2 | rs1 | funct3 | rd | opcode | Arithmetic instruction format |
| I-type | immed[11:0] | | rs1 | funct3 | rd | opcode | Loads & immediate arithmetic |
| S-type | immed[11:5] | rs2 | rs1 | funct3 | immed[4:0] | opcode | Stores |
| SB-type | immed[12,10:5] | rs2 | rs1 | funct3 | immed[4:1,11] | opcode | Conditional branch format |
| UJ-type | | immed[20,10:1,11,19:12] | | | rd | opcode | Unconditional jump format |
| U-type | | immed[31:12] | | | rd | opcode | Upper immediate format |

Figura 1: Tipos de instrução

Fonte: PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design: The Hardware/Software Interface*. RISC-V Edition. 2021, p. 249.

Na figura acima, os tipos de instrução podem ser vistos. Cada tipo agrupa e lê os bites da instrução recebida de maneira diferente, como pode ser observado. No anexo ANEXO A -, pode-se observar a qual grupo cada instrução pertence

3.3 Organização do Projeto

O projeto foi dividido em módulos pequenos, evitando blocos combinacionais multi-bit do Digital (especialmente na ALU), conforme recomendado. Principais módulos:

- pc – Contador de programa
- Instruction Memory – Memória de instruções (ROM do Digital)
- dmem – Memória de dados (RAM)
- regfile – Banco de registradores
- alu – Unidade lógica e aritmética
- control – Unidade de controle
- imm_gen – Gerador de imediatos
- pipeline_regs – Registradores de pipeline

3.4 Reset e Memórias Assíncronas

- O sinal reset zera o PC e os registradores de pipeline.
- As memórias de instruções e dados foram configuradas para permitir carregamento assíncrono no Digital, mantendo o estado interno da CPU durante a carga.

3.5 Validação e Depuração

Os módulos foram projetados para permitir inspeção de sinais internos, como:

- Saída da ALU
- Resultado do comparador de branches
- Conteúdo dos registradores
- Flags internas

Isso facilita a depuração no simulador Digital.

4 Elaboração do projeto

Os componentes acima, além de outros componentes auxiliares, foram escritos em Verilog, e adicionados ao Digital. As conexões foram feitas no próprio Digital, e utilizou-se os módulos de memória por ele disponibilizados.

4.1 PC e Instruction Memory

O componente do *pc*(*program counter*) foi escrito em Verilog, e a *Instruction Memory* usa um módulo ROM do Digital, com 10 bits de endereço e 32 bits de dados. Com isso, há $1024(2^{10})$ bytes, o que corresponde a 256 palavras de 4 bytes cada.

4.2 Alu (Unidade lógica e Aritmética)

A ALU (Arithmetic Logic Unit) é o módulo responsável por executar as operações aritméticas, lógicas e de deslocamento do processador. No projeto desenvolvido, a ALU foi implementada em Verilog de forma totalmente combinacional e modular, evitando o uso de blocos combinacionais multi-bit do Digital, conforme as recomendações da prática.

Além das operações escalares tradicionais do RISC-V, esta ALU também suporta operações vetoriais, permitindo a execução de operações paralelas sobre subpalavras de 8 e 16 bits dentro de um registrador de 32 bits.

A ALU possui duas entradas de dados de 32 bits, a e b, que representam os operandos da operação a ser realizada. O funcionamento da unidade é controlado pelo sinal aluctrl, composto por 6 bits, responsável por definir tanto o modo de operação quanto a instrução executada. Como saída, a ALU fornece o sinal result, de 32 bits, que contém o resultado da operação selecionada, além do sinal auxiliar zero, que indica se o resultado produzido é igual a zero e é utilizado principalmente em instruções de desvio condicional.

O sinal de controle aluctrl é dividido em dois campos distintos. Os dois bits mais significativos (aluctrl[5:4]) definem o modo de operação da ALU, podendo selecionar operações escalares de 32 bits (00), operações vetoriais com quatro elementos de 8 bits (01) ou operações vetoriais com dois elementos de 16 bits (10). Já os quatro bits menos significativos (aluctrl[3:0]) especificam a operação aritmética ou lógica a ser executada dentro do modo selecionado, como soma, subtração, operações lógicas e deslocamentos. Essa organização permite uma implementação flexível e facilita a expansão do conjunto de operações suportadas pela ALU.

O funcionamento interno implementado funciona como um circuito combinacional, descrito por um bloco always @(*), no qual o resultado é recalculado sempre que há alteração nas entradas ou nos sinais de controle. Internamente, o funcionamento é baseado na decodificação do sinal aluctrl, que seleciona o modo de operação (escalar ou vetorial) e, em seguida, a operação específica a ser executada.

No modo escalar, a ALU realiza operações aritméticas, lógicas e de deslocamento sobre os 32 bits completos dos operandos. Nos modos vetoriais, os operandos de 32 bits são particionados em subpalavras de 8 ou 16 bits, e as operações são executadas em paralelo sobre cada parte, com os resultados parciais sendo posteriormente concatenados para formar o valor final. Ao final da operação, o sinal zero é gerado por uma comparação direta do resultado, sendo utilizado para decisões de desvio condicional.

4.3 Instruções de Memória e Memória de Dados

No processador desenvolvido, as instruções de memória são responsáveis pelo acesso à memória de dados, seguindo o modelo load/store da arquitetura RISC-V. Nesse modelo, apenas instruções específicas realizam acesso à memória, enquanto as demais operam exclusivamente sobre registradores.

As instruções de leitura (load) utilizam a ALU para calcular o endereço efetivo, somando o conteúdo de um registrador base a um valor imediato gerado pelo módulo immgen. O endereço calculado é então utilizado para acessar a memória de dados (dmem), retornando o valor solicitado, que pode ser escrito no banco de registradores. De forma análoga, as instruções de escrita (store) também utilizam a ALU para o cálculo do endereço efetivo, armazenando na memória o valor proveniente de um registrador.

A memória de dados foi implementada utilizando o módulo de RAM disponibilizado pelo Digital e possui um tamanho de 2 elevado a 6 posições, totalizando 256 palavras.

Esse tamanho mostrou-se adequado e suficiente para os objetivos do projeto, permitindo o armazenamento de dados e resultados intermediários necessários à execução dos programas testados, sem aumentar desnecessariamente a complexidade do sistema.

O controle das operações de leitura e escrita, bem como os sinais de habilitação da memória, é realizado pela unidade de controle, a partir da decodificação da instrução em execução, garantindo o correto funcionamento das instruções de acesso à memória no processador.

4.4 Componentes Auxiliares

Como componentes auxiliares, foram implementados:

- Unidade de controle : responsável pelo controle central
- ALU control : Responsável pelo controle da ALU
- Hazard Unit : Responsável por lidar com problemas de hazard, especificamente, com instruções de uso após uma instrução de leitura
- Forward Unit : Faz o fowarding, isso é, encaminhamento dos valores
- Pipeline Control: Age conjuntamente a Hazard Unit, para casos de *stall*

5 Código

O código para este trabalho está disponibilizado no github, no link:

https://github.com/fillipeab/Entrega_RISC_V.git

A escolha para isso se deve a complexidade e a necessidade de compartilhamento do código entre os membros do grupo.

ANEXO A - RISC-V *instruction encoding*

Autor(es): David A. Patterson e John L. Hennessy

Título: *Computer Organization and Design: RISC-V Edition*

Página reproduzida: 248

| Format | Instruction | Opcode | Funct3 | Funct6/7 |
|---------|-------------|---------|--------|----------|
| R-type | add | 0110011 | 000 | 0000000 |
| | sub | 0110011 | 000 | 0100000 |
| | sll | 0110011 | 001 | 0000000 |
| | xor | 0110011 | 100 | 0000000 |
| | srl | 0110011 | 101 | 0000000 |
| | sra | 0110011 | 101 | 0000000 |
| | or | 0110011 | 110 | 0000000 |
| | and | 0110011 | 111 | 0000000 |
| | lrd | 0110011 | 011 | 0001000 |
| | scd | 0110011 | 011 | 0001100 |
| I-type | lb | 0000011 | 000 | n.a. |
| | lh | 0000011 | 001 | n.a. |
| | iw | 0000011 | 010 | n.a. |
| | id | 0000011 | 011 | n.a. |
| | ibu | 0000011 | 100 | n.a. |
| | ihu | 0000011 | 101 | n.a. |
| | iuw | 0000011 | 110 | n.a. |
| | addi | 0010011 | 000 | n.a. |
| | slli | 0010011 | 001 | 000000 |
| | xori | 0010011 | 100 | n.a. |
| | srlti | 0010011 | 101 | 000000 |
| | srai | 0010011 | 101 | 010000 |
| | ori | 0010011 | 110 | n.a. |
| | andi | 0010011 | 111 | n.a. |
| | jalr | 1100111 | 000 | n.a. |
| S-type | sb | 0100011 | 000 | n.a. |
| | sh | 0100011 | 001 | n.a. |
| | sw | 0100011 | 010 | n.a. |
| | sd | 0100011 | 111 | n.a. |
| SB-type | beq | 1100111 | 000 | n.a. |
| | bne | 1100111 | 001 | n.a. |
| | blt | 1100111 | 100 | n.a. |
| | bge | 1100111 | 101 | n.a. |
| | bltu | 1100111 | 110 | n.a. |
| | bgeu | 1100111 | 111 | n.a. |
| U-type | lui | 0110111 | n.a. | n.a. |
| UJ-type | jal | 1101111 | n.a. | n.a. |

FIGURE 2.18 RISC-V instruction encoding.

248

Figura 2: Página 248 contendo a tabela de formatos de instrução
Digitalização da página para referência educacional

Propósito educacional: Esta digitalização é utilizada exclusivamente para fins didáticos neste trabalho acadêmico, sem intenção comercial.