

# **LASER-RISCV: Implementação e Simulação de Processador Baseado na Arquitetura RISC V**

Ewerton Cavalcanti dos Santos



CENTRO DE INFORMÁTICA  
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2023



Ewerton Cavalcanti dos Santos

# LASER-RISCV: Implementação e Simulação de Processador Baseado na Arquitetura RISC V

Monografia apresentada ao curso Engenharia da Computação  
do Centro de Informática, da Universidade Federal da Paraíba,  
como requisito para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Ewerton Monteiro Salvador  
Coorientador: Alisson Vasconcelos de Brito

Junho de 2023

**Catalogação na publicação  
Seção de Catalogação e Classificação**

S2371 Santos, Ewerton Cavalcanti Dos.  
Laser-Riscv: implementação e simulação de  
processador baseado na arquitetura Risc V / Ewerton  
Cavalcanti Dos Santos. - João Pessoa, 2023.  
59 f. : il.

Orientação: Ewerton Salvador.  
Coorientação: Alisson Brito.  
Monografia (Graduação) - UFPB/CI.

1. RISC-V. 2. LOGISIM. 3. Arquitetura de  
computadores. 4. Microarquitetura. I. Salvador,  
Ewerton. II. Brito, Alisson. III. Título.

UFPB/CI

CDU 004.2

*”Todo mundo é gênio. Mas, se você julgar um peixe por sua capacidade de subir em uma árvore, ele vai passar a vida toda acreditando que é burro”, Autor Desconhecido.*



## DEDICATÓRIA

Querida Sarah Thárcylla,

Hoje, ao concluir esta etapa tão importante da minha jornada acadêmica, sinto-me imensamente grato por poder compartilhar este momento especial com você. É com grande alegria e amor que dedico este trabalho de conclusão de curso a você, minha inspiração constante e companheira de todas as horas.

Desde que nos conhecemos, você tem sido meu apoio incondicional, incentivando-me a seguir em frente mesmo nos momentos mais desafiadores. Seu amor, paciência e compreensão são verdadeiros presentes que fortalecem minha determinação e confiança.

Ao longo desses anos, você esteve ao meu lado, celebrando minhas vitórias e acolhendo-me em meus momentos de dúvida. Seu apoio inabalável e sua capacidade de enxergar o melhor em mim impulsionaram-me a alcançar este objetivo significativo.

Neste trabalho, busquei não apenas demonstrar meu conhecimento e habilidades, mas também transmitir um pouco do brilho que você trouxe para minha vida. Sua presença tornou cada desafio mais leve, cada estudo mais prazeroso e cada conquista mais significativa.

Agradeço por ser meu porto seguro, meu ombro amigo e minha maior fã. Sei que posso contar com você para sempre e, por isso, entrego a você esta dedicatória como um símbolo do meu amor eterno e da gratidão que sinto por ter você ao meu lado.

Que este trabalho seja uma pequena homenagem a todo o amor e apoio que você me proporcionou ao longo dessa jornada. Que cada palavra escrita seja um lembrete do quanto você é fundamental para o meu crescimento e sucesso.

Meu amor, estou empolgado para embarcar em novos capítulos da nossa vida juntos, com a certeza de que, com você ao meu lado, superaremos todos os obstáculos e conquistaremos grandes realizações. Obrigado por ser minha inspiração constante e meu verdadeiro amor.

Com todo o meu carinho,

Ewerton Santos.

## **AGRADECIMENTOS**

Gostaria de expressar meus sinceros agradecimentos às pessoas que foram de extrema importância em minha trajetória acadêmica e no desenvolvimento do meu Trabalho de Conclusão de Curso (TCC).

Primeiramente, gostaria de agradecer ao meu orientador, Ewerton Salvador, por sua orientação valiosa, paciência e apoio ao longo deste processo. Sua experiência e conhecimento foram fundamentais para o sucesso deste trabalho.

Também gostaria de agradecer ao meu co-orientador, Alisson Vasconcelos, por seu auxílio e contribuições significativas. Sua visão crítica e orientação técnica foram essenciais para a qualidade do meu TCC.

Uma menção especial à professora Thaís Gaudêncio, que me ajudou imensamente durante um período de crise depressiva. Sua compreensão, suporte e encorajamento foram fundamentais para superar essa fase difícil, na qual até mesmo corri o risco de ser jubilado. Sou imensamente grato pela sua ajuda e compaixão.

Também gostaria de agradecer aos professores Rômulo Calado e Verônica Maria, que me incentivaram a me desenvolver e alcançar meu máximo potencial nesta reta final do curso. Sua dedicação e incentivo foram inspiradores e fizeram uma diferença significativa em minha formação acadêmica.

Não posso deixar de mencionar meus pais, minha noiva e principalmente a Deus. Agradeço a meus pais pelo apoio incondicional, encorajamento constante e pelo investimento em minha educação. À minha noiva, agradeço por estar ao meu lado, compreender minhas demandas acadêmicas e fornecer apoio emocional durante todo o processo.

Por último, mas certamente não menos importante, agradeço a Deus por todas as oportunidades, bênçãos e força que Ele me concedeu ao longo dessa jornada acadêmica. Sua presença em minha vida tem sido a fonte de minha inspiração e motivação.

A todas essas pessoas, expresso minha profunda gratidão. Sem o apoio, orientação e incentivo de vocês, não teria sido possível concluir este trabalho e alcançar meus objetivos acadêmicos. Muito obrigado!

## RESUMO

O presente trabalho de conclusão de curso (TCC) tem como objetivo analisar a viabilidade e aplicação do uso de um subconjunto reduzido de instruções da arquitetura RISC-V, implementado no simulador Logisim, no ensino da disciplina de Arquitetura de Computadores na UFPB. Com o intuito de modernizar e tornar o ensino mais didático, busca-se explorar uma arquitetura mais atual e uma ferramenta de simulação. A pesquisa envolve a análise da viabilidade didática dessa implementação, juntamente com a estimativa de desempenho alcançada por meio do simulador. Os resultados obtidos são encorajadores, considerando as limitações do Logisim, e fornecem uma base sólida para o aprimoramento da qualidade do ensino nessa disciplina, elevando o nível dos alunos envolvidos.

**Palavras-chave:** <RISC-V>, <LOGISIM>, <Arquitetura de Computadores>, <Microarquitetura>.

## **ABSTRACT**

The present undergraduate thesis aims to analyze the feasibility and application of using a reduced subset of instructions from the RISC-V architecture, implemented in the Logisim simulator, in the teaching of Computer Architecture at UFPB. In order to modernize and make the teaching more didactic, we seek to explore a more current architecture and simulation tool. The research involves the analysis of the didactic viability of this implementation, along with the estimation of performance achieved through the simulator. The results obtained are encouraging, considering the limitations of Logisim, and provide a solid foundation for improving the quality of teaching in this discipline, raising the level of the involved students.

**Key-words:**<RISC-V>, <LOGISIM>, <Computer Architecture>,  
<Microarchitecture>.

## LISTA DE FIGURAS

1	LASER-RISCV - Implmentação em LOGISIM . . . . .	21
2	Formato de instruções RISC-V 32-bit . . . . .	28
3	Banco de registradores RISC-V . . . . .	29
4	Contador de programa, memória principal e banco de registradores . . . . .	32
5	Memória principal com redutor de sinal . . . . .	32
6	Um dos 32-bits do Endereço x0 do Banco de Registradores . . . . .	33
7	Funcionamento Interno do Extensor de Sinal (EXTEND) . . . . .	33
8	Conexão Entre Memória Principal, Banco de Registradores e Extensor de Sinal . . . . .	34
9	Conexão Entre Banco de Registradores, Extensor de Sinal e ULA . . . . .	34
10	Estrutura Interna da ULA no Logisim . . . . .	35
11	Carregar Dados da Memória . . . . .	36
12	Escrever Dado De Dolta no Banco de Registradores . . . . .	36
13	Endereço de PC + 1 . . . . .	37
14	Escrita de Dado do Banco de Registradores na Memória . . . . .	37
15	Cálculo de Endereço Alvo Para Instrução BEQ . . . . .	37
16	Estrutura Interno à ULA para "flag"Zero . . . . .	38
17	Processador completo . . . . .	39
18	Unidade de Controle . . . . .	39
19	Tabela Verdade do Decodificador da ULA . . . . .	40
20	Estrutura Interna do Decodificador da ULA . . . . .	41
21	Estrutura Interna da Lógica de Seleção de Endereço ( <b>LSED</b> ) . . . . .	43
22	Implementação da Tabela de Despacho 1 . . . . .	44
23	Implementação da Tabela de Despacho 2 . . . . .	44
24	Implementação de Memória <b>ROM</b> em Logisim . . . . .	45
25	Estrutura Interna do Decodificador de Instrução . . . . .	46
26	Fluxograma de Estados da "Main FSM" . . . . .	47
27	Estrutura Interna da "Main FSM" . . . . .	48

28	Programa Exemplo . . . . .	49
29	Programa Carregado em Memória . . . . .	50
30	Programa Executado . . . . .	51
31	Lista de Velocidades de Relógio Disponíveis no Logisim . . . . .	53

## **LISTA DE TABELAS**

1	Conjunto de Instruções . . . . .	27
2	Nome e números dos registradores . . . . .	30
3	Decodificador de Imediatos . . . . .	34
4	Ações Implementadas Pela Unidade da Lógica de Seleção de Endereço (LSED)	41
5	Tabela de Despacho 1 . . . . .	42
6	Tabela de Despacho 2 . . . . .	42
7	Organização da Memória de Microprogramas . . . . .	45
8	Codificação do Sinal "CtrlAdr" da Microinstrução . . . . .	46
9	Representação dos Imediatos . . . . .	49
10	Programa Traduzido em Linguagem de Máquina . . . . .	50
11	Conclusão Qualitativa . . . . .	52

## LISTA DE ABREVIATURAS

**imm** - Imediato com extensão de sinal em  $imm_{11:0}$

**uimm** - Imediato de 5-bit sem sinal em  $imm_{4:0}$

**upimm** - 20 bits superiores de um imediato de 32-bit, em  $imm_{31:12}$

**Address** - Endereço de memória:  $rs1 + \text{SignExt}(imm_{11:0})$

**[Address]** - Dado no endereço de memória "Address"

**BTA** - Endereço de ramo Alvo

**JTA** - Endereço de salto Alvo

**SignExt** - valor com extensão de sinal de 32 bits

**ZeroExt** - Valor com extensão por zeros de 32 bits

**csr** - Registrador de controle e estado

**FPGA** - Matriz de portas programáveis

**ISA** - Arquitetura do conjunto de instruções

**CLK** - Pulso de relógio

**ULA** - Unidade lógica e aritmética

**ROM** - Memória de apenas leitura

**LSED** - Lógica de seleção de endereço

**RISC** - Reduced Instruction Set Computer

# Sumário

<b>1 INTRODUÇÃO</b>	<b>17</b>
1.1 Definição do Problema . . . . .	17
1.2 Objetivo Geral . . . . .	18
1.3 Objetivos Específicos . . . . .	18
1.4 Organização do Trabalho . . . . .	18
<b>2 CONCEITOS GERAIS E TRABALHOS RELACIONADOS</b>	<b>19</b>
2.1 Arquitetura de Computadores . . . . .	19
2.2 Arquitetura RISC-V . . . . .	20
2.3 O LOGISIM . . . . .	20
2.4 Trabalhos Relacionados . . . . .	21
<b>3 METODOLOGIA</b>	<b>23</b>
3.1 Estudo de Viabilidade . . . . .	23
3.2 Estimativa de Desempenho . . . . .	23
<b>4 ARQUITETURA DO PROCESSADOR</b>	<b>25</b>
4.1 Load e Store (LW e SW) . . . . .	25
4.2 Operações lógicas (AND e OR) . . . . .	26
4.3 Operações aritméticas (ADD e SUB) . . . . .	26
4.4 Saltos de instruções (JAL e BEQ) . . . . .	26
4.5 Operações com Imediatos (ADDi, ANDi, ORi e SLTi) . . . . .	27
4.6 Banco de Registradores . . . . .	27
4.7 Memória Principal . . . . .	29
<b>5 MICROARQUITETURA</b>	<b>31</b>
5.1 Caminho de Dados . . . . .	31
5.2 Unidade de Controle . . . . .	38
<b>6 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS</b>	<b>48</b>
6.1 Teste de Execução de Programa . . . . .	48

6.2	Resultados de Viabilidade . . . . .	50
6.3	Resultados Estimados de Desempenho . . . . .	52
<b>7</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS</b>	<b>55</b>
7.1	Conclusão . . . . .	55
7.2	Resultados e Trabalhos Futuros . . . . .	55
7.3	Considerações Finais . . . . .	56
	<b>REFERÊNCIAS</b>	<b>56</b>

# 1 INTRODUÇÃO

A disciplina de Arquitetura de Computadores do Centro de Informática da UFPB (Universidade Federal da Paraíba) tem como principal objetivo apresentar através de exemplos didáticos como as linhas de código são executadas por um processador [16] [17]. Atualmente é utilizado um subconjunto de instruções MIPS como arquitetura base da disciplina de Arquitetura de Computadores que é relativamente simples, com 9 instruções, e com um caminho da dados didático, facilitando assim, a compreensão e estudo. Esta arquitetura foi criada em meados dos anos 1990s e significa *Microprocessor without interlocked pipeline stages* (microprocessador sem estágios intertravados de pipeline) [18] que naquele momento foi embarcado em dispositivos famosos como o Nintendo 64 [19] e o Playstation. Entretanto, o MIPS não possui uma comunidade tão atuante hoje em dia, afirmação essa que pode ser validada pela quantidade de artigos recentes relacionados a RISC-V quando comparado ao MIPS, além de ser uma implementação proprietária. O objeto de estudo desse trabalho, um subconjunto de instruções da arquitetura RISC-V que é uma arquitetura do tipo RISC, começou a ser desenvolvida no ano de 2010 na Universidade da Califórnia, em Berkeley. Desde então, ela foi continuamente melhorada, sendo livre para ser usada para qualquer finalidade - o que já caracteriza uma vantagem em relação ao MIPS. Por ser livre, possui uma comunidade extremamente engajada e atuante, tornando o seu estudo relativamente mais fácil, visto que a documentação é vasta.

## 1.1 Definição do Problema

Durante o curso de Arquitetura de Computadores na Universidade Federal da Paraíba, um dos principais desafios enfrentados pelos alunos é compreender o funcionamento de um processador e ser capaz de propor mudanças na implementação estudada. No entanto, essa mudança é feita de forma teórica, em que o aluno cria um desenho com base em uma representação do caminho de dados e explica as modificações por meio de um relatório.

Além disso, é importante mencionar que o MIPS, atualmente utilizado na disciplina, é uma tecnologia antiga, o que evidencia a oportunidade de renovação para acompanhar as demandas atuais. Nesse contexto, o RISC-V surge como uma tecnologia nova e promissora, apresentando um grande potencial e sendo totalmente aberta em termos de uso. Essa característica torna o RISC-V um atrativo significativo para ser incorporado como uma ferramenta de ensino, permitindo uma abordagem mais atualizada e relevante para os alunos.

Dessa forma, a necessidade de compreensão prática do funcionamento do processador e a demanda por ferramentas atualizadas para o ensino de Arquitetura de Com-

putadores motivam a busca por soluções que incorporem o uso do RISC-V como uma alternativa ao tradicional ensino baseado em MIPS bem como o uso de um simulador proporcionando que o aluno não fique apenas no campo teórico.

## 1.2 Objetivo Geral

Dada a problemática, este trabalho busca facilitar o ensino/aprendizagem da disciplina de Arquitetura de Computadores por meio da implementação de um subconjunto de instruções da arquitetura RISC-V [20] para fins pedagógicos [21], além de também, modernizar a estrutura e ferramentas que são usadas para este fim.

## 1.3 Objetivos Específicos

1. Delimitar um conjunto de instruções do RISC-V relativamente reduzido e viável para resolver problemas simples e também didático.
2. Implementar um caminho de dados que abranja o conjunto de instruções delimitado bem como o módulo de controle utilizando o simulador LOGISIM.
3. Analisar a dificuldade de entendimento e implementação a fim de se determinar a viabilidade didática do modelo.
4. Analisar de forma quantitativa o desempenho do modelo desenvolvido em comparação a outras tecnologias.

## 1.4 Organização do Trabalho

O Capítulo 2 apresenta a bibliografia em busca de conceitos científicos para este trabalho. No Capítulo 3, está descrita a metodologia, apresentando as técnicas de geração de resultados. No Capítulo 4, a arquitetura usada para implementação, a qual descreve o subconjunto de instruções selecionadas e descritas de forma estruturada para o trabalho, bem como uma descrição funcional de cada uma, além das estruturas de memória e banco de registradores usados pela arquitetura. O Capítulo 5, apresenta um detalhamento dos componentes, bem como suas implementações no simulador a nível de microarquitetura. Por fim, nos Capítulos 7 e 8, é analisado os resultados qualitativos e quantitativos com objetivo de tirar conclusões a respeito da viabilidade do uso da implementação juntamente com o simulador para fins didáticos.

## 2 CONCEITOS GERAIS E TRABALHOS RELACIONADOS

Este capítulo se dedica a apresentar conceitos gerais e trabalhos relacionados que são de suma importância para o entendimento do presente texto.

### 2.1 Arquitetura de Computadores

A arquitetura de um computador é definida por suas estruturas básicas, blocos lógicos, interligações entre eles, lógica digital da microarquitetura, e conjunto de instruções [16]. A arquitetura define o projeto e a organização do sistema, desde o circuito lógico até a estrutura do processador passando também pela organização da memória e as instruções que delimitam as funções que o sistema é capaz de executar [13].

Os pontos bases da arquitetura de um computador são a arquitetura do conjunto de instruções e o circuito lógico (caminho de dados) que torna essa arquitetura funcional . O conjunto de instruções de um computador é basicamente o escopo de funções e ações que o processador (computador) é capaz de realizar, sendo muitas vezes referenciada como o *"Assembly"* da arquitetura, que é a linguagem de montagem de máquina legível para um humano por meio da qual se pode desenvolver um programa que o computador consegue executar e gerar resultados. Já o caminho de dados é definido como o circuito lógico (que é em suma um circuito elétrico) que traduz a linguagem binária em um conjunto de operações com componentes de circuitos lógicos (and, xor, multiplexador, etc) fazendo instruções Assembly percorrer em um caminho dentro do circuito elétrico realizando operações lógicas e aritméticas sobre dados referenciados pelas instruções [12].

Existem duas grandes categorias na Arquitetura de computadores, são elas as arquiteturas do tipo RISC e as do tipo CISC. [10] As arquiteturas CISC são as mais antigas e mais usadas em computadores pessoais hoje em dia, significa *"Complex Instruction Set Computer"*, ou, em uma tradução literal, *"Computador com um Conjunto Complexo de Instruções"*. Esse nome se dá pelo fato de ser uma arquitetura de um número de instruções extremamente grande e muitas delas de um elevado nível de complexidade. Por se tratar de instruções complexas, elas são relativamente lentas para serem executadas.

Já a RISC, por outro lado, surgiu um pouco depois da CISC, cerca de 30 anos mais nova, e é uma das arquiteturas mais populares no meio acadêmico visto o crescente número de publicações nos últimos anos. Significa *"Reduced Instruction Set Computer"*; em português, *"Computador com um conjunto reduzido de instruções"*) é uma linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas. [11] Com um pequeno número de instruções e uma complexidade reduzida temos então instruções mais rápidas e objetivas, com a desvantagem de ter um código *Assembly* maior

e mais complexo o que vale a pena, visto que, temos uma arquitetura menos complexa e com melhor desempenho.

Não se pode deixar de citar, também, as arquiteturas híbridas que são basicamente arquiteturas CISC que incorporaram um núcleo RISC deixando assim a arquitetura ainda mais complexa, porém, com um desempenho melhor, quando desejado, sem perder a compatibilidade legado.[11]

## 2.2 Arquitetura RISC-V

Trata-se de uma arquitetura de conjunto de instruções (ISA) que através da inovação e colaboração promete criar uma nova era dos processadores. De código aberto, ou seja, livre para ser utilizada para qualquer finalidade, sem a necessidade de pagar royalties ou pedir autorizações, o projeto foi iniciado em 2010 na Universidade da Califórnia, em Berkeley. Foi concebida com o principal propósito as implementações de alto desempenho e baixo consumo de energia, como computação em nuvem, aparelhos móveis, embarcados e internet das coisas, tendo um conjunto de instruções limpo porém com várias opções de expansão, inclusive com ponto flutuante [20].

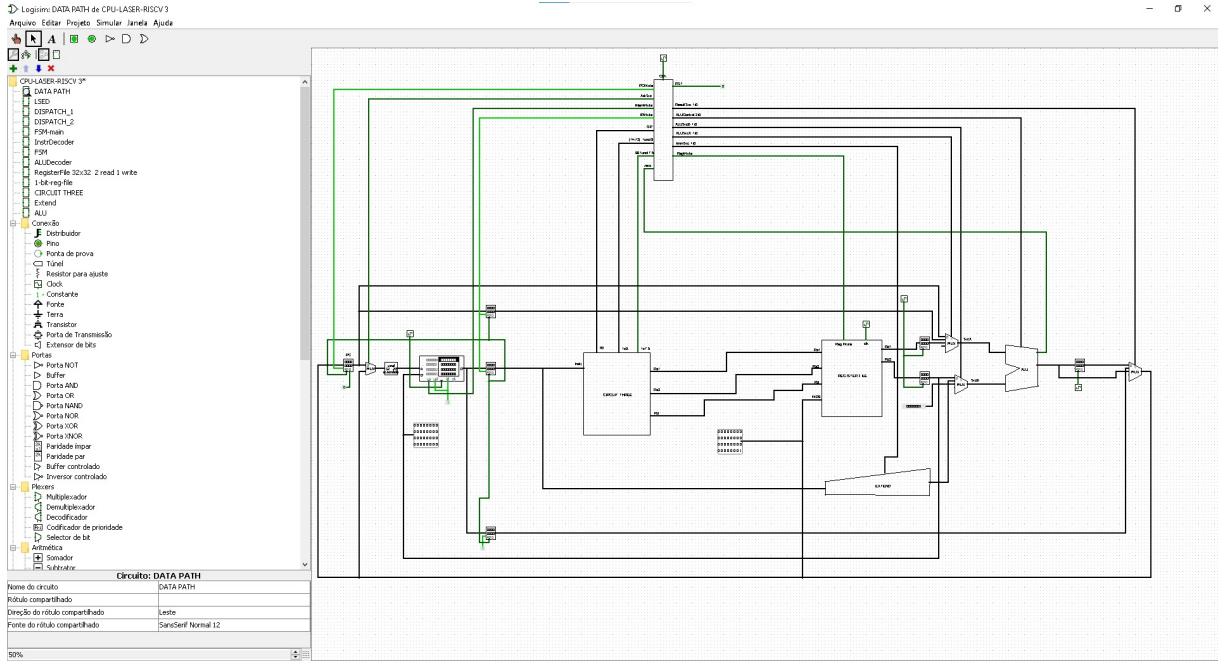
O RISC-V tem um conjunto de instruções padrão chamado RV32I que é a base para operações com números inteiros, e a partir desse conjunto, é possível fazer expansões, como por exemplo, a RV32M que é a expansão para multiplicação e divisão ou a RV32A que dá suporte para operações atômicas [20].

## 2.3 O LOGISIM

O Logisim, visto na **Figura 1**, é uma ferramenta de design e simulação de circuitos digitais voltada para a educação. Com uma interface intuitiva os usuários podem criar circuitos lógicos combinacionais e sequenciais usando uma variedade de componentes. A interface gráfica facilita a criação e visualização dos circuitos permitindo arrastar e soltar os componentes e conectá-los virtualmente, além disso, o Logisim oferece recursos de simulação permitindo aos usuários testar e depurar seus circuitos verificando as entradas e saídas esperadas. A ferramenta também possui recursos extras como a criação de subcircuitos personalizados e a exportação dos circuitos em diferentes formatos. Com sua abordagem amigável o Logisim é amplamente utilizado em disciplinas de eletrônica digital e arquitetura de computadores para facilitar o aprendizado e a experimentação com circuitos digitais [23].

O Logisim é um software gratuito e de código aberto distribuído sob a licença GPLv2. Isso significa que ele pode ser compartilhado livremente permitindo que os usuários o utilizem e distribuam sem restrições, no entanto, a licença não permite modificações

**Figura 1: LASER-RISCV - Implmentação em LOGISIM**



**Fonte:** Logisim

no código fonte sem a devida autorização. Essa abordagem de licenciamento promove a transparência e a colaboração permitindo que estudantes educadores e entusiastas de eletrônica digital tenham acesso a uma ferramenta poderosa para aprender projetar e simular circuitos digitais. O Logisim possui uma interface intuitiva que facilita a criação e visualização dos circuitos além de oferecer recursos de simulação para testar e depurar o funcionamento dos mesmos. Com o Logisim os usuários podem explorar os conceitos de eletrônica digital e arquitetura de computadores de forma acessível e gratuita. É uma ferramenta valiosa para o ensino e aprendizado dessas disciplinas, proporcionando uma experiência dinâmica e interativa. Sua disponibilidade como software de código aberto garante a continuidade do acesso e uso do Logisim, beneficiando a comunidade acadêmica e entusiastas de eletrônica em todo o mundo [22].

## 2.4 Trabalhos Relacionados

Um conjunto de trabalhos publicados serviu de inspiração, como o ”A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems” que apresenta uma arquitetura baseada em RISC-V tolerante a falhas projetado para estar embarcado em sistemas aeroespaciais que levantou uma questão interessante quanto a desempenho mediante a falhas de arquiteturas de processadores. [8]

Quando se fala de baixo custo, remete-se a eficiência energética e o artigo científico

a publicação [9] foi a principal fonte de estudo do tema. Propõe-se uma extensão ISA que permite a coexistência de instruções de 16 bits comprimidas e instruções 32 bits mantendo o baixo custo e eficiência energética baseado na arquitetura RISC-V.

Outro trabalho relevante e motivador desta pesquisa é o [21] que assim como este trabalho, também explora a arquitetura risc-v com o objetivo de fins educacionais. No entanto, há uma diferença importante. O ”rvsh”utiliza uma implementação monociclo escrita em HDL (linguagem de descrição de hardware) o que o torna mais próximo de uma implementação real. Essa abordagem é especialmente significativa, uma vez que, as ferramentas de EDA (eletrônica de design automatizado) também utilizam esse tipo de linguagem. Portanto o rvsh oferece uma perspectiva mais prática e alinhada com a implementação de processadores reais.

O artigo intitulado ”RISC-V SIMPLE: Projeto e Desenvolvimento de Processadores RISC-V com a ISA RV32IMF Usando as Microarquiteturas Uniciclo, Multiciclo e Pipeline em FPGA”foi também uma das inspirações para a pesquisa atual. Ele descreve a construção de uma plataforma didática para o ensino de Arquitetura de Computadores através de uma implementação em Verilog de um processador RISC-V. Foi utilizada 3 tipos de microarquitetura diferentes, são elas Uniciclo, Multiciclo e Pipelined o que permite os estudantes e pesquisadores terem uma visão das diferentes estratégias que são utilizadas por cada implementação e os impactos que causam na complexidade e desempenho final [24].

### 3 METODOLOGIA

Neste capítulo, será descrito a metodologia adotada nesta pesquisa, que aborda os desafios das limitações do Logisim e a complexidade das modificações necessárias pelos alunos da disciplina de arquitetura de computadores para a implementação deste trabalho.

#### 3.1 Estudo de Viabilidade

Uma das metodologias adotadas nesta pesquisa envolve lidar com as restrições de componentes e módulos, bem como o desempenho e a capacidade de simulação do Logisim, durante o processo de análise e alteração da arquitetura do processador. Os alunos aplicarão técnicas de otimização e utilizarão seu conhecimento teórico em circuitos lógicos e arquitetura de computadores para realizar uma análise de complexidade das modificações necessárias à implementação. Essa abordagem visa garantir a viabilidade da execução da atividade de arquitetura de computadores usando a implementação proposta, levando em consideração métricas como a facilidade de compreensão do projeto e da ferramenta, a capacidade didática de ilustração do caminho de dados e do bloco de controle, o esforço requerido para realizar mudanças no projeto, a capacidade de simulação e execução de programas, e o poder computacional necessário para a execução.

#### 3.2 Estimativa de Desempenho

Além disso, como uma segunda metodologia empregada, utiliza-se o cálculo estimativo de MIPS (milhões de instruções por segundo) com base na frequência de relógio do Logisim, juntamente com uma projeção para uma implementação em uma FPGA específica. A estimativa de desempenho do processador em termos de MIPS leva em consideração a frequência de operação do circuito no Logisim. Com base nessa estimativa, os alunos podem realizar uma projeção para uma implementação em uma FPGA selecionada, considerando as características e restrições desse dispositivo. Essa abordagem complementar proporciona uma análise mais abrangente do desempenho do processador e auxilia na tomada de decisões durante o projeto e modificação da arquitetura. Para esta análise, será utilizada como métrica principal a velocidade de execução de programa, que será calculada em milhões de instruções por segundo [26].

A performance é definida como o inverso do tempo de execução de um programa [13], como descrito na equação (1) e por sua vez, a performance relativa será dada através da equação (2).

$$\Pi = \frac{1}{\tau} \quad (1)$$

$$\frac{\Pi_A}{\Pi_B} = \frac{\tau_B}{\tau_A} = n \quad (2)$$

Onde caso n seja maior que 1, isso significa que a performance de A é maior que a de B, ou seja o tempo de execução é menor e se for igual, intuitivamente, os tempos e performances serão iguais.

Já o tempo de execução, que foi anteriormente referenciado, é calculado com base no número de ciclos de relógio por programa, tempo de um único ciclo de relógio e frequência de relógio [13], como dado na equação (3).

$$\tau = K \times t_k = \frac{K}{r} \quad (3)$$

Onde K é o número de ciclos de relógio por programa,  $t_k$  é o tempo que leva para finalizar um único ciclo de relógio e r é a frequência do relógio.

Podendo compilar estas informações na chamada equação clássica (4).

$$\tau = InstructionCount \times CPI \times t_k = \frac{InstructionCount \times CPI}{r} \quad (4)$$

Onde "InstructionCount" é o número de instruções executadas pelo programa e CPI é a média de ciclos de relógio que gasta uma instrução.

A partir dessas equações pode ser deduzida uma terceira equação que é a média de quantas instruções por segundo o processador é capaz de executar (5).

$$MIPS = \frac{r}{CPI \times 1.000.000} \quad (5)$$

A partir daí se pode calcular quantas milhões de instruções por segundo o processador é capaz de realizar baseado na frequência de relógio e número de ciclos médio que leva para executar uma instrução.

## 4 ARQUITETURA DO PROCESSADOR

Com objetivo de avaliar a viabilidade do uso do simulador Logisim no ensino da disciplina de Arquitetura de computadores foi selecionado um subconjunto de instruções do conjunto RV32I, com base no livro ”Digital Design and Computer Architecture: RISC-V Edition” de Harris e Harris, [12] especificamente no Capítulo 7, que apresenta uma implementação multi-ciclo. Essa abordagem visa simplificar o estudo inicial, focando em problemas mais simples, enquanto atendem as demandas do escopo da disciplina de Arquitetura de Computadores.

O objetivo educacional é permitir que os estudantes compreendam os conceitos fundamentais antes de abordar implementações mais complexas. Para isso, foi adotado um caminho de dados simplificado, que também é descrito no mesmo capítulo, para a implementação desse subconjunto escolhido.

As instruções escolhidas para esse subconjunto incluem as operações aritméticas ADD (adição) e SUB (subtração), as operações lógicas OR (OU) e AND (E), as instruções de controle JAL (jump and link) e BEQ (branch if equal), além das instruções de acesso à memória LW (load word) e SW (store word). Também foram incluídas as instruções imediatas ADDi (adição imediata), ANDi (E imediato), ORi (OU imediato) e SLTi (set less than immediate).

Esse subconjunto selecionado oferece um ponto de partida adequado para o estudo de arquitetura de computadores, permitindo a resolução de problemas simples inicialmente, antes de se aumentar a complexidade da implementações e agrangência.

### 4.1 Load e Store (LW e SW)

São instruções primordiais no conjunto escolhido aquelas que possibilitam carregar (LW) ou guardar (SW) dados de tamanho 32 bits (palavra) na memória. Na ausência dessas instruções, seria impossível solucionar qualquer problema, por mais simples que seja pois precisamos nos comunicar com a memória principal visto que ela é a ponte entre o núcleo do processador e seu sistema operacional e essas duas instruções são as que nos possibilita ler e escrever da memória. Por isso, se faz necessário adicioná-las ao subconjunto selecionado.

A instrução que carrega a palavra da memória no registrador (load word) é do tipo I, a qual tem a estrutura descrita na **Figura 2** na sua segunda linha e na Tabela 1, podemos ver sua descrição detalhada em linguagem de máquina. Já a instrução que faz o caminho inverso, ou seja, guarda um dado que está no registrador para dentro da memória (store word) é do tipo S a qual também sua estrutura base, descrita na **Figura 2** terceira linha e detalhada na **Tabela 1** segunda linha.

## 4.2 Operações lógicas (AND e OR)

Para tomada de decisões e automação de casos, se faz necessário instruções que operem de forma lógica. O operador AND realiza a operação lógica "E" qual só será verdadeira se ambos os operandos forem verdadeiros. Já o operador lógico OR realiza a operação "OU", o qual terá resultado verdadeiro se ao menos um dos operandos for verdadeiro. Ambas instruções são do tipo R que podemos ver como estão estruturadas na **Figura 2** na primeira linha e temos também uma descrição mais detalhada na Tabela 1 nas linhas 3 e 4 respectivamente.

## 4.3 Operações aritméticas (ADD e SUB)

Para efetuar cálculos aritméticos simples são imprescindíveis instruções aritméticas de soma (ADD) e subtração (SUB). Assim como as operações lógicas, vistas anteriormente, as aritméticas também são do tipo R e também descritas na mesma estrutura na **Figura 2** porém, estão detalhadas nas linhas 5 e 6 da **Tabela 1**.

## 4.4 Saltos de instruções (JAL e BEQ)

Para desvio de fluxo se faz necessário saltos de instruções, sejam eles condicionais (BEQ) ou incondicionais (JAL). Essas instruções permitem a criação de bifurcações no código, as quais referem-se a pontos no código onde a execução se divide em caminhos distintos com base em resultados de condições ou até mesmo saltos incondicionais.

O BEQ (Branch if Equal) é uma instrução do tipo B cuja estrutura está descrita na linha 4 da Figura 2 e melhor detalhada na penúltima linha da Tabela 1 e realiza um teste de lógica com dois registradores e dependendo do resultado do teste, o BEQ faz um salto proporcional ao imediato contido em instruções do tipo **I, S, B ou J**. Se a comparação dos registradores for igual, o salto é realizado, caso contrário, a execução segue o fluxo para a próxima instrução sequencialmente.

O JAL (Jump and Link) é do tipo J e é descrita na penúltima linha da Figura 2, onde são apresentados os campos e a organização dessa instrução e em detalhes na Tabela 1. O JAL, realiza um salto incondicional no fluxo de execução do programa direcionando a execução para um outro endereço de memória, também proporcional ao imediato contido na instrução. Além disso, o JAL também armazena o endereço da próxima instrução em um registrador de livre escolha. Ao ser executado, o JAL, desvia o fluxo de execução do programa independente de qualquer condição, o que o caracteriza como instrução de desvio incondicional o que pode ser usado para implementação de chamadas de função e sub-rotinas.

**Tabela 1: Conjunto de Instruções**

OP	funct3	funct7	Tipo	Instrução	Descrição	Operação
0000011 (3)	010	-	I	lw rd, imm(rs1)	load word	rd = [Address]31:0
0100011 (35)	010	-	S	sw rs2, imm(rs1)	store word	[Address]31:0 = rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1   rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	110	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 — rs2
0010011 (19)	000	—	I	addi rd, rs1, imm	add imm	rd = rs1 + imm
0010011 (19)	111	—	I	andi rd, rs1, imm	and imm	rd = rs1 & imm
0010011 (19)	110	—	I	ori rd, rs1, imm	or imm	rd = rs1   imm
0010011 (19)	010	—	I	slti rd, rs1, imm	X <imm	rd = rs1 <imm
1100011 (99)	000	—	B	beq rs1, rs2, label	branch if =	PC = BTA
1101111 (111)	—	—	J	jal rd, label	j & l	rd=PC+1

Fonte: Appendix B: RISC-V ISA Summary [12]

Estas duas instruções são essenciais para tornar possível múltiplas lógicas em um mesmo programa, oq as torna, portanto, indispensáveis no nosso subconjunto.

#### 4.5 Operações com Imediatos (ADDi, ANDi, ORi e SLTi)

Para poder trabalhar com valores arbitrários é preciso utilizar instruções com imediatos que são aquelas cujo utilizam valores fixos, contido na própria instrução, para efetuar operações.

As instruções ADDi, ANDi e ORi fazem operações lógicas e aritméticas que estamos usualmente acostumados, utilizando um valor imediato e um registrador como operandos. Já a SLTi, faz uma comparação entre o valor do registrador e o valor contido no imediato. Caso o valor contido no registrador seja menor que o valor contido no imediato, o resultado da operação será 32b'1 (representação em binário de 1 em 32 bit), caso contrário o resultado será zero.

Todas as 4 instruções anteriormente listadas são do tipo I, mesmo tipo da instrução load word, que como já vimos, está descrita sua estrutura na segunda linha da Figura 2. Estas novas instruções estão detalhadas nas linhas 7, 8, 9 e 10, respectivamente, na nossa Tabela de instruções 1.

#### 4.6 Banco de Registradores

O banco de registradores, visto na **Figura 3**, é uma estrutura digital composta de registradores, que por sua vez, são feitos com flip-flops os quais têm como objetivo

**Figura 2: Formato de instruções RISC-V 32-bit**

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
imm <sub>11:0</sub>		rs1	funct3	rd	op	I-Type
imm <sub>11:5</sub>	rs2	rs1	funct3	imm <sub>4:0</sub>	op	S-Type
imm <sub>12,10:5</sub>	rs2	rs1	funct3	imm <sub>4:1,11</sub>	op	B-Type
imm <sub>31:12</sub>				rd	op	U-Type
imm <sub>20,10:1,11,19:12</sub>				rd	op	J-Type
fs3	funct2	fs2	fs1	funct3	fd	op
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Fonte: Appendix B: RISC-V ISA Summary [12]

armazenar informações com alta eficiência e rapidez. Dentro do banco, eles obedecem uma estrutura que os permitem ser acessados e modificados de forma organizada e estruturada.

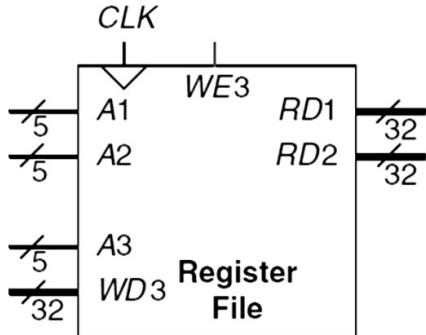
Este banco permite que seja possível ler em duas posições diferentes e escrever em uma terceira ao mesmo tempo baseada nas suas 3 entradas de endereço de 5-bit. As entradas são utilizadas para definir qual endereço do banco estamos tratando, uma entrada de dados de 32-bit, a qual é utilizada para carregar dados à um registrador previamente selecionado e por fim, duas saídas de 32-bit as quais fazem as leituras simultâneas de dois registradores também previamente selecionados.

A estrutura interna e funções definidas pela arquitetura estão descritas na **Tabela 2**, onde se pode dar destaque para alguns registradores como os  $t_{0-6}$ , que são registradores temporários que são sobreescritos após chamadas de funções, registradores  $s_{0-12}$ , que são registradores que se mantém seus valores mesmo após chamadas de funções e também o registrador zero que é um registrador particular o qual contém sempre o valor zero não podendo ser sobreescrito.

A referência "Número do registrador" que é possível ver na segunda coluna da tabela de registradores, anteriormente citada, será utilizada nos campos rs1, rs2, e rd, que são registrador fonte 1 (register source 1), registrador fonte 2 (register source 2) e registrador destino, visto tanto na **Tabela 1** quanto na **Figura 2** que mostram o detalhamento das instruções que possuímos e a estrutura geral dos tipos de instruções da arquitetura, respectivamente. Essas nomenclaturas são utilizadas para referenciar qual registrador estamos utilizando na instrução indicada como fontes e destino dos operandos.

A pesar de serem apenas 32 registradores, eles têm um grande impacto na velocidade com que as operações são realizadas, reduzindo acessos à memória principal.

**Figura 3: Banco de registradores RISC-V**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

#### 4.7 Memória Principal

A memória principal de uma arquitetura de computador desempenha um papel de extrema importância, tendo como responsabilidade armazenar os dados e instruções as quais se fazem necessárias para processar informações com a execução de programas.

Ela fornece um espaço de armazenamento rápido e de baixo custo quando comparado ao registradores. A pesar de não serem tão rápidas quanto os flip-flops, ainda possuem um desempenho interessante tendo assim sua relevância destacada principalmente para armazenamento de instruções. Quando executamos um programa, carregamos suas instruções e dados na memória principal a partir de memórias secundárias que são mais lentas, menos eficientes e ainda mais baratas. Então o nosso processador pode acessar a memória principal para carregar informações as quais necessita para executar tarefas e manipular dados.

A memória principal está estruturada em células de 32-bit cada, tornando assim, desnecessário o montador, ou o próprio programador, se preocupar com alinhamento de memória, pois cada endereço já possui uma instrução completa. Essa característica pode trazer desvantagens. Quanto maior a célula unitária de memória, maior o desperdício de memória pois comumente valores armazenados não ocupam todos os bits disponíveis na célula (com exceção das instruções). Isso pode resultar em espaços vazios que poderiam ser utilizados para armazenar outros valores o que é considerado ineficiente quanto a utilização do espaço. Também não é possível acessar apenas parte de dados menor que 32-bit, utilizando essa arquitetura de memória, técnica essa que é utilizada para resolver alguns problemas de concatenação de dados.

Tendo, então, disponível esse conjunto de instruções, já é possível resolver muitos problemas simples e poder mostrar na prática, ponta a ponta, como um processador pode ser pensado, estruturado, testado e simulado.

**Tabela 2: Nome e números dos registradores**

Nome	Número do Registrador	Uso
zero	x0	Contém o valor 0
ra	x1	Endereço de retorno
sp	x2	Apontador da pilha
gp	x3	Apontador global
tp	x4	Apontador de thread
t <sub>0-2</sub>	x5–7	Registradores temporários
s0/fp	x8	Registrador salvo / Apontador de quadro
s <sub>1</sub>	x9	Registrador salvo
a <sub>0-1</sub>	x10–11	Argumentos de Funções / Valor de retorno
a <sub>2-7</sub>	x12–17	Argumento de funções
s <sub>2-11</sub>	x18–27	Registradores salvos
t <sub>3-6</sub>	x28–31	Registradores temporários

**Fonte: Appendix B: RISC-V ISA Summary [12]**

## 5 MICROARQUITETURA

A seção de microarquitetura é parte fundamental para um projeto de processador. Nela será apresentado detalhes internos da organização do processador com foco nos componentes e estruturas que realizam um papel de suma importância para o seu funcionamento. Abordaremos aspectos da unidade de controle, caminho de dados, estrutura de registradores, unidade lógica e aritmética bem como mecanismos de extensão de sinal.

Esta seção trará uma visão e entendimento profundo das estruturas internas do processador e como podem influenciar diretamente o seu desempenho. Isso dará uma visão detalhada das estratégias do projeto e possibilidade de avaliar a qualidade em relação as requisitos funcionais do sistema.

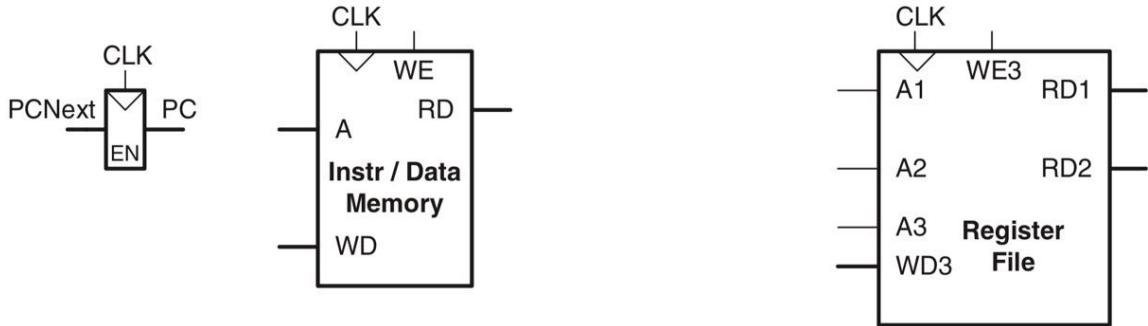
### 5.1 Caminho de Dados

Baseado no livro ”Digital Design and Computer Architecture: RISC-V Edition”, onde existem 3 tipos de implementações, que são elas, monociclo, multiciclo e pipilined, o caminho de dados objeto de estudo, é baseado na implementação multiciclo encontrada no **Capítulo 7** [12], por se tratar de uma implementação que têm um controle relativamente simples de ser implementado e entendido (quando comparado ao pipelined), sem ficar tão distante da realidade de como um processador real funciona, como é a implementação monociclo. Algumas das vantagens do uso da implementação multiciclo em detrimento da mono ciclo são: não necessitar de memórias separadas para dados e instruções, visto que é usada a mesma memória para ambas tarefas; não requer um ciclo de relógio longo quanto o necessário na abordagem monociclo tendo assim suporte para altas frequências de relógio.

Para começar a montar o caminho de dados, é recomendável começar pelas memórias e a arquitetura de estados do processador como mostrado na **Figura 4**. Como já citado acima, será utilizado a mesma memória tanto para armazenar instruções quanto para armazenar dados. Esta abordagem de usar uma combinação de funções para a mesma memória é mais realista pois reflete como processadores modernos funcionam na prática [27].

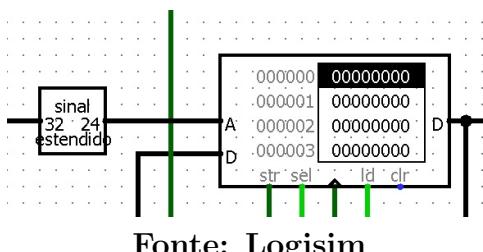
Analisando a Figura 4, é observada uma pequena estrutura a esquerda. Este componente se trata de um registrador chamado contador de programa (Program Counter ou PC) e tem como função armazenar o endereço de memória da próxima instrução que será executada. Este registrador possui duas entradas, uma chamada ”PCNext” que é o endereço da instrução que será executada no próximo ciclo e outra chamada ”enable” (EN) a qual habilita ou desabilita a escrita no registrador e possui uma saída ”PC” que é o endereço de memória da instrução a ser executada no ciclo atual. É observado também

**Figura 4: Contador de programa, memória principal e banco de registradores**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 5: Memória principal com redutor de sinal**



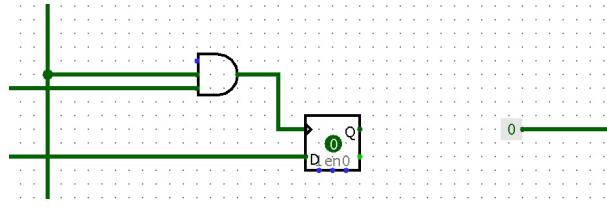
Fonte: Logisim

que se trata de um componente sensível ao **CLK**, nesse caso, à borda de subida.

Ainda na **Figura 4**, o componente do meio é a memória principal o qual é sensível a borda de subida do **CLK** e que possui uma entrada de endereço "A", uma entrada de dados "WD", uma saída de dados "RD", todas de largura 32-bit e um sinal de controle "WE" o qual habilita ou desabilita a escrita. Ela é responsável por armazenar as instruções e dados do programa. Como é visto na **Figura 5** foi necessário fazer uma adaptação pois o componente de memória disponibilizado pelo Logisim possui um endereço máximo de 24-bit, fazendo assim necessário ignorar os 8 bits mais significativos do endereço de memória "A". Essa ação deixará a implementação mais limitada à um pouco menos que 17 mil endereços de memória, o que já é suficiente para executar uma grande gama de programas.

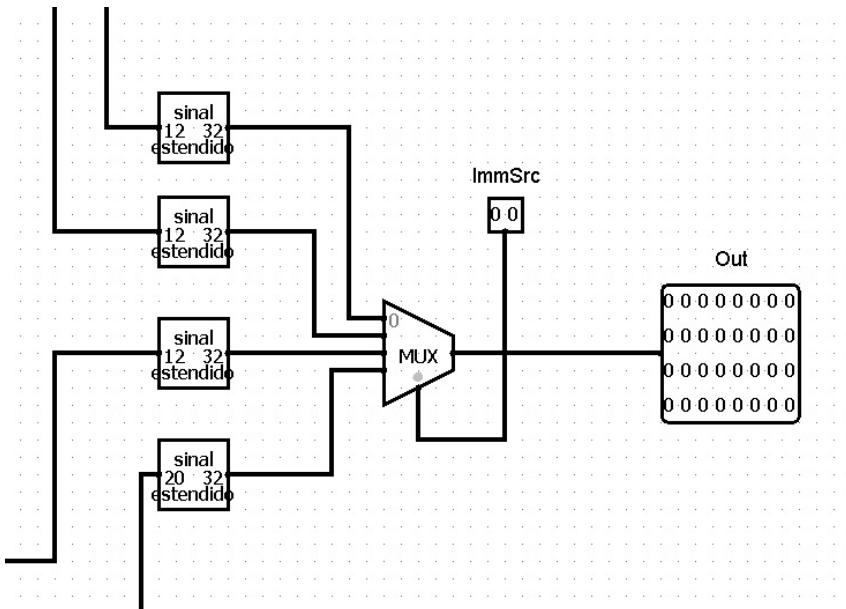
Nota-se, por fim, à direita na **Figura 4** o banco de registradores o qual já foi discutido seu funcionamento na **Seção 4.6**. Por sua vez, para implementá-lo no Logisim foi um dos desafios desse presente trabalho, pois, no simulador em questão não existe um componente pré definido que atenda os requisitos funcionais desse componente. Foi implementada com base no artigo "Register File Block Diagram" [14], onde é utilizado uma matriz de Flip-Flop's D que são sensíveis à borda de subida, com o detalhe de que, o endereço x0 do banco de registradores, não pode ser sob reescrito pois precisa ter um valor constante zero, como já bem vimos na **Tabela 2** no Capítulo de Arquitetura. Para isso foi desconectada a entrada de todos os bits e suas saídas foram conectadas ao nível

**Figura 6: Um dos 32-bits do Endereço x0 do Banco de Registradores**



Fonte: Logisim

**Figura 7: Funcionamento Interno do Extensor de Sinal (EXTEND)**



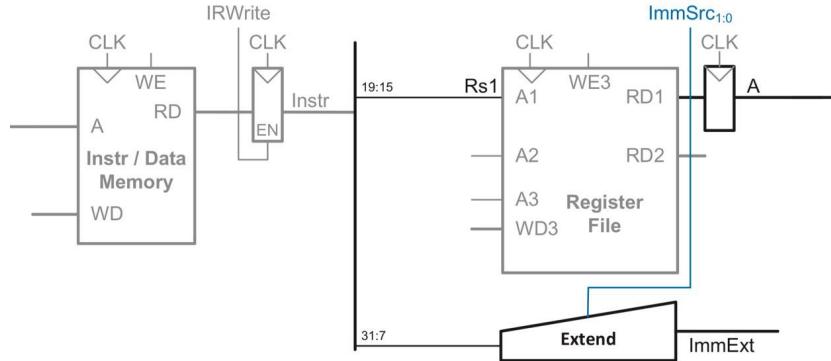
Fonte: Logisim

lógico zero, como podemos ver na **Figura 6**.

Para manter a instrução disponível ao longo de todos os ciclos da execução, se fez necessária a adição de um registrador não arquitetural, que significa, que não pode ser usado a nível de "assembly" sendo um registrador interno como se pode ver na **Figura 8**. Com a finalidade de guardar o valor da saída A do banco de registradores, também foi colocado outro registrador não arquitetural nesta saída, como também podemos ver na mesma figura. Estes registradores relatados acima foram implementados no simulador com o mesmo componente utilizado pelo contador de programa explicado anteriormente.

Na mesma imagem, é visto também uma estrutura chamada "Extend" que é responsável por estender o valor imediato contido na instrução para um tamanho de 32-bit que será usado posteriormente para cálculos na **ULA**. Para realizar essa implementação foi utilizado a mesma estrutura utilizada antes da entrada de endereço da memória principal na **Figura 5** porém, dessa vez, ao em vez de diminuir o tamanho será aumentado para 32-bit. Como visto na **Figura 2** o imediato pode estar em diferentes posições da ins-

**Figura 8: Conexão Entre Memória Principal, Banco de Registradores e Extensor de Sinal**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

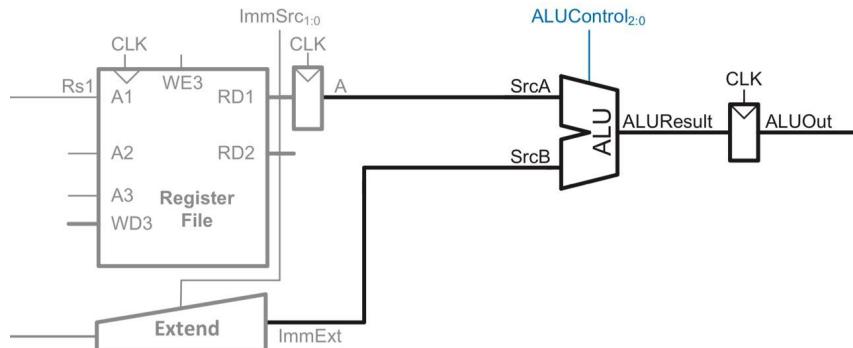
**Tabela 3: Decodificador de Imediatos**

Instrução	Opcode	ImmSrc	ImmExt	Tipo
Tipo-I	00X0011	00	$\{\{20\{\text{Instr}[31]\}\}, \text{Instr}[31:20]\}$	I
SW	0100011	01	$\{\{21\{\text{Instr}[31]\}\}, \text{Instr}[7], \text{Instr}[30:25], \text{Instr}[11:8]\}$	S
BEQ	1100011	10	$\{\{21\{\text{Instr}[31]\}\}, \text{Instr}[7], \text{Instr}[30:25], \text{Instr}[11:8]\}$	B
JAL	1101111	11	$\{\{13\{\text{Instr}[31]\}\}, \text{Instr}[19:12], \text{Instr}[20], \text{Instr}[30:21]\}$	J

trução e também possuir diferentes tamanhos. Para lidar com essas variações, o extensor de sinal (Extend) está equipado com um sinal de controle chamada "ImmSrc" de 2-bit, o qual, segundo a **Tabela 3**, irá selecionar a posição e tamanho correto do imediato baseado no "Opcode" da instrução em execução. Para realizar essa implementação, foi utilizado 4 tipos de extensores de sinais para abranger os tipos possíveis presentes na tabela e foi utilizado o "ImmSrc" como um seletor de multiplexador como mostrado na **Figura 7**.

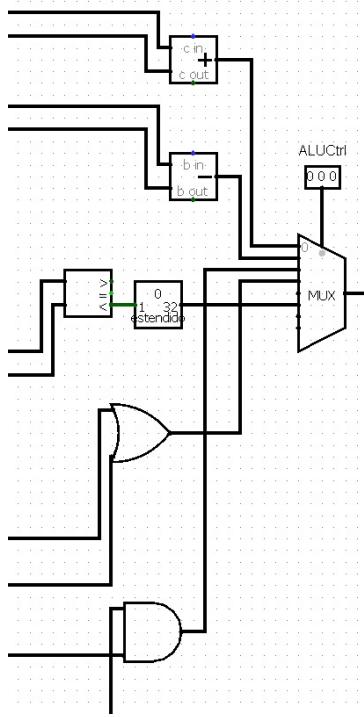
Avançando no caminho de dados, observa-se na **Figura 9** um dos principais componentes do caminho de dados, a ULA. Ela é responsável pela realização de operações no

**Figura 9: Conexão Entre Banco de Registradores, Extensor de Sinal e ULA**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

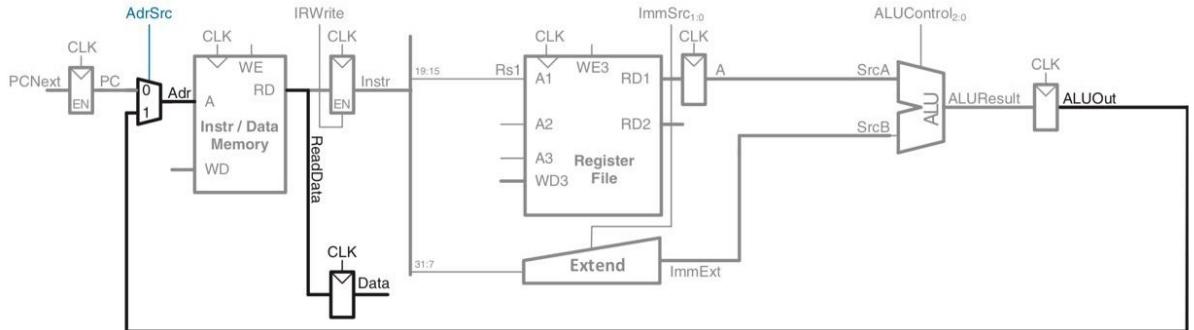
**Figura 10: Estrutura Interna da ULA no Logisim**



sistema e possui duas entradas "SrcA" e "SrcB" os quais estão conectados às saídas "A" da memória e do extensor de sinal, respectivamente. Também destacam-se mais duas coisas, uma delas sendo a presença de um sinal de controle chamado "ALUControl<sub>2:0</sub>" o qual controla qual tipo de operação a ULA irá realizar no ciclo atual. Também é evidente mais um registrador não arquitetural o qual tem como objetivo guardar o resultado da saída da ULA para ser utilizado nos próximos ciclos de relógio. O simulador em questão, não disponibiliza uma estrutura de ULA mas nos permite utilizar operadores pré construídos como soma, subtração, comparação e operadores lógicos que já são suficientes para implementar as operações que se dispõe a realizar esta ISA. Como é visto na **Figura 10** foi utilizado os diferentes blocos de operações que o Logisim dispõe e conectados a um multiplexador fazendo sinal de controle "ALUControl" funcionar como um seletor.

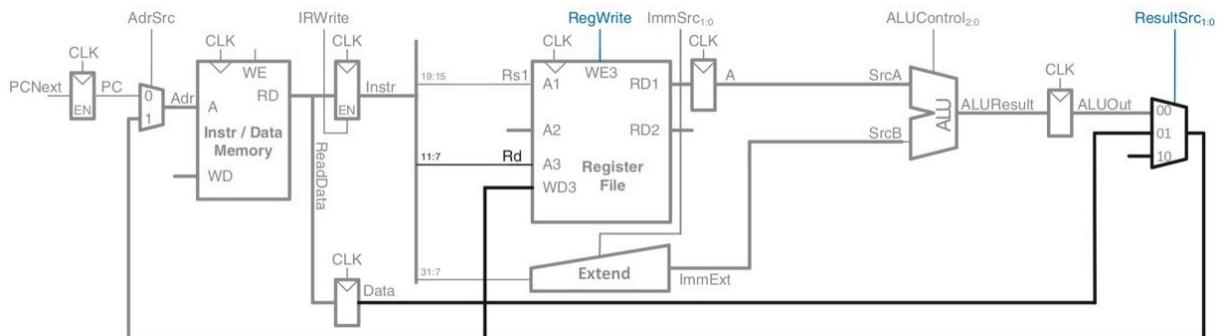
Muitas instruções, após calcular um endereço de memória, necessitam ler o dado contido nesse endereço. Para isso, se faz necessária uma conexão do registrador da saída da ULA com a entrada "A" da memória principal, como mostrado na **Figura 11**. O dado é carregado no registrador não arquitetural que está conectado na saída da memória. Note também que, para isso ser realizado, foi necessária a adição de um multiplexador com um sinal de controle "Adr", que tem como objetivo selecionar se o endereço que será lido da memória virá de "PC" ou da "ULA". Após isso, para finalmente carregar um dado da memória em um registrador é necessário outro registrador intermediário não arquitetural e também um outro multiplexador na saída da "ULA" com um sinal de controle "ResultSrc<sub>1:0</sub>", como pode-se notar na **Figura 12** o qual irá definir de qual fonte

**Figura 11: Carregar Dados da Memória**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 12: Escrever Dado De Delta no Banco de Registradores**



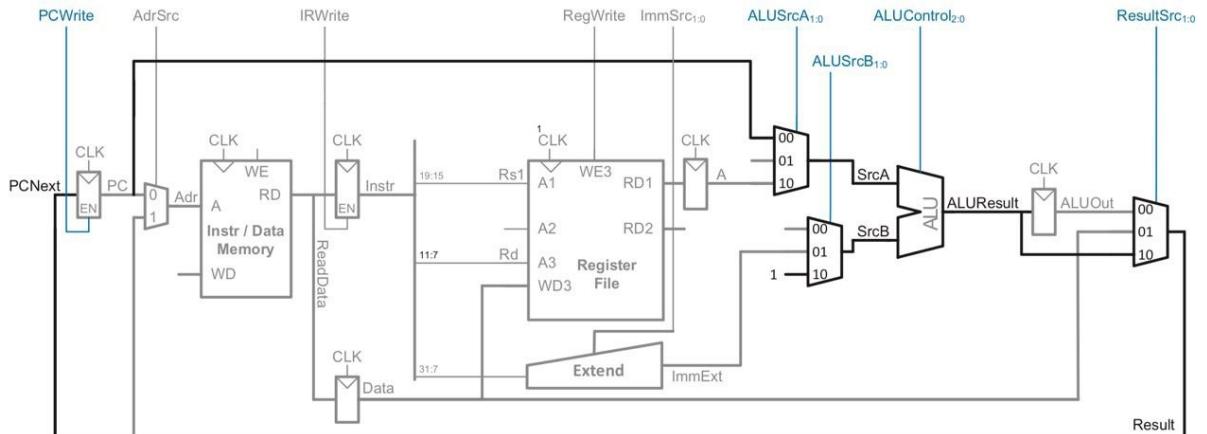
Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

a entrada de dados do banco de registrador irá ler.

Enquanto tudo isso acontece, é necessário incrementar o endereço do contador de programa +1 (lembrando que como as células da memória principal são de 32-bit, os incrementos devem acontecer de um em um endereço, e não de quatro em quatro, caso as células fossem de 8 bits), para possibilitar essa tarefa, foi necessário a adição de dois multiplexadores, um em cada entrada da "ULA". O que foi colocado para a entrada "SrcA" tem um sinal de controle que se chama "ALUSrcA<sub>1:0</sub>" e tem com função selecionar entre a saída do banco de registradores ou a saída de "PC". Já o que foi posto na entrada do "SrcB" tem como função selecionar a saída do registrador (essa veremos mais a frente), do extensor de sinal ou uma constante 1.

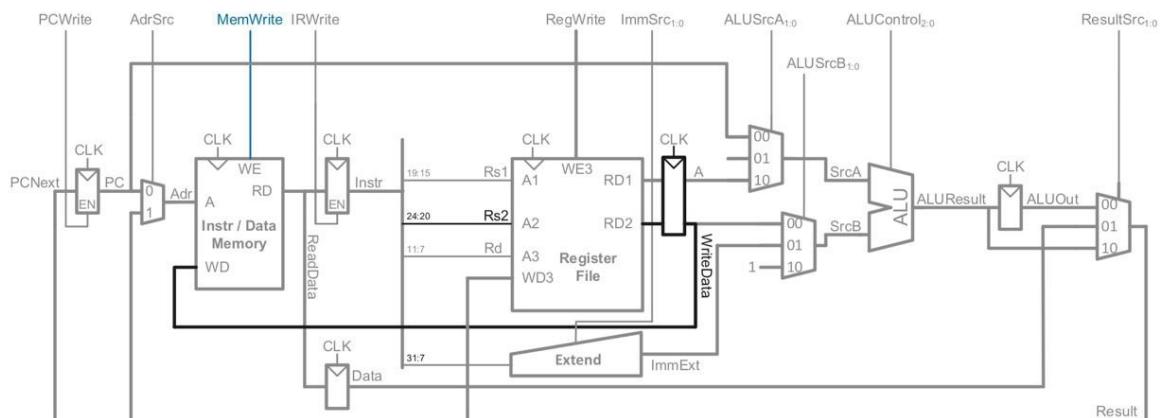
Agora para poder fazer o caminho inverso, que é guardar dados vindo do banco de registradores na memória, é necessário criar uma conexão entre a saída do banco e a entrada de dados "WD" da memória principal, conexão esta que é vista na **Figura 14**. Além dessa conexão, pode-se observar que foi criado outro registrador não arquitetural denominado de "WriteData" que tem como função guardar o dado proveniente da saída "RD2" do banco de registradores. Nota-se que tanto os registradores da saída do Banco

**Figura 13: Endereço de PC + 1**



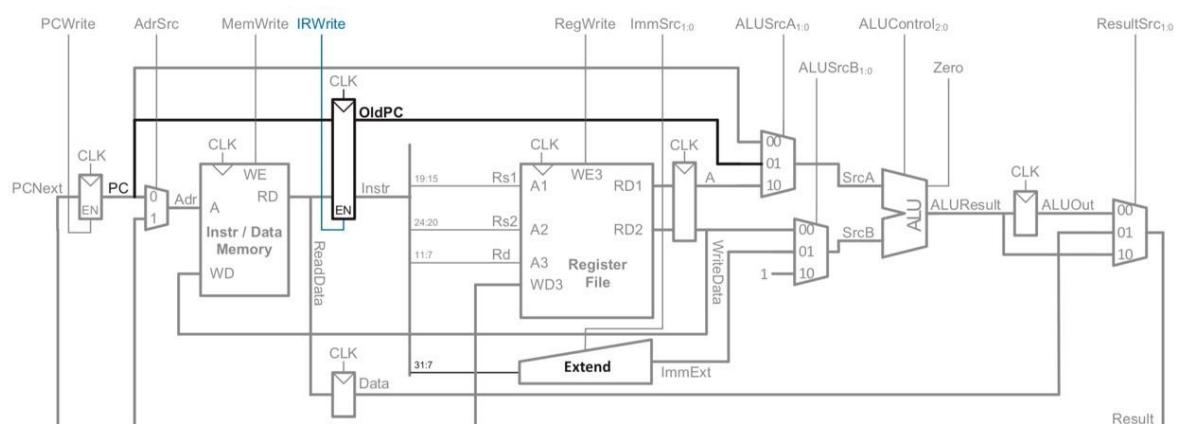
Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 14: Escrita de Dado do Banco de Registradores na Memória**



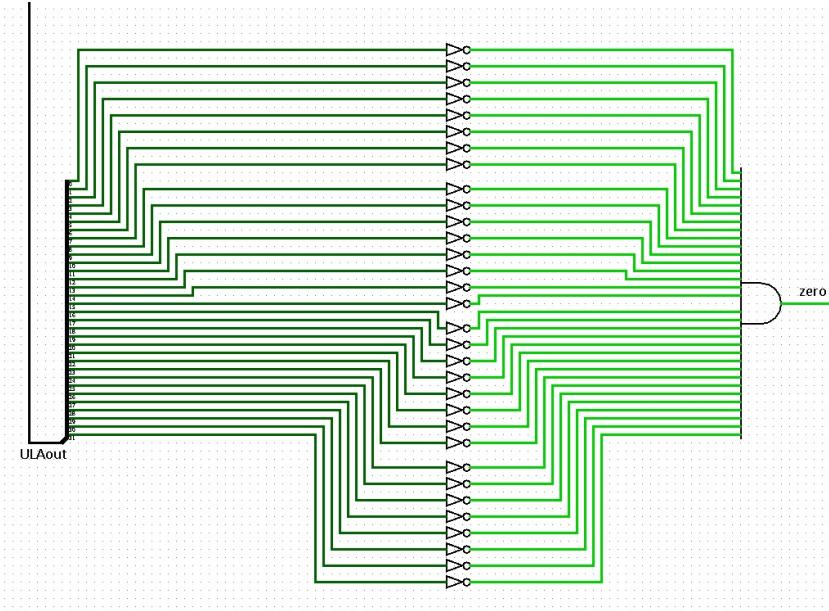
Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 15: Cálculo de Endereço Alvo Para Instrução BEQ**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 16: Estrutura Interno à ULA para ”flag”Zero**



**Fonte:** Logisim

de Registradores quanto o da saída da ULA, não possuem sinais de controle, portanto, todo ciclo de relógio eles são escritos para ser usados no ciclo seguinte.

Observa-se que no segundo ciclo de relógio a ULA não está sendo utilizada, portanto será utilizado para calcular  $PCTarget = PC + ImmExt$  que é um cálculo necessário para a instrução BEQ. Nesse ponto a instrução de PC já foi carregada e incrementada ( $PC+1$ ). Portanto para isso é necessário guardar o endereço de instrução anterior (OldPC) em um registrador não arquitetural cujo podemos ver na **Figura 15**. Note agora que o sinal de controle "ALUSrcA<sub>1:0</sub>" agora também pode selecionar "OldPc".

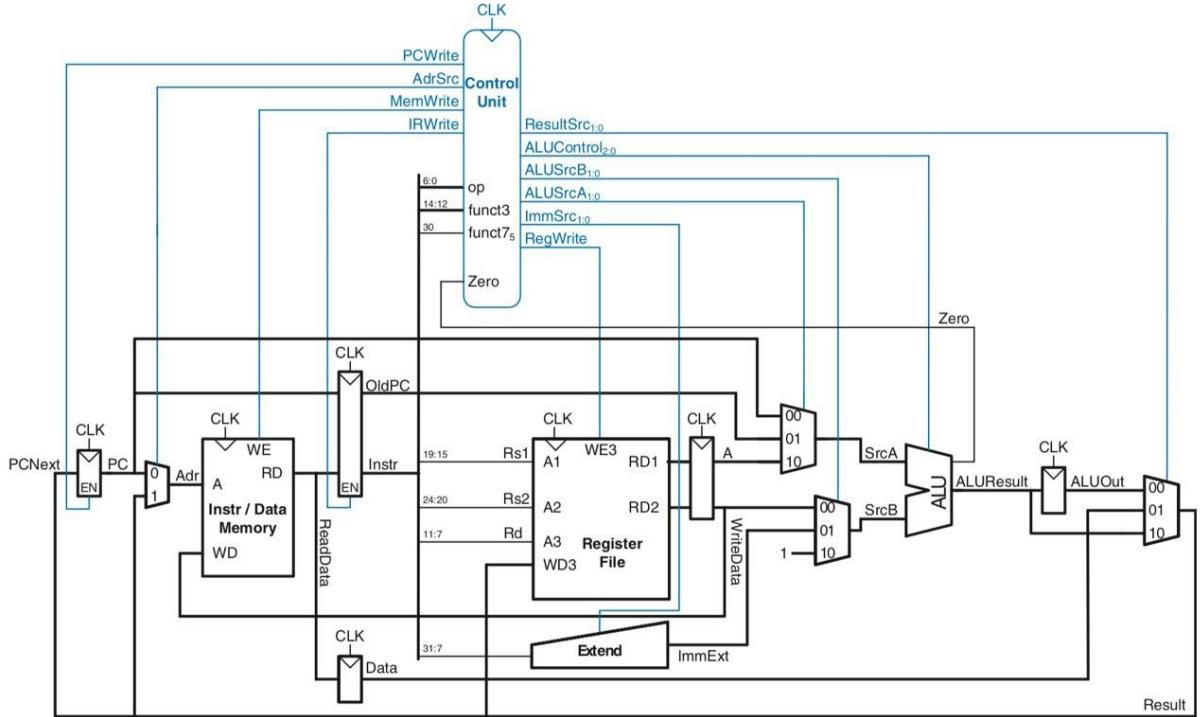
Por fim, pode-se notar que a ULA ganhou também uma saída chamada "zero". Esta saída funciona como uma "flag" de 1 bit que tem como valor "1" quando o resultado que sai da ULA for "0" e caso o resultado seja qualquer outro valor, a saída é "0". Esta estrutura no simulador foi implementada como descrito na **Figura 16** que é basicamente fazer uma operação AND (E) em todas as os 32 bit negados da saída da ULA.

Com isso está completo o projeto do caminho de dados multiciclo para a arquitetura em questão. Na próxima seção, será dado continuidade a construção do processador com o controle da máquina de estados finita.

## 5.2 Unidade de Controle

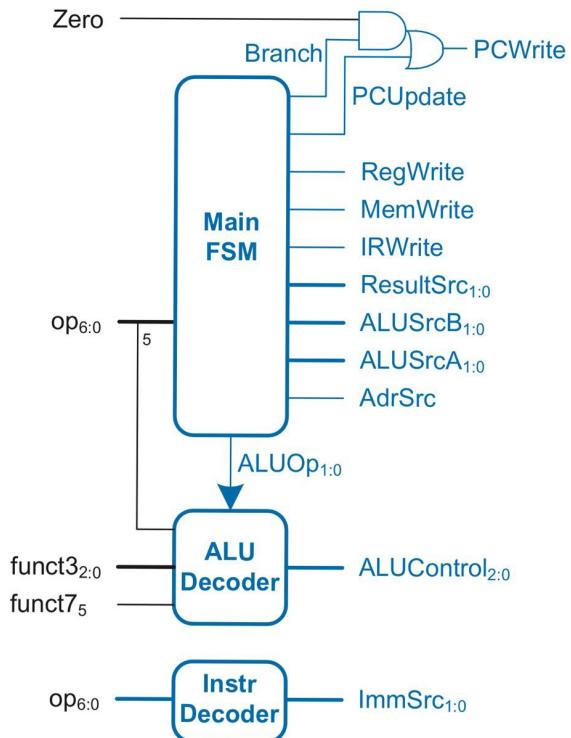
A unidade de controle decodifica os sinais baseado nos campos **op** (também conhecido como "OpCode"), **funct3** e **func7<sub>5</sub>** contidos nos campos Instr<sub>6:0</sub>, Instr<sub>14:12</sub> e Instr<sub>30</sub>

Figura 17: Processador completo



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 18:** Unidade de Controle



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 19: Tabela Verdade do Decodificador da ULA**

ALUOp	funct3	{op <sub>5</sub> , funct7 <sub>5</sub> }	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

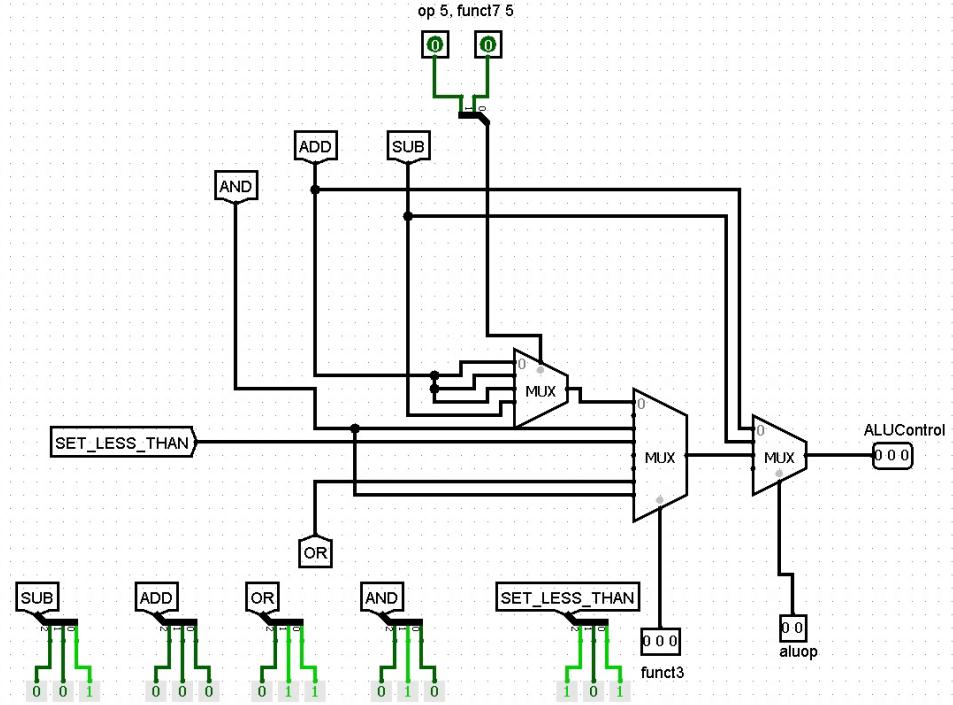
da instrução em execução. A **Figura 17** mostra como todo o caminho de dados se conecta à unidade de controle. Se vê caminho de dados em preto e a unidade de controle em azul.

Para fortalecer o objetivo didático dessa implementação, a unidade de controle foi implementada de forma microprogramada. Essa abordagem permite uma compreensão mais aprofundada e detalhada do funcionamento interno de um processador, facilitando a aprendizagem dos alunos

A unidade de controle é formada pela máquina de estados finita principal (Final State Machine ou FSM), decodificador da ULA (ALU Decoder) e decodificador de instrução (Instr Decoder) como pode-se notar na **Figura 18**. O decodificador da ULA é um circuito combinacional baseado na tabela verdade vista na **Figura 19** e foi implementado no simulador com uma combinação de multiplexadores como podemos ver na **Figura 20**. O decodificador de instrução ou decodificador do imediato é também um circuito combinacional baseado na **Tabela 3**, cujo já discutimos na **Seção 5.1** do Caminho de Dados. Foi implementado também com uma combinação de multiplexadores como mostra a **Figura 25**.

Já a máquina de estados finita (FSM) foi desenvolvida como uma máquina de Moore, onde as saídas só dependem do atual estado. Na **Figura 20** vemos o fluxo inteiro de estados da máquina em questão, os sinais descritos são apenas os relevantes para os estados em questão podendo considerar os outros sinais que não estão explícitos como "don't care" ou não importa. Como já bem se sabe, se trata de um circuito sequencial, pois os próximos estados dependem do estado anterior, é necessário uma estrutura de memória para ser implementada. Como mostra **Figura 27**, um sistema que é composto por 4 componentes. A memória ROM que é utilizada para armazenar uma lista de estados (microinstruções), um registrador não arquitetural que guarda o estado atual(mPC), um somador que é empregado para incrementar o valor do atual estado, além disso, existe uma

**Figura 20: Estrutura Interna do Decodificador da ULA**



Fonte: Logisim

**Tabela 4: Ações Implementadas Pela Unidade da Lógica de Seleção de Endereço (LSED)**

Estado Atual	Ação
S0 ou S3	SEQ
S1	DISPATCH_1
S2	DISPATCH_2
S6,S8 ou S9	ALUWBT
S4,S5,S7 ou S10	FETCH

estrutura denominada "Lógica de seleção de endereço" (LSED), cujo objetivo é determinar o próximo endereço selecionado na memória ROM com base no estado atual. Em suma, a memória armazena os possíveis estados, o registrador indica o atual estado, o somador incrementa o estado atual e a "Lógica de seleção de endereço" determina o próximo estado a ser acessado na memória ROM, possibilitando assim o bom funcionamento da máquina de estados.

Esta **LSED** implementa 5 tipos de ações diferentes, que podem ser observadas na **Tabela 4**, baseadas no estado atual que podem ser observados na **Figura 26**. A ação "DISPATCH\_1" o qual esta descrita em detalhes na **Tabela 5** implementa a seleção de caminho a ser tomado no estado "S1: Decode", visto na **Figura 26**, já a ação "DISPATCH\_2" implementa a tomada de decisão do estado "S2: MemAdr" também visto na

**Tabela 5: Tabela de Despacho 1**

OpCode	Próximo Estado
0X00011	S2: MemAdr
0110011	S6: ExecuterR
0010011	S8: ExecuteI
1101111	S9: JAL

**Tabela 6: Tabela de Despacho 2**

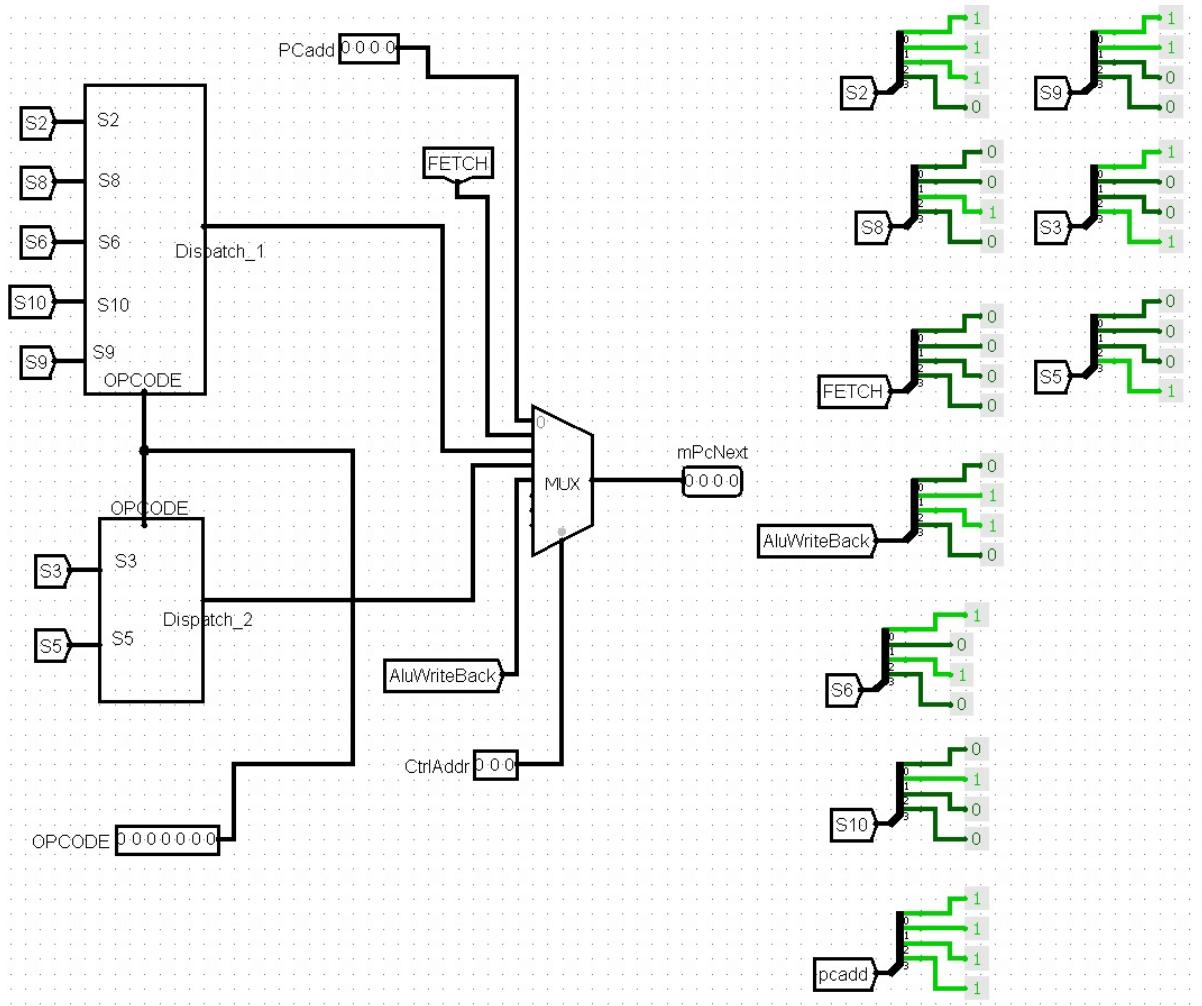
OpCode	Próximo Estado
0000011	S3: MemRead
0100011	S2: MemWrite

**Figura 26** e detalhada na **Tabela 6**. Observa-se que as ações designadas por ambas tabelas de despacho (DISPATCH\_1 e DISPATCH\_2), vistas nas **Tabelas 5 e 6**, dependem unicamente do **OpCode** da instrução em questão. Já a ação "ALUWB" implementa a seleção de um endereço específico que contém a microinstrução correspondente ao estado "S7: ALUWB" visto também na **Figura 26**. O "SEQ" refere-se ao resultado do somador que é visto na **Figura 27**, o qual soma o endereço contido em  $mPC + 1$ , que significa que o endereço da próxima microinstrução é o endereço subsequente. Por fim o "Fetch" faz com que o ciclo volte ao inicio fazendo "mPc" apontar para a primeira microinstrução da memória (Endereço x0). Como se pode observar na **Figura 27**, existe um sinal chamado "CtrlAdr" que provém da microinstrução selecionada e serve de realimentação para a LSED. Este sinal indica qual ação a LSED tomará no atual ciclo de relógio que é a codificação do que foi dito como "Estado Atual" anteriormente e é codificado segundo a **Tabela 8**.

Na **Figura 21** observa-se a implementação no simulador do **LSED** onde o sinal de controle "CtrlAdr" (3-bit) é usado como um seletor em um **MUX** o qual implementa a **Tabela 8** e ao lado direito da mesma imagem, há uma lista de estados decodificados em endereço de memória indicados na **Tabela 7**. Os sub circuitos "**DISPATCH\_1**" e "**DISPATCH\_2**" que podem ser vistos em detalhe nas **Figuras 22 e 23** implementam as **Tabelas 5 e 6** respectivamente, os quais dependem unicamente do "OpCode" da instrução em questão. Verifica-se que a única entrada do **MUX** que não está codificada ao lado direito da **Figura 21** é o "PCAdd". Isso ocorre porque ele é um sinal proveniente de um somador com o endereço atual do micropograma ( $mPC$ ), ou seja, " $mPC + 1$ ", o qual implementa a ação de "SEQ", que é o endereço subsequente.

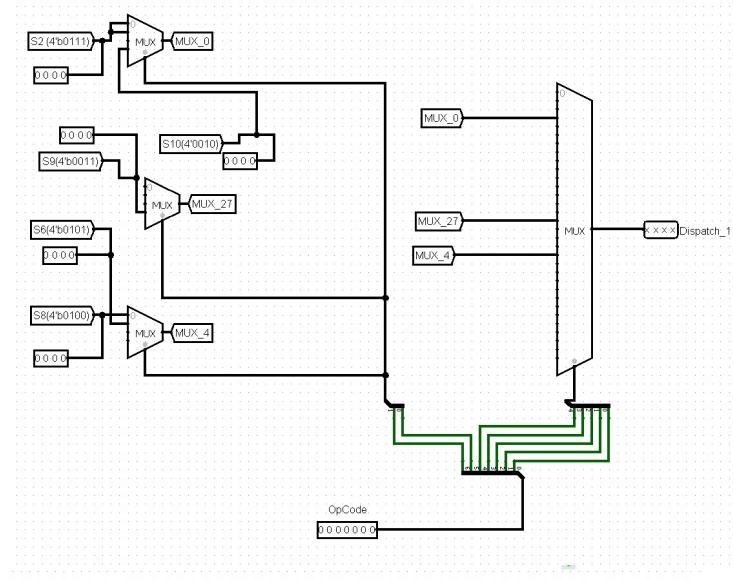
Para o correto funcionamento das lógicas apresentadas até então, faz-se necessário definir a estrutura de como as microinstruções estão dispostas em cada endereço da memória ROM [25]. As correspondências de endereço de memória do micropograma (microinstrução), equivalência do micropograma em estado da **FSM** e sinal de controle

**Figura 21: Estrutura Interna da Lógica de Seleção de Endereço (LSED)**



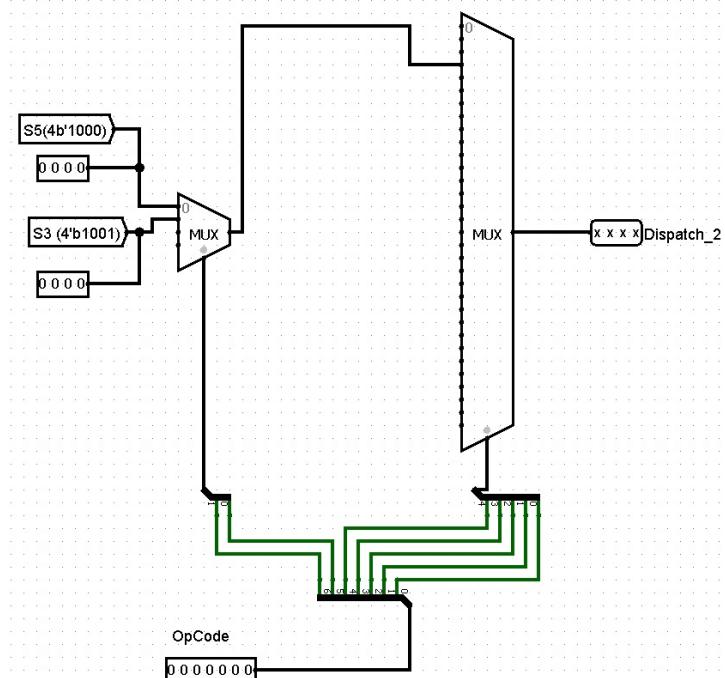
**Fonte: Logisim**

Figura 22: Implementação da Tabela de Despacho 1



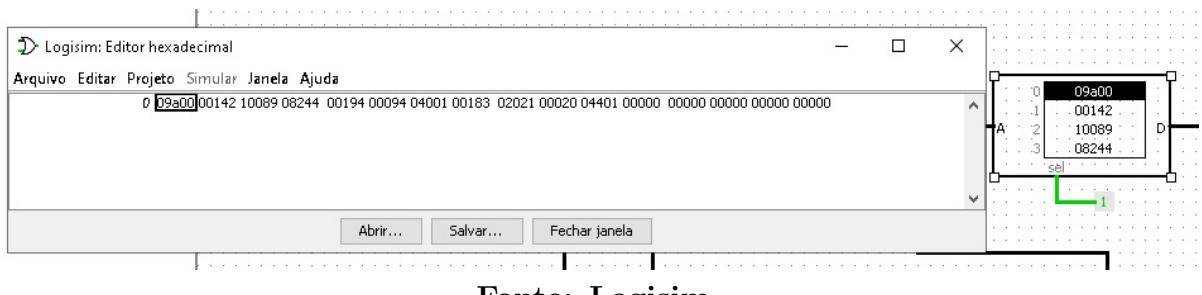
Fonte: Logisim

Figura 23: Implementação da Tabela de Despacho 2



Fonte: Logisim

**Figura 24: Implementação de Memória ROM em Logisim**



Fonte: Logisim

**Tabela 7: Organização da Memória de Microprogramas**

Endereço de Memória	Estado (Figura 26)	CtrlAdr (Tabela 8)
0000	S0: Fetch	000
0001	S1: Decode	010
0010	S10: BEQ	001
0011	S9: JAL	100
0100	S8: ExecuteI	100
0101	S6: ExecuteR	100
0110	S7: ALUWB	001
0111	S2: MemAdr	011
1000	S5: MemWrite	001
1001	S3: MemRead	000
1010	S4: MemWB	001

”CtrlAdr” que, como visto na **Tabela 8**, codifica a próxima ação que será tomada pelo LSED, podem ser vistas na **Tabela 7**. Esta tabela define exatamente qual endereço de memória será ocupado por um determinado estado (microinstrução) além de mostrar o sinal de controle ”CtrlAdr” o qual será decodificado para definir qual próximo estado a FSM tomará. Por fim os sinais de controle da ”Main FSM” (**Figura 18**) necessários para execução de cada microprograma, estão definidos em cada estado (ou microinstrução) na **Figura 26**, os sinais os quais não estão descritos, podem ser considerados como não importa (“don’t care”), como já visto.

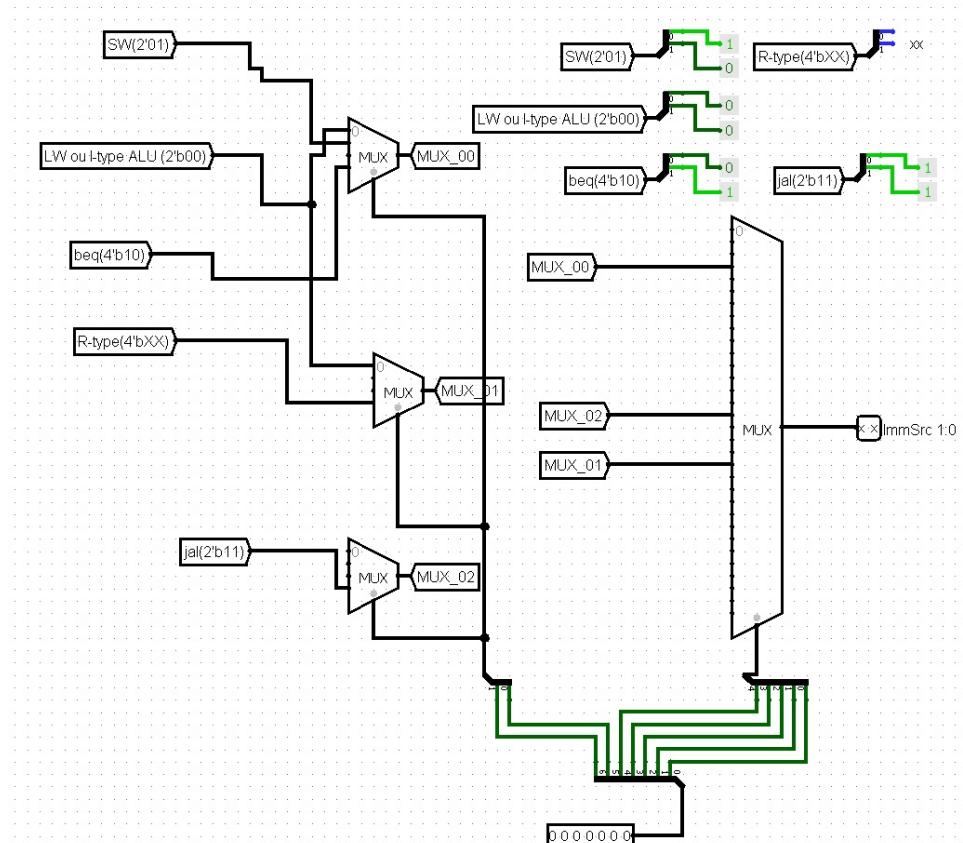
A nível de implementação no Logisim, a memória **ROM**, usada na ”Main FSM” (**Figura 27**) é um bloco pré definido da estrutura do simulador, o qual pode ter o seu conteúdo modificado, como pode ser visto em detalhes na **Figura 24**.

Para concluir, pode-se observar que as instruções do processador precisam de 3 ciclos para instruções do tipo B, 4 ciclos para instruções do tipo R, I, J, S e 5 ciclos para load word. Com isso finalizamos a unidade de controle.

Tabela 8: Codificação do Sinal "CtrlAdr" da Microinstrução

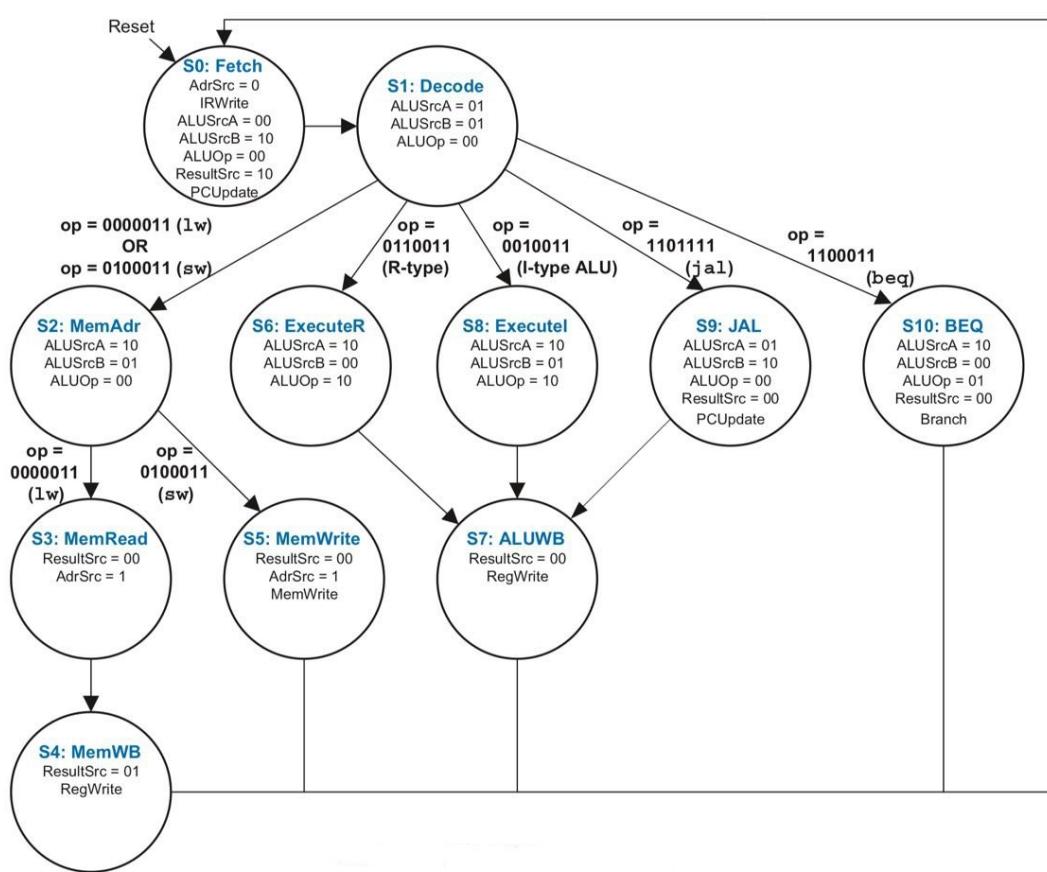
CtrlAdr	Ação
000	SEQ
001	FETCH
010	DISPATCH_1
011	DISPATCH_2
001	ALUWUB

Figura 25: Estrutura Interna do Decodificador de Instrução



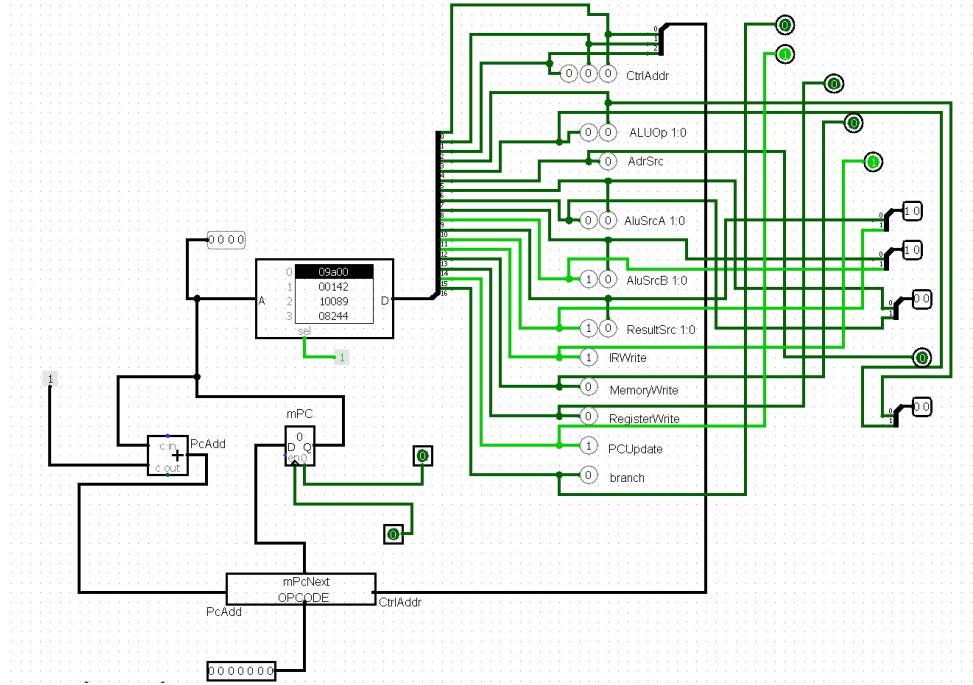
Fonte: Logisim

**Figura 26: Fluxograma de Estados da "Main FSM"**



Fonte: Digital Design and Computer Architecture: RISC-V Edition [12]

**Figura 27: Estrutura Interna da "Main FSM"**



Fonte: Logisim

## 6 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Este capítulo apresenta a análise de resultados e é uma parte essencial de uma produção científica. Aqui, os dados coletados ao longo da pesquisa serão apresentados de forma estruturada e detalhada, possibilitando a análise e interpretação dos resultados obtidos. Essa fase é de suma importância para a análise dos objetivos propostos anteriormente no presente trabalho e formular conclusões com base nessa avaliação, fornecendo também aos pesquisadores uma análise crítica dos resultados, contribuindo assim para a construção do conhecimento coletivo.

### 6.1 Teste de Execução de Programa

Implementado o caminho de dados e unidade de controle, que encontra-se em um repositório público [4], operando com as instruções contidas na **Tabela 1**, foi definido o programa da **Figura 28** como objeto de teste, afim de comprovar o correto funcionamento do processador. Este dito programa tem como objetivo ler o dado de uma determinada posição de memória, somá-lo à uma constante e salvar o resultado da soma na mesma posição de memória anteriormente lida. Esta rotina é repetida até que o valor guardado na memória atinja um determinado valor (valor esse estabelecido por uma segunda constante). Os valores referentes aos imediatos (1-6) vistos na **Figura 28** estão descritos na **Tabela 9**. Os registradores **zero**, **t0**, **t1**, **s1** e **s2**, utilizados no programa, têm seus

**Figura 28: Programa Exemplo**

```

1 addi t1, zero, imm1;
2 addi s1, zero, imm2;
3 addi s2, zero, imm3;
4 lw t0, 0(s1);
5 add t0, t0, t1;
6 sw t0, 0(s1)
7 beq t0, s2, imm4;
8 jal zero imm5;
9 add zero, zero, zero;
10 jal zero imm6;

```

**Tabela 9: Representação dos Imediatos**

Imediato	Formato	Valor (decimal)	Valor (binário)
imm1	11:0	2	000000000010
imm2	11:0	70	000001000110
imm3	11:0	6	0000000000110
imm4	11:0	2	000000000010
imm5	19:0	-4	11111111111111111110
imm6	19:0	-1	11111111111111111111

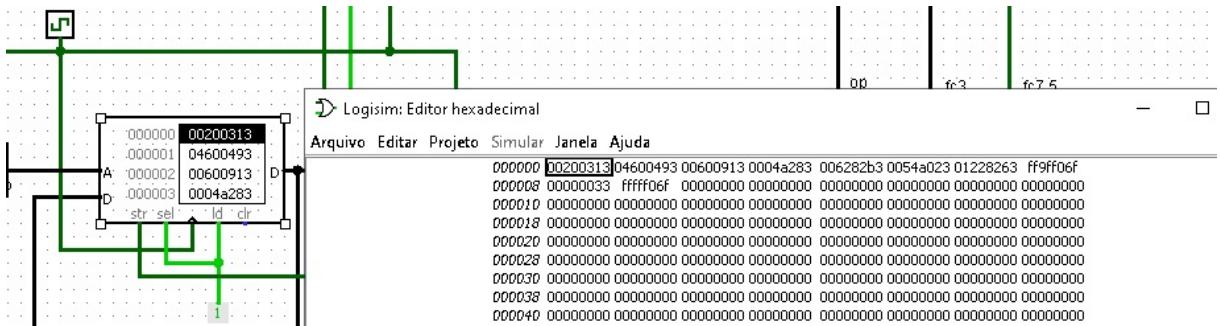
endereços codificados na **Tabela 2**.

As linhas de 1 a 3 do programa, (**Figura 28**) têm como função salvar os imediatos (constantes) nos respectivos registradores **t1** (valor que será usado como somador), **s1** (endereço de memória arbitrário) e **s2** (valor que será usado para salto de instrução). A linha 4 carrega o dado contido no endereço de memória **s1**, no registrador **t0**. Na linha 5 os valores contidos nos registradores **t0** e **t1** são somados e o resultado é salvo no registrador **t0**. Já na linha 6, o resultado da soma que está contido em **t0** é salvo de voltado ao endereço de memória contido em **s1**. A linha 7, verifica se os valores contidos nos registradores **t0** e **s2** são iguais, caso negativo, a linha 8 é executada, que soma o valor contido no **imm5** com o endereço de memória atual, fazendo com que, o novo endereço de memória seja deslocado de volta para a linha 4 repetindo assim o processo. Ainda na linha 7, caso os valores sejam iguais (caso afirmativo), o endereço de memória atual é somado ao **imm4** fazendo com que o novo endereço de memória seja o da linha 9. Esta linha, por sua vez, soma o valor contido no registrador **zero** com ele mesmo e o resultado é descartado, pois o registrador **zero** é o valor zero e de somente leitura, o que quer dizer que essa linha não faz nada (“No Operation”). Por fim, a linha 10 tem como objetivo fazer o endereço de memória voltar para a linha 9 somando o **imm6** ao endereço de memória atual fazendo com que o programa termine em um laço infinito.

**Tabela 10: Programa Traduzido em Linguagem de Máquina**

Linha (Figura 27)	Tradução (binário)	Tradução (hexadecimal)
1	32b'0000000000001000000000001100010011	8h'00200313
2	32b'00000100011000000000010010010011	8h'04600493
3	32b'000000000011000000000100100010011	8h'00600913
4	32b'0000000000000000100101000101000011	8h'0004A283
5	32b'000000000011000101000001010110011	8h'006282B3
6	32b'00000000001010100101000000100011	8h'0054A023
7	32b'000000001001000101000001001100011	8h'01228263
8	32b'111111110011111111000001101111	8h'FF9FF06F
9	32b'00000000000000000000000000000000110011	8h'00000033
10	32b'1111111111111111000001101111	8h'FFFFF06F

**Figura 29: Programa Carregado em Memória**



**Fonte: Logisim**

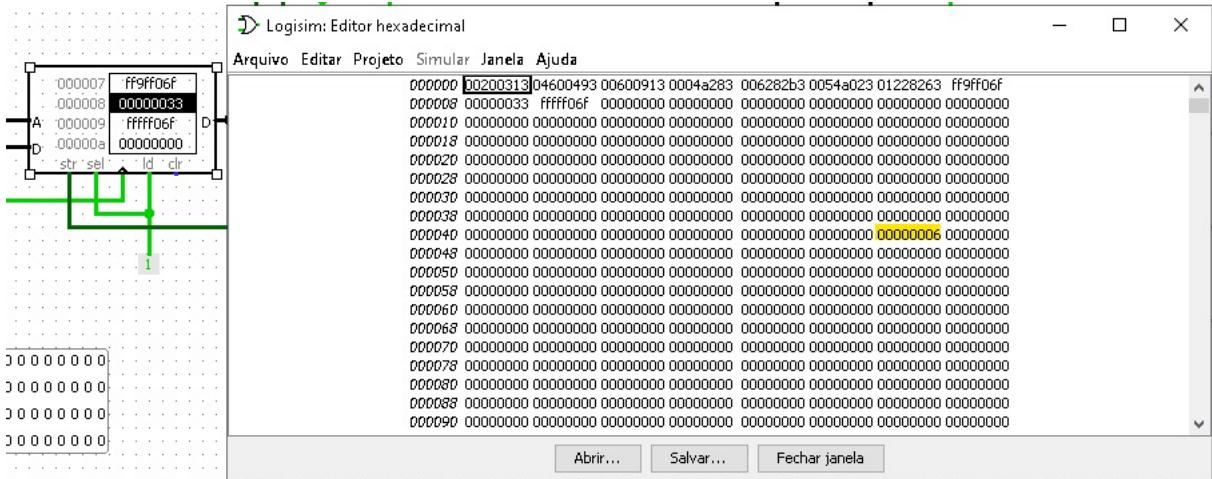
Visto o programa estruturado, agora é necessário fazer a tradução para a linguagem de máquina (**Figura 10**), para assim, poder ser carregado em memória e executado pelo simulador. Cada linha do programa da **Figura 28** foi traduzido com o auxílio das **Tabelas 1 e 2** e **Figura 2**. Tendo o programa traduzido, ele deve ser carregado na memória principal da implementação no Logisim como visto na **Figura 29**. Observa-se, também, que a memória do Logisim pede a tradução em hexadecimal. Após carregar o programa e habilitar a simulação é visto na **Figura 30** que o programa foi executado corretamente pois observa-se o resultado (**8h'00000006**) na posição de memória desejada, validando assim, o funcionamento do processador.

## 6.2 Resultados de Viabilidade

Analizando primeiramente o poder computacional necessário para executar o simulador LOGISIM, nota-se que em sua própria documentação onde fala:

”Ele roda em qualquer máquina que suporte Java 5 ou posterior; versões especiais são lançadas para MacOS X e Windows. A natureza multiplataforma é

**Figura 30: Programa Executado**



**Fonte: Logisim**

importante para estudantes que possuem uma variedade de sistemas de computadores em casa/dormitório.” [2]

ficando claro, portanto, que é um software portátil e de fácil acesso.

O impacto de um simulador vem da considerável diferença entre apenas observar uma microarquitetura e manipular de fato uma microarquitetura. Interagir com o controle de um caminho de dados, mesmo que de forma simulada, permite que alunos compreendam de forma bem mais clara e natural uma série de detalhes que comumente passam desapercebidos quando o estudo feito é apenas teórico.

Agora será analisado a complexidade e nível de dificuldade para adicionar uma nova instrução ao subconjunto estabelecido do processador questão. Para realizar essa tarefa se faz necessário modificações tanto no caminho de dados quanto na unidade de controle como adição de um novo ”OPCODE”(que implica em uma nova codificação), novos registradores não arquiteturais, extensor de sinal, ULA e memórias. Modificações essas, detalhadas e analisadas nos próximos parágrafos.

Preparar uma nova decodificação caso a nova instrução tenha um novo tipo que a nossa arquitetura não prevê, essa tarefa seria tão complexa quanto a complexidade de adicionar um novo ”OPCODE” o que implicaria em um novo caminho na máquina de estados, julgando assim uma tarefa de complexidade alta caso a nova instrução seja de um novo tipo pois iria ser necessária a modificação de quase todas a estrutura, principalmente, de máquina de estados que implica em uma modificação na unidade de controle.

A próxima possível modificação, seria a adição de novos registradores para lidar com a necessidade de armazenamento intermediário no caminho de dados, essa tarefa no LOGISIM tem complexidade baixa, visto que se trata de um simulador de bloco, seria

**Tabela 11: Conclusão Qualitativa**

Modificação	Descrição	Grau de Dificuldade
OPCODE	Código presente na instrução que indica o seu tipo.	Alto
ULA	Unidade responsável por efetuar operações lógicas e aritméticas.	Baixo
BANCO DE REGISTRADORES	Componente responsável por implementar as múltiplas leituras e escritas nos registradores.	Medio
MEMÓRIA PRINCIPAL	Componente responsável por salvar e manter disponível o programa em execução bem como.	Baixo/Alto
EXTENSOR DE SINAL	Responsável por transformar os imediatos para 32-bit	Baixo

apenas arrastar e conectar corretamente o novo registrador onde deseja.

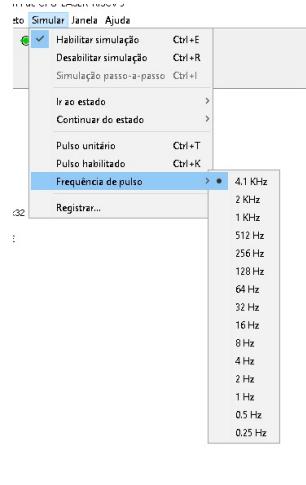
Já modificações em unidades funcionais como extensor de sinal, ULA, memória principal e banco de registradores, nesse caso, o nível de complexidade irá depender de qual unidade iremos modificar, no caso das unidades de extensor de sinal e ULA, a complexidade é baixa visto que têm uma implementação simples como já vimos anteriormente, porém quando se fala de banco de registradores, é uma tarefa de complexidade média visto que se trata de um banco de registradores implementado em flip-flop's. Na memória existe uma limitação relacionada ao LOGISIM, caso a modificação necessária seja suportada pelo simulador, a complexidade é baixa, caso contrário, terá uma alta complexidade pois teríamos que implementar uma memória do zero atendendo os requisitos desejados. O extensor de sinal tem complexidade baixa de modificação devido ao suporte dos componentes presentes no LOGISIM os quais foram utilizados para a implementação do extensor.

Por fim, é notório que o nível de dificuldade de implementação da nova instrução é inversamente proporcional à semelhança que essa nova instrução tem com outra pré existentes, ou seja, quanto mais diferente a nova instrução for, mais complexo de modificar a nossa implementação a fim de comporta-la corretamente. A **Tabela 11** mostra a relação discutida anteriormente de forma compilada de componentes e nível de complexidade de modificação.

### 6.3 Resultados Estimados de Desempenho

Para estimar o desempenho do processador implementado no Logisim será utilizar a equação (5) (Milhões de instruções por segundo), encontrada na nossa seção de metodologia, pois, é a métrica mais encontrada em documentos de processadores comerciais que

**Figura 31: Lista de Velocidades de Relógio Disponíveis no Logisim**



**Fonte: Documentação oficial do Logisim [2]**

possibilita a comparação de desempenho com processadores que estão ou que já passaram pelo mercado.

Devido a limitações do próprio Logisim, como visto na Figura 31, a frequência de relógio mais rápida disponível é de 4,1Khz, portanto (6).

$$r = 4,1 \text{ kHz} \quad (6)$$

Como apresentado também na seção de microarquitetura, toda as instruções do nosso conjunto são executadas em 4 ciclos de relógio, com exceção da lw e beq que são executadas em 5 e 3 ciclos de relógio, respectivamente. Com isso se pode calcular que a média de ciclos de relógio por instrução do conjunto é de 4 ciclos (7).

$$CPI = 4 \quad (7)$$

Com esses dados é possível calcular quantas instruções por segundo o processador é capaz de executar simulado no LOGISIM utilizando a equação (5).

$$MIPS = \frac{4100}{4 \times 1.000.000} = 0,001025 \quad (8)$$

Porém, expandindo a implementação dessa arquitetura em uma FPGA (Altera Cyclone II foi usada como base para este cálculo [3]), que têm uma velocidade de relógio padrão de 50Mhz, seria obtido um valor próximo do descrito na equação abaixo.

$$MIPS = \frac{50000000}{4 \times 1.000.000} = 12,5 \quad (9)$$

O que significa que seria possível executar aproximadamente 12,5 milhões de instruções por segundo.

Analizando os resultados das equações, pode-se notar que apesar da quantidade limitada de instruções por segundo, para fins didáticos é um resultado satisfatório a pesar de não expressar o real poder de um processador. Um dos principais motivos para isso acontecer são os recursos limitados de velocidade de relógio.

Portanto se tem um desempenho satisfatório quando o assunto é didática e ensino, visto que podemos ter em mãos a simulação de um processador real que quando implementado, mesmo com a simplicidade do seu conjunto de instruções, pode ser competitivo. Não se pode deixar de ressaltar que uma outra implementação com suporte à pipeline teria um desempenho bem superior com o ônus de ter um sistema de controle e caminho de dados mais complexo. Esse maior desempenho se da pelo fato que quanto a segunda microinstrução de um programa começa a ser executada, em paralelo a primeira microinstrução do próximo programa já também é executada, e assim por diante.

## **7 CONCLUSÕES E TRABALHOS FUTUROS**

### **7.1 Conclusão**

Por meio dessa pesquisa, foi possível constatar que, apesar da implementação ser composta por apenas 12 instruções, a mesma é suficiente para o escopo didático. Essa implementação possibilita a resolução de problemas de média complexidade, gerando um ambiente adequado para o ensino dos conceitos relacionados a Arquitetura de Computadores.

Ao longo do presente trabalho, foi explorado um caminho de dados e uma unidade de controle relativamente simples, que devem ser facilmente compreendidas. O desenvolvimento e análise desses componentes permitem a profunda compreensão dos fundamentos de processadores e arquitetura de computadores. Por meio dessa implementação, foi possível abordar conceitos como mecanismos de tradução de instruções em linguagem de máquina, alinhamento de memória, construção de banco de registradores, extensor de sinal, unidade lógica e aritmética, implementação de máquina de estados em nível lógico, entre outros.

Porém, é importante ressaltar que esta implementação tem limitações devido à restrições do Logisim. Apesar de ser capaz de lidar com problemas de média complexidade, existe limitações quanto a escalabilidade e flexibilidade. Novos componentes ou instruções mais complexas podem não ser tão facilmente implementados dentro do ambiente Logisim.

Em linhas gerais, essa pesquisa remonta a importância didática da implementação de um processador em um simulador com um conjunto básico de instruções sendo considerada valiosa para fins educacionais permitindo a profunda compreensão dos conceitos de arquitetura de computadores.

### **7.2 Resultados e Trabalhos Futuros**

Através da análise dos resultados obtidos, pode-se concluir que a implementação do processador utilizando o Logisim apresentou limitações em termos de velocidade de clock, com a frequência máxima disponível de 4,1 kHz. Isso resultou em um desempenho relativamente baixo, refletido na métrica MIPS (milhões de instruções por segundo), que foi calculada em 0,001025 MIPS.

No entanto, é importante ressaltar que, se considerada uma eventual implementação em uma FPGA [3] com uma frequência de relógio padrão de 50 MHz, o desempenho projetado seria de aproximadamente 12,5 MIPS, o que representa um aumento significativo em relação à implementação no Logisim.

Com base nas limitações identificadas e possíveis direções que podem ser exploradas em trabalhos futuros temos:

**1. Uso de Simuladores mais avançados:**

Utilizar simuladores com maior poder e mais funcionalidades que permita simular tanto a lógica mas também aspectos físicos como consumo de energia e atraso de propagação e que permita a utilização de frequências de relógio mais altas.

**2. Expansão do Conjunto de Instruções:**

Considerar o aumento do número de instruções a fim de expandir a versatilidade e capacidade de resolver problemas mais complexos.

**3. Prototipação em FPGA:**

Considerar desenvolver uma prototipação do presente trabalho em FPGA a fim de analisar o comportamento físico e podendo assim comparar com implementações físicas usando mais métricas.

Explorando esses pontos, será possível aprimorar o desempenho do processador, além de melhorar a coleta de informações, aumentar a sua versatilidade verificar seu potencial de competitividade em ambientes de implementação reais. Estes trabalhos futuros irão contribuir para a evolução do projeto, proporcionando uma sólida base para o desenvolvimento de sistemas embarcados e a compreensão profunda dos conceitos de arquitetura de computadores.

### **7.3 Considerações Finais**

A presente pesquisa, ressaltou a grande importância do estudo da arquitetura de computadores. Ficou evidenciado que, mesmo com uma implementação simples e didática, é possível obter resultados promissores. Encoraja-se, portanto, o contínuo desenvolvimento da área através da exploração de novas técnicas, algoritmos e arquiteturas que possam levar a maiores avanços.

## REFERÊNCIAS

- [1] TUMELERO, Naína. Quer aprender a delimitar a metodologia TCC?. Disponível em:<<https://blog.mettzer.com/metodologia-tcc/>>. Acesso em: 11 de Nov 2022.
- [2] Documentação oficial do logisim disponível em:<<http://www.cburch.com/logisim/>>. Acesso em: 07 de Jun 2023.
- [3] Conheça a FPGA Altera Cyclone II em:<<https://www.makerhero.com/blog/conheca-a-fpga-altera-cyclone-ii/>>. Acesso em: 19 de Jun 2023.
- [4] Repositório de implementação do presente trabalho em:<<https://github.com/ewertonsantos/CPU-LASER-RISCV/>>. Acesso em: 19 de Jun 2023.
- [5] Histórico do número de instruções por segundo para processadores comerciais disponível em:<[https://computer.fandom.com/wiki/Instructions\\_per\\_second](https://computer.fandom.com/wiki/Instructions_per_second)>. Acesso em: 07 de Jun 2023.
- [6] Micro Magic apresenta chip RISC-V que alcança 5GHz usando apenas 1 Watt disponível em:<<https://mundoconectado.com.br/noticias/v/16508/micro-magic-apresenta-chip-risc-v-que-alanca-5ghz-usando-apenas-1-watt>>. Acesso em: 07 de Jun 2023.
- [7] Liga Koreana de Processadores Extreme League em:<[https://hwbot.org/submission/5179170\\_phantom\\_k\\_7\\_zip\\_core\\_i9\\_13900ks\\_255873\\_mips](https://hwbot.org/submission/5179170_phantom_k_7_zip_core_i9_13900ks_255873_mips)>. Acesso em: 07 de Jun 2023.
- [8] SANTOS, Douglas Almeida et al. A low-cost fault-tolerant RISC-V processor for space systems. em: 2020 15th Design and Technology of Integrated Systems in Nanoscale Era (DTIS). IEEE, 2020. p. 1-5.
- [9] PATSIDIS, Karyofyllis et al. A low-cost synthesizable RISC-V dual-issue processor core leveraging the compressed Instruction Set Extension. *Microprocessors and Microsystems*, v. 61, p. 1-10, 2018.
- [10] wikipedia.CISC:<<https://pt.wikipedia.org/wiki/CISC>>. Acesso em: 22 de Dez 2022.
- [11] wikipedia.CISC:<<https://pt.wikipedia.org/wiki/RISCC>>. Acesso em: 22 de Dez 2022.
- [12] Harris, S., & Harris, D. (2016). *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann.

- [13] PATTERSON, David. Computer organization and design RISC-V edition: the hardware. 2017.
- [14] Register File Block Diagram disponível em:<[http://www.cs.uni.edu/~fiernup/cs041s05/lectures/lec8\\_Register\\_File.htm](http://www.cs.uni.edu/~fiernup/cs041s05/lectures/lec8_Register_File.htm)>. Acesso em: 07 de Jun 2023.
- [15] MATTOS, Mauro Marcelo; FARIAS, Jorge Sampaio; RENALDI, Filipe. VXT: um ambiente didático para ensino de conceitos básicos de sistemas operacionais e arquitetura de computadores. In: WORKSHOP DE COMPUTAÇÃO DA REGIÃO SUL. 2004.
- [16] MURDOCCA, Miles J.; HEURING, Vincent P. Introdução à arquitetura de computadores. Elsevier, 2001.
- [17] MORANDI, Diana; RAABE, André Luis Alice; ZEFERINO, Cesar Albenes. Processadores para ensino de conceitos básicos de arquitetura de computadores. In: Workshop sobre Educação em Arquitetura de Computadores-WEAC. sn, 2006. p. 17-24.
- [18] HENNESSY, John et al. MIPS: A microprocessor architecture. ACM SIGMICRO Newsletter, v. 13, n. 4, p. 17-22, 1982.
- [19] PARRY, Mark E.; SATO, Yoshinobu. Nintendo Company Limited: The Launch of Nintendo 64.
- [20] WATERMAN, Andrew Shell. Design of the RISC-V instruction set architecture. University of California, Berkeley, 2016.
- [21] DE SOUZA, Eduardo Michel Deves et al. RVSH-Um processador RISC-V para fins didáticos. Anais do Computer on the Beach, v. 14, p. 450-452, 2023.
- [22] AL BUSAIDI, Sayyid Samir; AHMAD, Afaq. Free Open Source Software Logisim-A Perfect Tool for Teaching and Learning of Digital Logic Circuit Design Course—Experience and Status. In: 4th FREE & OPEN SOURCE SOFTWARE CONFERENCE (FOSSC'2019-OMAN). 2019. p. 3.
- [23] MIQUELINI, RAFAEL AUGUSTO ALBUQUERQUE; FERRARI, Hélio Oliveira. LOGISIM: FERRAMENTA PARA SIMULAÇÃO DE CIRCUITOS COMBINACIONAIS e SEQUENCIAIS DIGITAIS. Intercursos Revista Científica, v. 20, n. 2, 2021.
- [24] BEGGS, Arthur de Matos. RISC-V SiMPLE: projeto e desenvolvimento de processadores RISC-V com a ISA RV32IMF usando as microarquiteturas uniciclo, multiciclo e pipeline em FPGA. 2021.
- [25] BRAIN, Marshall. How microprocessors work. Howstuffworks Inc, 2002.

- [26] VAN DEN ENDEN, Adrianus Wilhelmus Maria. Efficiency in multirate and complex digital signal processing. 2003.
- [27] WESTPHAL, Rafael. Modelagem de Hierarquias de Memoria. 2009.