Simulador Interativo para Programação na Arquitetura de Processadores RISC-V

Eduardo Michel Deves de Souza
Universidade do Vale do Itajaí, Brasil
eduardomichel@edu.univali.br

Larissa de Sousa Gouvea
Universidade do Vale do Itajaí, Brasil
larissagouvea@edu.univali.br

Cesar Albenes Zeferino
Universidade do Vale do Itajaí, Brasil
zeferino@univali.br

Douglas Rossi de Melo
Universidade do Vale do Itajaí, Brasil
drm@univali.br

Abstract

The RISC-V architecture, recognized for its modularity and efficiency, faces challenges due to the limited availability of educational resources. To address this issue, an interactive web simulator was developed based on the RV32I instruction set, using the Angular framework, with an interface integrated into a core comprising a code parser, assembler, and processor, enabling both continuous and step-by-step execution. The simulator's validation confirmed the proper functioning of essential features, including code editing, assembling, execution, and visualization of registers and memory. This simulator is intended for students, educators, and embedded systems designers in projects involving the RISC-V architecture, serving as a teaching tool in computer architecture courses.

Palavras-chave

Arquitetura de Computadores, Linguagem de Montagem, Simulação, RISC-V.

1 Introdução

Sistemas embarcados são dispositivos programáveis para funções específicas, amplamente utilizados no cotidiano. Seu principal componente, o processador, define as operações por meio de conjuntos de instruções conhecidos como ISA (Instruction Set Architecture), que servem de interface entre hardware e software [1]. Dentro das ISAs, as arquiteturas RISC (Reduced Instruction Set Computer) se destacam por sua simplicidade e eficiência, em contraste com arquiteturas mais complexas [2].

Entre as arquiteturas RISC, o RISC-V tem se consolidado como uma alternativa aberta e gratuita às tradicionais x86 e ARM, com crescente popularidade em sistemas embarcados e computação de alto desempenho [3]. Projetada para modularidade, a arquitetura RISC-V conta com uma ampla comunidade global de desenvolvedores e, recentemente, o Ministério da Ciência, Tecnologia e Inovação do Brasil tornou-se membro *premium*, fortalecendo sua participação no ecossistema de semicondutores [4]. Essa arquitetura aberta oferece uma base promissora para avanços tecnológicos em diversas áreas [5].

A arquitetura RISC-V, por ser mais nova em relação a outras, como x86 e ARM, ainda enfrenta limitações quanto à disponibilidade de recursos didáticos e ferramentas acessíveis [6]. Essa falta de materiais educativos e suporte prático representa um obstáculo significativo para o aprendizado e a adoção do RISC-V, restringindo seu uso tanto em contextos acadêmicos quanto industriais.

Diante do problema apresentado, este projeto propôs ampliar a disponibilidade de recursos para a arquitetura RISC-V por meio do desenvolvimento de um simulador web capaz de executar a linguagem de montagem RISC-V. Para isso, foi utilizado o conjunto base RV32I (RISC-V 32-bit Integer Base Integer Instruction Set), considerado estável e imutável porque suas instruções foram congeladas e não estão sujeitas a alterações futuras [7].

O artigo é dividido em seis seções: a Seção 1 apresentou o problema, os objetivos e a relevância da arquitetura RISC-V. A Seção 2 aborda conceitos teóricos sobre sistemas embarcados, processadores e simulação. A Seção 3 analisa trabalhos relacionados. A Seção 4 descreve o desenvolvimento do simulador. A Seção 5 apresenta os resultados. A Seção 6 conclui com objetivos alcançados, contribuições e perspectivas futuras.

2 Fundamentação Teórica

Entre os avanços tecnológicos no campo dos sistemas embarcados, destaca-se o uso de SoCs (System-on-a-Chip), que integram CPU (Central Processing Unit), memória e interfaces de entrada/saída em um único chip. Essa integração reduz custos, aumenta a eficiência e economiza espaço físico, tornando os SoCs ideais para dispositivos móveis e eletrônicos de consumo [8]. Um exemplo significativo são os SoCs Snapdragon, amplamente utilizados em *smartphones* e *tablets* [9].

2.1 Processadores

O processador é o componente central de computadores e sistemas embarcados, responsável pela execução de programas e pela interação com a memória e dispositivos periféricos. Sua eficiência impacta diretamente o desempenho do sistema, sendo essencial para a operação de qualquer tecnologia computacional [10]. A interface entre software e hardware é definida pelo conjunto de instruções, que especifica modos de endereçamento, tipos de dados e operações, como aritméticas, lógicas, de controle de fluxo e de carregamento/armazenamento. Essas instruções influenciam diretamente a programação e o desempenho dos sistemas [1].

As arquiteturas de processadores seguem dois paradigmas principais: RISC (Reduced Instruction Set Computer) e CISC (Complex Instruction Set Computer). Processadores RISC, como os da linha ARM, são projetados para executar instruções simples de maneira rápida e eficiente, o que os torna ideais para dispositivos móveis [11]. Já os processadores CISC, como a arquitetura x86 da Intel, oferecem instruções mais complexas que ampliam a versatilidade, sendo comuns em computadores pessoais e servidores.

2.2 RISC-V

O RISC-V é uma arquitetura aberta e gratuita, inicialmente projetada para pesquisa e educação, que ganhou ampla adoção industrial devido à sua flexibilidade e modularidade [5]. Sua base, o conjunto de instruções RV32I, oferece uma fundação estável e imutável, projetada para ser a referência fundamental da arquitetura, garantindo confiabilidade e compatibilidade com extensões futuras [7]. Operando com 32 bits, o RV32I define 32 registradores de uso geral e inclui instruções essenciais para operações aritméticas, lógicas, de controle de fluxo e acesso à memória.

A modularidade do RISC-V se destaca por permitir extensões específicas, que incorporam funcionalidades de multiplicação (M), ponto flutuante de precisão simples (F) e dupla precisão (D), além do conjunto base de instruções (I), como no RV32IMFD [5]. No conjunto de instruções base RV32I, há seis formatos principais de instrução (R, I, S, B, U e J), que simplificam a codificação e decodificação de operações, contribuindo para implementações mais eficientes [7].

As pseudo-instruções no RISC-V simplificam a escrita de código em linguagem de montagem, facilitando a programação sem aumentar a complexidade da arquitetura. Elas são variações de instruções reais, criadas para melhorar a legibilidade e produtividade dos desenvolvedores, enquanto o conjunto de instruções base permanece enxuto e eficiente [7].

2.3 Simulação

A simulação é uma técnica essencial para reproduzir operações de sistemas reais, permitindo validar o funcionamento, avaliar o desempenho e explorar diferentes configurações. Por meio de modelos matemáticos ou lógicos, ela possibilita a análise de cenários hipotéticos em ambientes controlados, reduzindo os riscos e custos de experimentos físicos [12]. Essa abordagem é especialmente útil em sistemas complexos, onde métodos analíticos não são suficientes, tornando necessário o uso de simuladores baseados em computadores para representar e estudar o comportamento desses sistemas [13].

No campo dos processadores, a simulação é crucial para desenvolver e validar arquiteturas. Ferramentas como MARS [14] e SPIM [15] são projetadas para simular a execução de código em processadores baseados na arquitetura MIPS, uma arquitetura RISC amplamente usada no ensino e na pesquisa. Esses simuladores permitem a escrita, montagem e execução de programas em linguagem de montagem, além de oferecer interfaces que mostram registradores, memória e o fluxo de execução, facilitando o aprendizado e a depuração de algoritmos.

3 Trabalhos Relacionados

Esta seção apresenta uma análise de simuladores existentes, destacando características e funcionalidades em relação aos objetivos deste trabalho.

O RARS (RISC-V Assembler, Simulator, and Runtime) é uma ferramenta destinada à montagem e simulação de programas em linguagem de montagem RISC-V, com foco em desenvolvedores iniciantes. Ele suporta as ISAs RV32IMFDN e RV64IMFDN, permite o uso de pseudo-instruções e facilita o gerenciamento de múltiplos arquivos por meio de abas. No entanto, sua utilização é limitada a ambientes *desktop*, pois não possui versões web ou móveis, restringindo seu acesso a sistemas operacionais específicos [16].

O Vulcan é um simulador RISC-V voltado para o ensino, com suporte parcial às extensões RV32I, RV32M, RV32A e RV32F. Ele oferece um editor de código integrado e exibe, de forma paralela, o contador de programa (PC), o código de máquina e as instruções originais, além de permitir a visualização de registradores inteiros, de ponto flutuante e da memória. Suas limitações incluem a dependência de tela cheia devido a restrições do Flutter Web e a ausência de simulação interativa e *feedback*, restringindo seu uso para aplicações mais avançadas [17].

O emulsiV é um simulador visual baseado na arquitetura RISC-V, utilizado na ESEO, com foco no ensino. Ele destaca-se por ilustrar o caminho de dados do processador de forma animada, exibindo etapas como *fetch*, *decode*, ULA (Unidade lógica e aritmética), mem/reg e PC, facilitando o entendimento dos conceitos básicos. No entanto, sua interface é confusa, exigindo tempo para adaptação, e sua documentação é limitada, fornecendo apenas informações parciais. Além disso, o simulador não suporta todas as pseudo-instruções, nem indica claramente quais são compatíveis.

O Simulador BRISC-V, desenvolvido pela Boston University, é uma ferramenta educacional web baseada em JavaScript que facilita o ensino da linguagem de montagem RISC-V [18]. Ele suporta o conjunto de instruções RV32I, permitindo compilar código C para montagem, executar instruções passo a passo e analisar o estado do processador. Embora acessível e interativo, não permite desfazer o último passo executado, exigindo reiniciar a simulação para corrigir erros. Além disso, só é possível carregar códigos de arquivos, sem opção de editar diretamente na plataforma.

O WebRISC-V é um simulador web avançado projetado para ensinar e visualizar o Caminho de Dados com Pipeline na linguagem de montagem RISC-V. Ele suporta as extensões RV64I e RV64M e apresenta o recurso "Pipeline Table", que rastreia a execução de aplicações em um pipeline de cinco estágios, oferecendo visibilidade detalhada dos registradores, memória e estados internos [19]. Apesar de sua funcionalidade avançada, o simulador requer consulta à documentação para uso eficaz, o que pode ser desafiador para iniciantes. Além disso, a separação entre as visualizações de registradores e memória dificulta a análise simultânea, e o suporte a pseudo-instruções é limitado.

O Ripes é um simulador visual baseado na ISA RISC-V, com foco em ensinar conceitos de microarquitetura. Ele oferece três abas principais: editor, processador e memória, permitindo visualizar o caminho de dados, o espaço de endereços e o programa de entrada lado a lado com seu correspondente executável. Seu diferencial está na integração de montador, compilador e simulador de cache, além de modelos de processadores que variam de ciclo único a pipelines de cinco estágios [20]. Embora completo e versátil, sua complexidade pode ser excessiva para o propósito deste trabalho, que busca um simulador mais simplificado e acessível.

Os simuladores analisados apresentaram limitações que comprometem sua usabilidade e alcance. O RARS requer instalação em *desktop*, o que pode dificultar seu uso em ambientes educacionais diversos. O Vulcan sofre com problemas de layout e limitações no suporte a registradores. O emulsiV possui uma interface confusa e carece de documentação adequada. Já o BRISC-V, embora funcional, dificulta a edição e execução direta de códigos, enquanto o

WebRISC-V apresenta uma interface fragmentada que torna a análise mais complexa. Essas limitações reforçam a relevância de uma ferramenta mais acessível e integrada, como o simulador proposto neste trabalho.

O simulador desenvolvido neste trabalho foi projetado para ser uma ferramenta acessível e prática, voltada ao ensino e aprendizado da arquitetura RISC-V. Sua implementação web, utilizando Angular e TypeScript, elimina a necessidade de servidores ou conexões com a internet, permitindo que o usuário explore a arquitetura RISC-V de forma local e independente. Suportando o conjunto de instruções RV32I, o simulador oferece modos de simulação passo a passo e execução contínua, além de proporcionar uma visualização integrada de registradores e memória por meio de uma interface intuitiva. Adicionalmente, ele permite a exportação de código escrito em linguagem de montagem para sua representação em linguagem de máquina.

4 Desenvolvimento

O simulador foi desenvolvido utilizando o *framework* Angular, cuja arquitetura baseada em componentes facilitou a criação de módulos independentes e reutilizáveis. Ele oferece uma interface gráfica intuitiva para edição, montagem, execução e exportação de código RISC-V em formatos como *.asm*, *.bin* e *.hex*. A ferramenta suporta execução passo a passo, com visualização integrada de memória e registradores, refletindo o estado da simulação em tempo real. Por ser compatível com navegadores baseados em Google Chrome e Mozilla Firefox, não requer instalação de software adicional.

O desenvolvimento do simulador foi organizado em dois grandes módulos: a interface gráfica e o núcleo do simulador. A interface foi projetada com base em protótipos criados no Figma, priorizando clareza e organização no design, enquanto o núcleo foi estruturado para garantir a análise, montagem e execução de código RISC-V, em conformidade com a ISA RV32I. O núcleo é composto pelo analisador de código, que organiza as instruções fornecidas; o montador, que converte a linguagem de montagem para linguagem de máquina; e o processador, que executa as instruções e atualiza os estados de registradores e memória.

4.1 Interface do Simulador

A interface do simulador foi desenvolvida para oferecer uma experiência prática e direta, permitindo que o usuário escreva, monte e execute código em linguagem de montagem RISC-V com facilidade. Ela é composta por elementos como o menu de opções, que centraliza as funcionalidades principais; o painel de edição e execução, onde o código pode ser inserido, montado e analisado; o painel de registradores, que exibe os estados atualizados dos registradores; e o console, responsável por apresentar mensagens do sistema e erros. Estruturada de forma modular com o *framework* Angular, cada componente opera de maneira independente, garantindo a interação direta com o núcleo do simulador e refletindo em tempo real as operações realizadas.

4.1.1 Menu de Opções. O menu de opções é o painel central do simulador, reúne as principais funcionalidades que permitem ao usuário controlar a simulação de maneira prática e eficiente. Por meio de botões intuitivos, o menu possibilita ações como abrir e salvar arquivos, exportar o código em diferentes formatos, montar o código, iniciar sua execução de forma completa ou passo a passo, reverter o último passo realizado e reiniciar o simulador. A Figura 1 exibe o protótipo do menu de opções, mostrando todos os botões disponíveis para o usuário interagir com o simulador.

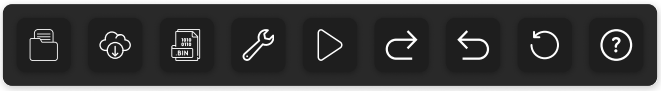

Figura 1: Protótipo do menu de opções.

4.1.2 Painel de Edição e Painel de Execução. O painel de edição permite ao usuário inserir ou modificar código em linguagem de montagem RISC-V. Ele foi projetado para oferecer uma interface intuitiva e funcional, centralizando as ferramentas necessárias para edição direta do código. Na Figura 2, é mostrado o protótipo do painel de edição.

Figura 2: Protótipo do painel de edição

O painel de execução apresenta o código montado e o estado atual da memória do processador. Ele é subdividido em duas áreas principais: o segmento de texto, que exibe as instruções convertidas para linguagem de máquina, e o segmento de dados, que organiza e apresenta a memória. Funcionalidades adicionais incluem a navegação por páginas, com cada página contendo 128 endereços de memória, e a opção de alternar entre os formatos hexadecimal e decimal. A Figura 3 apresenta o protótipo do painel de execução.

4.1.3 Painel de Registradores. O painel de registradores apresenta os 32 registradores do processador RISC-V e o contador de programa, permitindo ao usuário acompanhar o estado atualizado de cada um durante a execução do código. Sempre que um registrador é alterado, o painel destaca a mudança com uma cor de fundo na linha correspondente e realiza a rolagem automática até o registrador modificado, facilitando a identificação das atualizações. A Figura 4 mostra o protótipo do painel de registradores.

4.1.4 Console. O console do simulador é responsável por exibir as saídas do sistema, incluindo mensagens de erro e outros *feedbacks* importantes durante o uso do simulador. Além de ser uma interface

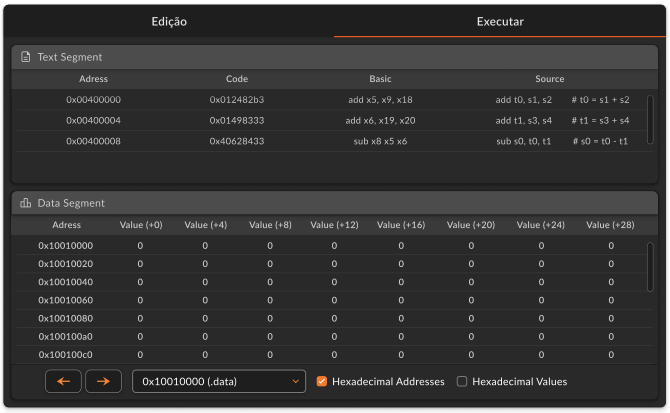

Figura 3: Protótipo do painel de execução

Nome	Número	Valor
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000

Figura 4: Protótipo do painel de registradores

de comunicação, onde o usuário pode acompanhar resultados de operações como montagem de código, geração de binário, e importação e exportação de arquivos, o console também permite a entrada de dados pelo usuário. A Figura 5 exibe o protótipo do console.

Figura 5: Protótipo do console

4.1.5 Interface Completa do Simulador RISC-V. A Figura 6 apresenta o protótipo da interface do simulador com o painel de edição em evidência. Esse painel de edição é o ponto inicial de interação do usuário, permitindo a escrita ou carregamento de código em linguagem de montagem RISC-V. Ele integra o menu de opções, que possibilita salvar e abrir arquivos, montar o código e iniciar a execução, juntamente com o console e o painel de registradores, elementos que complementam o ambiente de trabalho do simulador.

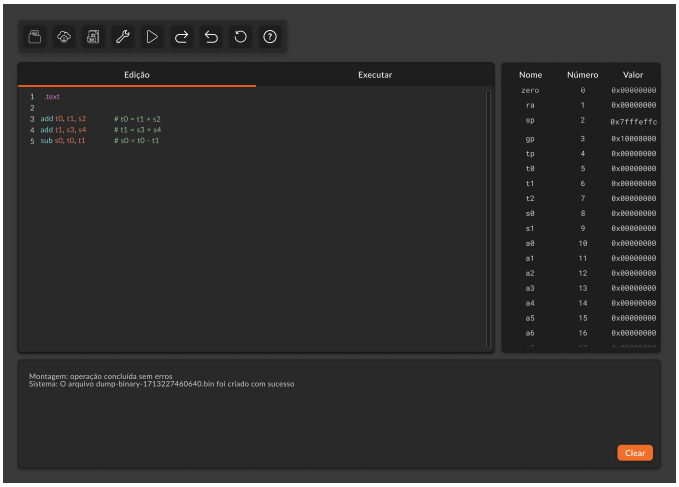

Figura 6: Protótipo de interface do simulador exibindo o painel de edição

A Figura 7 exibe o protótipo da interface com o painel de execução em evidência. O painel de execução não apenas apresenta o código montado, mas também as seções de visualização de memória e registradores, permitindo ao usuário acompanhar as mudanças no estado do simulador em tempo real.

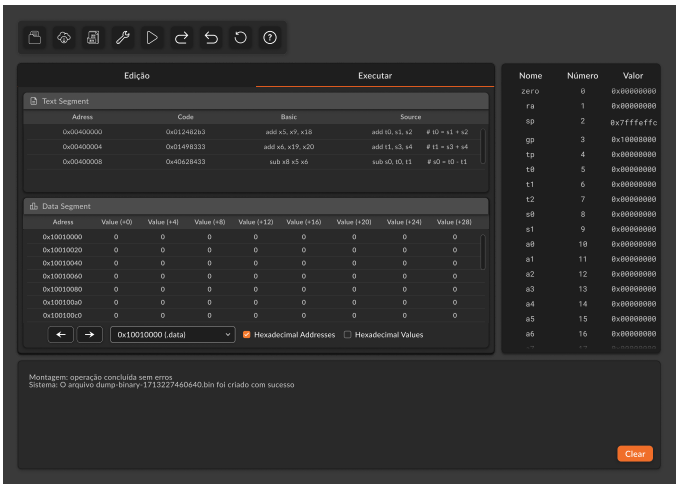

Figura 7: Protótipo de interface do simulador exibindo o painel de execução

4.2 Núcleo do Simulador

O núcleo do simulador é formado pelo Analisador de Código, Montador e Processador, que atuam de maneira integrada para processar

e executar o código inserido pelo usuário. O Analisador de Código interpreta o código e o prepara para a montagem; o Montador converte as instruções em linguagem de máquina; e o Processador executa o código, atualizando os registradores e a memória. O resultado final desse fluxo é armazenado no objeto final do programa, que reúne o código binário, dados do programa e o estado atualizado da simulação. A Figura 8 ilustra a modelagem do núcleo do simulador, destacando como esses componentes interagem entre si.

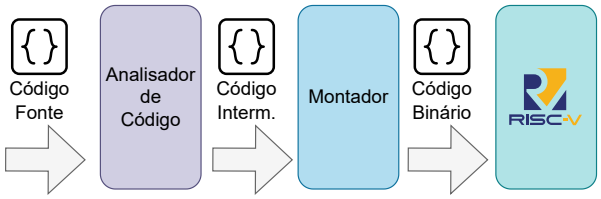

Figura 8: Modelagem do núcleo do simulador

A seguir, cada um desses componentes será detalhado, com a apresentação de sua modelagem, incluindo diagramas de estados e caso de uso. Para o objeto final do programa, será apresentado o diagrama de classes.

4.2.1 Analisador de Código. O Analisador de Código interpreta o código escrito pelo usuário e o converte em um objeto intermediário, que será utilizado nas etapas de montagem e execução. Esse processo começa com a separação do código em linhas e elementos individuais, permitindo uma análise detalhada de cada instrução. Ele lida com os segmentos de texto e de dados, identifica pseudo-instruções e as transforma em instruções compatíveis com o processador RISC-V RV32I. Além disso, o Analisador cria uma tabela de símbolos, registrando rótulos, variáveis e endereços de memória, essenciais para o funcionamento correto do programa. A Figura 9 apresenta a modelagem do Analisador de Código.

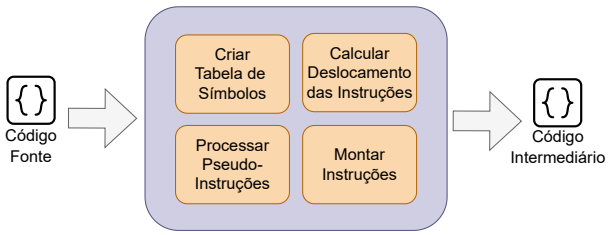

Figura 9: Modelagem do analisador de código

Após calcular os deslocamentos necessários, o Analisador organiza todas as informações no objeto intermediário, que contém as instruções, os dados do programa e o estado inicial do processador, incluindo registradores e memória. Esse objeto intermediário é a base para o objeto final do programa, integrando todas as informações necessárias para as etapas seguintes de montagem e execução no núcleo do simulador.

4.2.2 Montador. O Montador converte o objeto intermediário, gerado pelo Analisador de Código, em linguagem de máquina. Cada instrução é analisada para identificar seu *opcode*, responsável por determinar a operação a ser realizada, como somas, comparações ou manipulação de memória. A partir do *opcode*, o Montador classifica a instrução em um dos formatos da ISA RISC-V (R, I, S, B, U ou J) e organiza as informações necessárias, como os registradores de origem (`rs1`, `rs2`), o registrador de destino (`rd`) e os valores imediatos. Essas informações são então convertidas em um código binário de 32 bits. A Figura 10 apresenta a modelagem completa do Montador.

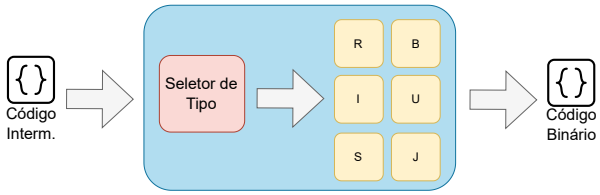

Figura 10: Modelagem do montador

Cada formato possui uma estrutura específica. Por exemplo, no formato R, são utilizados dois registradores de origem e um registrador de destino, enquanto no formato I, há um registrador de origem, um registrador de destino e um valor imediato. O Montador organiza essas informações conforme as características de cada formato e as transforma em código binário. Após a conversão, o código binário é armazenado no objeto final do programa, que passa a conter o código de máquina completo, os dados e os rótulos do programa.

4.2.3 Processador. O processador executa o código em linguagem de máquina utilizando o objeto final do programa. Ele busca as instruções na memória por meio do contador de programa, decodifica o *opcode* para determinar a operação (como soma, subtração ou manipulação de memória) e, em seguida, processa os valores nos registradores indicados (`rs1`, `rs2`) e os valores imediatos na ULA. Após cada operação, o estado dos registradores e da memória é atualizado. A Figura 11 ilustra a modelagem do processador.

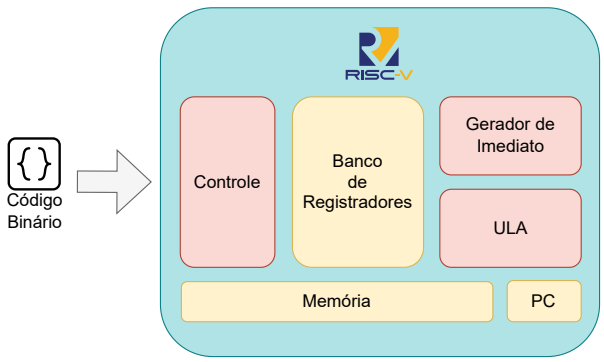

Figura 11: Modelagem do processador

O fluxo de execução inclui desvios e saltos condicionais com base nos resultados das operações, atualizando o contador de programa para manter o controle correto do programa. Durante a execução, o objeto final do programa é continuamente atualizado com o estado mais recente dos registradores e da memória. Essas mudanças são refletidas em tempo real na interface gráfica, permitindo ao usuário acompanhar o estado atual do simulador e o fluxo de controle do programa.

4.2.4 Objeto Final do Programa. O objeto final do programa centraliza todas as informações essenciais para a simulação, como o código binário gerado pelo montador, os dados do programa, a tabela de símbolos, a memória, os registradores e o contador de programa. Ele também armazena o índice do último registrador modificado, permitindo que a interface gráfica reflita o estado atualizado do simulador. Qualquer alteração no código, montagem ou execução atualiza esse objeto, integrando o núcleo do simulador com a interface. A Figura 12 apresenta o diagrama de classes do objeto final do programa.

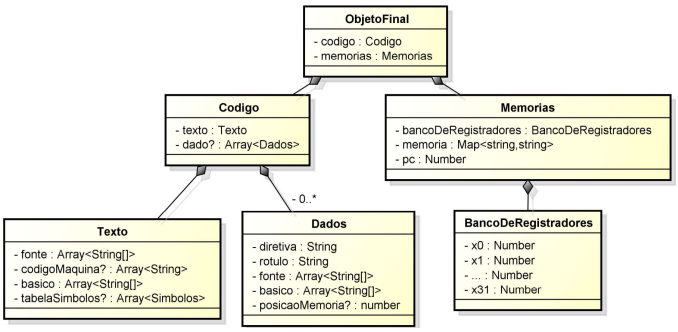

Figura 12: Diagrama de classes do objeto final do programa

A integração do simulador foi realizada por meio de serviços que conectam a interface ao núcleo. Esses serviços acionam o núcleo ao escrever, montar e executar o código, retornando o objeto final do programa, que atualiza em tempo real o estado do processador, registradores e memória na interface.

4.3 Recursos do Simulador

O simulador inclui funcionalidades projetadas para facilitar o aprendizado de RISC-V, como suporte a diretivas, destaque visual de instruções e registradores, exibição de erros e flexibilidade na referência a registradores. Esses recursos tornam o uso mais intuitivo e ajudam no acompanhamento da execução do código.

4.3.1 Suporte a Diretivas. O suporte às diretivas *.ascii*, *.string* e *.word* permite organizar dados no segmento de memória. A diretiva *.ascii* armazena sequências de caracteres, como mostrado no Quadro 1, enquanto *.string* funciona de forma idêntica, sendo útil para declarar strings no código.

Já a diretiva *.word*, ilustrada no Quadro 2, é utilizada para armazenar valores numéricos de 32 bits. Ela permite a declaração de múltiplos valores em uma mesma linha ou sua continuidade em linhas subsequentes, facilitando a organização de listas ou tabelas no código.

```
.data
string1: .ascii  "teste"
string2: .string "exemplo"
```

Quadro 1: Exemplo de uso das diretivas *.ascii* e *.string*

```
.data
numeros1: .word 10, 20
numeros2: .word 1, 2, 3
                4, 5, 6
```

Quadro 2: Exemplo de uso da diretiva *.word*

4.3.2 Realce de Instruções e Registradores. Para facilitar o acompanhamento da execução do código, o simulador altera a cor de fundo da instrução em processamento no painel de execução, destacando-a visualmente para o usuário. Esse recurso permite identificar claramente qual linha de código está sendo executada em tempo real. Além disso, após cada operação, os registradores que sofrem alterações recebem uma mudança na cor de fundo da linha correspondente no painel de registradores, indicando quais valores foram modificados. Essas alterações visuais proporcionam maior clareza ao fluxo de execução e tornam o processo de aprendizado e depuração do código mais eficiente e intuitivo.

4.3.3 Identificação e Exibição de Erros. Durante a montagem do código, o simulador realiza verificações detalhadas para identificar inconsistências, como símbolos ausentes, rótulos duplicados ou diretivas mal posicionadas. Ao encontrar um erro, o processo é interrompido, e uma mensagem explicativa é exibida no console, facilitando a identificação do problema e permitindo ao usuário corrigir rapidamente o código.

4.3.4 Referência a Registradores. O simulador permite referenciar os registradores por números ou nomes simbólicos, proporcionando flexibilidade na escrita e leitura do código. No Quadro 3, os registradores são referenciados diretamente por seus números, o que é útil para usuários familiarizados com essa abordagem.

```
.text
add x5, x0, x1
sub x10, x2, x3
```

Quadro 3: Exemplo de código utilizando os números dos registradores

Por outro lado, o uso de nomes simbólicos, como *t0* e *a0*, demonstrado no Quadro 4, torna o código mais legível, especialmente para iniciantes ou durante o ensino da arquitetura RISC-V.

```
.text
add t0, zero, ra
sub a0, sp, gp
```

Quadro 4: Exemplo de código utilizando os nomes simbólicos dos registradores

5 Resultados

Os resultados do simulador RISC-V foram avaliados por meio de testes sistemáticos, garantindo que os requisitos estabelecidos fossem atendidos. Além disso, verificou-se o correto funcionamento de funcionalidades como visualização de memória e registradores, e realizou-se uma análise comparativa com outros simuladores, incluindo testes com códigos de referência e avaliação de desempenho em trabalhos similares.

5.1 Avaliação e Validação

A validação do simulador foi realizada utilizando códigos em linguagem RISC-V retirados do livro Computer Organization and Design RISC-V Edition: The Hardware Software Interface [2]. Esses códigos foram usados para verificar o cumprimento dos requisitos estabelecidos, incluindo a edição, montagem e execução do código. Durante os testes, o sistema demonstrou a capacidade de atualizar corretamente os registradores e a memória, enquanto o console apresentou mensagens claras relacionadas ao estado do simulador e possíveis erros. A Figura 13 mostra o simulador em execução no navegador Google Chrome.

Para a análise comparativa, foi utilizado um código de referência extraído da Seção 2.8 de [2], apresentado no Quadro 5. Esse código realiza operações aritméticas simples, como soma e subtração entre registradores, e manipula a pilha para salvar e restaurar estados durante a execução de um procedimento. Com um tamanho de 728 bytes, o código foi capaz de rodar em diferentes simuladores mencionados nos trabalhos relacionados, permitindo validar o funcionamento correto do núcleo e comparar desempenho e funcionalidades, garantindo a robustez do simulador desenvolvido.

5.2 Análise Comparativa

Para a análise comparativa, utilizou-se o código apresentado no Quadro 5, que realiza operações aritméticas e manipulação de pilha. A comparação avaliou o uso de memória RAM durante a execução, com testes realizados no navegador Google Chrome para simuladores web e ferramentas de monitoramento de memória para simuladores *desktop*.

A Tabela 1 apresenta os resultados da comparação entre diferentes simuladores RISC-V. Os resultados destacam o simulador desenvolvido neste trabalho como o mais eficiente no uso de memória RAM, consumindo cerca de 26 MB, enquanto simuladores como RIPES e WebRISC-V apresentaram os maiores consumos, com 647 MB e valores similares, respectivamente. Outros simuladores, como o Vulcan e o BRISC-V, exibiram resultados intermediários, com o Vulcan se destacando em eficiência após o simulador proposto. Essa eficiência é atribuída ao foco exclusivo no conjunto de instruções RV32I e à ausência de funcionalidades extras que demandam mais memória, como visualizações avançadas do caminho de dados.

O design do simulador priorizou simplicidade e eficiência, resultando em uma compilação final compacta de 172 KB e a capacidade de operar localmente, sem servidores ou conexão com a internet. Essa abordagem torna o simulador ideal para sistemas com recursos limitados, proporcionando uma alternativa leve e funcional em relação a outros simuladores que possuem interfaces mais complexas e maior consumo de memória.

```
    .text
    j main
leaf_example:
    addi sp, sp, -12     # ajusta a pilha
    sw x5, 8(sp)         # salva x5
    sw x6, 4(sp)         # salva x6
    sw x20, 0(sp)        # salva x20

    add x5, x10, x11     # x5 = g + h
    add x6, x12, x13     # x6 = i + j
    sub x20, x5, x6      # f = (g + h) - (i + j)
    addi x10, x20, 0     # x10 = f

    lw x20, 0(sp)        # restaura x20
    lw x6, 4(sp)         # restaura x6
    lw x5, 8(sp)         # restaura x5
    addi sp, sp, 12      # ajusta a pilha
    jalr x0, 0(x1)       # retorna

main:
    addi x5, zero, 1     # x5 = 1
    addi x6, zero, 2     # x6 = 2
    addi x20, zero, 3    # x20 = 3

    addi x10, zero, 4    # g = 4
    addi x11, zero, 3    # h = 3
    addi x12, zero, 2    # i = 2
    addi x13, zero, 1    # j = 1

    jal leaf_example     # chama funcao

    nop                  # no-op
```

Quadro 5: Código RISC-V para operações aritméticas e manipulação de pilha

Tabela 1: Comparação de simuladores quanto à memória utilizada

Simulador	Memória Utilizada (MB)
RARS [16]	289
Vulcan [17]	173
emulsiV [21]	372
BRISC-V [18]	339
WebRISC-V [19]	542
RIPES [20]	647
Este Trabalho	26

6 Conclusão

A arquitetura de processadores tem avançado significativamente, enfrentando desafios como eficiência energética, desempenho e acessibilidade de aprendizado. Arquiteturas de instrução reduzida, como o RISC-V, destacam-se pela modularidade que favorece inovações em sistemas embarcados e processadores. No entanto, a ampla adoção do RISC-V é limitada pela falta de recursos educacionais e ferramentas interativas que apoiem o ensino e a prática.

Para abordar essa lacuna, foi desenvolvido um simulador web interativo focado no conjunto de instruções RV32I da arquitetura RISC-V. Projetado para ser acessível e intuitivo, o simulador facilita o aprendizado da linguagem de montagem RISC-V, permitindo que seja utilizado diretamente no navegador sem necessidade de instalações adicionais, aumentando sua acessibilidade para estudantes e desenvolvedores.

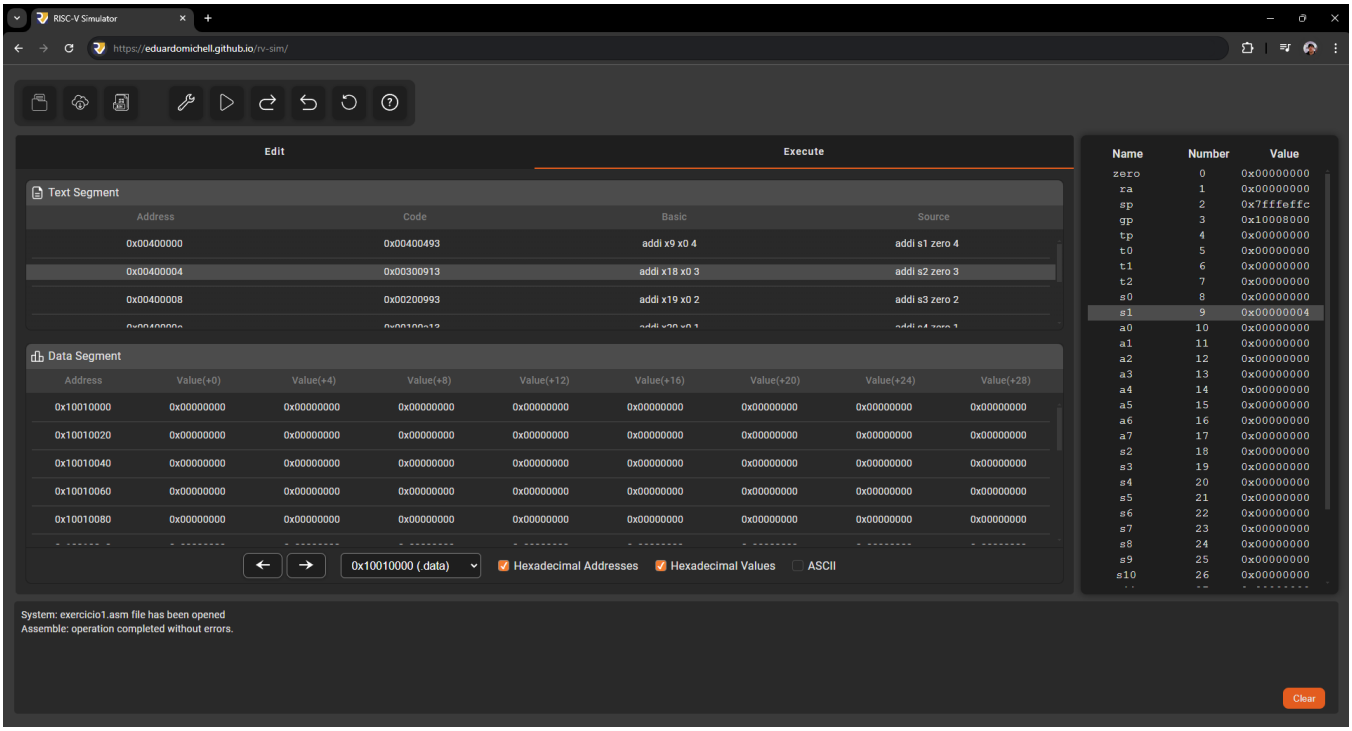

Figura 13: Simulador em execução no navegador Google Chrome

Os resultados obtidos demonstram que o simulador atende aos objetivos propostos, executando as instruções do conjunto RV32I de maneira funcional e compatível com diferentes navegadores e sistemas operacionais. Para trabalhos futuros, será realizada a avaliação da experiência dos usuários e a análise da usabilidade do simulador. Para isso, pretende-se aplicar questionários a estudantes da área, possibilitando a obtenção de dados sobre a facilidade de uso, compreensão das funcionalidades e impacto na aprendizagem da arquitetura RISC-V.

7 Agradecimentos

Este trabalho foi financiado, em parte, pela Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina – FAPESC (contratos 2023TR000880 e 2024TR001897) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (processos 313513/2021-0, 350208/2022-0, 408641/2023-1 e 350794/2023-5).

Referências

[1] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier Science, 2017.

[2] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2020.

[3] Dan Robinson. Could risc-v become a force in hpc? we talk to the experts. https://www.theregister.com/2023/02/08/riscv_hpc/, 2023. Acessado em: 26/08/2023.

[4] Ministério da Ciência, Tecnologia e Inovações. Brasil se torna membro da aliança risc-v international, 2024. URL https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/noticias/2024/03/brasil-se-torna-membro-da-alianca-risc-v-international. Acessado em: 01 jul. 2024.

[5] SiFive Inc. Andrew Waterman, Krste Asanović. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*, volume 2. riscv.org, 2019. URL https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf.

[6] Ed Sperling. Risc-v: What's missing and who's competing. https://semiengineering.com/risc-v-whats-missing-and-whos-competing/, 2020. Acessado em: 22/08/2023.

[7] D.A. Patterson and A. Waterman. *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon LLC, 2017.

[8] D. Greaves. *Modern System-on-Chip Design on Arm*. ARM Education Media, 2021.

[9] Qualcomm. Snapdragon Overview. https://www.qualcomm.com/snapdragon/overview, 2023. Acessado em: 08/04/2024.

[10] J. Nurmi. *Processor Design: System-On-Chip Computing for ASICs and FPGAs*. Springer Netherlands, 2010.

[11] M. Wolf. *Computers as Components: Principles of Embedded Computing System Design*. The Morgan Kaufmann series in computer architecture and design. Morgan Kaufmann, 2023.

[12] A.M. Law. *Simulation Modeling and Analysis, Sixth Edition*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill Education, 2024.

[13] J. Banks. *Discrete-event System Simulation*. Pearson, 2014.

[14] Kenneth Vollmar and Pete Sanderson. Mars: an education-oriented mips assembly language simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 239–243, 2006.

[15] James R Larus. Spim s20: A mips r2000 simulator. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

[16] Benjamin Landers. Rars. https://github.com/TheThirdOne/rars, 2017. Acessado em: 05/04/2024.

[17] Victor Miguel de Morais Costa. Vulcan. https://github.com/vmmc2/Vulcan, 2020. Acessado em: 07/04/2024.

[18] Rashmi Agrawal, Sahan Bandara, Alan Ehret, Mihailo Isakov, Miguel Mark, and Michel A. Kinsy. The brisc-v platform: A practical teaching approach for computer architecture. In *Proceedings of the Workshop on Computer Architecture Education*, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Roberto Giorgi and Gianfranco Mariotti. Webrisc-v: a web-based education-oriented risc-v pipeline simulation environment. In *Proceedings of the Workshop on Computer Architecture Education*, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Morten B. Petersen. Ripes: A visual computer architecture simulator. In *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*, pages 1–8, 2021.

[21] Guillaume Savaton. emulsiv. https://eseo-tech.github.io/emulsiV/, 2020. Acessado em: 07/04/2024.