UM SIMULADOR WEB DE ASSEMBLY PARA PROPÓSITOS EDUCACIONAIS

A ASSEMBLY WEB SIMULATOR FOR EDUCATIONAL PURPOSES

Antonio Marlilson Moreira Amancio*
Ricardo Lenz Cesar**

RESUMO

O ensino da linguagem Assembly, essencial para a compreensão do funcionamento interno de computadores, pode ser desafiador devido à sua complexidade e proximidade ao hardware. Para tanto, este trabalho apresenta o desenvolvimento de um simulador web interativo para Assembly, projetado com foco em acessibilidade, usabilidade e aplicação prática no contexto educacional. A ferramenta permite que estudantes escrevam, testem e depurem códigos Assembly em tempo real, utilizando uma interface intuitiva com documentação inclusa. Além disso, o simulador incorpora funcionalidades modernas, como visualização gráfica, controle de execução e responsividade para dispositivos variados. Testado com exemplos práticos, o simulador demonstra potencial para tornar o aprendizado mais acessível e motivador, contribuindo para o ensino de linguagens de baixo nível e áreas correlatas.

Palavras-chave: Assembly. Simuladores Educacionais. Web. Arquitetura de Computadores. Ensino de Programação.

ABSTRACT

The teaching of Assembly language, essential for understanding the internal workings of computers, can be challenging due to its complexity and close relationship with hardware. To address this, this work presents the development of an interactive web-based Assembly simulator, designed with a focus on accessibility, usability, and practical application in an educational context. The tool enables students to write, test, and debug Assembly code in real time, using an intuitive interface with built-in documentation. Additionally, the simulator incorporates modern features such as graphical visualization, execution control, and responsiveness for various devices. Tested with practical examples, the simulator demonstrates the potential to make learning more accessible and engaging, contributing to the teaching of low-level languages and related fields.

* Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: antoniomarlilson@gmail.com
** Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: ricardo.lenz@ifce.edu.br

Keywords: Assembly. Educational Simulators. Web. Computer Architecture. Programming Education.

1 INTRODUÇÃO

Nos cursos de graduação na área de computação, muitos alunos enfrentam dificuldades ao lidar com os desafios da programação, uma vez que tal atividade pode transparecer como um campo por demais abstrato e conceitual a vários iniciantes e que exige um nível alto de prática e raciocínio lógico. De acordo com Watson e Li (2014), a literatura mostra que as dificuldades influenciam nos elevados índices de reprovações e desistências em disciplinas iniciais de linguagem de programação.

Linguagens de programação de alto nível permitem abstrações maiores e se aproximam mais das necessidades imediatas do desenvolvedor. Em contrapartida, distanciam-se mais da máquina e suas peculiaridades. Diversos cursos de computação, entretanto, têm o interesse em aproximar os alunos de um entendimento maior à respeito da máquina. Nesse sentido, o aprendizado de uma linguagem de programação mais próxima do hardware é fundamental para dar substância na formação integral do aluno. Em particular, a linguagem de programação Assembly (também referenciada em português como linguagem de montagem) é um fundamento importante na aproximação com a máquina, embora também possa ser desafiante.

A linguagem Assembly é uma representação compreensível da linguagem de máquina, sendo utilizada frequentemente em situações onde se trabalha com certos microcontroladores e outros dispositivos computacionais. Também é usada, mesmo em sistemas tradicionais de computadores de mesa, em contextos ligados a desenvolvimento de *drivers* e módulos de sistemas operacionais. Ademais, seu uso também ocorre em implementações que visam uma otimização maior do processador, como na escrita de rotinas numéricas especializadas ou demais tarefas específicas que algum processador possa oferecer além do usual representado em linguagens de alto nível. Embora decifrável para humanos, a linguagem Assembly segue mais propriamente a perspectiva de projeto do hardware e não de abstrações que facilitem o entendimento de códigos maiores para o programador, sendo muitas vezes considerada de difícil entendimento, possuindo características diferentes em comparação com outras linguagens de nível mais alto (CAMARGO, 2013).

Segundo Santos (2008), Assembly é uma linguagem que permite ter um controle direto sobre o hardware de um computador, resultando em economia de código e, consequentemente, de memória. Além disso, um código enxuto, em Assembly, é frequentemente mais rápido do que códigos gerados automaticamente por compiladores de linguagens de alto nível.

O ensino de disciplinas ligadas à parte física do computador envolve algumas dificuldades como a limitação de recursos para que o estudante faça a realização de testes e a complexidade em tratar com dispositivos minúsculos (SILVA; MENESES, 2009). Dessa forma, ferramentas de programação podem ser utilizadas para simplificar o aprendizado, entusiasmar ou motivar os estudantes. Conforme observado por Silva e Meneses (2009), é uma prática comum empregar

ambientes computacionais que permitem a simulação de operações de hardware, tornando o processo de aprendizado mais acessível e compreensível para os estudantes.

A simulação de um contexto computacional possibilita o estudo de sistemas complexos por meio de análises de modelos realistas, com o propósito de compreender as informações sobre o seu funcionamento e testar suposições na busca de resoluções para problemas (CELESTINO; VALENTE, 2021). Ela pode assumir formas física ou digital/virtual, de modo que os simuladores e os softwares de simulação computacional podem ser compreendidos como instrumentos capazes de reproduzir digitalmente uma atividade, com o intuito de capacitar uma pessoa para eventos reais (CLAYTON; GIZELIS, 2005).

Dessa maneira, o presente trabalho propõe o desenvolvimento de um simulador de Assembly com o objetivo principal de facilitar o aprendizado dessa linguagem. Propõe-se um simulador web interativo com vários recursos, incluindo editor de código, capacidade de executar (simular) o código, recursos de *debug*, capacidade de saída gráfica (tela para visualização) e suporte a diferentes tipos de instruções de arquiteturas, desde manipulações aritméticas com registradores a operações de pilha, memória, etc. Por fim, oferecendo ainda possibilidades para o aluno experimentar trabalhar com paralelismo através de instruções do tipo SIMD, à semelhança de instruções MMX, SSE e AVX. Essas características permitem aos usuários da ferramenta explorar e experimentar diferentes estilos de programação de forma interativa e facilitada para fins educacionais, cobrindo várias possibilidades e aspectos conforme as necessidades acadêmicas.

Os objetivos específicos deste projeto incluem a criação de uma interface de usuário intuitiva que simplifique a experiência do usuário e a implementação de uma simulação interativa que permita ao estudante escrever, testar e depurar código Assembly em tempo real. Ademais, documentação e exemplos práticos também estão incorporados no projeto para ajudar no processo de aprendizado do estudante. O intuito do simulador ser fornecido mediante interface web é para que seja acessível de forma mais universal, permitindo que estudantes o utilizem em diferentes sistemas e máquinas com facilidade.

O presente trabalho é dividido nas seguintes seções: a Seção 2 compreende a Fundamentação Teórica, que elucida os conceitos trabalhados na construção do simulador; a Seção 3 expõe estudos relacionados à matéria; a Seção 4 explica a metodologia para desenvolvimento da aplicação; a Seção 5 apresenta os resultados obtidos no projeto e descreve o funcionamento do simulador, além de avaliar a implementação realizada; por fim, a Seção 6 aborda as conclusões do trabalho e possibilidades futuras.

2 REFERENCIAL TEÓRICO

2.1 Linguagem de Montagem (Assembly)

Assembly é uma linguagem de baixo nível que se aproxima diretamente da linguagem de máquina, permitindo que os programadores tenham controle mais preciso sobre o hardware. Enquanto as linguagens de alto nível são mais abstratas, Assembly proporciona um nível de

controle sobre o processador e os recursos do sistema, essencial para otimizar o desempenho e acessar características específicas do hardware (GASNAS; GLOBA, 2024; HALL; SLONKA, 2020).

Em Assembly, os programas utilizam mnemônicos, que são representações simbólicas das instruções do processador. Cada mnemônico corresponde a uma instrução de máquina. Assim, para trabalhar em Assembly é necessário conhecer sobre as instruções da máquina sendo programada, já que as instruções variam conforme o tipo de processador (GASNAş, 2022). As instruções de um processador x86, por exemplo, são diferentes das de um processador ARM. Essa variação destaca a importância do conhecimento da arquitetura ao programar em Assembly, já que o desenvolvimento e a otimização de código dependem diretamente das características do processador utilizado (HALL; SLONKA, 2020).

O código Assembly é comumente dividido em seções, como a seção de dados e a seção de texto. A seção de dados armazena variáveis e constantes, enquanto a seção de texto contém as instruções que serão executadas. As instruções podem envolver operações aritméticas, manipulação de registradores e controle do fluxo de execução, tanto de forma condicional quanto incondicional (GASNAş, 2022).

A linguagem Assembly é fundamental no desenvolvimento de sistemas de baixo nível, como sistemas operacionais e drivers de dispositivos. Muitos sistemas operacionais utilizam componentes escritos em Assembly para garantir a comunicação eficiente com o hardware subjacente, aproveitando ao máximo os recursos do processador e do sistema (GASNAS; GLOBA, 2024; HALL; SLONKA, 2020).

2.2 Tecnologias Web

As tecnologias de base empregadas em sistemas web são HTML, CSS e JavaScript, que são responsáveis pela estrutura, estilo e interatividade dos sites, respectivamente. HTML (HyperText Markup Language) é a linguagem de marcação que define a estrutura e o conteúdo dos *sites* através de *tags* (elementos) e seus atributos. CSS (Cascading Style Sheets) é a linguagem de estilo que define a aparência e o layout dos sites. Ela usa regras para aplicar estilos aos elementos HTML, como cores, fontes, bordas, espaçamentos, alinhamentos e outros. Cada regra tem ainda um seletor que indica quais elementos serão afetados. Por fim, JavaScript é a linguagem de programação de base nos browsers, permitindo adicionar interatividade, dinamismo e lógica aos sites. A linguagem permite manipular elementos HTML, propriedades CSS, responder a eventos do usuário, comunicar-se com servidores, armazenar dados, entre outras funções. Em cima dessas tecnologias de base há um ecossistema bastante diverso de frameworks, bibliotecas, pré-processadores, transpiladores e outros, implementando assim também diversos componentes reutilizáveis.

O uso de tecnologias web num sistema pode ser interessante pelo fato de serem tecnologias bem padronizadas, disponíveis em muitos meios e plataformas, facilitando o acesso ao programa.

2.3 Instruções SIMD

SIMD (Single Instruction, Multiple Data) é uma das categorias propostas pela taxonomia de Flynn para indicar a capacidade de com uma única instrução realizar o processamento de vários dados em paralelo. Vários processadores possuem pacotes de instruções SIMD, como por exemplo as instruções MMX, SSE e AVX de processadores da arquitetura x86.

A importância de instruções que seguem esse paradigma SIMD é de permitir um aumento significativo de desempenho para diversas aplicações, em especial aplicações de processamento de imagem, jogos, multimídia, simulações, etc. De fato, vários compiladores oferecem suporte a SIMD. Nesse sentido, o aluno que vai estudar Arquitetura de Computadores e se depara com esse tipo de mecanismo pode experimentar em primeira mão o trabalho com instruções SIMD se as mesmas forem oferecidas pela plataforma, enriquecendo assim a sua experiência.

Em contraste com a abordagem SIMD, o mecanismo tradicional de execução segue o modelo SISD (Single Instruction, Single Data) na classificação de Flynn, onde cada instrução vai trabalhando com um dado, uma após a outra, de forma sequencial (serial).

Como exemplos de instruções SIMD, pode-se citar a instrução PADDB do MMX[1], (adição de 8 bytes em paralelo), ou ADDPS (adição de 4 floats em paralelo) do SSE, ou ainda VADDPS (adição de 8 floats em paralelo) do AVX. As variações de instruções para esses diferentes pacotes SIMD podem se dar em função de vários fatores; os registradores suportados no padrão MMX trabalham com 64 bits, permitindo empacotar 8 bytes num mesmo registrador; no SSE, trabalham com 128 bits; no AVX, 256 bits (e, no AVX 2, 512 bits). Um registrador com 128 bits no SSE, por exemplo, pode empacotar 16 bytes ou 4 números do tipo int (de 4 bytes cada). Isso permite, por exemplo, somar 4 números int em paralelo.

Outra variação nas instruções SIMD ainda refere-se à possibilidade de trabalhar com aritmética saturada ou não. Numa aritmética sem saturação, a soma 255 + 1 em bytes do tipo unsigned resulta no valor 0; 255 + 2 seria o valor 1; etc. Mas, com saturação, a soma 255 + 1 resulta em 255; 255 + 2 daria 255 também; etc. Isso pode ser importante em vários casos, por exemplo em processamento de imagens onde os bytes podem representar canais R, G, B e a aritmética com saturação permite que a imagem seja corretamente manipulada (um canal R com valor 255, se somado com 1, permanece 255 com a saturação; sem a saturação, iria para 0, o que significaria a ausência do vermelho, gerando um efeito frequentemente indesejado pelo desenvolvedor).

2.4 Editor de Código em tecnologia Web

Aplicações que trabalham com edição de código necessitam naturalmente de algum editor de texto apropriado. Editores que ofereçam efeito de sintaxe colorida e outras funcionalidades podem ajudar bastante no trabalho de desenvolvedores. Os editores de código modernos disponíveis em bibliotecas JavaScript frequentemente incluem recursos como autocompletar, verificação

[1] Essas e outras instruções SIMD são detalhadas na documentação dos manuais oferecidos pela Intel em <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.

de erros em tempo real, navegação simplificada entre arquivos e suporte a extensões, permitindo que os desenvolvedores personalizem seu ambiente de trabalho de acordo com suas necessidades. Abaixo, são apresentados alguns exemplos de editores de código amplamente utilizados em implementações de ferramentas web:

- **Ace Editor:** O Ace Editor[2] é um componente JavaScript que fornece um editor de texto leve e altamente personalizável. Ele oferece efeito de sintaxe colorida para várias linguagens, e permite a configuração de diferentes estilos para comandos, strings, números e comentários.
- **CodeMirror:** CodeMirror[3] é uma biblioteca JavaScript que oferece um editor de texto com diversas opções. Ele fornece realce colorido de sintaxe e personalizável para uma variedade de linguagens. Os desenvolvedores podem configurar estilos específicos para diferentes elementos do código, melhorando a legibilidade.
- **Monaco Editor:** Desenvolvido pela Microsoft, o Monaco Editor[4] é a base do Visual Studio Code. Ele fornece efeito de sintaxe colorida e permite personalização extensiva. É um componente mais pesado que os outros, mas com muitos recursos.

Essas ferramentas não apenas facilitam a leitura do código, mas também oferecem funcionalidades adicionais, como autocompletar, verificação de erros em tempo real e navegação simplificada. No desenvolvimento moderno, a integração de editores de texto avançados em plataformas web, como o simulador desenvolvido neste projeto, permite que os usuários tenham acesso a um ambiente de desenvolvimento mais robusto diretamente no navegador, sem a necessidade de instalações manuais complexas. Isso aumenta a acessibilidade e a flexibilidade para os alunos que buscam aprender e praticar programação de maneira interativa.

3 TRABALHOS RELACIONADOS

Há várias ferramentas relacionadas que podem amenizar algumas dificuldades frequentemente enfrentadas em determinados tópicos de Arquitetura de Computadores. Algumas ferramentas, como o simulador de circuitos Logisim[5] (ou seu *fork* mais atual, Logisim-Evolution[6]), são populares em cursos da área. São ferramentas *open source* feitas em Java.

A necessidade de expor alunos à programação direta com o processador por meio da linguagem de Assembly demanda ferramentas que ofereçam um editor de código para esse tipo de tarefa, com algum simulador para permitir o aluno executar o seu código. O espaço que pode ser explorado de ferramentas ligadas a editores, simuladores, debuggers (depuradores), assemblers, disassemblers, etc. é bastante amplo, ainda mais quando se perpassa por diferentes

[2] <https://ace.c9.io/>
[3] <https://codemirror.net/>
[4] <https://microsoft.github.io/monaco-editor/>
[5] <https://cburch.com/logisim/index.html>
[6] <https://github.com/logisim-evolution/logisim-evolution>

arquiteturas sendo consideradas. O aluno explorando esse tema pode acabar se deparando com a *toolchain* (cadeia de ferramentas de desenvolvimento) da GNU, que inclui compilador (gcc)[7], assembler (gas), debugger (gdb), linker (ld), etc., que funcionam como comandos de terminal e estão disponíveis para várias arquiteturas como x86-64, ARM e MIPS. Outras ferramentas, como IDA Pro[8], combinam um disassembler, debugger e decompilador, ou ainda Radare2[9], que combina assembler / disassembler com debugger, editor hexadecimal e outros recursos visando engenharia reversa para várias arquiteturas (x86, ARM, Java, SPARC, etc) e sistemas (Linux, Windows, etc). Em alguns cursos mais específicos essas ferramentas podem ser usadas, porém são mais avançadas, mais distantes da realidade de um aluno começando a aprender sobre Arquitetura de Computadores e tópicos relacionados.

Uma arquitetura tradicionalmente bastante usada no ensino é a arquitetura MIPS, com certa tradição no universo acadêmico e usada também em sistemas embarcados. Essa arquitetura foi alvo do Spim, um simulador de MIPS de Larus (1990). A ferramenta oferece um editor de Assembly, recursos como execução passo a passo, visualização de registradores e memória, etc. Uma versão posterior feita com Qt foi lançada (QtSpim). O autor original, contudo, já se aposentou[10].

Vollmar e Sanderson (2006) desenvolveram a ferramenta MARS (MIPS Assembler and Runtime Simulator)[11], que fornece um ambiente para desenvolver com Assembly, feito em Java. Várias extensões foram feitas sobre MARS, como Sales et al. (2010), Araujo et al. (2014) e Lim e Smitha (2019).

O trabalho de Jonjic (2012) sobre o μMPS2, um simulador MIPS implementado com C++, Boost e Qt, provê uma base para experimentação em cursos de Arquitetura de Computadores e Sistemas Operacionais. A ideia é fornecer um simulador MIPS que seja fácil para alunos desenvolverem experimentos básicos numa disciplina de Sistemas Operacionais, uma vez que usar um hardware real para isso pode envolver inúmeros outros fatores de complexidade. Conforme o autor comenta, a proposta do μMPS é um pouco diferente de outros simuladores como Spim ou Mars, que foram projetados para executar um código Assembly diretamente neles, ao invés de interpretar código de máquina real compilado para arquitetura MIPS. O μMPS, por sua vez, foca mais no ambiente de simulação, que conta com dispositivos periféricos e recursos de debugging, sendo necessário, porém, que se o aluno use alguma *toolchain* (compilador, linker, etc) para a produção dos arquivos binários que irão ser executados. Um aluno poderia usar essas ferramentas para produzir algo para executar com o simulador Qemu também (a ser abordado logo mais adiante); contudo, a diferença do μMPS é que enquanto o Qemu é voltado para executar sistemas reais, focando em desempenho, o μMPS é voltado para o ensino, com

[7] O gcc envolve uma coleção de compiladores, com C, C++, Fortran, Go, etc., além de gerar código para inúmeras arquiteturas e suportar tecnologias como OpenMP para programação paralela.

[8] Software proprietário bem conhecido; ver <https://hex-rays.com/ida-pro>.

[9] <https://www.radare.org>

[10] <https://spimsimulator.sourceforge.net/>

[11] A ferramenta foi sendo atualizada até 2014; os autores se aposentaram. Mas ainda é possível acessar a ferramenta por meio da hospedagem em <https://github.com/dpetersanderson/MARS>.

infraestrutura mais adaptada para experimentação e extensões e simulação de recursos fáceis de se usar (JONJIC, 2012).

Outros trabalhos com MIPS incluem o WebMIPS, de Branovic, Giorgi e Martinelli (2004), sendo feito para web (com código ASP executado no servidor); o CG MIPS, de Silva e Meneses (2009); o DrMIPS, de Nova, Ferreira e Araújo (2013); o MIPSers, de Kho e Uy (2017), feito também em Java; e o WIMS (Web-based Interactive MIPS Simulator), de Assis e Nogueira (2024). Conforme os autores deste último observam em relação a MIPS, há certa falta de simuladores online educativos com interface amigável; muitos, como MARS, requerem instalação ainda (como várias aplicações Java) e usam editores de código antigos, limitando sua usabilidade diante de opções mais modernas (ASSIS; NOGUEIRA, 2024).

A arquitetura MIPS é um exemplo simples e popular de RISC. Foi usada por vários anos em diversos sistemas, desde *workstations* da Silicon Graphics a videogames. Em anos recentes, porém, sua presença diminuiu. A empresa MIPS Technologies trocou a arquitetura MIPS pela mais moderna RISC-V [12].

A arquitetura RISC-V é um exemplo simples e mais moderno de RISC. RISC-V é um padrão aberto, modular, com implementações *open source* e um grupo internacional de mais de 100 empresas coordenando o projeto. Alguns autores de livros de Arquitetura de Computadores passaram a adotar também RISC-V (HENNESSY; PATTERSON, 2017).

Assim, seguindo a nova linha RISC-V, um projeto baseado no MARS foi criado chamado de RARS (RISC-V Assembler and Runtime Simulator) [13], continuando na implementação com Java. A arquitetura RISC-V também é foco do sistema Ripes de Petersen (2021), que possui funcionalidades diversas como editor com Assembly, simulador de pipeline e simulador de cache. O trabalho é *open source* e implementado com Qt[14]. Uma versão experimental para web é fornecida por meio de Qt com WebAssembly, embora a implementação não tenha funcionado no período de escrita do presente trabalho. O Orbit[15], outro trabalho com editor Assembly e simulador RISC-V é feito com JavaScript para web.

Do lado ARM, arquitetura bastante presente em dispositivos móveis, uma ferramenta chamada VisUAL[16] foi implementada com Java e usada em meio acadêmico, em curso com a disciplina de Arquitetura de Computadores no Imperial College London. A ferramenta oferece editor de código com sintaxe colorida, indentação automática, execução, visualização de registradores e memória, etc. Uma outra versão, VisUAL 2[17], foi iniciada visando reimplementar o VisUAL original em F#, que é transpilado para JavaScript, usando tecnologias web como o editor Monaco e o *framework* Electron para acessar recursos nativos, como arquivos. Outro trabalho[18] ligado a ARM ainda é uma implementação também com tecnologia web feita com

[12] <https://www.eejournal.com/article/wait-what-mips-becomes-risc-v/>
[13] <https://github.com/TheThirdOne/rars>
[14] Repositório em: <https://github.com/mortbopet/Ripes>
[15] <https://github.com/aidandempsey/orbit>
[16] <https://salmanarif.bitbucket.io/visual/>
[17] <https://github.com/ImperialCollegeLondon/Visual2>
[18] <https://github.com/DashCampbell/Arm-Assembly-Compiler-Simulator>

frontend NextJS e backend em Rust usando Tauri.

Para outras arquiteturas, pode-se citar o EdSim51[19], relativo à popular família de microcontroladores 8051 lançada pela Intel na década de 1980; o GnuSim8085[20], uma ferramenta implementada com C e Gtk para estudantes trabalharem com o processador Intel 8085, fornecendo editor, assembler e debugger; também nessa linha, o trabalho Simaeac, de Verona, Martini e Gonçalves (2009), que se baseia no 8085; e, por fim, um editor Assembly para web com simulador para o processador Intel 8080, implementado com TypeScript[21].

Há trabalhos que buscam simular arquiteturas para execução de sistemas reais, como o Bochs de Lawton (1996), que começou como um emulador de PC (x86) para ambientes Unix, ou Qemu de Bellard (2005), que emula diversas arquiteturas (x86, PowerPC, ARM, etc.), permitindo executar sistemas operacionais completos como Windows ou Linux em sua emulação. O Gem5, de Binkert et al. (2011), é outra iniciativa, sendo uma plataforma modular criada originalmente com foco na pesquisa da área, fazendo simulação de sistemas ARM, x86, etc., contando com suporte da AMD, ARM, IBM, Intel, entre outras. É uma ferramenta com foco mais na pesquisa e indústria.

Outros projetos ainda com foco na emulação de determinadas arquiteturas incluem o PCjs[22] (que emula o IBM PC 8080 / 8088 em JavaScript) e o v86[23], que em sua emulação chega a converter o código de máquina x86 para WebAssembly para maior desempenho no *browser*.

Simuladores de arquiteturas reais são importantes para facilitar testar softwares de diferentes processadores sem precisar do hardware físico (o que pode ajudar também na redução de custos), além de facilitarem certas análises de desempenho, vulnerabilidades, debugging, etc. Em alguns casos, cumprem um papel importante num ensino mais avançado focado em arquiteturas reais da indústria.

Por outro lado, arquiteturas fictícias podem facilitar muito mais para o estudante, removendo algumas complexidades secundárias em prol da compreensão dos conceitos fundamentais. Elas tornam o ensino mais acessível, além de permitir um aprendizado progressivo, onde diferentes construções mais avançadas são gradualmente integradas. E podem ser ajustadas para se adequar a diferentes objetivos de aprendizado.

Seja inspirando-se em projetos existentes ou elaborando algo do zero, alguns autores de livros na área frequentemente criam seus próprios projetos de arquitetura de computador para fins didáticos, como Y86 de Bryant Bryant e O'Hallaron (2011), Marie de Null e Lobur (2014) ou o computador de Mano em Mano (1993). Algumas dessas arquiteturas são então implementadas como simuladores, como por exemplo MarieSim de Null e Lobur (2003), e um simulador com assembler feito com Qt da máquina de Mano[24]. Outros, ainda, chegam a implementar simuladores de um subconjunto de uma arquitetura existente, como por exemplo o

[19] <https://edsim51.com/>
[20] <https://github.com/GNUSim8085/GNUSim8085>
[21] <https://github.com/mac501pl/8080Sim>
[22] <https://github.com/jeffpar/pcjs>
[23] <https://github.com/copy/v86>
[24] <https://github.com/mmahdibarghi/mano-simulator>

BCL[25], um subconjunto da arquitetura x86-64.

Camarmas-Alonso et al. (2021) criaram uma ferramenta web chamada Creator, um simulador genérico para Assembly, feito inicialmente para MIPS e depois expandido para RISC-V. A ferramenta foi feita com Bootstrap e Vue, e permite também editar as instruções. O Yasp[26] é uma ferramenta de Assembly com editor e simulador feita para web, voltada para ensino. A ferramenta simula de forma visual uma placa com LEDs, permitindo que estudantes possam trabalhar com isso. Por fim, o CompSim (ESMERALDO; LISBOA, 2017) é uma ferramenta implementada com Python[27], disponibilizada como *freeware* [28] no momento de escrita do presente trabalho (os autores pretendem lançar futuramente uma versão *open source* quando o código estiver mais maduro).

O que se pode extrair dessa grande variedade de trabalhos? É possível notar, por exemplo, que vários trabalhos mais antigos usavam Java como meio para serem *multi-plataforma*, enquanto que mais recentemente vários têm migrado ou proposto implementações que usam tecnologia web, com código diretamente em JavaScript ou transpilado para o mesmo. Seja criando aplicações para executar no desktop (via Electron ou Tauri), seja para executar diretamente *online* (com a vantagem de não precisar instalar ou fazer atualizações manuais), o uso de tecnologia web é evidente nos trabalhos mais atuais. Também o aspecto de fazer algo *open source*, modular, expansível, ajuda muito. Projetos *open source*, vários frequentemente hospedados no Github, podem ser muito mais convidativos. Vários são incrementados com novos módulos e funcionalidades e podem, assim, ter uma vida útil enriquecida.

É possível observar, também, que há muitos trabalhos envolvendo MIPS, uma arquitetura tradicional de ensino, embora a mesma tenha sido de certa forma ultrapassada pelo RISC-V, uma arquitetura mais moderna, modular e aberta. Há trabalhos para diversas outras arquiteturas reais específicas como x86 ou ARM, entre outras. Mas, como foi comentado anteriormente, dependendo de cada curso e seu contexto, às vezes pode ser mais interessante o uso de ferramentas com um Assembly menos específico de uma arquitetura e mais geral e acessível, envolvendo os aspectos que forem apropriados de abordar na disciplina, distanciando-se de complexidades secundárias desnecessárias de certas arquiteturas reais. Assim, a ferramenta proposta no presente trabalho adota essa linha, propondo um projeto que seja *open source*, modular, expansível, feito com modernas tecnologias web e com recursos gráficos e instruções SIMD, com documentação inclusa e podendo executar de forma online, atendendo assim a tais demandas.

4 METODOLOGIA

A proposta visa criar uma ferramenta interativa e educacional que integre um editor de código Assembly, um interpretador para execução em tempo real e a capacidade de expansão por

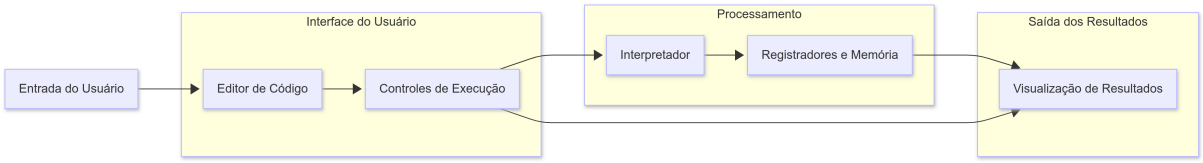
[25] <https://github.com/usd-cs/below-c-level>
[26] <https://github.com/yasp/yasp>
[27] A referência apenas menciona a linguagem Python; o trabalho parece ter sido feito com a biblioteca Tk para a interface gráfica.
[28] <http://compsim.crato.ifce.edu.br/download.html>

meio de pacotes de instruções adicionais. A combinação desses elementos permitirá aos usuários escrever, testar e depurar código Assembly de forma intuitiva e eficiente. Uma visão geral da arquitetura do sistema é dada à seguir.

4.1 Arquitetura do Sistema

A arquitetura geral do simulador proposto é apresentada na Figura 1. A figura mostra o fluxo de interação do sistema, desde a entrada do usuário até a saída dos resultados. Essa estrutura foi planejada para integrar a interface do usuário, o editor de código, o interpretador, os registradores e a visualização de resultados.

Figura 1 – Diagrama da Arquitetura do Simulador



Fonte: Elaborado pelo autor.

Os seguintes elementos podem assim ser destacados:

- **Usuário**: O ponto de interação inicial, onde o usuário insere o código.
- **Interface do Usuário**: A camada que apresenta as funcionalidades do simulador.
- **Editor de Código**: Permite a escrita e edição do código Assembly.
- **Interpretador**: Executa o código escrito pelo usuário.
- **Memória e Registradores**: Onde os dados são armazenados e manipulados durante a execução.
- **Saída e Visualização**: Mostra os resultados da execução do código.

4.2 Interface do Usuário

A interface do usuário é projetada com foco na usabilidade e na experiência do usuário. O editor de código é o componente central, permitindo aos usuários escrever e editar seu código de forma eficiente. Controles de execução, incluindo as funções "Executar", "Pausar"e "Passo a Passo", foram implementados para simplificar a execução do código. Ferramentas de depuração, como a definição de *breakpoints* e a visualização dos registradores, foram integradas para auxiliar os usuários na análise e compreensão do comportamento do código durante a execução.

Um painel de saída exibe os resultados da execução do programa, enquanto os painéis de registradores e memória proporcionam uma visão detalhada sobre como os dados são manipulados.

A responsividade da interface garante que a aplicação se adapte a diferentes tamanhos de tela e dispositivos, proporcionando uma experiência de uso consistente. Além disso, os usuários têm a capacidade de ajustar a velocidade de execução do código e a largura de bits, oferecendo maior controle sobre a simulação.

Para a implementação da visualização de dados, foi utilizada a biblioteca Chart.js[29]. Esta biblioteca permite a criação de gráficos interativos, que poderão ser trabalhados conforme a manipulação dos registradores, podendo servir em determinados contextos visuais conforme o desejo do programador.

A biblioteca Split.js[30] foi empregada para criar uma interface dividida, permitindo que os usuários visualizem simultaneamente o editor de código, a saída do programa e os registradores. Essa funcionalidade melhora a experiência do usuário, tornando a interação com a ferramenta mais intuitiva e eficiente.

4.3 Editor de Código

O editor de código é implementado utilizando a biblioteca CodeMirror[31]. A escolha desta biblioteca é fundamentada na sua facilidade de integração com diferentes plataformas, no suporte a uma variedade de linguagens de programação e na capacidade de fornecer realce de sintaxe, o que melhora a legibilidade do código. Além disso, o CodeMirror oferece recursos avançados, como autocompletar, que auxiliam os usuários na escrita de código Assembly, minimizando erros e aumentando a eficiência do processo de codificação. O editor também provê uma experiência de desenvolvimento responsiva, adaptando-se a diferentes tamanhos de tela e dispositivos, o que é essencial para atender a diferentes necessidades.

4.4 Interpretador do Código

Além do editor, o trabalho deve contar com um interpretador, sendo assim possível executar o código digitado. O interpretador deve ser capaz de aceitar uma ampla gama de instruções Assembly gerais, abrangendo operações aritméticas, como ADD e SUB, operações lógicas, como AND e OR, e instruções de controle de fluxo, como JMP e CALL.

O interpretador também deve permitir que o usuário possa manipular diretamente os registradores e a memória.

Adicionalmente, a implementação deve incluir mecanismos para lidar com erros de sintaxe e instruções inválidas, garantindo que os usuários recebam orientações claras sobre como corrigir problemas em seu código. A interface do usuário deve ser projetada para ser intuitiva, permitindo que os usuários interajam facilmente com o interpretador e visualizem os resultados de suas operações.

[29] <https://www.chartjs.org//>
[30] <https://split.js.org/>
[31] <https://codemirror.net/>

O interpretador foi desenvolvido com uma arquitetura modular, com partes específicas para operações aritméticas, lógicas, movimentação de dados e manipulação de pilha.

Para a leitura do código, a fase de análise léxica envolve o pré-processamento do código Assembly, que identifica *labels*, remove linhas vazias e comentários, e verifica a sintaxe das instruções. Essa etapa é essencial para estruturar o código de maneira compreensível antes da execução, garantindo que apenas instruções válidas sejam consideradas.

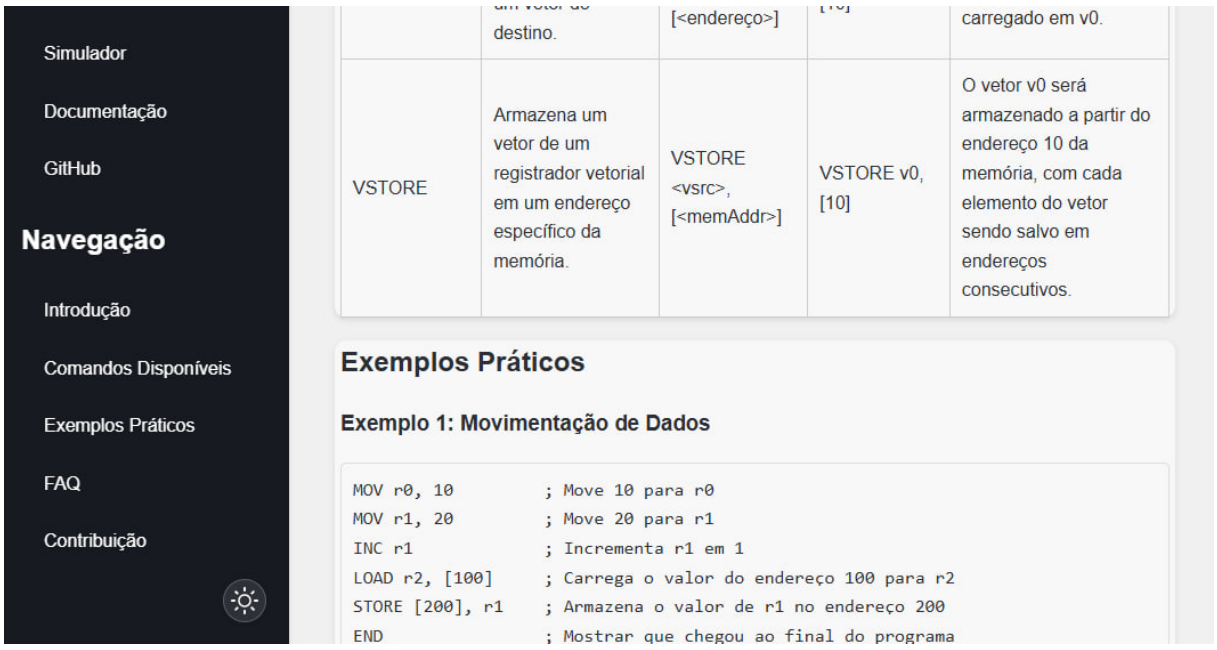
A fase de execução, por fim, é responsável por interpretar as instruções propriamente ditas e realizar as operações correspondentes. Cada instrução Assembly é mapeada para uma função específica que será chamada durante a execução. A implementação modular permite fácil adição de novas instruções e uma manutenção simplificada.

5 RESULTADOS

5.1 Funcionamento da Ferramenta

O simulador apresentado possui uma página principal para executar a simulação e uma segunda página de documentação, que detalha as suas funcionalidades, comandos e possui ainda vários exemplos práticos conforme ilustrado na Figura 2, abaixo.

Figura 2 – Página da documentação



Fonte: Elaborado pelo autor.

5.1.1 Interface do Simulador

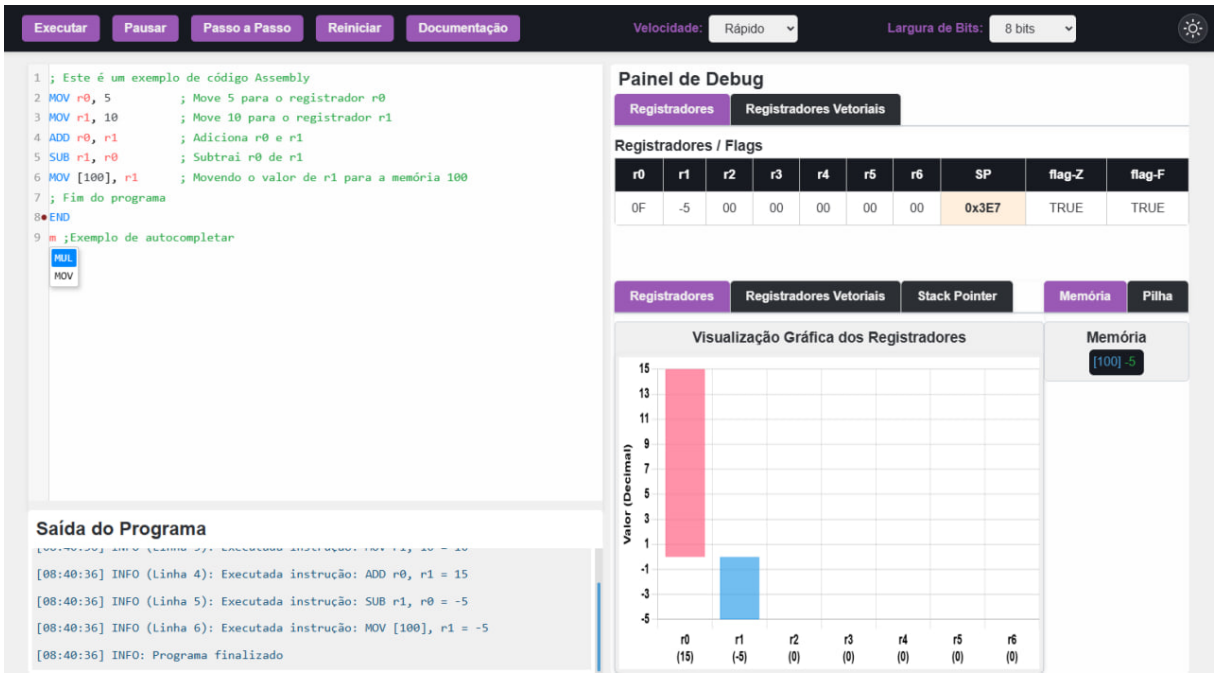
A interface do simulador é composta pelos seguintes componentes principais:

- **Editor de Código**: Permite que os usuários escrevam e editem códigos Assembly, com suporte a destaque de sintaxe, funcionalidade de breakpoints e auto completar.

- **Controles de Execução**: Botões para executar, pausar, depurar códigos passo a passo e reiniciar.
- **Painel de Registradores**: Representação visual da memória do sistema, mostrando alterações durante a execução.
- **Painel de Memória**: Mostra o conteúdo armazenado durante a execução.
- **Histórico de Execução**: Registro detalhado das instruções processadas, incluindo descrições e mudanças nos estados dos registradores e da memória.
- **Configurações Avançadas**: Opções para ajustar a largura de entrada dos registradores (ex.: 8, 16 ou 32 bits) e a velocidade de execução.

Esses componentes podem ser visualizados na Figura 3, que mostra a interface completa do simulador de Assembly.

Figura 3 – Interface do simulador de Assembly



Fonte: Elaborado pelo autor.

5.2 Exemplos Práticos de Simulação

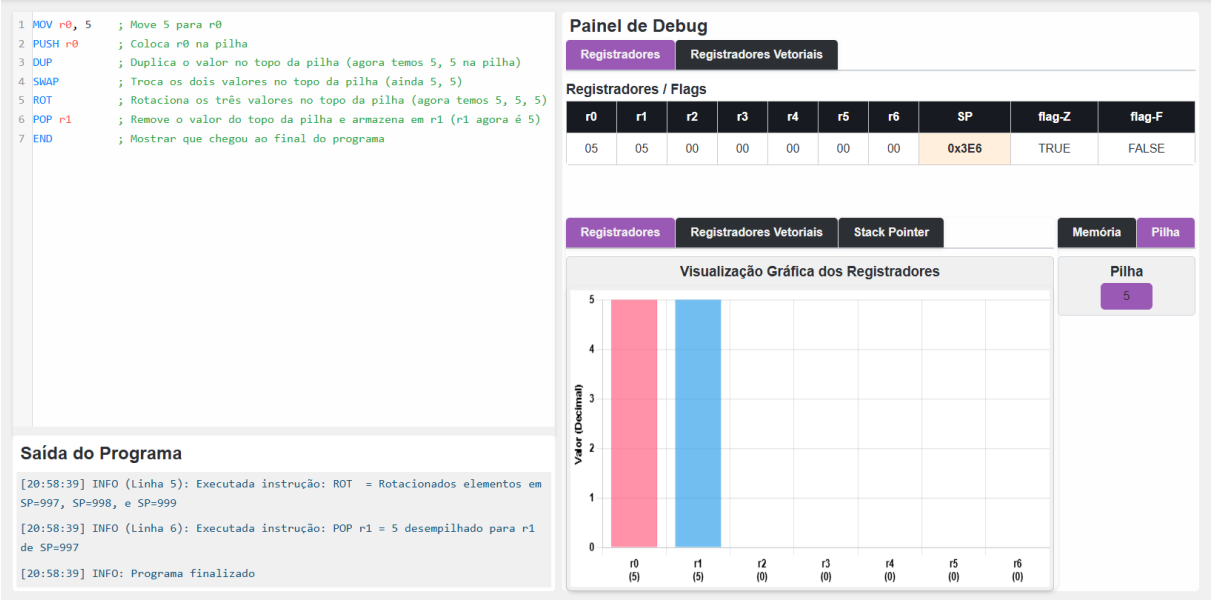
Para ilustrar alguns exemplos de código da ferramenta, dois exemplos foram selecionados da documentação e apresentados à seguir.

5.2.1 Exemplo 1: Manipulação de Pilha

Esse exemplo, retirado da página de documentação, demonstra como realizar operações básicas de manipulação da pilha, como duplicação, troca e remoção de valores, úteis em algorit-

mos que exigem armazenamento temporário ou cálculo intermediário. Conforme ilustrado na Figura 4.

Figura 4 – Manipulação de Pilha

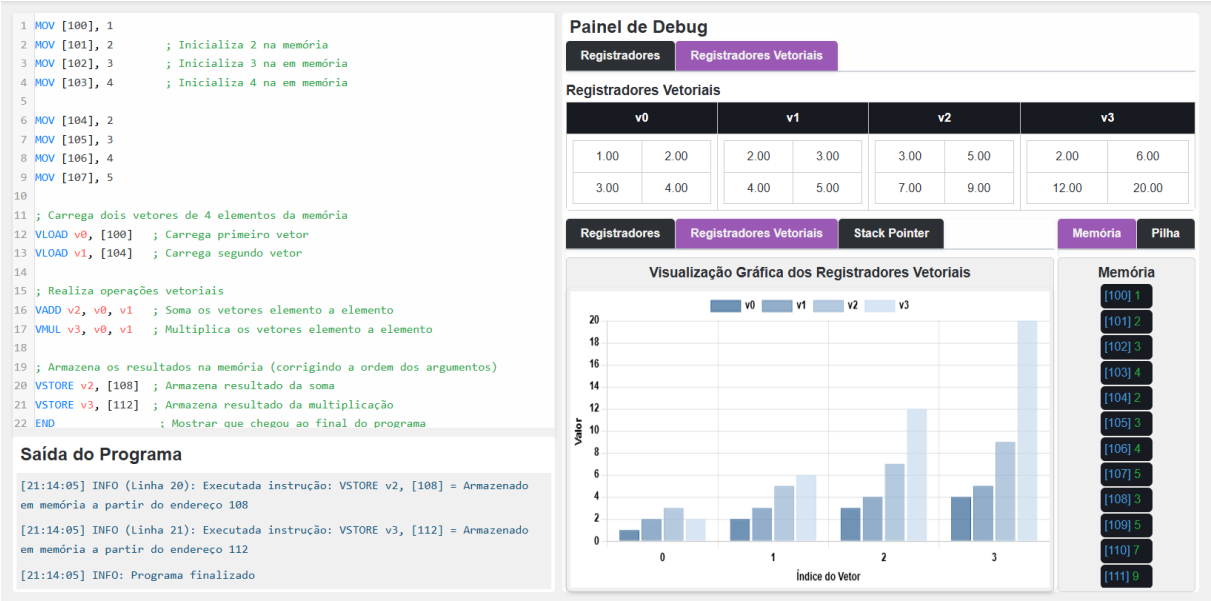


Fonte: Elaborado pelo autor.

5.2.2 *Exemplo 2: Adição Vetorial*

Este exemplo, retirado da página de documentação, utiliza instruções SIMD, inicializa dois vetores na memória, realiza a soma e a multiplicação elemento a elemento, e armazena os resultados nas posições de memória 108 e 112, respectivamente. Conforme ilustrado no trecho de código da Figura 5.

Figura 5 – Adição Vetorial



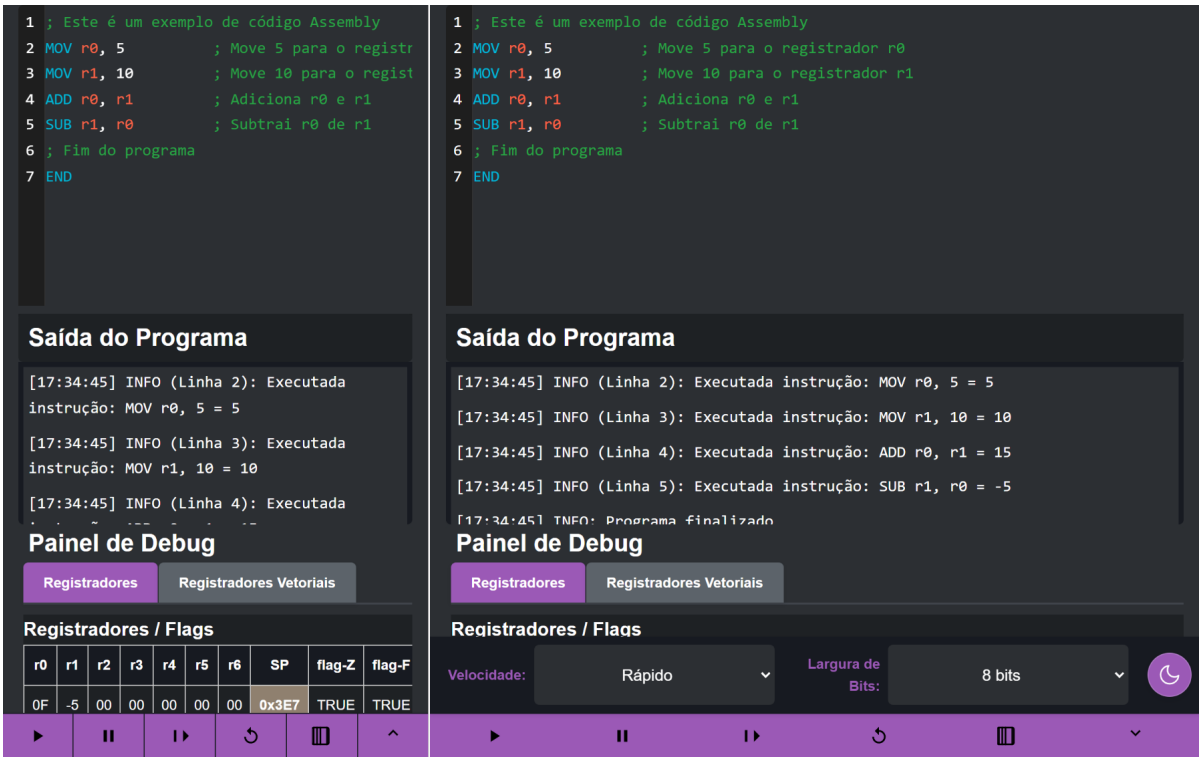
Fonte: Elaborado pelo autor.

5.3 Mudança de Tema e Responsividade

Foi implementada uma funcionalidade de troca de tema no simulador, permitindo ao usuário alternar entre o tema claro (*Light Mode*) e o escuro (*Dark Mode*) como podemos observar na Figura 6. Essa opção foi adicionada para oferecer uma experiência mais confortável, especialmente em ambientes com pouca iluminação, e para atender às preferências pessoais dos usuários.

Além disso, a responsividade do site foi otimizada para garantir que o simulador se adapte adequadamente a diferentes dispositivos, como smartphones, tablets e desktop. A utilização de técnicas como *media queries* e flexbox permitiu ajustar o layout do simulador para uma experiência de uso fluida e intuitiva em qualquer resolução de tela, garantindo acessibilidade e usabilidade em diversos dispositivos.

Figura 6 – Mudança de tema e responsividade



Fonte: Elaborado pelo autor.

6 CONCLUSÃO

O presente trabalho buscou contribuir para o ensino de linguagem Assembly em cursos de Arquitetura de Computadores ao desenvolver um simulador web interativo e didático que possibilita aos alunos praticar e compreender conceitos fundamentais da disciplina. A ferramenta foi projetada com foco na usabilidade, acessibilidade e aplicação prática, promovendo um ambiente educacional mais intuitivo e engajador.

O simulador utiliza tecnologias atuais para proporcionar uma experiência dinâmica, permitindo que os usuários editem, executem e analisem programas Assembly em tempo real.

Além disso, recursos como controle de execução, visualização gráfica e suporte a instruções SIMD agrega opções interessantes que podem ser abordadas no ensino de forma atrativa.

Atualmente, o simulador encontra-se disponível em uma plataforma online do GitHub Pages e possui seu código-fonte hospedado em um repositório público no GitHub, promovendo o acesso aberto e incentivando colaborações futuras.

Como sugestões de trabalhos futuros, pode-se expandir com novos conjuntos de instruções, bem como integrar funcionalidades didáticas adicionais como tutoriais e relatórios de desempenho. Além disso, pode-se criar um meio para validar e contabilizar tarefas de códigos desenvolvidos por alunos, viabilizando mais a ferramenta em determinados ambientes de ensino. Por fim, é possível aumentar a motivação para aprendizado com a introdução de módulos de E/S interessantes para jogos, como entrada de joystick, simulação de LEDs e saída gráfica geral de tela de vídeo, consolidando a ferramenta para fins didáticos diversos no ensino.

LINKS DA APLICAÇÃO

1. https://marlilsonm.github.io/Simulador_Assembly/ (Site do simulador web Assembly)
2. https://github.com/MarlilsonM/Simulador_Assembly (Repositório da Interface do Usuário)

REFERÊNCIAS

ARAUJO, M. R. D. et al. Mips x-ray: A mars simulator plug-in for teaching computer architecture. **International Journal of Recent Contributions from Engineering, Science & IT (iJES)**, v. 2, n. 2, p. 36–42, 2014.

ASSIS, R.; NOGUEIRA, B. Wims: A modern web-based mips simulator for improved learning in computer architecture and operating systems. **International Journal of Computer Architecture Education**, v. 13, n. 1, p. 1–6, 2024.

BELLARD, F. Qemu, a fast and portable dynamic translator. In: CALIFORNIA, USA. **USENIX annual technical conference, FREENIX Track**. [S.l.], 2005. v. 41, n. 46, p. 10–5555.

BINKERT, N. et al. The gem5 simulator. **ACM SIGARCH computer architecture news**, ACM New York, NY, USA, v. 39, n. 2, p. 1–7, 2011.

BRANOVIC, I.; GIORGI, R.; MARTINELLI, E. Webmips: a new web-based mips simulation environment for computer architecture education. In: **Proceedings of the 2004 Workshop on Computer Architecture Education: Held in Conjunction with the 31st International Symposium on Computer Architecture**. New York, NY, USA: Association for Computing Machinery, 2004. (WCAE '04), p. 19–es. ISBN 9781450347334. Disponível em: <https://doi.org/10.1145/1275571.1275596>.

BRYANT, R. E.; O'HALLARON, D. R. **Computer systems: a programmer's perspective**. [S.l.]: Prentice Hall, 2011.

CAMARGO, J. Simbler – um simulador de linguagem de montagem didÁtico aplicado ao ensino de informÁtica. **Interciência e Sociedade (Versão online) - ISSN: 2238-1295**, v. 3, p. 118–128, 12 2013.

CAMARMAS-ALONSO, D. et al. A new generic simulator for the teaching of assembly programming. In: IEEE. **2021 XLVII Latin American Computing Conference (CLEI)**. [S.l.], 2021. p. 1–9.

CELESTINO, M. S.; VALENTE, V. C. P. N. Aplicabilidade e benefÍcios de softwares e simuladores em processos de ensino-aprendizagem. **ETD - Educação Temática Digital**, SciELO, v. 23, p. 882–904, 10 2021. ISSN 1676-2592. Disponível em: <http://educa.fcc.org.br/scielo.php?script=sci_arttext&pid=S1676-25922021000400882&nrm=iso>.

CLAYTON, G.; GIZELIS, T.-I. Learning through simulation or simulated learning? an investigation into the effectiveness of simulations as a teaching tool in higher education. In: **Proceedings from British International Studies Association Conference**. [S.l.: s.n.], 2005.

ESMERALDO, G.; LISBOA, E. B. Uma ferramenta para exploração do ensino de organização e arquitetura de computadores. **International Journal of Computer Architecture Education**, v. 6, n. 1, p. 68–75, 2017.

GASNAS, A.; GLOBA, A. Reasons why assembly language remains fundamental in modern programming. **Acta et Commentationes Sciences of Education**, v. 36, n. 2, p. 78–86, 2024.

GASNAş, A. Explorarea metodelor de predare a cursului de programare „limbaje de asamblare". In: **Conference on Applied and Industrial Mathematics**. [S.l.: s.n.], 2022. p. 156–161.

HALL, B. R.; SLONKA, K. J. **Assembly Programming and Computer Architecture for Software Engineers**. 2nd. ed. Burlington, VT: Prospect Press, 2020. ISBN 978-1-943153-32-9.

HENNESSY, J. L.; PATTERSON, D. A. **Computer organization and design RISC-V edition: The hardware software interface**. [S.l.]: Elsevier Science & Technology Books, 2017.

JONJIC, T. **Design and Implementation of the uMPS2 Educational Emulator**. Tese (Doutorado), 2012.

KHO, N. M. D.; UY, R. L. Mipsers: Mips extension release 6 simulator. In: **2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)**. [S.l.: s.n.], 2017. p. 1–6.

LARUS, J. R. **Spim s20: A mips r2000 simulator**. [S.l.], 1990.

LAWTON, K. P. Bochs: A portable pc emulator for unix/x. **Linux Journal**, Belltown Media Houston, TX, v. 1996, n. 29es, p. 7–es, 1996.

LIM, D. X.; SMITHA, K. G. Pipelined mips simulation: A plug-in to mars simulator for supporting pipeline simulation and branch prediction. In: **2019 IEEE International Conference on Engineering, Technology and Education (TALE)**. [S.l.: s.n.], 2019. p. 1–7.

MANO, M. M. **Computer system architecture**. [S.l.]: Prentice-Hall, Inc., 1993.

NOVA, B.; FERREIRA, J. C.; ARAúJO, A. Tool to support computer architecture teaching and learning. In: **2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)**. [S.l.: s.n.], 2013. p. 1–8.

NULL, L.; LOBUR, J. Mariesim: The marie computer simulator. **J. Educ. Resour. Comput.**, Association for Computing Machinery, New York, NY, USA, v. 3, n. 2, p. 1–es, jun. 2003. ISSN 1531-4278. Disponível em: <https://doi.org/10.1145/982753.982754>.

NULL, L.; LOBUR, J. **Essentials of Computer Organization and Architecture**. [S.l.]: Jones & Bartlett Publishers, 2014.

PETERSEN, M. B. Ripes: A visual computer architecture simulator. In: **2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)**. [S.l.: s.n.], 2021. p. 1–8.

SALES, G. C. R. et al. Mips x-ray: A plug-in to mars simulator for datapath visualization. In: **2010 2nd International Conference on Education Technology and Computer**. [S.l.: s.n.], 2010. v. 2, p. V2–32–V2–36.

SANTOS, C. D. P. **Criação de Hardware de Comunicação Ethernet para Recepção de Som**. Tese (Doutorado) — Universidade do Minho, 2008.

SILVA, G. C.; MENESES, R. C. Um ambiente computacional de apoio à aprendizagem de instruções assembly. In: **I Encontro Nacional de Informática e Educação**. Cascavel - PR: [s.n.], 2009.

VERONA, A. B.; MARTINI, J. A.; GONçALVES, T. L. Simaeac: Um simulador acadêmico para ensino de arquitetura de computadores. **I ENINED - Encontro Nacional de Informática e Educação**, p. 424–432, 2009. ISSN 2175-5876.

VOLLMAR, K.; SANDERSON, P. Mars: an education-oriented mips assembly language simulator. Association for Computing Machinery, New York, NY, USA, p. 239–243, 2006. Disponível em: <https://doi.org/10.1145/1121341.1121415>.

WATSON, C.; LI, F. W. Taxas de falha na programação introdutória revisitadas. In: **Anais da conferência de 2014 sobre Inovação e tecnologia na educação em ciência da computação**. [S.l.: s.n.], 2014. p. 39–44.