Simulador Web para o Ensino de Arquitetura de Computadores com Suporte a Vetores e Cache

João Pedro R. Linares[1], **André Rauber Du Bois**[1]
Gerson Geraldo H. Cavalheiro[1]

[1]Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas
R. Gomes Carneiro, 01 – Pelotas – RS/Brasil – 96010-610

{jprlinares,gersonc,dubois}@inf.ufpel.edu.br

Abstract. *This article presents NeanderWeb-V, a web-based simulator of the didactic Neander-V architecture, which extends the original Neander with support for vectorization and cache. The tool is accessible directly through web browsers and was designed to support introductory teaching of Computer Architecture and Programming. Its main feature is the exposure of hardware-level parallelism mechanisms, allowing students to observe the impact of the parallel resources in the Neander-V architecture on program performance. Programs can be written, executed, and debugged directly in the browser, in a modular and informative environment. A case study conducted with first-semester Computer Science students indicated high usability and ease of use, reinforcing the pedagogical potential of the tool.*

Resumo. *Este artigo apresenta o NeanderWeb-V, um simulador baseado em web da arquitetura didática Neander-V, que estende o Neander original, com suporte à vetorização e cache. A ferramenta é acessível diretamente por navegadores e foi projetada para apoiar o ensino introdutório de Arquitetura de Computadores e Programação. Seu principal destaque é a exposição de mecanismos de paralelismo em nível de hardware, permitindo que estudantes observem o impacto dos recursos paralelos da arquitetura Neander-V sobre o desempenho dos programas. Os programas podem ser escritos, executados e depurados diretamente no navegador, em um ambiente modular e informativo. Um estudo de caso realizado com estudantes do primeiro semestre de curso na área da Computação indicou alta usabilidade e facilidade de uso, reforçando o potencial pedagógico da ferramenta.*

1. Introdução

O ensino de arquitetura de computadores, e mesmo dos princípios básicos de programação, é um dos principais desafios nas fases iniciais da formação em Computação. Isso se deve, em grande parte, à abstração imposta pelas linguagens de alto nível, que afastam os estudantes dos mecanismos fundamentais de execução no núcleo de processamento (CPU). Uma estratégia pedagógica para superar essa barreira é o uso de arquiteturas simplificadas, modeladas por simuladores didáticos. Entre elas, a arquitetura Neander [Weber 2000] se destaca por oferecer um modelo minimalista de processador, com os componentes essenciais, como registradores, memória e conjunto de instruções, mas contendo os recursos fundamentais dos sistemas computacionais.

Simuladores educacionais (Seção 2) são especialmente úteis em cursos introdutórios, pois não exigem instalação de softwares complexos e permitem a experimentação da execução de programas em ambientes visuais. No entanto, mesmo cumprindo seu papel na introdução às arquiteturas do tipo von Neumann e à lógica de programação de baixo nível, tais simuladores raramente abordam conceitos de paralelismo, cada vez mais presentes nas arquiteturas modernas. Por outro lado, ferramentas elaboradas, como o CPU Sim [Skrien 2001] e o EASE [Skillen et al. 2011], que implementam arquiteturas complexas, apresentam barreiras de entrada elevadas para estudantes iniciantes.

Recursos como caches, pipeline, execução superescalar e instruções vetoriais são exemplos de paralelismo explorado nas atuais arquiteturas. Apesar de amplamente utilizados, esses mecanismos permanecem pouco explorados em disciplinas introdutórias, em parte pela ausência de ferramentas didáticas adequadas que exponham tais recursos de forma compatível com o nível de abstração dos estudantes em início de formação.

Este trabalho apresenta a arquitetura hipotética Neander-V, uma extensão da arquitetura Neander, e seu simulador, o NeanderWeb-V. Desenvolvido com tecnologias web [Linares and Cavalheiro 2025], o simulador oferece um ambiente prático e intuitivo no qual os estudantes podem escrever, executar e depurar programas diretamente no navegador, visualizando o estado dos registradores, memória e caches em tempo real. A proposta busca preencher a lacuna entre simplicidade e contemporaneidade, unindo a clareza didática do Neander à introdução de conceitos modernos como o paralelismo de dados e a hierarquia de memória com cache.

O restante deste artigo está organizado como segue. A Seção 2 revisa alguns dos simuladores educacionais disponíveis, contextualizando a proposta. A Seção 3 detalha a arquitetura Neander-V, incluindo suas extensões vetoriais e suporte a caches. A Seção 4 apresenta a implementação e os recursos do simulador. A Seção 5 traz exemplos práticos e uma análise de desempenho. A Seção 6 discute as limitações e direções futuras, e a Seção 7 apresenta as considerações finais.

2. Trabalhos Relacionados

O uso de simuladores de arquitetura de computadores é relevante no contexto de disciplinas introdutórias em cursos de Computação, dada a dificuldade de alunos iniciantes em lidar com conceitos abstratos como ciclo de instruções, sinais de controle e organização interna da CPU. Esses simuladores são consolidados como grandes ferramentas pedagógicas ao permitirem a visualização gráfica de estruturas internas do processador, o acompanhamento passo a passo da execução de instruções e a exploração interativa de conceitos de baixo nível. Um aspecto destacado é a importância de apresentar aos estudantes um modelo simplificado de CPU, que possa ser manipulado por meio de interfaces visuais e feedback imediato [Djordjevic et al. 2005].

Entre outros, simuladores como EasyCPU [Yehezkel et al. 2009], LMC [Yurcik et al. 2001] e DrMIPS [Nova et al. 2013] foram desenvolvidos para suprir essa necessidade. A visualização dos componentes internos, a identificação de blocos funcionais e a observação do fluxo de dados e sinais em tempo real favorecem o entendimento de tópicos como pipelining e modos de endereçamento, bem como de temas como cálculo de desempenho e estrutura do conjunto de instruções.

Esses ambientes são úteis para promover o aprendizado, ao permitir que estudan-

tes escrevam e testem seus próprios programas em assembly, visualizem os efeitos de cada instrução na memória e nos registradores, e identifiquem de forma clara o papel dos sinais de controle e do fluxo de execução no processador. Outro ponto é a importância dos simuladores para o ensino de programação de baixo nível. Muitos cursos utilizam essas ferramentas para introduzir linguagem de máquina e assembly a estudantes com pouca ou nenhuma experiência prévia nesse tipo de linguagem. Abordagens que possuem execução passo a passo e representações visuais permitem que os estudantes compreendam a codificação de instruções, a resolução de rótulos e saltos, e o comportamento de instruções. Além disso, o processo de tradução manual de algoritmos em assembly ajuda a explicitar a lacuna semântica entre linguagens de alto e baixo nível.

Simuladores educacionais modernos incorporam recursos pensados especificamente para o processo didático. Entre os mais citados estão editores com realce de sintaxe, modos de execução diferenciados (como o passo a passo) e a possibilidade de observar dinamicamente os estados dos registradores e da memória. Algumas ferramentas ainda oferecem suporte à simulação de código automodificável, o que permite explorar aspectos avançados como a arquitetura de von Neumann e outros conceitos.

Além dos aspectos pedagógicos, questões logísticas e de acessibilidade são fatores decisivos na adoção dessas ferramentas. O crescimento de modalidades de ensino remoto e híbrido ampliou a demanda por soluções independentes de infraestrutura especializada. Simuladores que funcionam diretamente em navegadores têm destaque por eliminarem barreiras técnicas de instalação, configuração e compatibilidade entre sistemas, permitindo que estudantes utilizem dispositivos pessoais, como notebooks e celulares, para explorar conceitos de forma autônoma e interativa. Além disso, esse tipo de ferramenta favorece o estudo assíncrono e a flexibilidade curricular.

No contexto brasileiro, destaca-se a iniciativa pioneira na concepção da arquitetura Neander [Weber 2000]. Sua simplicidade permite que seus princípios sejam facilmente abordados, ajudando na assimilação inicial dos conceitos fundamentais da arquitetura de um processador. O Neander-X, uma extensão funcional da arquitetura original, foi apresentado por meio do simulador NeanderWin [Borges and Silva 2006]. Apesar de suas qualidades didáticas, o Neander apresenta limitações estruturais devido à sua simplicidade, como espaço de memória reduzido, ausência de suporte adequado a sub-rotinas, e modos de endereçamento mais sofisticados. A procura de superar essas limitações motivou o desenvolvimento do processador Sapiens, uma evolução do Neander-X, implementado no ambiente integrado SimuS [Silva and Borges 2016]. O Sapiens amplia significativamente os recursos da arquitetura original, como novos modos de endereçamento e instruções de controle mais robustas. O SimuS também incorpora uma interface com editor, compilador, depurador e simulador integrados, mantendo compatibilidade com os programas desenvolvidos para as arquiteturas anteriores. A Tabela 1 apresenta um comparativo entre essas e outras ferramentas difundidas. A análise evidencia a variedade de arquiteturas base adotadas, desde modelos minimalistas como LMC e Neander, até plataformas mais robustas como MIPS, destacando suas características didáticas.

A análise da Tabela 1 ajuda a identificar o papel do NeanderWeb-V no ensino. É possível ver que há tantos simuladores simples, como o LMC, que não oferecem caminhos para conceitos modernos, e os baseados em MIPS, que podem já incluir recursos como suporte a sub-rotinas, mas podem representar um salto muito grande de complexi-

Tabela 1. Simuladores educacionais de arquitetura de computadores.

Simulador	Ref.	Plataforma	Editor Embutido	Arquitetura Base
CPU Sim	[Skrien 2001]	Multi	Sim	Customizável
DrMIPS	[Nova et al. 2013]	Multi	Sim	MIPS
EduMIPS64	[Patti et al. 2012]	Multi	Não	MIPS
EASE	[Skillen et al. 2011]	Multi	Não	Customizável
LMC	[Yurcik et al. 2001]	Web	Sim	LMC
MARS	[Vollmar et al. 2006]	Multi	Sim	MIPS
NeanderWeb-V	[Este Artigo]	Web	Sim	Neander
SimuS	[Silva and Borges 2016]	Multi	Sim	Sapiens
WebMIPS	[Branovic et al. 2004]	Web	Sim	MIPS
WepSIM	[García et al. 2019]	Web	Sim	MIPS

dade para o aluno iniciante. O SimuS preenche uma parte dessa lacuna, sendo baseado no Neander-X, mas o passo seguinte à introdução ao paralelismo de hardware (dados e memória) de uma forma simples e acessível ainda permanece um tanto quanto inexplorado. Essa é a lacuna que o NeanderWeb-V procura preencher.

3. Arquitetura Neander-V

A arquitetura Neander-V foi projetada como uma evolução do processador Neander original, mantendo a compatibilidade, e sua simplicidade fundamental, enquanto introduz conceitos associados ao paralelismo em nível de hardware.

3.1. Arquitetura Base

A arquitetura Neander foi concebida com fins didáticos, oferecendo uma configuração mínima, que facilita a compreensão do funcionamento interno de um processador. Este processador conta com um único registrador escalar de dados, o **acumulador (AC)**, para uso geral. O AC recebe o resultado das operações realizadas pela **Unidade Lógica e Aritmética (ULA)**, sendo também seu parâmetro de entrada em muitos casos. O **apontador de programa (PC)** organiza a sequência de execução das instruções. A memória, com 256 bytes, é utilizada tanto para o armazenamento das instruções quanto para os dados. Por fim, o processador também possui flags de estado que sinalizam as condições resultantes das operações que alteram o conteúdo de AC, indicando a ocorrência de valores nulos (Z) ou negativos (N).

3.2. Extensão Vetorial

A primeira grande extensão do Neander-V é o suporte a operações vetoriais. Para isso, a arquitetura introduz o registrador vetorial de dados **VAC (Vector Accumulator)**, que armazena um vetor de 4 elementos de 8 bits. A exemplo do registrador AC, o conteúdo de VAC também é apresentado na interface. Um novo conjunto de instruções vetoriais (Tabela 2, linhas destacadas em cinza) foi adicionado para operar sobre os dados no VAC, permitindo a execução de uma única instrução sobre múltiplos dados (SIMD) e introduzindo o conceito de paralelismo de dados. A tabela inclui exemplos e descrição das instruções, bem como a informação se as instruções alteram os flags Negativo (N) e Zero (Z); as instruções vetoriais não alteram o registrador AC, assim, não alteram estes flags.

Tabela 2. Instruções da arquitetura Neander-V

Hex	Mnemônico	Exemplo	Descrição	N	Z
10	STA	STA 40	A posição 0x40 da memória recebe o conteúdo de AC.	✗	✗
20	LDA	LDA 41	Carrega o valor da memória no endereço 0x41 para AC.	✓	✓
30	ADD	ADD 42	Soma o valor da memória no endereço 0x42 ao AC.	✓	✓
40	OR	OR 43	Realiza OU lógico entre AC e o conteúdo do endereço 0x43.	✓	✓
50	AND	AND 44	Realiza E lógico entre AC e o conteúdo do endereço 0x44.	✓	✓
60	NOT	NOT	Inverte todos os bits do AC.	✓	✓
80	JMP	JMP 45	Salta incondicionalmente para o endereço 0x45.	✗	✗
90	JN	JN 46	Salta para 0x46 se o bit de sinal (N) estiver ativado.	✗	✗
A0	JZ	JZ 47	Salta para 0x47 se o bit de zero (Z) estiver ativado.	✗	✗
F0	HLT	HLT	Interrompe a execução do programa.	✗	✗
F1	VLD	VLD 50	Carrega vetor da memória (a partir de 0x50) para o registrador VAC.	✗	✗
F2	VST	VST 54	Armazena o vetor de VAC na memória a partir do endereço 0x54.	✗	✗
F3	VADD	VADD 51	Soma, elemento a elemento, VAC com o vetor armazenado a partir de 0x51.	✗	✗
F4	VEADD	VEADD	Soma o valor escalar do AC a todos os elementos de VAC.	✗	✗
F5	VOR	VOR 52	Realiza OU lógico, elemento a elemento, entre VAC e o vetor da memória.	✗	✗
F6	VAND	VAND 53	Realiza E lógico, elemento a elemento, entre VAC e o vetor da memória.	✗	✗
F7	VNOT	VNOT	Inverte bit a bit cada elemento do vetor VAC.	✗	✗

3.3. Introdução de Caches

A segunda extensão é a introdução da simulação de caches de dados, **D-Cache** com 16 bytes, e de instruções, **I-Cache** com 8 bytes, permitindo observar o impacto da localidade de referência na execução de um programa. A política de escrita adotada para a D-Cache é *write-through*, garantindo consistência com a memória principal, e a alocação ocorre apenas em casos de leitura com falha (*no-write-allocate*). As linhas de cache encontram-se alinhadas à memória principal em múltiplos de 8 (I-Cache) e de 16 (I-Data) bytes.

A interface exibe dinamicamente os contadores de *hits* e *misses*, os campos de *tag* e o conteúdo de cada cache, fornecendo feedback imediato durante a execução passo a passo do programa. Cada cache possui uma única linha, e os blocos são carregados a partir de endereços múltiplos do seu tamanho: múltiplos de 8 bytes para a I-Cache e de 16 bytes para a D-Cache, simplificando o mapeamento entre os endereços da memória principal e os blocos armazenáveis em cache. Em caso de *miss*, o bloco correspondente é carregado, substituindo integralmente o conteúdo anterior, o que é possível devido à política de escrita *write-through*.

4. Arquitetura do Simulador NeanderWeb-V

O NeanderWeb-V foi implementado utilizando tecnologias web (HTML5, CSS3 e JavaScript ES6). Este conjunto de ferramentas permite que o simulador seja executado

diretamente em navegadores modernos, sem necessidade de instalação ou configurações complexas, aumentando a acessibilidade para os estudantes.

4.1. Requisitos do Simulador

O desenvolvimento do NeanderWeb-V foi guiado por requisitos funcionais (RF) e não funcionais (RNF), estabelecidos com o objetivo de garantir tanto a fidelidade didática da simulação quanto a acessibilidade da ferramenta para estudantes em formação inicial.

Requisitos Funcionais (RF):

- **RF1**: Permitir a escrita, execução e depuração de programas em linguagem de montagem da arquitetura Neander-V.
- **RF2**: Oferecer suporte completo às instruções escalares e vetoriais da arquitetura.
- **RF3**: Simular o comportamento de caches de instruções (I-Cache) e de dados (D-Cache), com contabilização de *hits* e *misses*.
- **RF4**: Exibir dinamicamente o estado interno dos componentes (registradores, memória, caches e flags).
- **RF5**: Suportar modos de execução contínua e passo a passo.

Requisitos Não Funcionais (RNF):

- **RNF1**: Ser acessível via navegador moderno, sem dependências externas ou necessidade de instalação.
- **RNF2**: Possuir uma interface intuitiva, com realce de sintaxe, mensagens de erro informativas e suporte multilíngue.
- **RNF3**: Oferecer compatibilidade com diferentes tamanhos de tela e dispositivos, incluindo tablets e smartphones.
- **RNF4**: Ser modular e extensível, permitindo a futura adição de recursos como novos registradores, modos de endereçamento e instruções.

4.2. Interface Gráfica e Usabilidade

A interface foi concebida contando com painéis lógicos, permitindo fácil visualização do programa, memória de dados, registradores e caches, e interação do usuário. A visão geral da interface é apresentada na Figura 1. O usuário pode optar por visualizar os dados em decimal ou hexadecimal, podendo também escolher a língua de apresentação da interface, atualmente disponível em português e inglês.

O menu superior oferece diferentes funcionalidades, como carregar e salvar um arquivo a partir ou para o computador do usuário, alterar o modo de visualização, opções de execução, assim como um botão de ajuda, informando todas as instruções disponíveis no simulador e um exemplo de uso. São duas opções de execução disponíveis: executar todo o programa ou executar passo a passo. Em ambos os casos, partindo da posição de memória zero. O usuário também pode limpar a execução passada, deixando o estado da arquitetura limpo para uma nova execução. O painel inferior oferece informações de log, registrando os eventos observados durante a execução.

Acima do painel de log encontram-se as informações sobre o estado das memórias cache, incluindo a contabilização de hits e misses no cache, e do conteúdo dos registradores, AC, VAC e PC. Também é apresentado o status das flags negativo (N) e zero (Z) e do sistema, decimal ou hexadecimal, de apresentação dos valores numéricos.

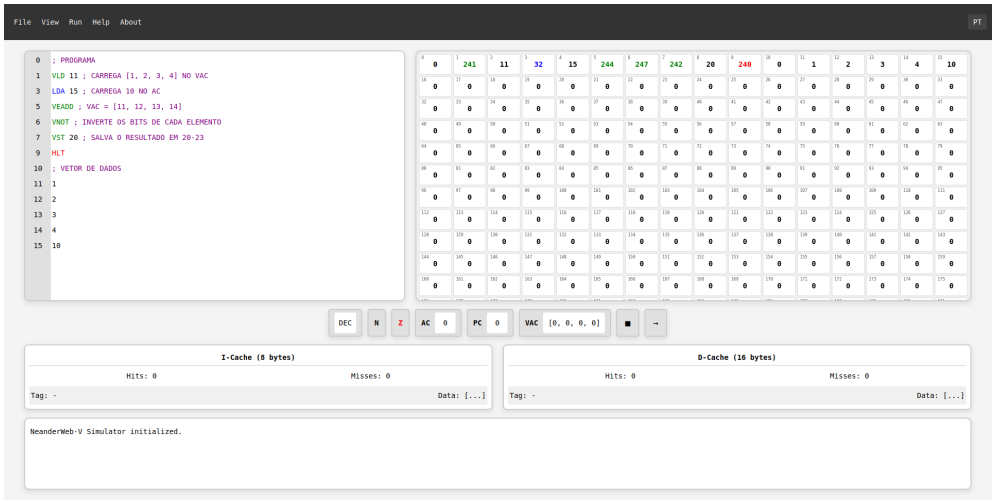

Figura 1. Interface NeanderWeb-V.

A caixa de texto à esquerda oferece o editor da memória, onde devem ser introduzidos tanto o código do programa como os dados em memória. A numeração apresentada corresponde ao endereço de memória inicial correspondendo à linha: desta forma, uma instrução com dois operandos, por exemplo `ADD 130`, ocupa dois bytes e um dado ou uma instrução sem parâmetros, por exemplo `42` e `HLT`, ocupam apenas um byte. O editor oferece realce de sintaxe, diferenciando instruções, operandos e comentários. No entanto, não há etapa de montagem, caso a execução seja lançada e exista uma instrução mal formada, irá ocorrer um erro de execução.

A janela à direita, apresenta os 256 bytes de memória, em uma matriz 16 x 16, permitindo a inspeção de dados antes e após a execução do programa. Esta janela não permite edição, sendo todas as alterações nela feitas no editor da esquerda.

Ao ativar o modo de execução passo a passo, são apresentados dois botões ao usuário presentes na imagem, ao lado do registrador vetorial. O botão de seta (→) avança a execução de uma instrução, já o outro botão (■), interrompe a execução do programa.

4.3. Mecanismo de Execução

Toda a lógica de simulação foi implementada em JavaScript. A implementação permite uma execução contínua, onde o programa executa até encontrar a instrução HLT ou um erro. Ainda que as alterações na memória, registradores e flag sejam apresentadas na interface, a velocidade da execução padrão não permite acompanhar e relacioná-las às instruções executadas. Por conta disso, foi implementada a execução passo a passo, o que permite avançar uma instrução por vez, destacando a instrução atual e permitindo acompanhar o efeito da execução das instruções nos componentes do sistema.

4.4. Verificação dos Requisitos

A implementação do NeanderWeb-V cumpre integralmente os requisitos funcionais e não funcionais propostos. A interface permite a escrita, execução e depuração de programas (RF1), com suporte total às instruções escalares e vetoriais descritas na Tabela 2 (RF2). A simulação de caches foi implementada com controle de *hits* e *misses*, refletindo o comportamento de acesso à memória (RF3). A visualização em tempo real dos registradores,

Neander-V (inversão de sinal)

```
0  VLD 80 ; VAC[0..3] <- MEM[80..83]
1  VNOT   ; VAC[0..3] <- not(VAC[0..3])
2  LDA 84 ; AC <- MEM[84] (contém 1)
3  VEADD  ; VAC[0..3] <- VAC[0..3] + AC
4  VST 80 ; MEM[80..83] <- VAC[0..3]
5  HLT
```

Descrição da Execução

O programa utiliza instruções vetoriais e escalares para inverter o sinal de todos os elementos de um vetor, aplicando a operação de complemento de 2. (*i*) VLD 0x80: O vetor armazenado nos endereços 0x80 a 0x83 é carregado para o registrador vetorial VAC. (*ii*) VNOT: Cada elemento do vetor é invertido bit a bit (complemento de 1). (*iii*) LDA 0x84: O valor escalar 1, armazenado no endereço 0x84, é carregado no acumulador AC. (*iv*) VEADD: O valor 1 do acumulador é somado a todos os elementos de VAC, completando a operação de complemento de 2 e, assim, invertendo o sinal de cada elemento do vetor. (*v*) VST 0x80: O vetor resultante, com sinais invertidos, é gravado de volta na memória a partir do endereço 0x80. (*vi*) HLT: O programa finaliza a execução.

Figura 2. Complemento de 2 de um vetor em Neander-V.

memória e caches, bem como o suporte a execução passo a passo, confirmam o atendimento aos RF4 e RF5.

Do ponto de vista não funcional, o uso exclusivo de tecnologias web garante a portabilidade e acessibilidade sem instalação (RNF1). A interface apresenta realce de sintaxe, mensagens de erro claras e opção de idioma, favorecendo a usabilidade (RNF2). O simulador é funcional em diferentes resoluções de tela e dispositivos móveis, embora com limitações esperadas em conforto de uso em telas pequenas (RNF3). Por fim, a implementação do simulador foi realizada de forma a permitir que novos recursos da arquitetura sejam facilmente adicionados (RNF4).

5. Estudos de Caso

Para demonstrar a aplicabilidade e as vantagens da arquitetura estendida, esta seção apresenta exemplos práticos de programas em Neander-V e uma análise quantitativa do impacto da vetorização. A seção também inclui uma avaliação preliminar realizada em uma turma de período inicial.

5.1. Programação em Neander-V

Exemplos de programas em Neander-V são apresentados nas Figuras 2 e 3. O primeiro código, Figura 2, inverte o sinal de todos os valores de um vetor de 4 elementos. O vetor está inicialmente armazenado em 4 posições de memória consecutivas a partir da posição de memória 0x80 e o valor inteiro 1 (um) encontra-se armazenado na posição 0x84. A inversão de sinal se faz por complemento de 2.

Os códigos na Figura 3 comparam a implementação da soma de um valor escalar a todos elementos de um vetor de 4 elementos em Neander (à esquerda) e em Neander-V (à direita). Nos dois casos, o vetor se encontra armazenado à partir da posição 0x80 e o valor a ser somado na posição 0x84. Na versão Neander, cada bloco de três instruções realiza a carga de uma das posições do vetor em AC, então opera a soma de AC com o valor a ser adicionado a cada posição, findando por armazenar na memória o resultado em AC. A versão Neander carrega o valor a ser somado em AC, carrega o vetor em VAC, processando então a soma e armazenando o resultado da soma na memória.

5.2. Análise Quantitativa

Além da clareza e redução do código-fonte, a principal motivação para o uso de instruções vetoriais é o ganho de desempenho. A arquitetura Neander-V, embora mantenha as características didáticas da arquitetura Neander original, permite demonstrar essa vantagem

Neander (soma escalar)

```
0  LDA 80 ; AC <- MEM[80]
1  ADD 84 ; AC <- AC + MEM[84]
2  STA 80 ; MEM[80] <- AC
3  LDA 81 ; AC <- MEM[81]
4  ADD 84 ; AC <- AC + MEM[84]
5  STA 81 ; MEM[81] <- AC
6  LDA 82 ; AC <- MEM[82]
7  ADD 84 ; AC <- AC + MEM[84]
8  STA 82 ; MEM[82] <- AC
9  LDA 83 ; AC <- MEM[83]
10 ADD 84 ; AC <- AC + MEM[84]
11 STA 83 ; MEM[83] <- AC
12 HLT
```

Neander-V (soma escalar vetorial)

```
0  VLD 80 ; VAC[0..3] <- MEM[80..83]
1  LDA 84 ; AC <- MEM[84]
2  VEADD  ; VAC[0..3] <- VAC[0..3] + AC
3  VST 80 ; MEM[80..83] <- VAC[0..3]
4  HLT
```

Figura 3. Código escalar e vetorial para soma de um escalar a um vetor.

Tabela 3. Acessos à memória e impacto no cache.

Programa	Acessos Totais	I-Cache		D-Cache (Alinhado)		D-Cache (Não alinhado)	
		Hits	Misses	Hits	Misses	Hits	Misses
Neander	37	12	1	23	1	20	4
Neander-V	11	7	1	2	1	3	2

de forma mensurável em linhas de código. Tendo como base a soma de um escalar a um vetor (Figura 3), o código Neander possui 13 linhas, enquanto o Neander-V possui apenas 5. Considerando que o incremento do tamanho do vetor implica, em ambos os casos, na alteração do programa, o número de linhas pode ser usado como métrica para estimar a complexidade do programa em termos de $O(n)$, com n representando o tamanho do vetor. No caso do Neander, a complexidade é linear, dada por $T_{\text{Neander}}(n) = 3n + 1 \Rightarrow O(n)$, enquanto no Neander-V, é sublinear, dada por $T_{\text{Neander-V}}(n) = 3 \cdot \left\lceil \frac{n}{k} \right\rceil + c \Rightarrow O(n)$, o que resulta em $T_{\text{Neander}}(n) \geq T_{\text{Neander-V}}(n)$. Nesta expressão, k representa o fator de vetorização da arquitetura, sendo isso o número de elementos do vetor que podem ser processados simultaneamente por uma única instrução, enquanto c corresponde a um custo fixo (overhead) de inicialização, que não depende do tamanho n do vetor.

Estes mesmos códigos permitem analisar os acessos à memória, contando tanto a busca de instruções quanto a de dados. Na Tabela 3 são apresentados dados correspondendo aos números de acesso totais à memória e número de hits e misses no cache resultante das execução dos programas escritos com código Neander e Neander-V. Para cada programa, foram executadas duas versões, uma na qual o dado escalar a ser somado no vetor e o próprio vetor encontram-se armazenados na mesma linha do cache (vetor iniciando em 0x80, dado a ser somado na posição 0x84), outra na qual há desalinhamento no dado a ser somado, estando este armazenado na posição de memória 0x7F. Estas alteração no endereço do dado a ser somado impacta no uso cache, havendo mais misses, no entanto, não há alteração no comportamento da I-Cache.

5.3. Avaliação da Usabilidade

A versão do simulador apresentada neste artigo foi apresentada aos estudantes de primeiro semestre em Engenharia de Computação na disciplina de Introdução à Engenharia de Computação, no semestre 2025/1, de nossa instituição. A turma já havia tido duas aulas

teóricas, sobre a arquitetura Neander e seu simulador.

A atividade com o Neander-V Web foi realizada em 60 minutos, em laboratório. 28 estudantes estavam presentes, dispostos livremente sobre 25 computadores, formando alguns pares de estudantes. O acesso ao simulador se deu via link web e os programas apresentados na Figura 3 foram previamente salvos no armazenamento local.

Os estudantes foram apresentados à ferramenta e então orientados a executar uma série de ações, envolvendo carregar os programas e executá-los. Ao final, os usuários preencheram um formulário de apreciação da ferramenta. Este formulário foi concebido com sete afirmações nas quais os usuários deveriam classificar, em uma de cinco escalas, de "Não concordo" a "Concordo", sua posição em relação a afirmação. As questões apresentadas e uma análise sumarizando as respostas encontram-se na Tabela 4. O formulário recebeu 21 respostas.

As avaliações com viés negativo são as relacionadas às afirmações 4 e 7, em que os usuários indicaram necessitar ajuda técnica e aprender muitas coisas antes de usar o simulador. No entanto, as respostas às questões 3 e 6 indicam que os usuários não tiveram dificuldades de utilizar o simulador e que acreditam que outras pessoas teriam a mesma facilidade. Como conclusão, foi levantada uma hipótese de que a complexidade do uso do simulador foi associada a necessidade de conhecer a arquitetura Neander-V para efetivamente aproveitar os recursos da ferramenta.

6. Limitações e Trabalhos Futuros

Como decisão de projeto para a arquitetura Neander-V, assumiu-se como princípio seguir a concepção de fornecer uma arquitetura minimalista, voltada à prática didática introdutória de Arquitetura e Organização de Computadores e Programação. Como consequência, o NeanderWeb-V enquanto simulador de arquitetura, apresenta as mesmas limitações do Neander original inerentes à sua capacidade de memória. Assim, o incremento de funcionalidades, como maior número de registradores vetoriais ou número mais elevado de instruções, vetoriais ou mesmo escalares, não se torna compatível com a proposta didática e introdutória da ferramenta. Desta forma, as limitações arquiteturais da arquitetura Neander-V não são consideradas como limitações deste trabalho.

Como limitações da ferramenta construída, considera-se seu desenvolvimento para ambiente web e seu potencial para compreensão do uso dos recursos paralelos disponíveis nas atuais arquiteturas de processadores. Ainda que as ferramentas como CSS, JavaScript e o próprio HTML ofereçam vários recursos para configurar a interface, em

Tabela 4. Resumo das respostas dos usuários ao questionário de avaliação.

Afirmação	Concordam (≥ 4)
1. Eu usaria esse simulador com frequência.	62%
2. Eu achei o simulador desnecessariamente complexo.	5%
3. Eu achei o simulador fácil de usar.	86%
4. Eu acho que precisaria de ajuda técnica para conseguir usar o simulador.	14%
5. Eu achei que as funcionalidades do simulador estão bem integradas.	86%
6. Eu imagino que a maioria das pessoas aprenderia a usar o simulador rapidamente.	76%
7. Eu precisei aprender muitas coisas antes de conseguir usar o simulador.	14%

função das configurações de resolução de apresentação da tela e do navegador, é possível que o usuário tenha que ajustar o zoom da sua janela do navegador manualmente. Também é possível utilizar a ferramenta por meio de dispositivos móveis, ainda que o conforto de utilização possa ser prejudicado pelas dimensões da tela.

No estudo de caso realizado, ainda que com uma amostra pequena, os usuários teste relataram satisfação em ter acesso ao simulador via interface web e identificaram aspectos de interface que devem ser considerados na primeira versão pública. Entre os pontos estão: (*i*) o desconforto de ser necessário paginar toda a memória para alcançar uma posição distante; (*ii*) ao inserir uma nova linha de instrução, toda a memória é deslocada o número de posições correspondentes, incluindo a área de dados; e (*iii*) a janela de visualização da memória (à esquerda) deve ser apresentada em tom acinzentado, e não branco, para dar a ideia de ser um campo não editável. Como trabalhos futuros, além de incluir melhorias na interface do simulador, a inclusão de alguns aspectos associados à arquitetura estão sendo considerados. Como prioritário, a visualização do pipeline de execução de instruções. Também está sendo avaliado estender a capacidade de memória da arquitetura, bem como introduzir novos componentes, como novos registradores, inclusive um segundo registrador vetorial, e outros modos de endereçamento, ampliando a capacidade de expressão da arquitetura. Em particular, devendo ser introduzidas instruções de laço, de acesso indireto à memória, de novas instruções vetoriais, e de suporte à chamada de sub-rotinas, via instruções do tipo CALL/RET, com suporte de um registrador de pilha. No entanto, para manter a concepção de dispor de uma ferramenta didática em nível introdutório, a continuidade da construção do simulador deve prever a habilitação de camadas da arquitetura, permitindo apresentação incremental dos conceitos associados e adequação do uso do simulador por diferentes disciplinas.

Por fim, projeta-se desenvolver uma versão desta arquitetura para um ambiente multiprocessado (multicore). Nesta etapa, um sistema dedicado à gestão de threads deverá ser incorporado, permitindo a edição da estratégia de escalonamento adotada.

7. Conclusão

Este artigo apresentou o NeanderWeb-V, um simulador web que estende a arquitetura Neander com suporte a vetorização (SIMD) e simulação de cache. A ferramenta foi concebida para apoio ao ensino introdutório de Arquitetura de Computadores e Programação, ao integrar conceitos fundamentais e avançados em um ambiente acessível e interativo.

O simulador permite que estudantes experimentem diretamente no navegador, eliminando barreiras técnicas e facilitando a compreensão prática da execução sequencial, paralelismo de dados e hierarquia de memória. Os ganhos de desempenho observados com as extensões vetoriais reforçam seu valor pedagógico.

Como trabalhos futuros, pretende-se incluir a visualização do pipeline de execução, como forma de representar explicitamente o paralelismo do hardware. Também está prevista a introdução de suporte a laços e sub-rotinas, o que deverá demandar a expansão da memória e, possivelmente, a criação de novas instruções. Para manter a proposta didática e minimalista da arquitetura original, considera-se definir camadas de instruções a serem habilitadas conforme os objetivos de ensino. Além dessas extensões arquiteturais, será realizada uma nova avaliação com estudantes, a fim de aferir de forma mais sistemática a usabilidade da ferramenta.

Referências

Borges, J. A. S. and Silva, G. P. (2006). NeanderWin: Um simulador didático para uma arquitetura do tipo acumulador. In *Workshop sobre Educação em Arquitetura de Computadores*, volume 1.

Branovic, I., Giorgi, R., and Martinelli, E. (2004). WebMIPS: A new web-based MIPS simulation environment for computer architecture education. In *Proc. of the 2004 Workshop on Computer Architecture Education*, pages 19–es.

Djordjevic, J., Nikolic, B., and Milenkovic, A. (2005). Flexible web-based educational system for teaching computer architecture and organization. *IEEE Transactions on Education*, 48(2):264–273.

García, F., Calderón-Mateos, A., Alonso-Monsalve, S., and Prieto-Cepeda, J. (2019). WepSIM: An online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming. *IEEE Transactions on Learning Technologies*, 13(1):211–218.

Linares, J. P. and Cavalheiro, G. G. H. (2025). Simulador da arquitetura neander em uma abordagem web. In *Escola Regional de Alto Desempenho da Região Sul*, pages 113–116. SBC.

Nova, B., Ferreira, J. C., and Araújo, A. (2013). Tool to support computer architecture teaching and learning. In *2013 1st Int. Conf. Port. Soc. Eng. Educ.*, pages 1–8. IEEE.

Patti, D., Spadaccini, A., Palesi, M., Fazzino, F., and Catania, V. (2012). Supporting undergraduate computer architecture students using a visual mips64 cpu simulator. *IEEE Transactions on Education*, 55(3):406–411.

Silva, G. P. and Borges, J. A. d. S. (2016). SimuS: Um simulador para o ensino de arquitetura de computadores. *Int. J. Comput. Archit. Educ.*, 5(1).

Skillen, N., Manickam, V., and Aravind, A. (2011). Ease: an extensible architecture simulation engine. In *Proc. of the 16th Western Canadian Conf. on Comput. Educ.*, pages 23–27.

Skrien, D. (2001). CPU Sim 3.1: A tool for simulating computer architectures for computer organization classes. *Journal on Educational Resources in Computing*, 1(4):46–59.

Vollmar, K., Sanderson, and Pete (2006). MARS: An education-oriented MIPS assembly language simulator. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, page 239–243, New York. ACM.

Weber, R. F. (2000). *Fundamentos de arquitetura de computadores*. Sagra Luzzatto.

Yehezkel, C., Eliahu, M., and Ronen, M. (2009). Easy CPU: Simulation-based learning of computer architecture at the introductory level. *IJEE*, 25(2):228–238.

Yurcik, W., Brumbaugh, and Larry (2001). A web-based little man computer simulator. In *Proc. of the 32nd Tech. Symp. on Computer Science Education*, pages 204–208.