# Desenvolvendo Ferramentas para Ensino de RISC-V com Python, Verilog, Matplotlib, SVG e ChatGPT

Guilherme A. R. de Figueiredo, Erick S. de Souza e Júlio H. F. Rodrigues, José A. Nacif , Ricardo Ferreira
Universidade Federal de Viçosa, Brasil
ricardo@ufv.br

***Resumo*—O uso de simuladores é um facilitador para o ensino de arquitetura de computadores. Porém, o desenvolvimento de simuladores e ferramentas de visualização proporciona uma visão mais aprofundada aos estudantes. Este trabalho apresenta diversas ferramentas para o ensino de projeto de processadores RISC-V. Para motivar os estudantes, utilizamos o ambiente Google Colab com a linguagem Python no desenvolvimento das ferramentas para implementações de alto nível, juntamente com a linguagem Verilog para implementações mais detalhadas do RISC-V. São apresentados exemplos de interfaces interativas com visualizações gráficas utilizando a biblioteca Matplotlib e o formato vetorial SVG. Devido à popularidade do Python, o desenvolvimento é facilitado com o apoio de modelos de linguagem como ChatGPT para geração parcial do código de interface e visualização. O trabalho foi desenvolvido em conjunto com os alunos da disciplina de Arquitetura de Computadores da Universidade Federal de Viçosa. Todas as ferramentas são abertas com o propósito de serem continuadas, servindo de base para o desenvolvimento de exercícios, exemplos e mais ferramentas para ensino.**

***Index Terms*—Google Colab, Software Pipelining, Escalonamento, GPU**

## I. Introdução

A maioria dos simuladores é desenvolvida no modelo "Catedral" [24] de forma centralizada e controlada por poucos indivíduos. Um desenvolvimento no modelo "Bazar", de forma descentralizada e colaborativa, pode envolver uma comunidade ampla, sendo o maior caso de sucesso o sistema Linux e os softwares livres [24]. O uso de software livre em arquitetura de computadores não é novidade [12], [13]. Com a popularidade do Python, dos Jupyter Notebooks e dos ambientes colaborativos como Google Colab, este trabalho propõe o desenvolvimento de ferramentas de simulação para o ensino dos processadores RISC-V. A reutilização de técnicas em diferentes contextos é um princípio fundamental no modelo "Bazar", que será explorado neste trabalho.

Quando os estudantes usam simuladores prontos, seria análogo a "ganhar o peixe", pois muitas vezes podem apenas aprender superficialmente, seguindo as etapas predefinidas do simulador, sem necessariamente entender os detalhes importantes para fixar o conhecimento. Essa abordagem é rápida e conveniente, mas não promove a compreensão profunda. Criar pequenos simuladores é como "ensinar alguém a pescar": os alunos precisam entender os conceitos, projetar e implementar, o que envolve uma compreensão mais profunda e permite que eles apliquem o conhecimento em diferentes contextos.

Outro ponto é usar metodologias de aprender por exemplos e aprender fazendo [25], que também é um dos pilares da metodologia "Bazar" [24], onde os ambientes colaborativos como Google Colab são bem apropriados. Quando um estudante modifica e testa seu código derivado de exemplos anteriores para criar novos recursos, essas ações proporcionam um reforço com retorno rápido para ele mesmo [10]. Um software é um meio dinâmico, um texto ou artigo científico não é [28]. A criação dos *notebook* computacionais, como o *Jupyter Notebook* com a biblioteca Ipython [22], promoveu a fusão dos conceitos de programação e documentação, tornando-se uma base popular para o uso e desenvolvimento de ferramentas de ensino e pesquisa.

Neste trabalho, nosso foco é aplicar as metodologias colaborativas de desenvolvimento no ensino do RISC-V [20], que é uma arquitetura de conjunto de instruções (ISA) de código aberto baseada em princípios estabelecidos de computadores com conjunto de instruções reduzido (RISC). A natureza aberta do RISC-V permite que instituições educacionais ofereçam experiências práticas em projeto e implementação de processadores, onde a compreensão da ISA e da arquitetura do processador são fundamentais. Além disso, a simplicidade do RISC-V facilita o projeto de circuito, reduz a complexidade e permite otimizações mais diretas de interações entre hardware e software.

Como existem vários trabalhos de simuladores já desenvolvidos para o tema [14], [19], [21], [23], [30], este trabalho propõe uma metodologia incremental para introduzir o uso do Google Colab no ensino de RISC-V, buscando preencher algumas lacunas no ensino para complementar o desenvolvimento de competências essenciais em arquitetura de computadores. Este trabalho complementa trabalhos anteriores que exploram o uso do Google Colab no ensino de Arquitetura e Ciência da Computação [4], [7], [9], [11].

As principais contribuições deste trabalho em relação aos anteriores são: saídas gráficas, uso de ChatGPT, exercícios específicos com RISC-V e modelagem em Python de um simulador de RISC-V com predição de desvio dinâmica. O material foi produzido em colaboração com os estudantes da disciplina de Arquitetura de Computadores da Universidade Federal de Viçosa.

Este artigo está organizado da seguinte forma: todas as seções incluem links para os Google Colab que podem ser estendidos de forma colaborativa. O objetivo não é fazer um simulador fechado, mas desenvolver exemplos que possam ser

melhorados, adaptados, estendidos ou até mesmo remodelados para outras funcionalidades. Assim, professores e estudantes colaboradores podem agregar mais recursos enquanto realizam atividades de ensino.

A Seção II ilustra uma ferramenta simples para exercícios de codificação em binário/hexadecimal das instruções Assembly RISC-V. Os exemplos demonstram como usar recursos do Ipython [22] para entrada e saída, incluindo saídas gráficas com animações. A Seção III apresenta uma ferramenta desenvolvida em Python para simular a execução de um preditor de desvios dinâmico incorporado em uma implementação em pipeline do RISC-V. O foco da ferramenta são os dois primeiros estágios do pipeline: busca e decodificação. A Seção IV ilustra a adição de uma interface visual com Matplotlib para uma implementação em Verilog do RISC-V mono-ciclo que segue a descrição do livro-texto de David Patterson e John Hennessy [20]. Na Seção V, adaptamos dois exemplos de simuladores de RISC-V em Verilog para incluir uma figura editável no formato vetorial SVG do livro-texto [20]. Por fim, as Seções VI e VII apresentam trabalhos relacionados complementares, as principais conclusões e sugestões de trabalhos futuros.

## II. Codificação Assembly

Neste seção iremos ilustrar vários exemplos de codificação, para mostrar passo a passo, os diversos recursos que podem ser utilizados no desenvolvimento de simuladores.

A etapa de codificação é basicamente fazer um montador assembler. O estudante precisa compreender o formato e consultar a documentação para incluir novas instruções RISC-V no seu montador.

### A. Codificador Simples

```
1:#@title Conversor Add rd,rs1,rs2 para Hex
2: f7 = "0000000"
3: f3 = "000"
4: opcode = "0110011" # add R-type f7 e f3

5:def rtype_add(Rd,Rs1,Rs2):
6:    clear_output()  # Limpa a saída anterior
7:    inst =  f7+Rs2+Rs1+f3+Rd+opcode
8:    display(Markdown("## "+bin2hex(inst)))

# Criação das Entrdas
9:  Rd = widgets.Text(description='rd:', value
   ='00000')
10: Rs1 = widgets.Text(description='rs1:',
   value='00000')
11: Rs2 = widgets.Text(description='rs2:',
   value='00000')

# Vincula a função rtype_add ao interact
12: interact(rtype_add, Rd=Rd,Rs1=Rs1,Rs2=Rs2)
```

Figura 1. Código python interativo para montar a instrução ADD

Suponha o formato tipo R do RISC-V com instruções com três operandos. Para facilitar, suponha a instrução ADD. O primeiro exemplo, ilustrado na Figura 1, faz a leitura dos valores de três registradores (em binário) para montar a instrução ADD em hexadecimal. As linhas 9-11 fazem a interface para ler os valores dos três registradores, onde $R_d = R_{s1} + R_{s2}$. A linha 7 mostra a parte mais importante que é a montagem dos campos da instrução, apenas concatenando as variáveis que representam cada um deles. As linhas 2-4 mostram a definição do ADD com *opcode* 0x33 e os campos $f_7$ e $f_3$ com o valor 0.

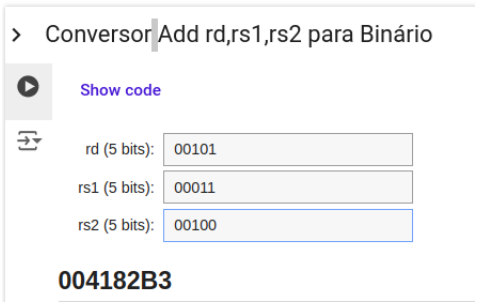



Figura 2. Interface gráfica simples para Codificação do ADD.

A interface é dinâmica com o método *interact*, ao alterarmos o valor, automaticamente o código é re-executado e mostra a nova saída como ilustrado na Figura 2.

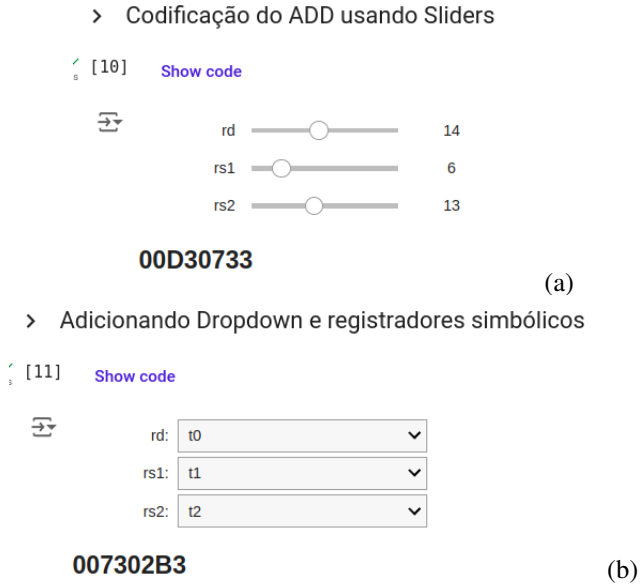
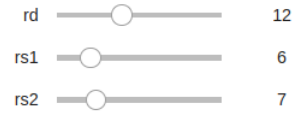



Figura 3. (a) Interface com sliders gerada pela ChatGPT; (b) Dropdown com Registradores Simbólicos.

Podemos agora ilustrar o uso da chatGPT para modificar a interface. Com o prompt "Modifique o código para a entrada usar sliders de 0 a 31 para entrar com os valores dos registradores", temos uma nova interface ilustrada na Figura 3(a). Podemos também adicionar uma opção com lista de valores (interface *dropdown*) e solicitar para fazer a conversão com os nomes dos registradores simbólicos do RISC-V 32 que incluem gp,sp, $t_1, t_2, \ldots$. Fizemos uso do seguinte prompt na chatGPT: "Adaptar o código para usar botões dropdown, considerando a codificação do processador RISC-V onde os registros são sp, gp, t1, t2, etc...fazer a correspondência com

os valores de 0 a 31.”. Como o RISC-V é bem conhecido e documentado, o código foi gerado corretamente. Mas deve ser conferido pelo usuário, uma vez que se trata de um modelo generativo. A chatGPT fez uso de um dicionário gerado automaticamente

```
register_map = {
    'zero': 0, 'ra': 1, 'sp': 2,
    ..., 't4': 29, 't5': 30, 't6': 31
}
```




**Exemplo para instrução add x12,x6,x7 com opcode 0x33**

| Func7 | Rs2 | Rs1 | Func3 | Rd | Opcode | Op Rd,Rs1,Rs2 |
|---|---|---|---|---|---|---|
| 0 | 7 | 6 | 0 | 12 | 0x33 | add x12,x6,x7 |
| 0000000 | 00111 | 00110 | 000 | 01100 | 0110011 | Em binário |
| 31..25 | 24..20 | 19..15 | 14..12 | 11..7 | 6..0 | bits |
| 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | Tamanho dos campos |

**Código hexadecimal: 00730633**

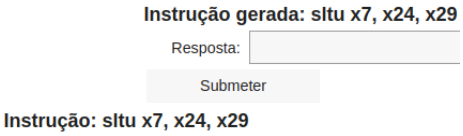
Figura 4. Saída no formato de tabela para codificação do ADD.

O próximo passo é ilustrar exemplos de como melhorar a apresentação dos dados com saída em formato de tabela, como ilustrado na Figura 4. Fizemos um exemplo com código estático e solicitamos a chatGPT para auxiliar na geração da interface de entrada e a parametrização do exemplo. O nosso código estático funciona para **add x5,x6,x7**. O prompt para chatgpt foi “dado este exemplo de código estático, criar uma função com sliders para os valores dos registradores que gerar a tabela para estes valores e também o código em hexadecimal da instrução com 8 dígitos.”. O código está ilustrado no repositório público deste projeto [6], iremos omitir os detalhes de código no texto para simplificar as explicações, mas a documentação está disponível no Google Colab.

Além do formato tipo R, os *notebook* disponíveis no repositório deste trabalho, ilustram vários exemplos com os outros formatos e outras atividades que podem ser estendidas, desde criação passo a passo com a conferência com biblioteca já desenvolvidas em Python [5]. Uma atividade pode ser a conversão no sentido inverso, onde o estudante digita o binário ou hexadecimal e a ferramenta deve exibir o mnemônico assembly. Este exemplo serve de base para várias extensões de codificação, incluindo o uso de ferramentas em Python que já fazem o ”dissassembler” [32].

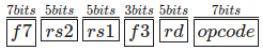
### *B. Gerador de Exercícios*

Podemos adaptar a ferramenta interativa de codificação para ser um gerador de exercícios. Cada vez que o estudante executar a célula terá um exemplo de uma instrução do tipo R, ele terá que codificar em hexadecimal e depois irá visualizar a resposta com a correção. O exemplo de código que mostramos




(a)

**Instrução: sltu x7, x24, x29**




| Func7 | Rs2 | Rs1 | Func3 | Rd | Opcode | Op Rd,Rs1,Rs2 |
|---|---|---|---|---|---|---|
| 0 | 29 | 24 | 0 | 7 | 0x33 | sltu x7,x24,x29 |
| 0000000 | 11101 | 11000 | 011 | 00111 | 0110011 | Em binário |
| 31..25 | 24..20 | 19..15 | 14..12 | 11..7 | 6..0 | bits |
| 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | Tamanho dos campos |

**Incorreto. A resposta correta é: 0x01DC33B3**

**Código binário correto: 0000 0001 1101 1100 0011 0011 1011 0011**

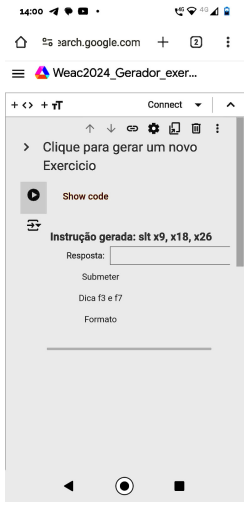
**Instrução digitada pelo usuário: 543456**

(b)

Figura 5. (a) Gerador de Exercício do tipo R; (b) Correção automática do exercício.

foi gerado com auxílio da chatGPT para adicionar uma opção aleatória para escolher valores para os registradores $r_d, r_{s1}, r_{s2}$ e escolher entre 10 exemplos de instruções do tipo R. As instruções são: *add*, *sub*, *sll*, *slt*, *sltu*, *xor*, *srl*, *sra*, *or* e *and*. A Figura 5(a) mostra a instrução sorteada e o campo para entrar com a resposta e a Figura 5(b) mostra a resposta com a correção do exercício.

(a)




(b)

Figura 6. (a) QR-code para o Google Colab do Gerador do Exercício do Tipo R para responder com a codificação hexadecimal; (b) Execução no Celular.

Os estudantes podem usar o gerador para treinar suas habilidades de codificação, assim como podem fazer a extensão do gerador para incorporar um conjunto maior de instruções do RISC-V. Outra vantagem que os geradores podem ser apresentados em *jupyter notebooks* minimalistas e com o uso de um QR code, os estudantes podem testar suas habilidades usando o celular. A Figura 6(a) ilustra o QR-code para acesso ao exercício do gerador das instruções do tipo R e a Figura 6(b) mostra a tela do *Google Colab* para execução no celular. Neste

gerador foram adicionados mais dois botões: dica e formato, para mostrar os campos f3-f7 e a ordem da divisão em campos do formato R, respectivamente. A ideia é ter níveis de ajuda antes de submeter a resposta.

## III. Preditor de Desvio Dinâmico

Esta seção apresenta um simulador de preditor de desvio dinâmico para o RISC-V. O simulador foi desenvolvido como trabalho prático da disciplina de Arquitetura de Computadores. A implementação foi realizada em Python com o principal objetivo de facilitar o rastreio de execução de um pequeno trecho com operações de desvio. O estudante pode acompanhar quais as ações a unidade de predição executa passo a passo, o rastreio da tabela de execução ciclo a ciclo e o conteúdo da tabela cache que armazena quais desvios estão cadastradas, qual é a predição e o destino de cada um deles. Este preditor foi adaptado de uma proposta para o processador MIPS [3].

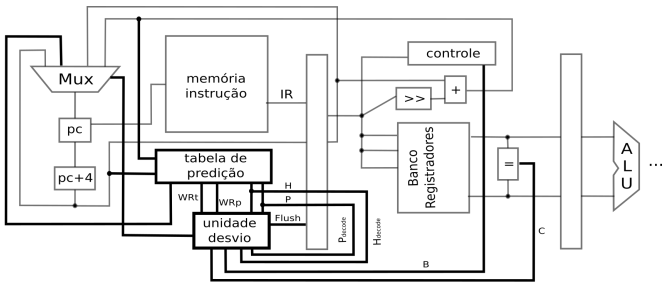



Figura 7. Caminho de Dados Simplificado com a inclusão da unidade de desvio e a tabela com as predições.

A Figura 7 mostra um esboço do caminho de dados com destaque para os estágios de busca e decodificação. O preditor é controlado pela unidade de desvio e armazena as predições na tabela no estágio de busca. Algumas ações são executadas no estágio de busca e outras no estágio de decodificação. Seis sinais são responsáveis pelo funcionamento. O sinal $H$ (ou Hit) no estágio de busca indica se a instrução atual está na tabela, ou seja, é um desvio já detectado. O sinal $P$ indica qual é a predição tomado ou não tomado (1 ou 0). Quando a instrução de desvio avança para o estágio de decodificação, estes sinais repassam seus valores para os sinas $H_d$ e $P_d$, respectivamente. No estágio de decodificação o sinal $B$ indica a presença de uma instrução de desvio que pode ser para cadastro ou para verificação (se já foi cadastrada). O sinal $C$ indica se o desvio foi tomado ou não tomado, que serve para atualizar a predição e abortar a execução, caso ocorra erro na predição.

A Tabela I resume as nove situações que podem ocorrer, sendo as três primeiras referentes ao estágio de busca e as 6 restantes ao estágio de decodificação. Para maiores explicações sobre o preditor, o leitor pode se referir à implementação MIPS descrita em [3].

O preditor simula um pequeno subconjunto de instruções do processador RISC-V, dentre elas a soma (addi e add) e quatro instruções de desvios (se igual, diferente, maior ou igual e menor), sendo facilmente expansível para suportar outras

Tabela I
Descrição das Ações da Unidade de Desvio.

| | Sinais de Entradas | | | | | | |
|---|---|---|---|---|---|---|---|
| | H | P | $H_d$ | $P_d$ | C | B | Descrição |
| 1 | 0 | X | 0 | X | X | 0 | Não cadastrada |
| 2 | 1 | 0 | 0 | X | X | 0 | Predição NT |
| 3 | 1 | 1 | 0 | X | X | 0 | Predição T |
| 4 | 0 | X | 0 | X | 0 | 1 | Cadastro NT |
| 5 | 0 | X | 0 | X | 1 | 1 | Cadastro T |
| 6 | 0 | X | 1 | 0 | 0 | 1 | Acerto Predição NT |
| 7 | 0 | X | 1 | 1 | 1 | 1 | Acerto Predição T |
| 8 | 0 | X | 1 | 0 | 1 | 1 | Erro Predição NT |
| 9 | 0 | X | 1 | 1 | 0 | 1 | Erro Predição T |

operações. Ele implementa um pipeline simples em Python com estágios de busca, decodificação, execução e escrita. Para criar novas instruções, o estudante deve fazer as respectivas alterações em cada estágio.

O estágio de busca é responsável por ler a próxima instrução a ser executada na memória de instruções. No próximo ciclo, a instrução seguirá para o estágio de decodificação, onde é determinado qual é o tipo da instrução e também para obter os valores dos registradores e imediatos. Durante o estágio de execução, a operação da instrução será realizada e, no último passo (escrita), o resultado poderá ser escrito nos registradores, completando o processamento da instrução. Importante destacar que os estágios formam uma *pipeline* e até quatro instruções podem estar ativas ocupando os quatro estágios. Uma extensão do trabalho é incorporar um estágio de memória.

Tabela II
Código de Instruções.

| Instrução | Operando 1 | Operando 2 | Operando 3 |
|---|---|---|---|
| add | 5 | 1 | 3 |
| add | 2 | 2 | 2 |
| beq | 5 | 4 | 24 |
| add | 4 | 4 | 1 |
| add | 6 | 3 | 2 |
| beq | 1 | 1 | 8 |
| addi | 5 | 4 | 1 |
| beq | 2 | 4 | 4 |
| end | | | |

Uma tabela de predição com 1 bit de predição foi utilizada tratamento do desvio. Além do bit de predição, a tabela armazena o endereço de destino e o *tag* do endereço da instrução para verificar qual instrução está armazenada em cada linha. Caso a predição esteja correta, a execução do *pipeline* continua sem interrupções. No caso contrário, apenas um ciclo é perdido e o bit de predição é atualizado. Essa tabela pode ser facilmente expandida para utilizar 2 bits de predição. O simulador necessitará apenas de pequenos ajustes na parte de atualização da predição. Os demais componentes - incluindo a interface, estágios do *pipeline* e entradas das instruções - não precisam ser modificados.

O formato de entrada para o simulador é uma lista de tuplas, onde cada tupla representa uma instrução com quatro campos, a instrução e três operandos, como representado na Tabela II.

Para a instrução **add**, a semântica $Reg[Op_1] = Reg[Op_2] + Reg[Op_3]$, onde $Reg$ é o banco de registradores. Para a instrução **addi** a semântica $Reg[Op_1] = Reg[Op_2] + Op_3$, ou seja, o $Op_3$ é uma constante imediata. Para as instruções de desvio temos a semântica $PC = Op_3$ se $Reg[Op_1] == Reg[Op_2]$ senão $PC = PC + 4$ para o **beq**. A entrada pode ser ajustada para suportar um analisador léxico e sintático que converta códigos do Assembly RISC-V para o formato de entrada. Lembrar que o conjunto de instruções inicial do simulador é pequeno para permitir expansões em trabalhos práticos. Os estudantes podem também adaptar montadores RISC-V em Python disponíveis em [5], [32], construir seu próprio montador ou apenas fazer pequenas extensões no simulador atual.

| inst | F | D | Ex | Wb | F | D | Ex | Wb | F | D | Ex | Wb | F | D | Ex | Wb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0: add r5 r1 r3 | 1 | 2 | 3 | 4 | | | | | | | | | | | | |
| 4: add r2 r2 r2 | 2 | 3 | 4 | 5 | 8 | 9 | 10 | | | | | | | | | |
| 8: beq r5 r4 24 | 3 | 4 | - | - | 9 | 10 | - | - | | | | | | | | |
| 12: add r4 r4 r1 | 4 | - | - | - | | | | | | | | | | | | |
| 16: add r6 r3 r2 | - | - | - | - | | | | | | | | | | | | |
| 20: beq r1 r1 8 | - | - | - | - | | | | | | | | | | | | |
| 24: addi r5 r4 1 | 5 | 6 | 7 | 8 | 10 | - | - | - | | | | | | | | |
| 28: beq r2 r4 4 | 6 | 7 | - | - | | | | | | | | | | | | |
| 32: end | 7 | - | - | - | | | | | | | | | | | | |

**Ciclo 10 - Situação 9. Erra predição BEQ de desviar. Atualiza a tabela. Flush.**

| Linha | Tag | Predição | Destino |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| 2 | 00 | 0 | 24 |
| 3 | 01 | 1 | 4 |

Figura 8. Simulador do Preditor de Desvio RISC-V.

Na Figura 8 é possível observar a tabela de rastreio de execução do simulador proposto. O rastreio ilustra a execução até o ciclo 10. Na lateral da tabela de rastreio temos o endereço de memória seguido das instruções do código em execução. As colunas representam os estágios do *pipeline* e na interseção com a linha é mostrado em qual ciclo a instrução está em um determinado estágio. Por exemplo, a instrução `4: add r2,r2,r2` está na linha 4 da memória de instrução e passou pelos estágios de busca (F), decodificação (D), execução (EX) e escrita (WB) nos ciclos 2, 3, 4 e 5, respectivamente. O código tem laços, e no ciclo 8, esta instrução retorna para execução. Neste exemplo, podemos observar até quatro execuções da mesma instrução em laços. O simulador vai avançando ciclo a ciclo. No exemplo da Figura 8, estamos no ciclo 10. Podemos observar que no ciclo 9, tínhamos a instrução **8: beq r5,r4,24** no estágio de busca e uma execução especulativa com o preditor de desvio sendo acionado. Portanto no ciclo seguinte, ciclo 10, a instrução **24: addi r5,r4,1** já será buscada. Ao mesmo tempo, a instrução beq está no estágio de decodificação onde a predição especulativa está sendo verificada. Neste exemplo ocorre um erro da predição de desvio, perdendo o ciclo atual e atualizando a tabela. O simulador mostra uma linha com o texto **Ciclo 10, situação 9, Erro na Predição BEQ Tomada, sendo o correto não tomado. Atualizar predição e realizar um *Flush***. Este comentário faz referência as nove possíveis ações do preditor de desvio que foram apresentadas na Tabela I.

Além disso podemos ver a memória cache que armazena a tabela de predição com quatro campos: linha, tag, predição tomada (1) e não tomado (0), destino. Neste exemplo temos duas instruções de desvio que já foram cadastradas na cache de predição, uma na linha 2 e outra na linha 3. A cache tem 4 linhas, neste exemplo. Como as instruções de 32 bits são armazenadas de 4 em 4 bytes, para cadastro na cache são descartados os 2 bits menos significativos. Portanto, a instrução Beq que está no endereço 8 ou em binário 00001000 será cadastrada na linha dois com o tag 0, pois será

| Tag | linha | ~~00~~ |
|---|---|---|
| 00000 | 10 | |

O campo predição tem o valor 0 pois a cache de predição é atualizada no ciclo 10. Como o desvio tinha sido executado especulativamente no ciclo 9 e foi abortado no ciclo 10, a predição passou a ser não tomada (0). No campo destino vemos o endereço 24 que é o destino do **beq** da linha 8.

Portanto, são visíveis todas as etapas do *pipeline* que executam em paralelo, permitindo observar diferentes operações ocupando simultaneamente distintos estágios do *pipeline*, o preenchimento progressivo da cache de predição e as ações da unidade de predição em cada ciclo, conforme o padrão apresentado na Tabela I.

O simulador foi desenvolvido para o rastreio e para o uso da tabela de predição, mas também pode ser utilizado em outros cenários de rastreamento como um gerador de exercícios com auto-correção para o ensino dos conceitos de preditores. Para a maioria dos alunos, este é um tópico que gera muitas dúvidas onde o usar e estender o simulador é uma ótima sugestão de trabalho de disciplina para fixar os conceitos de preditores.

## IV. Simulador RISC-V Monociclo com Matplotlib

Diversos projetos do processador RISC-V em Verilog estão disponíveis em repositórios públicos como o *GitHub*. Para desenvolver uma interface com Matplotlib voltada à depuração, foi utilizada uma versão base [19] que contém um subconjunto mínimo de instruções, conforme apresentado no livro texto de Patterson e Hennessy [20].

A proposta é ilustrar o uso Matplotlib para exibir um diagrama esquemático da arquitetura do processador de forma desacoplada do código do processador e integrada no ambiente Google Colab.

Vamos ilustrar esse processo usando como exemplo o trecho de código a seguir:

```
addi x5,x0,8
addi x2,x0,5
add x3,x5,x2
```

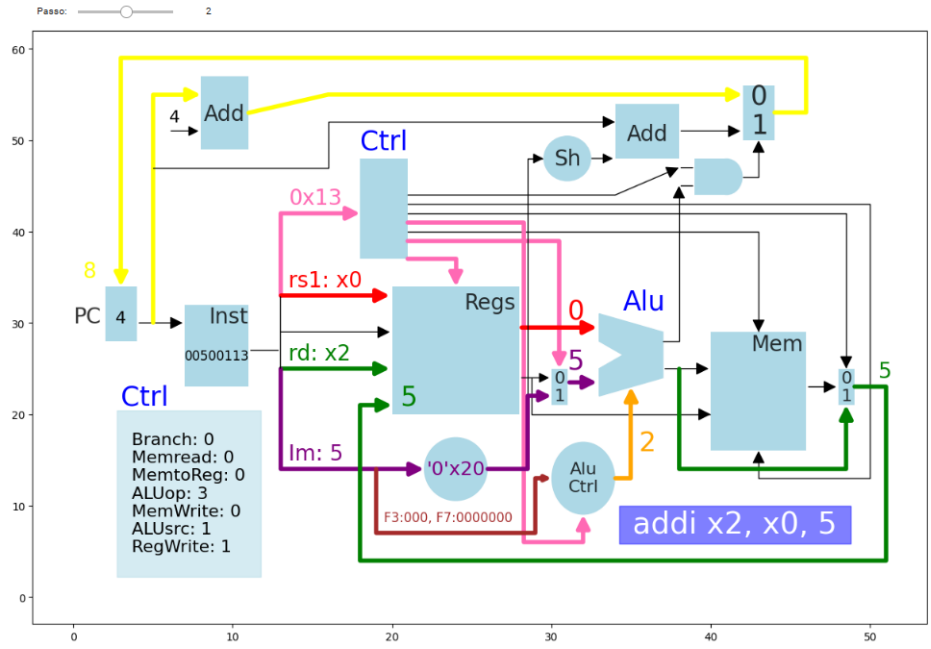



Figura 9. Exemplo do diagrama criado com Matplotlib no passo 2 da execução do código apresentado no início da Seção V.

O primeiro passo é gerar o arquivo de entrada que será utilizado pelo Matplotlib para exibir o diagrama. O simulador RISC-V em Verilog foi adaptado para registrar os dados em arquivo. Durante a simulação, o sistema realiza a leitura do código, executa as instruções e, a cada ciclo de relógio, grava em arquivo os valores dos sinais internos. Os dados gravados consistem em uma sequência de linhas contendo os valores das variáveis (sinais do processador) relevantes para a depuração no diagrama de blocos esquemático do caminho de dados.

Exemplos de sinais de interesse para entendimento do funcionamento do RISC-V são opcode, alucontrol, rs1, entre outros. Os valores dos sinais são separados por espaço e sinais não relacionadas são separados por vírgula.

Uma vez executada a simulação em Verilog, de forma desacoplada fazemos a geração da visualização. O arquivo com os valores dos sinais ciclo a ciclo é lido. A interface tem um botão deslizante (*slider*) com o número de ciclos da execução do programa, variando de 1 até o último ciclo. Isto permite que o usuário controle qual ciclo da simulação será exibido através da visualização do caminho de dados do processador. Todo o desenho do caminho de dados foi feito com a biblioteca Matplotlib. A cada alteração no *slider* para avançar ou retroceder no tempo de execução, é chamada uma função responsável por atualizar o diagrama para seu novo estado. O processador monociclo que está codificado em Verilog tem um conjunto de instruções: add, sub, sll, slt, sltu, xor, sra, srl, or, and, addi, slli, slti, sltui, xori, srai, srli, ori, andi, beq, bne, blt, bge, bltu, beeu, sw e lw.

Internamente no programa, a função que exibe cada ciclo da execução pode ser dividida em duas etapas: criação e personalização. Na etapa de criação é montado o esqueleto do diagrama, a partir de elementos do Matplotlib, como retângulos, elipses e setas. As setas são armazenadas em uma lista, para que possam ser acessadas na etapa de personalização. Nessa segunda etapa, é feita a estilização do diagrama de acordo com os valores armazenadas na linha correspondente aquele estado. São adicionados textos para cada sinal, além da alteração na cor das setas, para que elas ilustrem os caminhos que cada parte da instrução irá percorrer dentro do processador.

A figura 9 mostra a saída do diagrama para o ciclo 2 da execução do código apresentado no início dessa seção. A estrutura desacoplada do código do processador, aliado a divisão entre criação e personalização do diagrama permitem que ele seja adaptado para aceitar novas instruções ou dar enfoque em partes específicas do processador.

## V. Simuladores RISC-V com Edição Figuras SVG

Apesar da apresentação visual didática da interface Matplotlib, seu tempo de desenvolvimento pode desestimular os estudantes e professores para o desenvolvimento de novos diagramas. A Chatgpt gera código de qualidade para gráficos com Matplotlib, mas tem dificuldade de gerar desenho com diagramas de blocos. Esta seção propõem uma alternativa utilizando o formato SVG. Para manter a simplicidade, iremos aplicar uma ideia bem simples de substituição de texto.

Primeiro, o usuário faz o desenho do diagrama em um editor gráfico SVG que pode estar instalado localmente como o Inkscape [16] ou um editor online no navegador. Esta abordagem não requer nenhum conhecimento do formato SVG para gerar o desenho gráfico a partir de um código, como desenvolvido para versão matplotlib. O estudante ou professor só precisa posicionar textos com o padrão **@SINAL**, por exemplo @RS1, para escolher onde quer que seja mostrado os

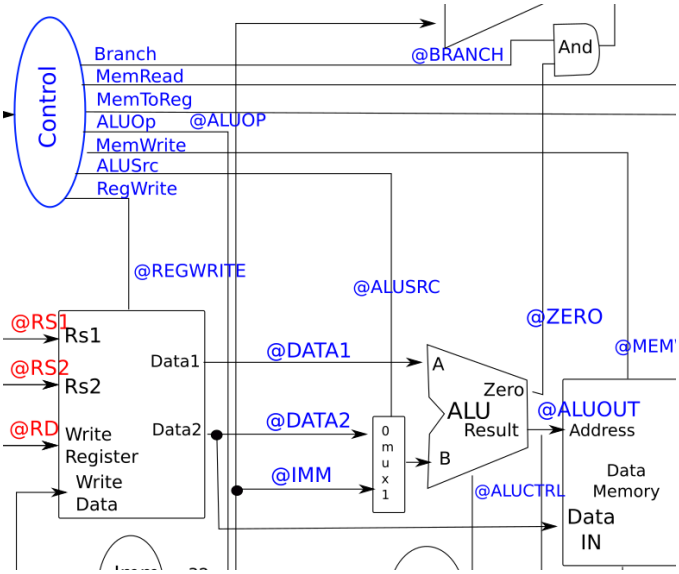



Figura 10. Exemplo de Esquemático editável em SVG com rótulos de sinais com @.

valores dos sinais internos, neste caso do campo do registrador RS1. A Figura 10 mostra um pedaço do caminho de dados do RISC-V monociclo que foi desenhado no editor Inkscape. Os sinais com "@" serão substituídos pelos valores dos sinais durante a simulação.

Para manter a simulação desacoplada da interface gráfica, o desenvolvimento seguiu duas etapas principais. A primeira consistiu na elaboração do esquemático, priorizando representações visuais similares às figuras do livro texto [20]. A segunda envolveu a adaptação do código Verilog para incluir rotinas de depuração que registram, a cada ciclo de relógio, os sinais relevantes para visualização no esquemático, como por exemplo:

```
$display("PC:%d,OPCODE:%b,WRITEDATA:%d,ZERO:%b,...,
 ALUCTRL:%d,F7:%b,F3:%b",
 CPU.Fetch.pc,opcode,CPU.writedata, CPU.zero,...,
 CPU.Alucontrol.alucontrol, f7_5, CPU.Decode.funct3);
```

Os nomes dos sinais são localizados no arquivo SVG e substituídos pelos seus respectivos valores. Os estudantes podem personalizar o sistema através de diversas modificações: adicionar novas instruções, incluir sinais para monitoramento, ajustar cores e tamanhos de fontes no arquivo SVG. Para isso, basta adaptar o código Verilog correspondente para incluir as instruções desejadas e o registro dos novos sinais. A ordem de registro dos sinais é flexível, sendo necessário apenas manter o identificador único e consistente entre o arquivo SVG e o código Verilog.

A Figura 11 mostra a simulação da terceira instrução do trecho de código a seguir:

```
addi x5,x0,8
addi x2,x0,5
add x3,x5,x2
```

onde $x_5 = 8$ na primeira instrução, depois $x_2 = 5$ na segunda instrução e a terceira que aparece na Figura 11 mostra o acesso à $x_5$ e $x_2$ no banco de registradores que tem os valores 8 e 5, cuja a soma será 13 que está na saída da ALU. O PC tem o valor 8 e o código do ALUCTRL é 2, ou seja, podemos mostrar vários detalhes. A proposta é gerar um exemplo base para que em trabalhos de disciplinas, os estudantes possam acrescentar instruções, sinais, fazer pequenos ajustes no caminho de dados e depois validar seus experimentos com a interface. A interface tem dois botões para navegar para frente e para trás no tempo, ciclo a ciclo. Como são geradas figuras no formato PNG, podemos gerar também animações em GIF e controlar o tamanho para ajuste seja na exibição em aula, celular ou laptop.

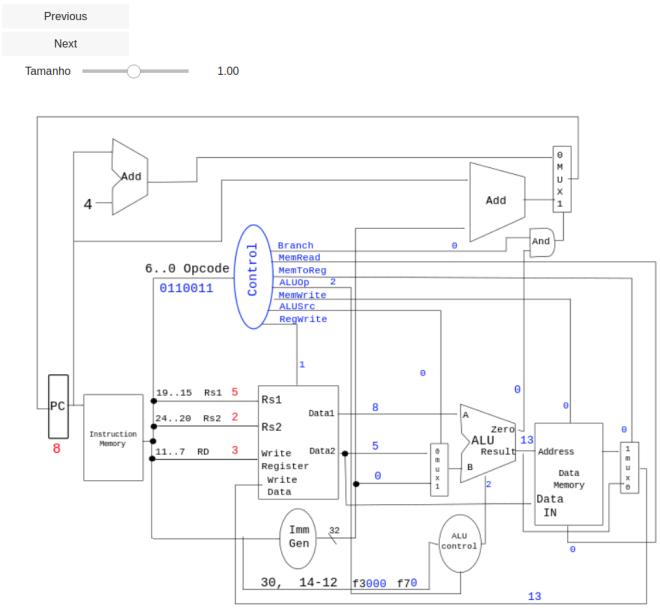



Figura 11. Exemplo do Esquemático RISC-V Monociclo simulando o **add x3,x5,x2**, onde $x_5 = 8$ e $x_2 = 5$.

Além da versão monociclo, elaboramos um exemplo com a versão Pipeline sem encaminhamento seguindo o padrão do livro texto [20]. O código Verilog está separado em seções para permitir extensões do simulador, seja na inclusão de novas instruções ou novas funcionalidades como predição de desvio, encaminhamento, branch-delay entre outras.

A Figura 12 ilustra o caminho de dados com Pipeline. Optamos por disponibilizar uma visualização mais leve, mas podemos alterar a interface para adicionar botões que possam selecionar mais de um modo de exibição para ter foco por exemplo em uma classe de instrução ou em um determinado estágio do pipeline e evitar excesso de informação.

## VI. Trabalhos Relacionados

Embora existam diversos simuladores e ferramentas online para o ensino de arquitetura de computadores [17], muitos deles não são de código aberto [8] ou não oferecem uma plataforma para extensões e personalizações simples. A utilização do Google Colab, com linguagens como Verilog ou Python, permite criar interfaces desacopladas que facilitam

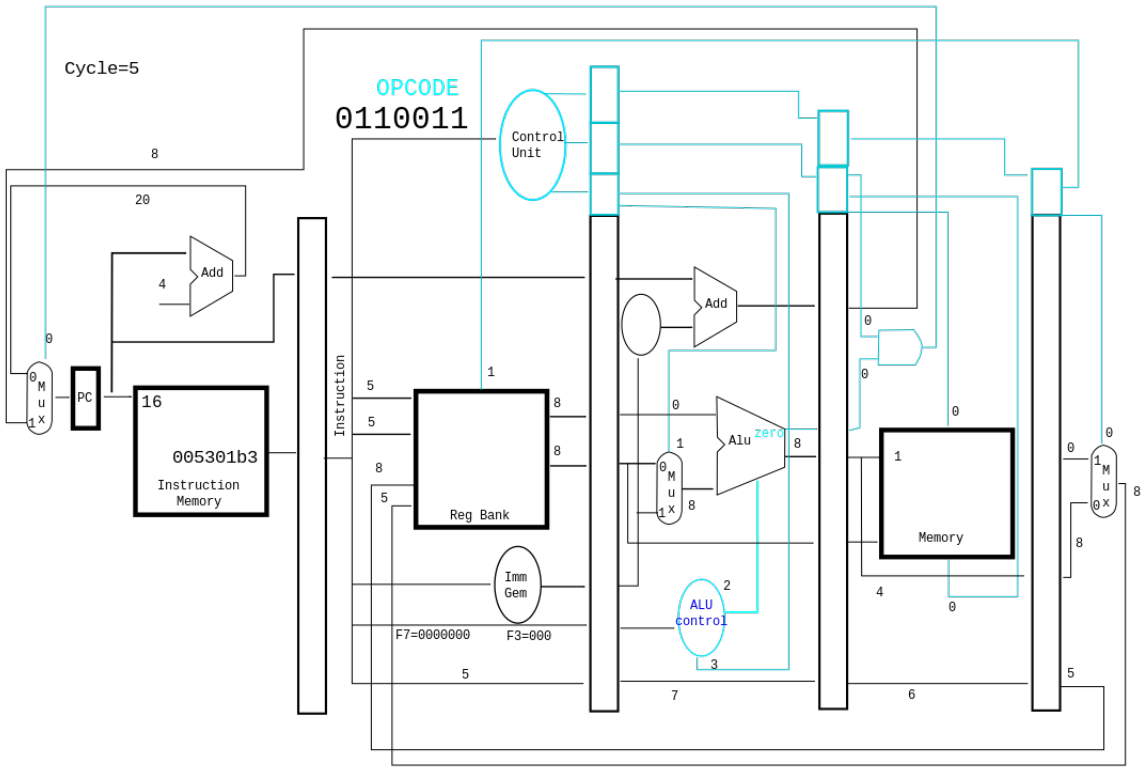



Figura 12. Exemplo do Caminho de Dados Pipeline com o Código: add x3,x6,x5 no fetch, add x5,x5,x5 no decode, addi x7,x0,8 na execução, addi x6,x0,4 na memória e addi x5,x0,8 no Write Back.

a compreensão mais aprofundada dos conceitos. Os alunos podem modelar, modificar e expandir simuladores existentes, promovendo um aprendizado mais ativo e personalizado.

A Tabela III resume os principais simuladores de MIPS e RISC-V, com análises complementares disponíveis em [8].

O Webrisc-V [14] opera via navegador, aceitando código assembly como entrada e oferecendo quatro configurações de visualização do caminho de dados: com e sem encaminhamento, com e sem branch-delay. Contudo, sua interface apresenta excessivos detalhes de conexões, não permite personalizações e requer conhecimentos avançados em PHP/-JavaScript e desenvolvimento web para extensões.

O Venus [30] funciona primariamente como montador e simulador em nível assembly, exibindo apenas a memória e o banco de registradores, sem representação do caminho de dados. Sua interface minimalista favorece a execução em dispositivos móveis para testes de códigos assembly simples.

O RISC-V Graphical implementa a versão monociclo do caminho de dados, permitindo execução passo a passo com visualização dos sinais. Reproduz fielmente o diagrama do livro texto [20], porém sua extensão demanda conhecimentos em Kotlin/JavaScript e desenvolvimento web.

O emulsiV [26], [27], simulador visual para Virgule, implementa um núcleo mínimo RISC-V. Voltado ao ensino de conceitos básicos de arquitetura, oferece visualizações parciais com animações das transferências entre unidades funcionais. Decompõe cada instrução em etapas (busca, decodificação, ALU, mem/reg, PC) para fins didáticos, sem focar na implementação pipeline.

O Compsim [8] suporta diversos processadores, incluindo RISC-V com 37 instruções. Disponibiliza visualização do banco de registradores, contador de instrução, unidade de controle, ALU e memória cache configurável. Incorpora RAM e recursos de E/S para comunicação com periféricos virtuais e físicos.

Tabela III
RESUMO DOS SIMULADORES DE PROCESSADORES MIPS E RISC-V.

| Nome | Simulador | Execução | Linguagem |
|---|---|---|---|
| Mars [31] | MIPS | Local | Java |
| HadesMIPS [3] | MIPS | Local | Java |
| V-Mips [1] | MIPS | Local | Java |
| Digi-MIPS [19] | MIPS | Navegador | Verilog |
| WebRISC-V [14] | RISC-V | Navegador | PHP |
| Venus [30] | RISC-V | Navegador | Kotlin/JavaScript |
| Gem5-RISC-V [23] | RISC-V | Local | Gem5 |
| RISC-V-graphical [15] | RISC-V | Navegador | Typescript |
| BRISC-V [2] | RISC-V | Navegador | - |
| emulsiV [26] | RISC-V | Navegador | Javascript |
| CompSim RISC-V [8] | RISC-V | Local | - |

O digi-MIPS [19] inclui a descrição do RISC-V em Verilog, mas a visualização é realizada de forma manual ou automática utilizando a ferramenta Yosys na interface do DigitalJS [18]. O BRISC-V [2], [29] é uma plataforma voltada para a exploração

do espaço de projeto de arquiteturas RISC-V em nível de transferência de registro (RTL). A plataforma consiste em módulos RTL de código aberto, parametrizáveis e sintetizáveis, para o design de sistemas baseados em RISC-V, com suporte a um ou mais núcleos. Seu objetivo é proporcionar uma plataforma altamente modular, com módulos parametrizáveis que permitem a rápida exploração de diferentes complexidades de núcleos RISC-V, cache multinível, organizações de memória, topologias de sistema e arquiteturas de roteadores. A plataforma pode ser usada tanto para simulação em RTL quanto para emulação em FPGA, com implementações em Verilog sintetizável sem a necessidade de blocos específicos de fornecedores.

Embora seu foco principal não seja o suporte didático para o ensino introdutório dos caminhos de dados do RISC-V, a plataforma inclui uma ferramenta de compilação RISC-V e uma interface gráfica para a configuração do sistema e simulação de código assembly RISC-V, suportando desde processadores de ciclo único simples até SoCs multicore com hierarquias de memória complexas e redes em chip (NoC). A modularidade da plataforma permite modificações incrementais nos módulos de hardware sem afetar o restante do sistema, facilitando a rápida instância de sistemas multicore RISC-V completos. Trabalhos futuros podem considerar a evolução da abordagem integrando o BRISC-V com ferramentas como o Google Colab, visando uma experiência de uso mais acessível e interativa.

## VII. Conclusão

Python e Google Colab formam uma combinação interessante para criar simuladores e material didático com um foco particular no RISC-V. Para projetos de modelagem e simulações mais simples podemos usar somente Python, já para projetos mais detalhados apresentamos um código híbrido com Verilog para a descrição do processador e Python para teste e interface. O Google Colab democratiza o acesso com uma plataforma na nuvem para qualquer dispositivo com conexão à internet, eliminando a necessidade de instalações locais. Além disso, a popularidade dos notebooks com Jupyter e Python, aliado ao seu uso em várias disciplinas, principalmente na área de inteligência artificial, diminui as barreiras para que os estudantes comecem a explorar e experimentar o desenvolvimento de simuladores. Integrando essas facilidades aos desafios da arquitetura de computadores, podemos incentivar os estudantes a aprofundar seus estudos em técnicas essenciais de design e implementação de processadores, criando simuladores RISC-V.

Este trabalho propõe uma abordagem colaborativa e incremental para o ensino de arquitetura de computadores, especificamente a ISA RISC-V, utilizando o ambiente Google Colab. Ao adotar a metodologia "Bazar", que enfatiza o aprendizado prático e a contribuição colaborativa, buscamos promover um entendimento mais profundo e aplicável dos conceitos de design e implementação de processadores. A reutilização de técnicas, a interatividade dos notebooks Jupyter e a possibilidade de evolução contínua dos materiais de ensino são aspectos que reforçam a eficácia dessa abordagem. Acreditamos que o envolvimento ativo dos estudantes no desenvolvimento e modificação de simuladores, em vez de apenas utilizá-los, resultará em uma aprendizagem mais robusta e significativa. As contribuições específicas deste trabalho, incluindo saídas gráficas, uso de ChatGPT, exercícios com RISC-V e a modelagem de um mini-simulador, demonstram o potencial de inovação e melhoria contínua no ensino de arquitetura de computadores. Esperamos que essa metodologia inspire outros educadores e estudantes a colaborar e expandir ainda mais esses recursos, fortalecendo a formação em arquitetura de computadores e promovendo uma compreensão mais prática e integrada dos sistemas computacionais.

## VIII. Agradecimentos

FAPEMIG (PIBIC, $APQ - 01577 - 22$), CNPq, NVIDIA, Funarbe. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

## Referências

[1] FA Alves, Danilo Almeida, Lucas Bragança, André BM Gomes, Ricardo S Ferreira, and José Augusto M Nacif. Ensinando arquiteturas vetoriais utilizando um simulador de instruções mips. *International Journal of Computer Architecture Education (IJCAE)*, 4(1):9–12, 2015.

[2] Sahan Bandara, Alan Ehret, Donato Kava, and Michel A Kinsy. Briscv: An open-source architecture design space exploration toolbox. *arXiv preprint arXiv:1908.09992*, 2019.

[3] Hector Perez Baranda, Jeronimo Costa Penha, and Ricardo Ferreira. Implementação de um preditor de desvio no mips 5 estágios. *International Journal of Computer Architecture Education*, 6, 2018.

[4] Michael Canesche, Lucas Bragança, Omar Paranaiba Vilela Neto, Jose A Nacif, and Ricardo Ferreira. Google colab cad4u: Hands-on cloud laboratories for digital design. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[5] Kaya Celebi. Python risc-v assembly code assembler package. https://github.com/kcelebi/riscv-assembler.

[6] Universidade Federal de Vicosa. Exemplos de simuladores em google colab para ensino risc-v. https://github.com/arduinoufv/GoogleColabRiscV.

[7] Ricardo Ferreira e José Augusto Nacif. Ensino de software pipelining e escalonamento em gpus com python. *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, 12(2):20 – 29, 2023.

[8] Guilherme Álvaro RM Esmeraldo, Robson Gonçalves Fechine Feitosa, Edna Natividade da Silva Barros, Eduardo Carlos P da S Proto, Harley Macedo de Mello, Edson Barbosa Lisboa, Esdras L Bispo Jr, and Gustavo Augusto Lima de Campos. Uma abordagem para ensino-aprendizado de projetos de sistemas computacionais com utilização do simulador compsim com suporte à arquitetura risc-v. *Revista Brasileira de Informática na Educação*, 31:271–288, 2023.

[9] Ricardo Ferreira, Michael Canesche, Peter Jamieson, Omar P Vilela Neto, and Jose AM Nacif. Examples and tutorials on using google colab and gradio to create online interactive student-learning modules. *Computer Applications in Engineering Education*, page e22729, 2024.

[10] Ricardo Ferreira, Jose Nacif, Salles Magalhaes, Thales de Almeida, and Racyus Pacifico. Be a simulator developer and go beyond in computing engineering. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE, 2015.

[11] Ricardo Ferreira, Carlos Sabino, Michael Canesche, Omar Paranaiba V Neto, and José Augusto Nacif. Aiot tool integration for enriching teaching resources and monitoring student engagement. *Internet of Things*, 26:101045, 2024.

[12] Ricardo S Ferreira, Antonio Carlos S Beck, Luigi Carro, Andre Toledo, and Aroldo Silva. A java framework to teach computer architecture. In *New Trends and Technologies in Computer-Aided Learning for Computer-Aided Design: IFIP TC10 Working Conference: EduTech 2005, October 20–21, Perth, Australia*, pages 25–35. Springer, 2005.

[13] Ricardo S Ferreira, Ulisses Chieppe, Giliardo C Freitas, and Cristiano Biancardi. Software livre no ensino de sistemas digitais e arquitetura de computadores. *Universidade Federal de Viçosa, Departamento de Ciência da Computação*, 2003.

[14] 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*, pages 1–6, 2019.

[15] Jesse Hines. Risc-v-graphical-datapath-simulator. https://jesse-r-s-hines.github.io/RISC-V-Graphical-Datapath-Simulator/.

[16] Dmitry Kirsanov. *The Book of Inkscape: The Definitive Guide to the Graphics Editor*. No Starch Press, 2021.

[17] Matthias Koenig and Roin Rasch. Digital teaching an embedded systems course by using simulators. In *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*, pages 1–7. IEEE, 2021.

[18] Marek Materzok. Digitaljs: A visual verilog simulator for teaching. In *Proceedings of the 8th Computer Science Education Research Conference*, pages 110–115, 2019.

[19] Fernando Passe, Michael Canesche, Omar Paranaiba Vilela Neto, Jose A Nacif, and Ricardo Ferreira. Mind the gap: Bridging verilog and computer architecture. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[20] David A Patterson and John L Hennessy. *Computer organization and design RiscV edition: the hardware software interface*. Morgan kaufmann, 2017.

[21] Jeronimo Costa Penha, Geraldo Fontes, and Ricardo Ferreira. Mipsfpga-um simulador mips incremental com validaçao em fpga. *International Journal in Computer Architecture Education (IJCAE)*, 5(1):19–25, 2016.

[22] Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in science & engineering*, 9(3):21–29, 2007.

[23] Cristóbal Ramírez, César Alejandro Hernández, Oscar Palomar, Osman Unsal, Marco Antonio Ramírez, and Adrián Cristal. A risc-v simulator and benchmark suite for designing and evaluating vector architectures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 17(4):1–30, 2020.

[24] Eric Steven Raymond. A catedral e o bazar. *The Linux Logic Home Page*, 12, 1998.

[25] Isabel Sanches. Do 'aprender para fazer'ao 'aprender fazendo': as práticas de educação inclusiva na escola. *Revista lusófona de educação*, 19(19), 2011.

[26] G Savaton. A visual simulator for teaching computer architecture using the risc-v instruction set. *Guillaume-Savaton-ESEO/emulsiV*, 2021.

[27] Guillaume Savaton. emulsiv is a visual simulator for a simple risc processor called virgule. https://eseo-tech.github.io/emulsiV/.

[28] James Somers. The scientific paper is obsolete. *The Atlantic*, 4, 2018.

[29] Boston University. Brisc-v (boston university risc-v) simulator and teaching tool. https://ascslab.org/research/briscv/simulator/simulator.html.

[30] Keyhan Vakil. Venus, risc-v simulator. https://venus.kvakil.me/.

[31] 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.

[32] Stefan Wallentowitz. Python model of the risc-v isa. https://github.com/wallento/riscv-python-model/tree/master?tab=readme-ov-file.