Desenvolvimento de um Simulador Educacional Visual para Arquiteturas Superescalares com Algoritmo de Tomasulo e Reorder Buffer

Felipe Vilhena Dias, Gabriel Cunha Schlegel, Iago Fereguetti e Lucas Henrique Rocha Hauck
Instituto de Ciências Exatas e Informática – Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte – MG, Brasil
Emails: 1466692.felipe.dias@sga.pucminas.br, gabrielcunha@sga.pucminas.br,
iago.fereguetti@sga.pucminas.br, lhauck@sga.pucminas.br

***Abstract*—O ensino de arquitetura de computadores avançada enfrenta o desafio de ilustrar processos dinâmicos, como a execução fora de ordem e a especulação de hardware, utilizando apenas diagramas estáticos. O algoritmo de Tomasulo, estendido com Buffer de Reordenamento (ROB), é fundamental para o entendimento de processadores superescalares modernos. Este artigo apresenta o desenvolvimento de um simulador visual em Python que permite a execução passo a passo de instruções MIPS, incluindo suporte a desvios condicionais especulativos. A ferramenta destaca-se pela interface gráfica que exibe, em tempo real, o estado das Estações de Reserva e do ROB. Os resultados experimentais, validados através de cronogramas de execução, demonstram a eficácia da ferramenta em elucidar a resolução de dependências de dados e a limpeza do pipeline (*flush*) em erros de predição.**

***Index Terms*—Tomasulo, Reorder Buffer, Simulação, Especulação, MIPS, Python.**

I. Introdução

O avanço contínuo na microarquitetura de processadores busca mitigar a disparidade entre a velocidade da CPU e a latência de memória. Técnicas de Paralelismo a Nível de Instrução (ILP), como a execução fora de ordem (*Out-of-Order Execution*), são essenciais nesse contexto [1]. O algoritmo de Tomasulo revolucionou a computação ao permitir o despacho dinâmico, e sua versão moderna com Buffer de Reordenamento (ROB) garante a consistência sequencial necessária para o tratamento de exceções e especulação de desvios.

Entretanto, a complexidade cognitiva para acompanhar o fluxo de dados entre Estações de Reserva (RS), Barramento de Dados Comum (CDB) e Banco de Registradores torna o aprendizado desafiador. Diagramas estáticos falham em capturar a natureza temporal dos conflitos e o comportamento de instruções especulativas.

Para preencher essa lacuna, este trabalho propõe um simulador interativo desenvolvido em Python. A ferramenta foca na visualização das estruturas internas, permitindo a análise imediata de métricas de desempenho e o comportamento do processador diante de predições de desvio incorretas.

Este artigo organiza-se da seguinte forma: A Seção II discute trabalhos correlatos. A Seção III detalha a implementação. A Seção IV apresenta estudos de caso com análise detalhada do pipeline. A Seção V conclui o trabalho.

II. Trabalhos Correlatos

A simulação é uma ferramenta indispensável no ensino de Arquitetura de Computadores. No cenário de pesquisa acadêmica, o **gem5** [3] é amplamente utilizado devido à sua precisão e suporte a múltiplas arquiteturas. No entanto, sua curva de aprendizado íngreme e falta de interface gráfica nativa o tornam pouco prático para o ensino introdutório.

Para fins didáticos, ferramentas visuais têm ganhado destaque. O **Ripes** [2] é uma referência moderna que oferece visualização detalhada do pipeline para a arquitetura RISC-V. Similarmente, trabalhos como o apresentado por Oliveira et al. [4] focam especificamente na visualização de políticas de substituição de memória cache, cobrindo uma lacuna importante no aprendizado da hierarquia de memória.

No contexto específico da arquitetura MIPS, o **EduMIPS64** [5] é uma ferramenta consolidada que permite a visualização do pipeline. Contudo, muitas dessas ferramentas educacionais limitam-se à execução em ordem (*in-order*) ou não detalham visualmente o funcionamento conjunto das Estações de Reserva e do ROB.

A proposta deste trabalho difere das anteriores ao focar exclusivamente na visualização do algoritmo de Tomasulo com suporte à especulação. Ao contrário de simuladores de propósito geral, nossa ferramenta isola a complexidade da execução fora de ordem, oferecendo uma interface dedicada para o acompanhamento da resolução de dependências e controle.

III. Arquitetura e Implementação

O simulador foi desenvolvido em Python 3 com interface `tkinter`. Para simular o hardware paralelo em software sequencial, as etapas do pipeline são processadas na ordem inversa dentro de cada ciclo de clock (`step()`): *Commit* → *Write Result* → *Execute* → *Issue*.

O suporte a especulação utiliza a estratégia *Predict Not Taken*. Instruções posteriores a um desvio (`BEQ`) são despachadas especulativamente. Caso o desvio seja tomado, o simulador realiza um *flush*, limpando as entradas inválidas no ROB e nas Estações de Reserva.

Para a avaliação quantitativa do desempenho, o simulador coleta, ciclo a ciclo, métricas cruciais de microarquitetura. Além do cálculo padrão de Instruções por Ciclo (IPC), a ferramenta monitora eventos de paralisação do despacho (*Issue Stalls*), classificados em bolhas por escassez de Estações de Reserva (RS Stalls) e bolhas por saturação do ROB. Adicionalmente, contadores específicos registram o número de erros de predição de desvio (*Branch Mispredictions*).

A Figura 1 apresenta a interface principal executando o cenário de teste aritmético.

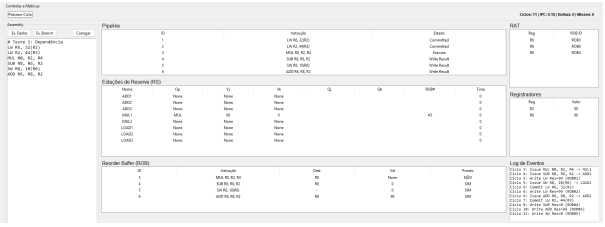

Fig. 1. Interface do simulador. O painel central exibe as dependências (Qj/Qk) nas Estações de Reserva durante a execução aritmética.

IV. Resultados e Discussão

Para validar o simulador, analisaram-se dois cenários distintos: dependência de dados (aritmética) e dependência de controle (especulação).

A. Cenário A: Dependências de Dados (RAW/WAW)

Utilizou-se o código abaixo com latências de $MUL = 6$, $SUB = 2$ e $ADD = 2$ ciclos.

```
LW R6, 32(R2) # I1
LW R2, 44(R3) # I2 (Produtor de R2)
MUL R0, R2, R4 # I3 (Usa R2, Lento)
SUB R8, R6, R2 # I4 (Usa R2, Rapido)
SW R8, 10(R6) # I5
ADD R6, R8, R2 # I6 (WAW em R6)
```

A Tabela I apresenta o rastreamento ciclo a ciclo. Observa-se que, embora a instrução `SUB` (I4) termine sua execução no ciclo 9 (antes da `MUL`), ela aguarda no ROB até o ciclo 15 para realizar o *Commit*, respeitando a ordem sequencial do programa. Isso valida a implementação correta do ROB.

TABLE I
Cronograma de Execução (Cenário A)

Instrução	Issue	Execute	Write Res.	Commit
I1: LW R6	1	2 – 4	5	6
I2: LW R2	2	3 – 5	6	7
I3: MUL R0	3	7 – 12	13	14
I4: SUB R8	4	**7 – 8**	**9**	**15**
I5: SW R8	5	10 – 12	13	16
I6: ADD R6	6	10 – 11	12	17

B. Cenário B: Especulação e Flush

Para testar a especulação, utilizou-se um desvio condicional (`BEQ`) configurado para ser tomado ($R1 = R2$), forçando um erro na predição padrão (*Not Taken*).

```
ADDI R1, R0, 10
ADDI R2, R0, 10
BEQ R1, R2, 2 # Desvio Tomado (Pula 2)
ADDI R3, R0, 5 # Especulada (Errada)
ADD R4, R1, R2 # Especulada (Errada)
SUB R5, R1, R2 # Destino Correto
```

A análise do log de eventos gerado pelo simulador confirma o comportamento esperado:

1) **Ciclos 3-5:** O processador despacha as instruções `ADDI R3` e `ADD R4` especulativamente.
2) **Ciclo 6:** O `BEQ` calcula o resultado "Igual".
3) **Ciclo 7 (Commit do BEQ):** O simulador detecta o erro de predição. O evento *FLUSH* é acionado, removendo as instruções `ADDI R3` e `ADD R4`.
4) **Ciclo 8:** O PC é corrigido e a instrução correta (`SUB R5`) é despachada.

Este comportamento pode ser visualizado na Figura 2.

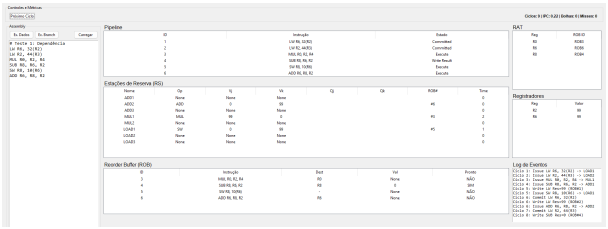

Fig. 2. Visualização da Especulação. Destaque para o log de eventos mostrando o "Flush" no ciclo 7 após a falha na predição.

A Tabela II e a Figura 3 resumem as métricas coletadas. Observa-se que o Cenário B apresenta um IPC superior devido à menor latência das instruções, apesar da penalidade do *branch miss*.

TABLE II
Métricas de Desempenho Coletadas

Cenário	Ciclos Totais	IPC	Misses de Branch
A (Dados)	17	0.35	0
B (Flush)	15	0.40	1

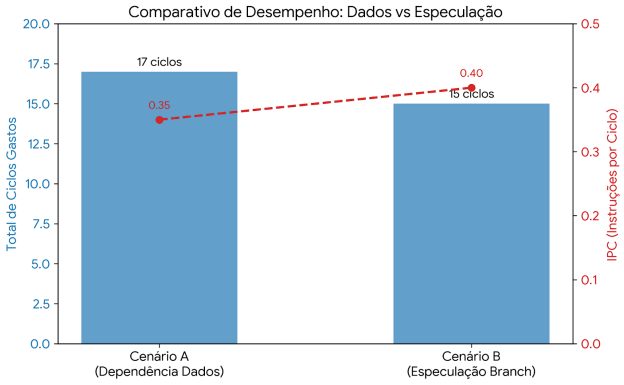

Fig. 3. Comparativo de Desempenho. As barras indicam o tempo total (menor é melhor) e a linha indica o IPC (maior é melhor).

V. Conclusão

O simulador desenvolvido demonstrou-se eficaz na ilustração de conceitos de microarquitetura avançada. A visualização gráfica permitiu observar claramente o descompasso entre a execução fora de ordem e o *commit* em ordem. Além disso, a implementação bem-sucedida do mecanismo de *flush* no ROB oferece aos estudantes uma compreensão prática do custo da especulação incorreta, um conceito frequentemente abstrato em aulas teóricas.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kaufmann, 2019.

[2] M. Borup, "Ripes: A visual processor simulator for RISC-V," *Journal of Open Source Software*, vol. 5, no. 52, p. 2537, 2020.

[3] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[4] G. L. M. de Oliveira et al., "A visual simulator for teaching cache memory concepts," in *IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1–5.

[5] A. Spadaccini et al., "EduMIPS64: A Visual and Interactive MIPS 64-bit Simulator," in *Proc. IEEE Int. Conf. on Microelectronic Systems Education*, 2009.