Simulador Educacional Visual para Arquiteturas Superescalares

Algoritmo de Tomasulo e Reorder Buffer (ROB)

Felipe Vilhena Dias, Gabriel Cunha Schlegel,
Iago Fereguetti, Lucas Henrique Rocha Hauck

PUC Minas - Instituto de Ciências Exatas e Informática

Contexto e Motivação

O Desafio do Ensino

O ensino de arquitetura de computadores avançada enfrenta barreiras significativas. Processos dinâmicos como a **Execução Fora de Ordem** e a **Especulação** são difíceis de visualizar apenas com diagramas estáticos de livros.

Problemas Identificados

- Complexidade cognitiva elevada para rastrear múltiplas instruções simultâneas.
- Dificuldade em visualizar o fluxo entre Estações de Reserva e ROB.
- Falta de ferramentas que mostrem *hazards* temporais em tempo real.

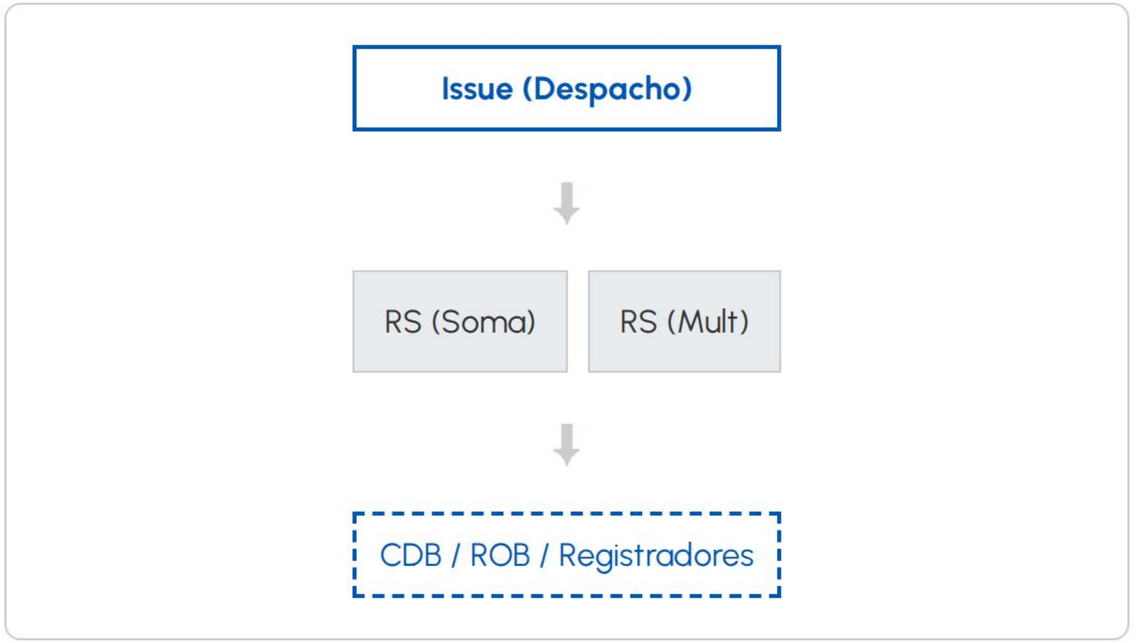
Fundamentação Teórica

Algoritmo de Tomasulo

Revolucionou a computação ao permitir o **despacho dinâmico**. Ele usa renomeação de registradores para resolver falsas dependências (WAR e WAW).

Reorder Buffer (ROB)

Componente essencial em processadores modernos. Garante que, embora a execução seja fora de ordem, a **graduação (commit)** seja sequencial, permitindo tratamento preciso de exceções.



Trabalhos Relacionados

Gem5

Extremamente preciso e usado na indústria. Porém, possui curva de aprendizado íngreme e falta de GUI nativa, inadequado para ensino introdutório.

Ripes / WebMIPS

Ripes é excelente para RISC-V visual. WebMIPS é acessível. Contudo, focam mais no pipeline estático ou ISA, e menos na dinâmica especulativa do Tomasulo.

Nossa Proposta

Foco exclusivo na visualização do Tomasulo + ROB. Interface dedicada para acompanhar resolução de dependências e controle especulativo.

Arquitetura do Simulador

Implementação

- Linguagem: **Python 3**
- Interface: **Tkinter** (Nativa)
- Estratégia: **Pipeline Reverso**

Para simular hardware paralelo em software sequencial, processamos as etapas inversamente dentro de cada ciclo:

`Commit → Write Result → Execute → Issue`

```
class TomasuloCore:
    def step(self):
        self.clock += 1
        # 1. COMMIT
        self.commit_stage()
        # 2. WRITE RESULT
        self.write_result_stage()
        # 3. EXECUTE
        self.execute_stage()
        # 4. ISSUE
        self.issue_stage()
```

Métricas de Desempenho

O simulador coleta métricas cruciais ciclo a ciclo para análise quantitativa.

17	0.35	2	1
CICLOS TOTAIS	IPC MÉDIO	BOLHAS (STALLS)	MISSES BRANCH

Stalls: Contabiliza paradas por falta de Estações de Reserva (RS) ou ROB cheio.

Misses: Contabiliza quantas vezes a predição de desvio falhou, causando flush.

Agora vamos testar a interface

Demonstração prática do simulador em execução

Resultados: Dependência de Dados

Cenário de Teste

```
LW R6, 32(R2) LW R2, 44(R3) MUL R0, R2, R4 # Lento SUB
R8, R6, R2 # Rápido SW R8, 10(R6) ADD R6, R8, R2
```

A instrução **SUB** (rápida) termina antes da **MUL** (lenta), provando a execução fora de ordem.

Trace de Execução

Instrução	Execução	Write	Commit
LW R6	2-4	5	6
LW R2	3-5	6	7
MUL R0	7-12	13	14
SUB R8	**7-8**	**9**	**15**
SW R8	10-12	13	16

*Note que SUB termina no ciclo 9, mas só faz commit no 15 (Ordem mantida).

Resultados: Especulação e Flush

O Cenário

Um desvio condicional `BEQ R1, R2, 2` é configurado. O simulador prevê "Não Tomado" e despacha instruções especulativamente.

O Flush

No ciclo 7, o erro é detectado. O evento **FLUSH** é acionado:

- Instruções especuladas são removidas do ROB.
- Estações de Reserva são limpas.
- PC é corrigido para o destino correto.

```
LOG: Ciclo 7: --- FLUSH! Predição falhou. (Erro #1) ---
```

Conclusão

Contribuições

- Ferramenta visual eficaz para ensino de microarquitetura avançada.
- Visualização clara do descompasso entre execução e graduação.
- Implementação prática de conceitos abstratos como Flush e Renomeação.

Trabalhos Futuros

- Integração com simulador de Cache (L1/L2).
- Implementação de preditor de desvios dinâmico (BPB).

Obrigado!

Dúvidas?