Análise de Preditores de Desvios de Direção Relatório Trabalho de Arquitetura*

Carlos A. Bitencourt[1], **Murilo Lázaro**[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
79070-900 – Campo Grande – MS – Brazil

`carlos.bitencourt@ufms.br, murilo.lazaro@ufms.br`

Abstract. *Write after, Write after.*

Resumo. *Escrever depois.*

1. Introdução

Na arquitetura de computadores, busca-se estratégias para a melhoria dos desempenho de programas. Compreender como o *hardware* executa os programas é essêncial para alcançar este objetivo [Patterson e Hennessy 2014]. Uma estratégia para a análise de análise de métricas de desempenho é a utilização de simuladores.

Neste trabalho, busca-se simular a execução de 3 programas com intuito de coletar métricas de desempenho com utilização do **Algoritmo de Tomasulo** em conjunto com a técnica de **Desvio de Predição Baseada em Direção**. Para a execução deste experimento utilizou-se como base o simulador TFSim [dos Reis e Duenha 2019]. Trata-se de um simulador funcional para execução fora de ordem e forma especulada, instruções MIPS6, obedecendo criteriosamente o Algoritmo de Tomasulo. Esse trabalho propõe a inclusão de novas funcionalidades, como a inclusão de preditores baseado em direção, criação de uma função que executa todos os ciclos das instruções que compõem um determinado programa e, realização do cálculo de métricas de desempenho, como, o Contador de instruções (IC), instruções por ciclo (IPC), MIPS e tempo de CPU.

*Dados complementares, código, vídeo: https://github.com/carlos-bitencourt/arq-project

2. Referencial Teórico

Na fundamentação teórica, para que se possa embasar uma análise adequada, faz-se necessário enumerar e descrever os conceitos utilizados na pesquisa como Algoritmo de Tomazulo, Preditores Baseado em Direção dentre outros.

2.1. Execução fora de ordem

Na execução de um programa, as instruções a serem executadas podem conter situações de riscos de conflito no *pipeline*, dentre eles a dependência verdadeira (*Read After Write* - RAW), antidependências (*Write After Read* - WAR) e dependências de saída (*Write After Write - WAW*) [Hennessy e Patterson 2019].

Tabela 1. Exemplo Trecho de Programa.

Linha	Instruções
1	fmul f0, f2, f4
2	fdiv f10, f0, f6
3	fsub.d f3,f2, f1
4	fadd.d f3, f4, f5
5	fadd.d f2, f7, f11

Ao considerar o trecho de código na Tabela 1 ocorre uma **dependência verdadeira** em **f0** na linha 1 e 2, instruções **fmul** e **fdiv**, pois o valor do f0 da instrução **fdiv**, linha 2, depende do resultado da instrução **fmul** na linha 1. Deste modo, em pipeline é necessário que instruções que possuem dependências verdadeiras aguardem a execução da instruções das quais dependem. Do mesmo modo, ainda na Tabela 1, existe uma **antidependência** caracterizada em **f2**, na linha 3 e 5, instruções **fsubd** e **faddd**. E, por fim, caracteriza-se como **dependência de saída** as instruções **f3** nas linhas 3 e 4, entre as instruções **fsub.d** e **fadd.d**. Tanto a **antidependência** quanto a **dependência de saída** representam um risco na execução fora de ordem pois podem gerar resultados corrompidos. Deste modo, uma forma de evitar erros de execução é a utilização da técnica de **renomeação de registradores** dado que a ordem destas instruções não é uma imposição restritiva.

2.2. Algoritmo de Tomasulo

O Algoritmo de Tomasulo foi desenvolvido por Robert Tomasulo, em 1967. Trata-se de uma algoritmo, implementado em hardware, que permite a execução simultânea de instruções em várias unidades de execução, permitindo assim, que as instruções do programa original sejam executadas fora de ordem. O objetivo desse algoritmo é melhorar o desempenho das operações de ponto flutuante, dado um determinado conjunto de instruções, de maneira a tornar os compiladores mais simples, ou seja, menos especializados.

O algoritmo propõe a minimização da quantidade de estados de espera entre as execuções das instruções em trechos de códigos que possui dependências, no caso das dependências de saídas e antidependências, são completamente resolvidas através da técnica de renomeação de registradores usada nas estações de reservas 1 e, nas instruções que possuem dependências verdadeiras tenta minimizar o número de stalls entre suas execuções.

Na Figura 1, observa-se uma estrutura do Algorimo de Tomasulo. O Buffer de reordenação (ROB) guarda as instruções que terminam suas execuções com o propósito de imposição da garantia da ordem de despacho de cada instrução em relação ao código original, impedindo assim, a execução fora de ordem. Somente após confirmação de instrução (**commit**) uma instrução terá o status de finalizada, realizando a escrita a dos dados produzidos nos registradores e/ou memória [Hennessy e Patterson 2019]. A estrura de especução, garante que nenhuma instrução que está sendo executada da forma especulada, possa escrever permanetemente na memória ou registrador. Em caso de erro de predição todas as instruções especuladas são removidas do buffer sem deixar inconsistente o armazenamento de dados.

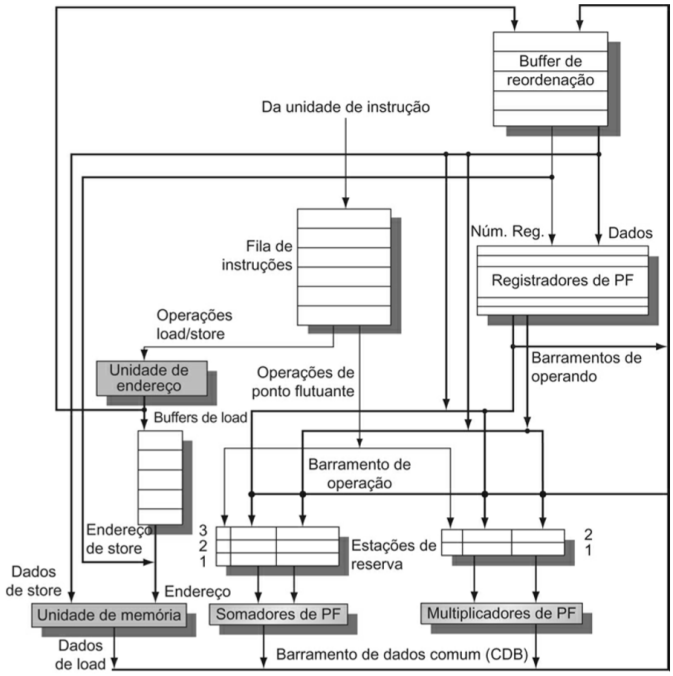

Figura 1. Estrutura do Harware de Tomasulo. Fonte: Hennessy e Patterson (2019).

Considerando este esquema, o Algoritmo de Tomasulo oferece duas vantagens importantes, a primeira é a distribuição da lógica de detecção de riscos, através das Estações de Reservas (ES) distribuídas e do uso do Barramento Comum (CDB), a segunda, trata-se da eliminação de stalls para os riscos de WAW e WAR. Isso pode ocorrer através da renomeação de registradores das ES's e, também pelo processo de armazenar operando na estação de reserva quando estiverem disponíveis. Deste modo, o Algoritmo de Tomasulo torna-se uma ferramenta importante para execução das instruções distribuídas.

2.2.1. Preditor de Desvio Baseados em Direção

Dentres o preditores, existem os estáticos e os dinâmicos. Os estáticos pressupõe que o desvio será independente do histórico de execução. Em contrapartida, os dinâmicos consideram o histórico.

Os Preditores de desvio baseado em direção, são estáticos e consideram a direção

do desvio na memória para tomada de decisão. Neste caso, a comparação é entre o endereço de memória atual e o endereço do destino do desvio se faz necessária.

Neste trabalho serão consideradas duas versões de preditores estáticos:

Desvios para endereços “maiores” *TAKEN* (V1) e Devios para endereços “maiores” *NOT TAKEN* (V2). Sendo que, o primeiro (V1), caso a comparação entre o endereço atual e o endereço de desvio indicar que o endereço de desvio é “maior”, então o preditor “pega(*TAKEN*)” o desvio, caso contrário “não pega (*NOT TAKEN*)” o desvio. Da mesma maneira, o segundo (V2), caso a comparação entre o enderço atual e o endereço de desvio indicar que o endereço é “maior”, então o preditor “não pega (*NOT TAKEN*)”, caso contrário o preditor “pega(*TAKEN*)”.

3. Metodologia

Para alcançar o propósito do trabalho, foram realizados ajustes no sistema TFSim [dos Reis e Duenha 2019]. Dentre eles de um botão ALL CYCLES, cálculo de algumas métricas de desempenho e a implementação de dois novos preditores estáticos descritas abaixo:

1. Inclusão do botão *ALL CYCLE* - realiza a execução de todos os ciclos de execução de trecho de Código, em MIPS.
2. Métricas de Desempenho - contador de instruções, Instruções por ciclo (IPC), Milhões de Instruções por Segundo (MIPS) e tempo de execução de CPU; Disponível na parte não gráfica do sistema.
3. Inclusão de dois novos Preditores estáticos 2.2.1:
   (a) Desvio para cima TAKEN (V1).
   (b) Desvio para cima NOT TAKEN (V2).
4. Inclusão do menu ’Trabalho’ (Figura 2) - para auxiliar na execução dos programas do trabalho, foi criado um menu, contento as opções de seleção de 3 novos programas e a opção de escolha dos desvios de predição desenvolvidos.

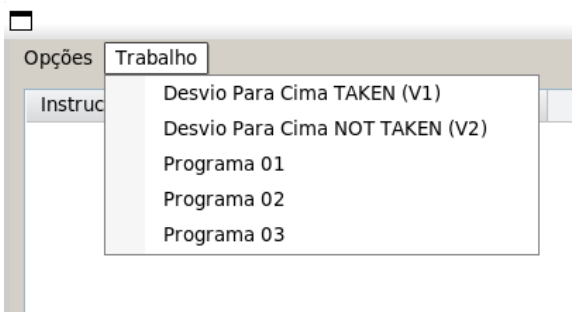

Figura 2. Insterface do Nova Menu TFSim. Fonte: Elaboração Própria (2022).

Deste modo, na Figura 3, pode-se observar a interface do sistema que está organizada como segue:

1. Fila de Instruções: mostra as instruções que estão sendo executadas durante a simulação, contendo informações como (despacho, execução, escrita e complemento);

2. Estações de Reserva (RS): contém as informações de operandos, como nome da instrução, caso a instrução está ocupada ou não (*Busy*), Op, V_j, V_k, Q_j, Q_k e A;
3. Banco de registradores: informa os valores de todos os registradores e qual estação de reserva (ou posição do ROB) está programado para escrever em cada registrador envolvido em uma determinada operação;
4. Estrutura de Memória: Conjunto de valores das posições de memória;
5. Buffer de Reordenação (ROB): informa quais instruções estão sendo executadas for a de ordem, apresentando seu destino e valor resultante (caso já tenham sido computadas);
6. Área onde são exibidos os valores do ciclo de clock. Em um versão futura poderá ser trazida as métricas implementadas de modo não gráfica.
7. Botões de controle de execução: *START* – inicia a execução do primeiro código de sequência de códigos. *NEXT CYCLE* – executa um ciclo por vez na sequência dada de códigos. *ALL CYCLE* – executa todos os ciclos de uma determinada sequência, sem sair do programa e *EXIT* – encerra a execução.

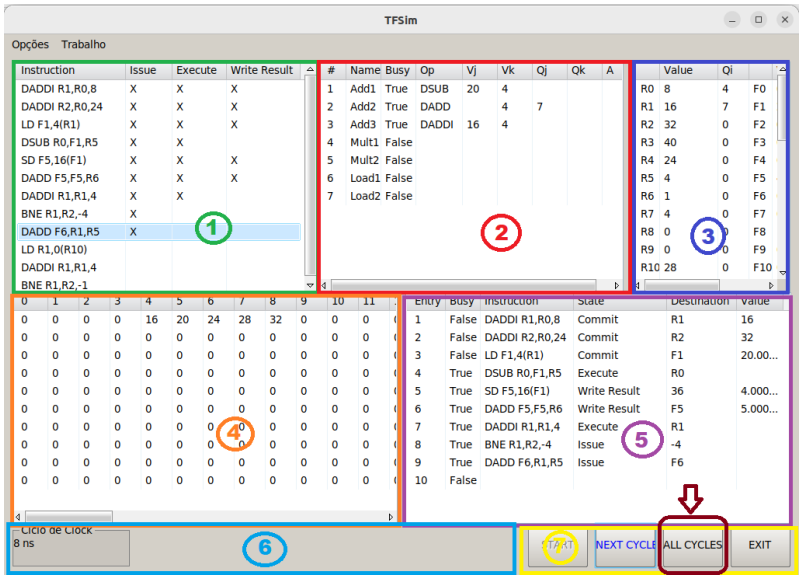

Figura 3. Insterface do TFSim. Fonte: Adaptado de dos Reis e Duenha (2019).

4. Análise e Discussão

...

5. Conclusão

* Trabalhos futuros, implementação de metricas na interface. * Botão Limpar Execução.

Referências

dos Reis, L. O. P. e Duenha, L. D. (2019). Tfsim: um simulador do algoritmo de tomasulo para apoio ao ensino de arquiteturas superescalares. *International Journal of Computer Architecture Education (IJCAE)*, 8(1):17–26.

Hennessy, J. L. e Patterson, D. A. (2019). *Arquitetura de Computadores: Uma Abordagem Quantitativa*. [tradução Daniel Vieira], Elsevier, Rio de Janeiro, 6ª edição.

Patterson, D. A. e Hennessy, J. L. (2014). *Organização e Projetos de Computadores: A interface hardware / software*. [tradução Daniel Vieira], Elsevier, Rio de Janeiro, 4ª edição.