Relatório do Trabalho Prático II

Arquitetura de Computadores III

Gabriel Lopes Ferreira
PUC Minas - ICEI
Belo Horizonte, Brazil
fulano@sga.pucminas.br

Luiz Junio Veloso Dos Santos
PUC Minas - ICEI
Belo Horizonte, Brazil
ljvsantos@sga.pucminas.br

Matheus Luiz Oliveira Spindula
PUC Minas - ICEI
Belo Horizonte, Brazil
fulano@sga.pucminas.br

Rebeca Neto
PUC Minas - ICEI
Belo Horizonte, Brazil
rbsneto@sga.pucminas.br

***Resumo*—Este é um relatório dos resultados obtidos ao realizar testes em um Simulador de Pipeline Superescalar com objetivo de estudo do funcionamento desse tipo de Pipeline.**

***Index Terms*—Pipeline, Superescalar, Tomasulo, Simulador**

I. Introdução

This document is a model and instructions for LaTeX. Please observe the conference page limits.

II. O simulador

O objetivo do simulador é auxiliar estudantes a entender o conceito de superescalaridade na arquitetura de microprocessadores. O simulador é baseado no Algoritmo de Tomasulo, na versão alfa o autor busca com o auxilio de avaliações e contribuições, aprimorar seu projeto. O simulador nos permite testar diferentes tipos de arquitetura, como aumentar o número de instruções no pipeline, alterar o número de unidades operacionais e o número de instruções na janela. Além disso podemos ver o caminho percorrido pela instrução no pipeline superescalar de uma maneira gráfica e também é possível ver o impacto de um programa com dependências e conflitos de registradores.

III. Metodologia adotada

Para analisarmos a execução, realizamos diversos testes com trace padrões da ferramenta e trace manipulados. Também fizemos alterações na arquitetura para analisarmos o impacto de diferentes arquiteturas na execução de um programa, e de diferentes programas para visualizarmos melhor problemas como dependências de leitura pós-escrita.

IV. Análise dos testes

A. Teste 01

Neste teste, usamos uma arquitetura superescalar com *2-way Pipeline*, *2-line buffer*, um trace com 21 instruções com todas instruções idênticas (ADD R1,R2,R3), ou sejam todas leem e escrevem no mesmo local, comparamos o impacto de utilizar 1 ALU e 2 ALU.

No teste acima, apesar de muitas instruções serem despachadas para o *Instruction Queue*, poucas vão para a *Reservation Station* (Janela Distribuída), já que existe somente 1 ALU e sua janela comporta somente 2 instruções e somente uma é executada em cada ciclo. Após sair da ALU o resultado vai para o *Reorder Buffer*, para que possa ser escrito no banco de registradores. No *Reorder Buffer* podemos ver a primeira instrução com um "Y" na coluna *ready*, e um "Result" na coluna V, indicando que seu resultado está pronto e que pode ser escrita no banco

Na segunda parte do teste, foi adicionada uma segunda ALU, isso fez com que um maior número de instruções fossem despachadas para as *Reservation Stations*, gerando um menor acumulo de instruções no *Instruction Queue*, o fato de ter 2 ALU faz com que ocorra um paralelismo de instruções, 2 em cada Janela. No *Reorder Buffer* podemos ver diversas instruções com "Result" na coluna V indicando que já estão com o resultado pronto, porém somente a primeira está com "Y" na coluna *ready*.

B. Teste 02

Neste teste, usamos uma arquitetura com *2-way Pipeline*, *3-line Buffer* e um trace com 21 instruções todas fazendo uso da ALU (ADD RX,R2,R3), porém diferentemente do teste 1, todas as instruções escrevem em local diferente. Comparamos o impacto ao utilizar 1 ALU e 2 ALU.

No teste acima, vemos muitas instruções paradas no *Instruction Queue* aguardando serem despachadas, já que existe somente 1 ALU e sua janela há espaço para duas instruções. Já no *Instruction Queue* vemos somente 3 instruções, vemos um Result na primeira, e um Y no *ready* indicando que está fazendo a escrita no banco de registradores.

Nessa parte do teste, foi adicionada outra ALU, que impactou de forma bem significativa o desempenho do trace. Na print podemos ver somente duas instruções aguardando no *Instruction Queue*, e 4 instruções na etapa de execução duas em cada ALU. Já no *Reorder Buffer* vemos duas instruções com Result na coluna V, e estas mesmas duas instruções com Y na coluna *ready*, além disso vemos outras 4 instruções no *Reorder buffer*, aguardando seu resultado.

C. Teste 03

Neste teste, usamos uma arquitetura com *2-way Pipeline*, *3-line Buffer* e um trace com 21 instruções todas fazendo uso da ALU (instruções do tipo ADD), todas instruções apresentam dependências com a instrução anterior. Comparamos o impacto ao utilizar 1 ALU e 2 ALU.

No teste acima, vemos um grande número de instruções paradas no *Instruction Queue* aguardando a liberação para uma ALU. Na ALU podemos perceber na coluna Qk, que ambas as instruções aguardam dependências, visto que na primeira vemos um ROB 0, indicando que sua dependência, está no *Reorder Buffer* linha 0, e na outra vemos um ROB 1. Já no *Reorder Buffer* vemos uma instrução com Result e um Y.

Nessa parte do teste adicionamos uma segunda ALU, que não trouxe nenhum impacto devido as dependências, podemos ver um grande número de instruções no *Instruction Queue*, nas ALU's podemos ver 3 instruções paradas aguardando suas dependências e somente uma instrução sendo executada, já no *Reorder buffer* vemos 4 instruções aguardando seus resultado para escreverem no banco.

D. Teste 04

Neste teste, utilizamos *2-way Pipeline*, *3-line Buffer* e um trace com 21 instruções todas um ADD de ponto flutuante, e escrevendo em mesmo local, sem dependências, comparamos o impacto ao utilizar 1 FP e 2 FP.

No teste acima, vemos um grande número de instruções parada no *Instruction Queue* aguardando a finalização de alguma instrução na FP1, já na FP podemos ver duas instruções sendo executadas, já no *Reorder Buffer* vemos somente duas instruções aguardando a finalização para realizar a escrita.

Na imagem acima vemos o teste com duas unidades FP que trouxe um grande impacto para a execução do trace. Podemos perceber no banco de registradores (Register File), um registrador com o escrito "ROB 4", mostrando que o seu valor está nessa posição do *Reorder buffer*, isso ocorre devido a renomeação de registradores. Podemos ver um número parecido de instruções na *Instruction Queue*, porém nas unidades de FP vemos duas instruções sendo executadas em cada uma das FP's. No *Reorder buffer* vemos duas instruções a mais do que no teste com uma FP, e apenas uma instrução escrevendo no banco.

E. Teste 05

Neste teste diferente dos anteriores, onde testamos vários tipos de problemas separadamente, utilizamos o trace padrão do simulador o "mips_assembly4.txt" em que suas operações são mais aleatórias e misturam diversos tipo de operações. Usamos uma arquitetura com *2-way Pipeline* e *3-line buffer*, realizamos testes com uma de cada unidade funcional e depois com duas de cada unidade.

Na imagem acima é possível ver diversas instruções na *Instruction Queue* aguardando a liberação de uma unidade operacional. No register file, vemos duas instruções com ROB significando que seus valores estão no Reorder Buffer. Na etapa de execução podemos ver diversas instruções em estados diferentes, como algumas aguardando dependências e outras sendo executadas ou aguardando na fila para executarem. No *Reorder buffer* é possível notar duas instruções com "Result", fora de ordem, e as mesmas com Y, além disso é possível ver um Store com o valor de um registrador, indicando que ele está guardando aquele valor na memoria.

Neste ultimo teste, repetimos o teste anterior porém com duas unidades funcionais de cada tipo, podemos ver somente duas instruções na *Instruction Queue* aguardando o despacho, nas *Reservation stations* vemos um menor acumulo de operações devido a maior quantidade de unidades operacionais, já no *Reorder buffer* vemos um maior número de instruções aguardando para serem escritas no banco de registradores.

V. Análise do simulador

A. Pontos Positivos

- Auxilia muito na visualização de conceitos do "superescalar", e aprendizagem de estudantes de arquitetura de computadores.

B. Pontos Negativos

- A fonte usada no simulador é um pouco pequena, os elementos dele é pequeno.
- Falta de um possibilidade dar zoom.
- Não mostra um contador de ciclos.
- Sempre que finalizar um teste é necessário reiniciar o programa para fazer outro teste e novamente colocar toda a arquitetura que sera usada.

C. Melhorias futuras

- Implementar uma funcionalidade de zoom no simulador e aumentar a fonte em todo o simulador.
- Para melhorar o simulador seria aconselhável que uma variável que conta o número de ciclos seja inserida, notamos que ja existe um contador interno, mas que o valor dele não esta sendo mostrado para o usuário, modificamos o código fonte do simulador para que essa variável fosse exibida na tela. Contudo foi possível perceber que ele não possui muitos tratamentos, onde se o programa parar, ele continuara contando a cada click no botão "next", independentemente do que esta sendo executado.

VI. Conclusão

Referências

[1] Roberto Miranda, and Eduardo Gregório, Superescalar Simulator (ALPHA version) based on Tomasulo's Algorithm, https://github.com/robertomap/SuperscalarSIM