UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GERALDO FULGÊNCIO DE OLIVEIRA NETO

Proposta de Conjunto de Simulações para Análise de Desempenho de Processadores Superescalares e Ensino de Arquitetura de Computadores

Trabalho de Conclusão de Mestrado apresentado como requisito parcial para obtenção do grau de Mestre em Informática

Prof. Dr. Flávio Rech Wagner
Orientador

Porto Alegre, março de 2004.

CIP- CATALOGAÇÃO NA PUBLICAÇÃO

Oliveira Neto, Geraldo Fulgêncio de
Proposta de Conjunto de Simulações para Análise de Desempenho de Processadores Superescalares e Ensino de Arquitetura de Computadores/Geraldo Fulgêncio de Oliveira Neto. – Porto Alegre: Programa de Pós- Graduação em Computação, 2004

117f.: il.

Trabalho de Conclusão (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós- Graduação em Computação, Porto Alegre, BR-RS, 2004. Orientador: Flávio Rech Wagner.
1. Arquiteturas Superescalares. 2. Simplescalar. 3. Arquitetura de Computadores. 4. Simuladores. I. Wagner, Flávio Rech. II. Título.

AGRADECIMENTOS

Gostaria de agradecer aos componentes da banca pela boa vontade na leitura e avaliação deste trabalho de conclusão, a toda equipe da biblioteca e do Instituto de Informática pelo auxílio e completa disponibilização de todo o material necessário na confecção deste e ao meu orientador prof. dr. Flávio Rech Wagner pelas críticas e sugestões que contribuíram de forma decisiva na realização deste trabalho, além do interesse e empenho que serviram de exemplo no trato com o ensino e pesquisa. E agradecer, de maneira geral, a todo corpo docente e administrativo do Instituto de Informática que demonstrou as razões para ser reconhecido como centro de excelência em informática do nosso país.

SUMÁRIO

LISTA DE ABREVIATURAS

ISA Instruction Set Architecture

CISC Complex Instruction Set Computers

RISC Reduced Instruction Set Computers

BHT Branch History Table

DHT Direct History Table

HRT History Register Table

BTB Branch Target Buffer

LRU Least Recently Used

MIPS Milhões de Instruções por segundo

PISA Portable Instruction Set Architecture

SPEC Standard Performance Evaluation Corporation

BTA Branch Target Buffer

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O objetivo deste trabalho é a definição de um conjunto de roteiros para o ensino de arquitetura de computadores com enfoque em arquiteturas superescalares. O procedimento é baseado em simulação e verificação da influência dos parâmetros arquiteturais dos processadores, em termos funcionais e de desempenho.

É dada ênfase a conceitos como memória cache, predição de desvio, execução fora de ordem, unidades funcionais e etc. Através do estudo e avaliação dos parâmetros que constituem estes conceitos, procura-se através dos roteiros identificar as configurações com melhor desempenho.

Para a implementação destes roteiros é adotado o conjunto de ferramentas de simulação SimpleScalar. Este conjunto, além de estar disponibilizado em código aberto na página oficial das ferramentas, traz como vantagem a possibilidade de alteração do código para fins de pesquisa.

Este trabalho e os roteiros que o compõem têm como objetivos auxiliar professores e estimular os alunos através de simulações, como forma didática de testar conceitos vistos em sala de aula. Os roteiros são apresentados com os respectivos resultados de simulação e incrementados com comentários e sugestões de um conjunto de perguntas e respostas para que o trabalho possa ter a continuidade necessária, partindo da sala de aula para a simulação, busca de respostas e culminando com um relatório final a ser avaliado.

Palavras-chave: Arquiteturas Superescalares, Simplescalar, Arquitetura de Computadores, Simuladores.

Proposal of a set of simulations for the performance analysis of superscalar processors and computer architecture education

ABSTRACT

The goal of this work is the definition of a set of scripts for education in computer architectures, with emphasis in superscalar architectures. The approach is based on simulation and on verification of the influence of the architectural parameters of the processors, on functional terms and on performance.

Emphasis is given to concepts as cache memory, branch prediction, out-of-order execution, functional units, etc. Through the study and evaluation of parameters that are basic to these concepts, architectures with better performance are identified through the scripts.

For the implementation of these scripts, the SimpleScalar simulation tool set is adopted. This set, available as open source code in its official web page, brings as advantage the possibility of alteration of its code for research purposes.

This work and the scripts that it contains have as goal to help professors and to stimulate students through simulation, as a didactic way of testing concepts presented in the classroom. The scripts are presented with the respective simulation results and developed with comments and suggestions of a set of questions and answers. Therefore, the student's work can have the necessary continuity, from the classroom to the simulation, the search for answers, and culminating with a final report to be evaluated by the professor.

Keywords: Simplescalar, simulators, superescalar processor, computer architecture

1 INTRODUÇÃO

Com a evolução da tecnologia o ensino de Arquitetura de Computadores tem se tornado uma tarefa difícil, devido à complexidade dos sistemas que são estudados e à falta de interatividade que os alunos têm com o sistema a ser estudado. Uma possibilidade docente interessante é o uso de simuladores, que permitem aos alunos experimentar com o sistema e aplicar os conhecimentos adquiridos para resolver problemas mais ou menos complexos.

O SimpleScalar [AUS 97] é um conjunto de ferramentas de simulação para processadores do estado-da-arte. A microarquitetura do processador simulado é configurável (execução fora de ordem, predição de desvio, graus de superescalaridade, hierarquia de memória, podendo ser analisada em módulos funcionais...). O SimpleScalar pode ser executado em diversos sistemas operacionais.

Neste trabalho serão apresentadas propostas de simulações com o SimpleScalar para utilização em cursos com forte apelo a áreas como Ciência da Computação, Sistemas de Informação, Engenharia da Computação e Análise de Sistemas, nas cadeiras de Organização e Arquitetura de Computadores. Neste caso os alunos poderão ter uma visão e avaliação quantitativa dos elementos que influem no desempenho global do processador.

Os processadores atuais obtêm incrementos espetaculares em seu desempenho mediante múltiplos e complexos mecanismos em sua microarquitetura. Além do que, o desempenho final do processador é determinado pela interação mútua entre estes mecanismos durante a execução de um programa e não pode ser determinado facilmente a priori.

Uma forma clássica de explicar o funcionamento dos mecanismos internos de um computador consiste em utilizar exemplos “de papel”. Estes são suficientes para que, em muitos casos, os alunos entendam cada mecanismo separado, mas trazem consigo o fato de proporcionarem um conhecimento parcial e pouco integrado do modelo de funcionamento do processador. É fácil que os alunos entendam um mecanismo sem saber valorizar o impacto que pode ter no desempenho final do processador, ou sem saber relacionar este impacto com as características do programa. Por exemplo: é fácil entender que a execução fora de ordem evita um problema concreto de dependência de dados. Todavia não é fácil explicar que características de um programa fazem com que a fila de instruções se sature. Este modelo de aprendizagem leva os alunos a esquecer facilmente tanto os mecanismos, como problemas e conceitos que estão por trás [MOU 2002].

O perfil dos cursos de tecnologia da informação e a experiência docente demonstram que os alunos aprendem melhor e estão mais motivados quando lhes propomos problemas a resolver que sejam os mais reais possíveis. Desta forma o aluno deve fazer uma seleção ativa e aplicar de forma efetiva aqueles conhecimentos básicos que foram introduzidos em classe. Esta prática permite integrar conceitos e técnicas em um esquema construído por eles mesmos e adquirindo um conhecimento mais sólido e duradouro.

Entretanto reproduzir a execução de um programa sobre um processador complexo, preenchendo tabelas à mão é uma tarefa complexa, pois existem muitos detalhes a considerar e é fácil esquecer algum deles. Por outro lado, os exemplos que podemos provar são poucos e pequenos. O mesmo problema se dá quando o professor realiza na classe uma explanação detalhada do funcionamento de um mecanismo mediante um exemplo. Na classe, cada aluno tem seu próprio ritmo de aprendizagem e o tempo dedicado à explicação pode tornar-se bastante ineficiente. Em casa, o aluno não tem uma resposta imediata da validade de sua própria análise, o que o impede de aprender de forma eficiente.

O ideal é que os alunos possam interagir com o sistema estudado para poder compreender melhor. Os simuladores proporcionam este tipo de disponibilidade a baixo custo e com uma possibilidade de acesso aos detalhes internos do sistema que geralmente não oferecem os sistemas reais. Os simuladores podem incorporar ajudas que, em tempo real, guiem os alunos à solução dos problemas encontrados. Quando os objetivos são simples e estão bem definidos, alcançar–lhes supõe a melhor prova de validação para os alunos que estão entendendo como o sistema funciona [MOU 2002].

Existem diversas formas de usar o simulador para realizar atividades docentes efetivas. Podemos utilizá-lo para realização de demonstrações na classe ou em trabalhos de auto-aprendizagem. Se o simulador disponível possui suficiente ajuda on-line, os alunos podem resolver pequenos problemas de forma autônoma. Esta ajuda pode ser complementada com a documentação adicional, em forma de manual do usuário, ou em forma de tutorial dirigido, no qual o aluno aprende a manejar o simulador resolvendo um exemplo.

Uma vez que os alunos dominam o simulador é possível realizar práticas mais ambiciosas. Estas poderiam envolver toda interface Software/ Hardware, como poderia ser a otimização de uma aplicação mediante a modificação tanto dos códigos binários como da microarquitetura do processador. Além de utilizar o simulador em nível de usuário, também é possível conceber sua implementação para melhor analisar como se implementam os mecanismos existentes, ou melhor, implementar novos mecanismos. O nível de conhecimentos requeridos para esta tarefa é alto, mas permite uma grande flexibilidade no desenho de práticas.

Contribuiu na motivação para este estudo a dificuldade encontrada na demonstração dos efeitos de mudança de parâmetros no projeto de processadores, em especial em arquiteturas superescalares, seja pela necessidade de recursos materiais adicionais em instituições de nível superior para o ensino de Arquitetura e Organização de Processadores, ou ainda pela dificuldade de encontrar ferramentas que auxiliem de forma didática a compreensão dos fenômenos oriundos dos projetos de processadores. A

proposta neste caso identificada foi essencialmente a difusão do estudo da cadeira, bem como a complementação prática.

Este trabalho tem como objetivo principal a apresentação de uma proposta de roteiros de simulações para auxiliar o ensino de conceitos ligados à Arquitetura de Computadores. Utiliza-se para tanto um conjunto de ferramentas de simulação criadas por D. C. Burger e T. M. Austin, denominado SimpleScalar [BUR97] e distribuído em código aberto na página oficial destas ferramentas.

Esta proposta está dividida em 5 partes: no capítulo 2 são analisados conceitos básicos para compreensão dos parâmetros a serem avaliados, no capítulo 3 é realizada a apresentação e descrição do conjunto de ferramentas SimpleScalar, além de instruções de como proceder à instalação deste conjunto de ferramentas, no capítulo 4 é apresentado o simulador WinDLX para comparações entre os simuladores, no capítulo 5 está sugerido um conjunto de simulações com resultados e análise para que professores e instrutores ligados a esta área possam ter parâmetros no uso desta poderosa ferramenta didática, e no capítulo 6 são realizadas considerações finais e conclusões extraídas neste trabalho.

2 CONCEITOS BÁSICOS

Desde o surgimento dos computadores estuda-se o paralelismo. Já na década de 40, Von Neumann debatia algoritmos para soluções de equações diferenciais com técnica de paralelismo. As idéias de executar tarefas simultaneamente e de antecipar atividades são amplamente utilizadas atualmente nas arquiteturas. Neste contexto são discutidos os seguintes conceitos: busca antecipada de instruções (lookahead); superposição de fases de execução de instrução (overlap), múltiplas unidades funcionais, estágios de execução (pipeline), processamento vetorial e multiprocessamento.

Assim, dentre os principais motivos para o surgimento da computação paralela citam-se: desempenho, modularidade, tolerância a falhas e etc. Os níveis de paralelismo mais evidentes são:

- Hardware (múltiplas unidades funcionais e pipeline)
- Micromáquina (microcódigo)
- Arquitetura (conjunto de recursos visíveis ao programador)
- Sistema operacional (concorrência de processos)
- Linguagem de programação (execução de partes do programa por unidades distintas)

2.1 Arquiteturas do Conjunto de Instruções (*ISA´s*)

A busca por processadores mais velozes que permitam a execução de programas de forma mais rápida tem levado os pesquisadores a desenvolverem novas e mais elaboradas organizações internas, o que, por sua vez, leva ao crescimento da complexidade da arquitetura destes processadores. Devido a esta complexidade, os projetistas adotaram o uso de várias camadas de abstração para facilitar a avaliação e a implementação destas arquiteturas.

Com diferentes camadas de abstração é possível visualizar o projeto do *hardware* de um processador em um nível mais alto, como, por exemplo, focar a atenção nas Unidades Lógicas Aritméticas (*Arithmetic Logic Units - ALUs*), no banco de registradores, e em outros blocos operacionais do processador. Do mesmo modo é possível também utilizar um nível de abstração mais baixo, como, por exemplo, visualizar a organização dos componentes eletrônicos utilizados na construção de uma *ALU*.

Dentre as diversas camadas de abstração destacamos a Arquitetura do Conjunto de Instruções (*Instruction Set Architecture - ISA*). A *ISA* descreve os vários aspectos do processador que são visíveis pelo programador, entre eles o conjunto de instruções suportadas [PAT 96].

Uma das grandes vantagens em utilizar abstrações, no projeto de computadores, é a possibilidade de construir diferentes processadores que possuem a mesma *ISA,* porém com arquiteturas internas diferentes. A família de processadores Intel 80x86 é um bom exemplo. É possível executar o código escrito para um processador 8086, projetado em 1978, em um Pentium Pro que possui uma arquitetura interna substancialmente diferente [PAT 98].

2.2 Simuladores de ISAs

Máquinas capazes de explorar o paralelismo na camada *ISA* realizam o Paralelismo a Nível de Instruções (*Instruction-Level Parallelism – ILP*).

Simuladores de *ISAs* são utilizados para prototipação de novas arquiteturas de processador. A utilização de ferramentas de software que permitam a simulação do funcionamento de um processador, tais como simuladores de *ISAs*, é uma alternativa viável em termos de tempo e custo ao invés da implementação diretamente em silício de arquiteturas experimentais.

Simuladores de *ISAs* podem ser classificados em dois grandes grupos, dependendo do modo como eles simulam a operação de um processador: os simuladores *execution-driven* e os simuladores *trace-driven*.

Em uma simulação *execution-driven*, o simulador que modela a máquina a ser estudada executa, completamente, os programas de teste usados para o estudo. Esta característica contrasta com uma simulação *trace-driven*, onde um simulador mais simples (um pré-processador) gera dados intermediários (*instruction traces*) que serão utilizados para alimentar um programa que modela a máquina que será estudada [SOU 99].

A simulação trace-driven é uma técnica para modelagem de desempenho e é particularmente útil para a modelagem de sistemas de memória de grande desempenho.

Na simulação trace-driven, é criado um rastreio das referências de memórias executadas, usualmente para cada simulação ou para uma execução instrumentada. O rastreio inclui quais instruções foram executadas (fixado o endereço das instruções), tanto quanto os endereços dos dados acessados [PAT 2003].

Existem outras caracterizações dos simuladores de *ISAs* que não serão abordadas aqui por fugir do objetivo principal deste trabalho: simuladores funcionais e simuladores de performance, simuladores *cycle time* e simuladores *instruction schedulers*, entre outros [AUS 97].

É fundamental que um simulador seja flexível de forma que os projetistas possam avaliar eficientemente múltiplas alternativas (número de unidades funcionais, características do preditor de desvio, hierarquia da cache, etc.) e suas interações. O ambiente deve permitir aos projetistas determinar o impacto destas alternativas no desempenho das várias organizações de processador em estudo. É importante que o

simulador execute o código rapidamente, pois cada experimento pode envolver a execução de centenas de milhões de instruções. Na prática, um experimento deve demorar horas e não dias [MOU 99].

2.3 Arquiteturas de Pipeline

A maioria das vantagens associadas aos processadores RISC, na realidade, advém de outras inovações em arquitetura que são possibilitadas por ela, e não pelo próprio número reduzido de instruções. A mais importante destas estratégias é o *pipeline*, apesar da memória *cache* embutida no chip também contribuir substancialmente para o desempenho como um todo. O *pipeline* é a extensão de uma idéia conhecida como "prefetch", na qual a próxima instrução a ser processada é carregada da memória e decodificada ao mesmo tempo em que se executa a instrução que está sendo processada.

Um processador completamente *pipeline* utiliza a mesma estratégia de uma linha de montagem de automóveis (na qual os veículos em vários estágios de montagem são trabalhados simultaneamente, porém cada um em uma operação específica, diferente, a ser efetuada em cada carro por vez). Portanto um *pipeline* de cinco estágios pode trabalhar com cinco instruções ao mesmo tempo. Enquanto a primeira instrução tem seus resultados guardados nos registradores de destino, a segunda pode estar executando uma operação aritmética. Após o *pipeline* ter sido preenchido inicialmente, a demora para completar as instruções é meramente o tempo de uma operação (idealmente um ciclo de *clock*), ao invés do tempo necessário para que todas as operações fossem processadas em seqüência.

A operação de um *pipeline* RISC hipotético de cinco estágios está ilustrada no diagrama de tempo da Figura 2.1. As instruções passam pelo *pipeline* da esquerda para a direita e são executadas em uma seqüência de cinco estágios de um ciclo. O tempo é medido de cima para baixo do diagrama em termos de ciclos que irão sendo completados.

	Busca Instrução	***Decodifica Instrução***	***Busca Operandos / Dados***	***Executa Instrução***	***Escreve o Resultado***
Ciclo 1	***Instrução 1***				
Ciclo 2	***Instrução 2***	***Instrução 1***			
Ciclo 3	***Instrução 3***	***Instrução 2***	***Instrução 1***		
Ciclo 4	***Instrução 4***	***Instrução 3***	***Instrução 2***	***Instrução 1***	
Ciclo 5	***Instrução 5***	***Instrução 4***	***Instrução 3***	***Instrução 2***	***Instrução 1***

Fluxo da Instrução⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒

Figura 2.1: Esquema Simplificado da Arquitetura Pipelined de 5 Estágios

No primeiro estágio, o processador busca uma instrução da memória e a transfere para o registrador de instruções, conforme indicado pelo contador do programa. No segundo estágio as instruções são decodificadas. No terceiro estágio é realizada a busca

de operandos caso a decodificação tenha identificado uma instrução aritmética ou busca de dados caso seja uma instrução load/ store. No quarto estágio a unidade aritmética e lógica (ULA) executa a operação, por exemplo uma adição, utilizando os operandos buscados no estágio anterior. Finalmente, no quinto estágio de escrita, os resultados são transferidos da saída da ULA ao registrador de destino.

No *pipeline* da Figura 2.1, cada instrução leva um total de cinco ciclos para completar sua execução. Mas à medida que as instruções se movem passo a passo através do *pipeline* e são executadas em paralelo, uma nova instrução é completada a cada um dos ciclos. Desta maneira, cinco instruções podem ser completadas em somente nove ciclos do relógio, opondo-se aos 25 ciclos que seriam necessários se elas fossem executadas separadamente em um processador que não fosse *pipeline*. O tempo que os quatro ciclos iniciais requerem para preencher o *pipeline* é conhecido como latência do processador. Em geral, *pipelines* mais longos oferecem menor granulação e, portanto, um processamento mais rápido, porém possuem uma latência mais alta.

Uma exigência importante da arquitetura *pipeline* é que para colher *os* benefícios da mesma as instruções enviadas ao *pipeline,* em qualquer momento, devem ser independentes. Se uma das instruções necessita do resultado de outra como seu operando, então estas duas instruções devem estar separadas de maneira que uma instrução não tenha que esperar por resultado de outra, pois dependendo dos estágios poderá haver travamento do *pipeline.* A segunda instrução teria que esperar para ler seus operandos de entrada até que a primeira completasse o estágio de escrita no registrador de destino.

Por outro lado, os processadores *pipeline* podem oferecer um desempenho mais próximo do ideal permitido pelo pipeline, se o compilador (ou o programador) tiver evitado tais dependências entre as instruções. Esta é outra razão pela qual desenhar um compilador otimizado para processadores *pipeline* (e para outras arquiteturas complexas que exploram o paralelismo) pode ser considerada uma tarefa desafiadora. Por estas e outras razões, o desenvolvimento do software tende a se atrasar em relação aos rápidos melhoramentos no desempenho do hardware. Também os sistemas operacionais mais antigos e a maioria dos pacotes de aplicações falharam em explorar por completo o potencial das novas plataformas baseadas em RISC.

Os processadores CISC já se beneficiam há bastante tempo da estratégia "prefetch", porém não se imaginava possível um *pipeline* completo nestes processadores porque eles não possuíam o pequeno conjunto de instruções simplificadas do RISC. Portanto, não haveria um conjunto fixo de operações comuns a todas as instruções.

Diferentemente dos processadores RISC, os CISC também não utilizavam de uma estratégia *load/store* uniforme para todas as instruções. Contudo, os últimos processadores CISC (por exemplo, o Intel Pentium e o 68050 da Motorola) possuem, de fato, *pipelines* e *cache* no chip. Isto foi conseguido com uma arquitetura híbrida, a qual a Intel chamou de CRISC (*complex reduced instruction set computing*).

Parece provável que os futuros processadores Intel da série x86 (e seus concorrentes como a Chips and Technologies, Cyrix e Advance Micro Devices) irão estender esta técnica adicionando *pipelines* mais potentes, expandindo assim a porção RISC de seus conjuntos complexos de instruções. Em termos de arquitetura, os

processadores serão realmente RISC, com áreas CISC embutidas que executarão as exceções que não forem próprias para o *pipeline*.

2.3.1 Desempenho no pipeline

A forma mais eficiente de acelerar a máquina, até aqui, é a construção do hardware partindo de várias unidades de execução de instruções, e colocá-las como numa linha de montagem conforme descrito na seção anterior.

Infelizmente estudos mostraram que cerca de até 30% de todas as instruções são desvios, e estes causam dano ao pipeline. Desvios podem ser classificados em três categorias: incondicionais, condicionais e loop. Um desvio incondicional diz ao computador para parar de buscar instruções em endereços seqüenciais e, consecutivamente, ir para algum endereço específico. Um desvio condicional testa alguma condição e desvia se a condição é satisfeita. Um exemplo típico é uma instrução que testa algum registrador e desvia se o registrador contém zero. Se o registrador não contém zero, o desvio não é feito e o controle continua na seqüência corrente.

Instruções de loop tipicamente decrementam um contador de iteração, e então desviam de volta ao início do loop se ele é diferente de zero (isto é, ainda existem mais iterações a serem feitas). Instruções de loop são casos especiais importantes de desvios condicionais, pois é conhecido a priori que os desvios quase sempre têm sucesso.

				CICLO					
	1º.Ciclo	**2º.Ciclo**	**3º.Ciclo**	**4º.Ciclo**	**5º.Ciclo**	**6º.Ciclo**	**7º.Ciclo**	**8º.Ciclo**	**9º.Ciclo**
Busca de instrução	**Inst.1**	**Inst.2**	**B**	**Inst.4**				**Inst.5**	
Decod. de instrução		**Inst.1**	**Inst.2**	**B**					
Busca de operando			**Inst.1**	**Inst.2**	**B**				
Execução				**Inst.1**	**Inst.2**	**B**			
Escreve Resultado					**Inst.1**	**Inst.2**	**B**		

Figura 2.2: Um pipeline de cinco estágios.

No pipeline da Figura 2.2 os números são usados para rotular as instruções. A instrução marcada B é um desvio condicional.

Considere o que acontece com o pipeline da Figura 2.2 quando uma instrução de desvio condicional é encontrada. A próxima instrução a ser executada pode ser a que se segue ao desvio, mas pode também estar no endereço para o qual deve ser desviado o controle, chamado de **destino do desvio.** Como a unidade que busca a instrução não sabe se o desvio irá ocorrer até que sua condição seja calculada, ela pára e não pode ir adiante até que o desvio seja executado. Consequentemente, o pipeline esvazia. Somente depois do ciclo 7 ter sido completado é que se sabe qual a instrução a seguir.

O desvio efetivamente causou a perda de quatro ciclos, chamados de **penalidade de desvio.** Com uma em cada três instruções sendo um desvio, fica claro que a perda de desempenho é substancial.

Grande quantidade de pesquisa tem tratado do problema de recuperar parte desse desempenho. A coisa mais simples a fazer é assumir que o desvio não será realizado, e apenas continuar enchendo o pipeline, como se o desvio fosse uma simples instrução aritmética. Se acontecer que o desvio realmente não foi realizado, não perdemos nada.

Se for, teremos que apagar as instruções atualmente no pipeline, algo chamado **squashing,** e começar de novo.

Squashing causa problemas próprios. Em algumas máquinas, como um subproduto do cálculo de endereço, um registrador pode ser modificado. Se a instrução que está sendo apagada modificou um ou mais registradores, estes devem ser restaurados, o que significa que deve haver um mecanismo de guardar seus valores originais. O desvio e formas de contornar o problema serão analisados no capitulo 2.6.

2.3.2 Superpipeline

Uma técnica alternativa para atingir alto desempenho no processamento é intitulada como *superpipeline*, termo utilizado pela primeira vez em 1988 [JOU 88]. Essa técnica explora o fato de que alguns estágios de *pipeline* executam tarefas que requerem menos de meio ciclo de *clock*. Assim, um processador que implemente esta técnica terá a velocidade de clock dobrada, e permitirá um aumento no desempenho por ter duas tarefas executadas em um único ciclo de clock.

Uma máquina usando *pipeline* básico executa uma instrução por ciclo de clock e tem um estágio de *pipeline* por clock também. Note que, ainda que várias instruções estejam sendo executadas concorrentemente, somente uma instrução está no estágio de execução de cada vez.

A implementação super*pipeline* é capaz de executar dois estágios de *pipeline* de cada vez. Uma forma alternativa de enxergar isso é entender que a função executada em cada estágio pode ser dividida em duas partes não sobrepostas e cada uma pode executar em meio ciclo de clock. Uma implementação super*pipeline* que se comporta deste modo é chamada de grau 2.

Veremos na próxima seção que a implementação superescalar pode executar duas ou mais instâncias de cada estágio em paralelo.

2.4 Arquitetura Superescalar

2.4.1 Introdução

Os termos escalar e vetorial são utilizados para distinguir os modelos de execução: os processadores escalares podem processar uma única instrução manipulando operandos discretos, enquanto que processadores vetoriais e matriciais utilizam a técnica de processar uma mesma instrução sobre vários dados em paralelo.

Além destes dois modelos, existe o que executa várias instruções concorrentemente, sem que elas precisem ter o mesmo código de operação (não vetoriais). Este modelo é utilizado pelos computadores superescalares, assim chamados por processarem instruções escalares em grande grau de paralelismo.

Computadores superescalares são máquinas projetadas para melhorar o desempenho de execução de instruções escalares. Em outras palavras, eles exploram o paralelismo em baixo nível, e assim, podem executar diversas instruções de máquina simultaneamente, contanto que todas sejam provenientes do mesmo programa objeto.

Como, na maioria das aplicações, a massa de operações são quantidades escalares, a "filosofia" superescalar representa o próximo passo na condução dos processadores de propósito geral para o alto desempenho, que podem, então, executar eficientemente uma larga gama de programas de aplicação não se limitando a aplicações vetoriais ou matriciais.

Um outro fator que permite a otimização do desempenho em processadores superescalares é a execução Fora-De-Ordem que faz o processador reordenar as instruções para melhor aproveitar o pipeline e evitar a inserção de bolhas. Ele necessita uma unidade de reordenação de instruções antes de executá-las. Ela pode existir em todos os tipos de pipeline. Neste tipo de execução teremos a possibilidade de redução de bolhas produzidas por instruções que dependem do resultado de outras instruções .

2.4.2 Limitações

A técnica superescalar depende da capacidade de executar múltiplas instruções em paralelo. O termo **instruction-level parallelism** refere-se ao grau no qual, em média, instruções de um programa podem ser executadas em paralelo. A combinação das técnicas de otimização baseada em compilador e baseada em *hardware* pode ser utilizada para maximizar o nível de paralelismo das instruções. Antes de analisar a técnica utilizada pelas máquinas superescalares para incrementar o nível de paralelismo, é necessário entender as limitações de paralelismo que o sistema deverá suportar.

Johnson [JOH 91] lista cinco limitações: dependência verdadeira de dados, dependência de desvios, conflitos de recursos, dependência de saída e antidependência.

Estas limitações são chamadas por alguns autores de dependências verdadeiras no caso das três primeiras, e de falsas dependências, no caso das duas últimas.

2.5 Arquiteturas CISC X RISC

Numa implementação da arquitetura superescalar de um processador, instruções comuns como aritmética inteira e de ponto flutuante, leitura e escrita na memória, e desvios condicionais, podem ser inicializadas simultaneamente e executadas independentemente, e tal implementação aumenta a quantidade de instruções executadas por intermédio de *pipeline* [STA 96.].

As implementações da arquitetura superescalar são feitas baseadas principalmente na arquitetura RISC, porém, elas podem ser feitas tanto na RISC como na CISC. O pequeno conjunto de instruções da arquitetura RISC torna mais eficiente a utilização de pipeline e da superescalaridade.

O principal objetivo de um projetista de computadores é minimizar o tempo de execução de qualquer programa. Podemos expressar este tempo de execução, *T*, como:

$$T = N \times C \times S$$

Onde *N* é o número de instruções que precisam ser executadas, *C* é o número médio de ciclos do relógio (*clock cycles*) do processador por instrução, e *S* é o número de segundos por ciclo.

Em uma primeira aproximação, *N, C,* e *S* são afetados primariamente pela tecnologia de compiladores, pela *ISA*, e pela tecnologia de implementação, respectivamente [RAU 89].

Uma maneira de reduzir *N* com o custo de um pequeno aumento em *C,* é explorar o paralelismo disponível em máquinas microprogramáveis horizontais [SAL 76]. Isto pode ser feito definindo instruções complexas que exploram o micro-paralelismo interno destas máquinas com o intuito de que *N* diminuirá mais bruscamente do que *C* aumentará. Esta é a idéia geral por trás dos Computadores com Conjunto de Instruções Complexas (*Complex Instruction Set Computers* - CISC).

Microprogramação pode ser utilizada para implementar *ISAs* CISC poderosas, com instruções complexas capazes de comandar várias operações envolvendo muitos operandos em registradores ou memória principal. Uma instrução CISC típica, por exemplo, pode ler um valor da memória, adicionar o valor ao conteúdo de um registrador interno e escrever o resultado em outra posição de memória.

Por outro lado, Computadores com Conjunto de Instruções Reduzidas (*Reduced Instruction Set Computer* – RISC) reduzem *T* utilizando instruções muito simples que reduzem *C* e, devido a sua simplicidade, *S* também. O aumento resultante em *N* pode ser minimizado por um ajuste fino entre a tecnologia de compiladores e a implementação das RISC *ISAs* correspondentes [PAT 85].

Instruções RISC geralmente comandam uma operação simples, não envolvendo mais do que um acesso à memória. Devido à sua simplicidade, instruções RISC não são implementadas usando microprogramação. Máquinas RISC são, portanto, equivalentes a máquinas microprogramáveis verticais expostas ao programador da camada *ISA*.

Melhorias na tecnologia de implementação também podem reduzir *T* através da redução de *S*. Entretanto, assumindo o uso da mais rápida tecnologia, qualquer aumento adicional no desempenho demandará a exploração do paralelismo disponível nos programas [SOU 99].

2.6 Predição de Desvios

2.6.1 Apresentação

Em alguns processadores, a unidade de controle realiza dinamicamente a predição de desvios. Usualmente, essas técnicas são mais eficientes do que as estáticas. Técnicas dinâmicas armazenam informações das instruções de desvio coletadas em tempo de execução e quando o desvio for novamente executado, o mecanismo de predição verifica o que ocorreu no passado mais recente e, baseado nessa informação, prediz qual o resultado será produzido pela instrução de desvio.

Predição de desvios pode ser pensada como uma forma sofisticada de tentar "adivinhar" o resultado de instruções de desvios através de especulação. Uma vez feita a predição, o processador pode especular a execução das instruções que dependem do resultado do desvio. Sem predição de desvios o processador tem que aguardar até que os desvios sejam resolvidos. Conforme já dito os desvios são instruções que ocorrem nos programas com uma freqüência muito alta (±30%). Devido a isso, um grande esforço foi despendido no sentido de se desenvolver técnicas eficientes para minimizar os *stalls* causados por esse tipo de instrução[YEH 91]**.**

Predição de desvios se tornou uma área de grande interesse devido a seu efeito no desempenho de pipeline e dos processadores superescalares. Vários métodos foram propostos para especular o caminho de um fluxo de instrução depois de um desvio. Esta

seção visa apresentar algumas das principais técnicas e métodos implementados de predição estáticas e dinâmicas de desvios, mostrando as principais características de cada uma dessas técnicas.

2.6.2 Predição Estática

Nesta técnica o resultado previsto para um desvio é sempre o mesmo e pode ser executado no nível de software, durante a compilação, ou no nível de hardware, em tempo de execução. As predições estáticas podem ser implementadas segundo três variações: o desvio sempre ocorrerá, nunca ocorrerá e o código da operação determina a predição.

A primeira técnica explora o fato de que a maioria dos desvios condicionais provocam uma transferência no fluxo de controle, prevendo que a sucessora de um comando de desvio é a instrução contida no endereço alvo. Logo, a unidade de busca de instruções busca o trecho de código alvejado ao invés do trecho de código adjacente ao comando de desvio. Esta técnica foi utilizada pelo IBM 360/91.

A segunda técnica de predição é implementada na maioria dos processadores que fazem busca antecipada de instruções (*look-ahead processors*). Assumindo que a transferência de controle nunca ocorrerá, a unidade de instruções continua buscando as instruções adjacentes ao comando de desvio. Esta técnica foi implementada em vários processadores, como por exemplo, o i960CA, MC68020 e o VAX 11/780.

A terceira técnica, ao invés de fixar uma única direção, leva em conta o código de operação do comando de ramificação para decidir se o fluxo de controle será transferido para o endereço alvo ou não, já que alguns códigos de operação de desvios tem uma tendência maior para um dos fluxos. Essa técnica utiliza resultados de estudos sobre o comportamento dos diversos tipos de comandos de desvio, levando em consideração a probabilidade de o controle ser transferido.

Com o objetivo de fixar a direção que será utilizada, o projetista monitora o comportamento dos comandos de desvio durante a simulação de programas representativos e determina o número de vezes que cada tipo de comando provocou uma transferência de controle. Com isso, utiliza uma freqüência dinâmica de cada tipo de desvio e a predição é fixada e armazenada, por exemplo, em uma memória ROM no interior do processador, a qual é consultada em tempo de execução para decidir se o desvio ocorrerá ou não.

2.6.3 Predição Dinâmica

Existem duas tendências em torno dos tipos de técnicas de redução dos custos dos desvios: técnicas implementadas em software e técnicas implementadas em hardware. O argumento empregado para defesa do uso das técnicas implementadas em software é que o projeto do hardware pode ser bastante simplificado, se o problema da redução de custo for transferido para um outro nível hierárquico do sistema. Por outro lado, os defensores da outra estratégia apontam que o decrescente custo do hardware justifica a implementação dessas técnicas diretamente na unidade de controle dos processadores.

Uma outra alternativa que tem atraído a atenção dos pesquisadores consiste em usar uma mistura dos dois tipos de técnicas para reduzir ainda mais o custo de execução dos comandos de desvio, o que tem apresentado ótimos resultados, se comparado às

técnicas estáticas ou as técnicas dinâmicas em isolado. Nesta seção serão apresentadas as técnicas de predição dinâmica para análise com simuladores.

Técnicas abordadas nesta seção [LEE 2003]:

- DHT – Direct History Table
- Predição via tabela com endereços e alvos dos desvios (BTA – Branch Target Address, BTB – Branch Target Buffer)
- Predição dinâmica na história do desvio (BHT – Branch History Table)
- Predição dinâmica de desvios utilizando contadores saturados (bimod)
- Predição dinâmica de desvios em 2 níveis (2lev)
- Preditores híbridos e multi-híbridos (comb)

2.6.3.1 DHT (Direct History Table)

Direct History Table (DHT) é o mais simples método de predição dinâmica de desvio. É baseado em uma tabela global de predição. Usualmente, os bits menos significativos do endereço da instrução são usados como índice da tabela. Assim, a predição do desvio é o "palpite" dado na linha correspondente da tabela [LEE 2003].

Quando o resultado do desvio passa a ser conhecido, a tabela, na linha correspondente, é atualizada para o novo valor.

2.6.3.2 BTA e BTB - buffer

É simples, embora com baixa precisão, prever um resultado do tipo: tomado ou não tomado, pois envolve apenas 50% de possibilidade de erro. Outro ponto fundamental é saber o destino do desvio. Portanto são duas questões diferentes: Irá desviar? Para onde?

Se o desvio for não tomado, o destino será a próxima instrução (PC++). Porém se o desvio for tomado, o destino tem que ser calculado em muitos casos, o que pode levar alguns ciclos de CPU. Para evitar isso, uma cache pode armazenar o destino do desvio, Branch Target Addresses - BTA ou Branch Target Buffer - BTB. PowerPC, UltraSparc e Pentium são alguns exemplos de processadores que possuem um BTA. Dependendo da organização da cache que guarda os BTAs, as implementações podem diferir. Por exemplo:

- PowerPC604 usa 64 entradas totalmente associativa
- PowerPC620 usa 256 entradas, 2-way
- Pentium usa 256 com 4-way
- Pentium Pro usa 512 com 4-way

No caso de predição via tabela com alvos dos desvios uma técnica alternativa para previsão é a que emprega uma tabela contendo os alvos das instruções de desvios. Denominada BTB - *Branch Target Buffer*. Essa tabela é uma evolução da tabela que contém a história dos desvios.

Como anteriormente, a tabela BTB inclui campos para identificar a instrução de desvio e para armazenar a história das recentes execuções do comando de desvio (ou o contador saturado). Adicionalmente, a BTB inclui um campo contendo informações

sobre a instrução sucessora do desvio: geralmente o campo armazena o endereço efetivo da sucessora; em outras implementações, a instrução sucessora também. A Figura 2.3 abaixo apresenta o diagrama de uma BTB.

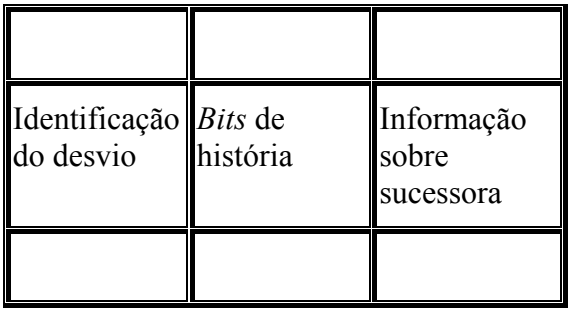


Figura 2.3: Organização da Branch Target Buffer

A BTB torna o processador mais eficiente por causa do potencial oferecido pelas informações sobre a sucessora do desvio. Em paralelo com a busca da próxima instrução, podemos detectar antecipadamente se ela é um desvio e se esse for o caso, qual o endereço f da instrução que deve suceder o comando de desvio que está sendo buscado. Através dessa antecipação, o estágio de busca pode ser prontamente redirecionado para o fluxo com maior probabilidade de execução, reduzindo, desse modo, a incidência de instruções introduzidas indevidamente nos estágios iniciais do pipeline.

A BTB funciona da seguinte maneira: o estágio de busca compara o endereço da instrução que está buscando com os endereços que estão na BTB. Se o endereço está na BTB então uma previsão é feita em função dos bits de história correspondentes. Se a previsão diz que o desvio será tomado, então o endereço no campo de destino será usado para acessar a próxima instrução. Quando o desvio é resolvido, no estágio de execução, a BTB pode ser corrigida com a informação correta sobre o que aconteceu com o desvio, caso a previsão feita anteriormente tenha sido incorreta.
O funcionamento é semelhante ao do mecanismo com BHT associativa, com apenas algumas diferenças. No caso de *miss*, o endereço destino também é inserido na entrada juntamente com o endereço do desvio. Quando acontece uma previsão incorreta, o endereço destino é atualizado para refletir o destino correto do desvio.
Em uma BHT associativa ou em uma BTB, a taxa de acertos depende de dois fatores:

- freqüência com que as informações de previsão são encontradas na tabela (*hit ratio*)
- freqüência de acertos na previsão de cada desvio.

Logo, a taxa de acerto é dada por:

$$T A = HitRatio * F A$$

onde:

HitRatio = frequência com que as informações são encontradas na tabela.

FA = freqüência de acerto na previsão de desvios e;

O segundo fator depende em parte do número de bits de predição. Com um número maior de bits é possível registrar a história dos desvios a partir de um passado mais distante. O primeiro fator depende da configuração da tabela de predição, ou seja, do número de entradas.

2.6.3.3 Predição dinâmica usando história do desvio (BHT - Branch History Table)

Essa técnica verifica o que ocorreu com as k mais recentes execuções de um desvio e realiza uma predição do resultado que será produzido pela corrente execução do desvio. Os k mais recentes resultados de cada desvio ficam armazenados numa Tabela da História dos Desvios (BHT - Branch History Table), mostrada na Figura 2.4, que é atualizada após a conclusão da instrução de desvio.

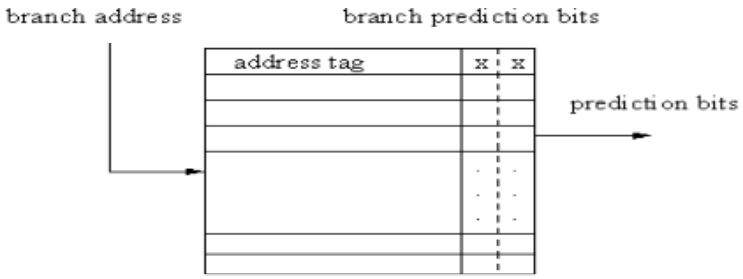


Figura 2.4: Predição dinâmica usando história do desvio

Fisicamente, as entradas contendo a história dos desvios podem ser armazenadas num conjunto de registradores ou então numa memória cache no interior do processador.

Quando uma instrução é acessada, ela é pré-decodificada para que seja determinado se aquela é uma instrução de desvio. Se for o caso, os bits menos significativos do endereço do desvio são utilizados para indexar a BHT. O bit na entrada selecionada fornece a previsão do desvio: por exemplo, 0 (zero) indica que o desvio será previsto como tomado, enquanto 1 (um) indicaria desvio não-tomado. A instrução a ser acessada no próximo ciclo é determinada de acordo com esta indicação.

No estágio de execução, o estado do bit na BHT é comparado com o resultado do desvio, para verificar se a previsão foi correta. Caso seja, a execução prossegue normalmente e, caso contrário, as instruções buscadas antecipadamente são descartadas e a busca é redirecionada para o destino correto.

Um esquema bastante simples de predição consiste em utilizar o resultado da última execução da instrução de desvio. Nesse caso, um bit seria suficiente para armazenar o resultado anterior da instrução de desvio. Se a predição indicar que o desvio deve ser tomado e se o estágio de execução indicar o contrário, a tabela BHT é atualizada, as instruções nos estágios precedentes são descartadas e o estágio de busca inicia a transferência de instruções pertencentes ao fluxo apropriado. Se a instrução de desvio estiver sendo executada pela primeira vez, utiliza-se uma das duas técnicas estáticas apresentadas previamente e em seguida, inclui-se o desvio na BHT. O autômato para esse mecanismo é muito simples e é mostrado na Figura 2.5.

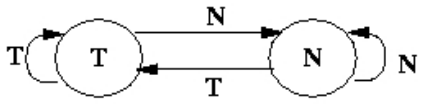


Figura 2.5: Autômato para predição com 1 bit de história

O número de bits de história (previsão) é um fator de extrema relevância na escolha do algoritmo de previsão. Acima foi mostrado um autômato para previsão com 1 bit de história. O maior problema em se usar esta técnica é quando se faz necessário prever o destino de desvios de controle do laços, e o laço é executado mais de uma vez (loops aninhados). Para n iterações de um loop, as primeiras n-1 iterações são tomadas e o desvio ao final do loop é previsto corretamente. Entretanto, ao final da última iteração e o desvio é não-tomado e a previsão é incorreta, uma vez que, durante a última execução o desvio foi tomado.

A próxima vez que o loop é executado, o desvio de controle é tomado e mais uma vez o algoritmo erra a previsão, pois da última vez o desvio fora não-tomado. Usando-se 1 bit de história é necessário uma previsão errada para que o algoritmo passe a prever a situação inversa. Se a previsão inicial é T e o resultado é não-tomado, o algoritmo passa a prever não-tomado na próxima execução.

Outro esquema que poderia ser utilizado é o mecanismo com 2 bits de história, assim é possível registrar o resultado das duas últimas execuções, e a próxima previsão é modificada apenas se as duas últimas previsões foram incorretas. A Figura 2.6 abaixo mostra o autômato utilizado para a previsão com 2 bits. Nos estados onde os dois bits coincidem, a previsão segue o resultado indicado por ambos. Nos estados onde os dois bits diferem, a previsão segue a indicação do bit que registra o estado mais antigo. Estudos realizados mostram que, com 2 bits de previsão, é possível alcançar uma taxa média de acerto individual de 90%.

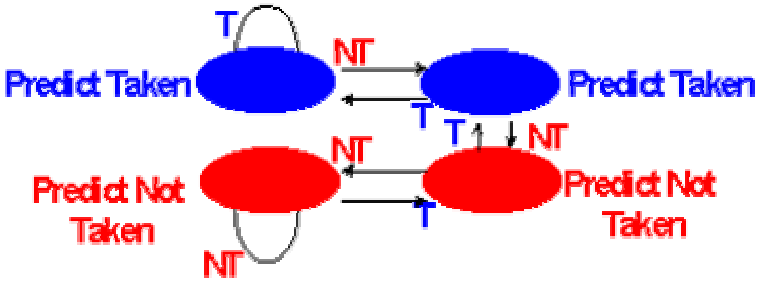


Figura 2.6: Autômato de predição com 2 bits de história

2.6.3.4 Predição Dinâmica de Desvios utilizando Contadores Saturados

Nesta técnica, os desvios possuem um contador de controle que permitirá, a partir do comportamento do desvio, indicar um “palpite” para o desvio.

Se o contador não for negativo, a técnica prediz que o desvio será tomado. Caso contrário, a predição indicará que a instrução adjacente é a sucessora.

Após a execução do comando de desvio, o contador será incrementado ou

decrementado, se o desvio for respectivamente tomado ou não. A operação incrementar / decrementar é inabilitada quando um dos valores extremos do contador for atingido.

Esta técnica retorna predições mais precisas do que as técnicas baseadas nos bits de história. Aumentando o limite de variação dos contadores não implica necessariamente em percentagens de acerto mais intensas: contadores com maior capacidade normalmente apresentam uma inércia maior para neutralizar o efeito provocado por uma outra instrução de desvio mapeada na mesma entrada.

2.6.3.5 Predição dinâmica em dois níveis

Esta técnica é usada no Pentium PRO, onde se armazena em uma tabela (usando a parte menos significativa de k bits para endereçar a tabela) o comportamento passado do desvio. Uma tabela local é indexada e é obtido o comportamento passado (se tomado o desvio ou não). Este comportamento é usado para indexar uma segunda tabela (Pattern History Table) onde há um contador com a predição do desvio. Neste caso é realizada uma atualização em caso de acerto (deslocamento do bit) [YEH 91].

Nessa técnica, o primeiro nível armazena a história dos últimos K desvios encontrados. O segundo nível armazena o que aconteceu com as últimas *j* ocorrências de um padrão específico para os K desvios.

O primeiro nível é denominado *Local History Table* e o segundo nível de *Pattern Table.* O endereço de um desvio é mapeado para acessar o primeiro nível. Após mapear a entrada correta, o registrador de história (*Global Branch History)* fornece o padrão de bits que irá determinar qual entrada será acessada no segundo nível.

Ao acessar o segundo nível, o mecanismo dispõe então do bit de predição que indicará o caminho a ser seguido pelo estágio de busca para acessar as instruções seguintes[LEE 2003].

Após a instrução de desvio ser executada, o resultado (tomado, não-tomado) é deslocado para dentro do registrador de história, da entrada correspondente no primeiro nível, modificando o padrão para os desvios que mapearão aquela entrada futuramente. A lógica de transição de estado avalia o resultado do desvio juntamente com a predição feita anteriormente para este, e fornece o novo bit de predição que será usado posteriormente. Isto pode ser verificado com auxílio do esquema geral demonstrado na Figura 2.7.

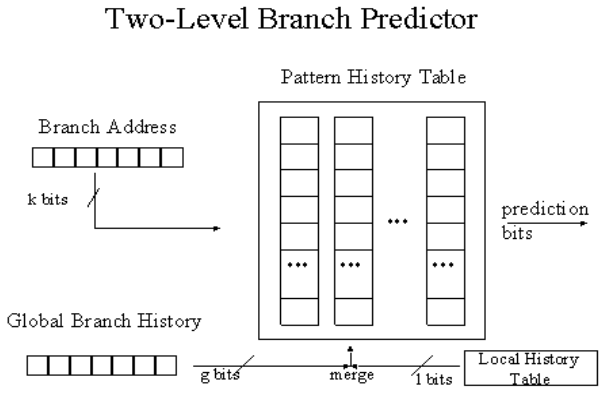


Figura 2.7: Esquema Geral de Preditor Dois Níveis

2.6.3.6 Preditores híbridos e multi-híbridos

Os preditores híbridos e multi-híbridos pareceram somente a partir de 1996 nos projetos de processadores e atualmente são os que apresentam as taxas de acerto mais precisas, podendo atingir patamares superiores a 99%. Um fator que pesa muito nas técnicas híbridas é o custo e grau de complexidade.

Preditores híbridos incluem diversas técnicas, todas operando em paralelo, mas somente a técnica com maior probabilidade de acerto é a que fornece o resultado da predição para a unidade de busca de instruções. Um exemplo de preditores híbridos é a combinação de uma DHT com uma BHT, mostrado na Figura 2.8.

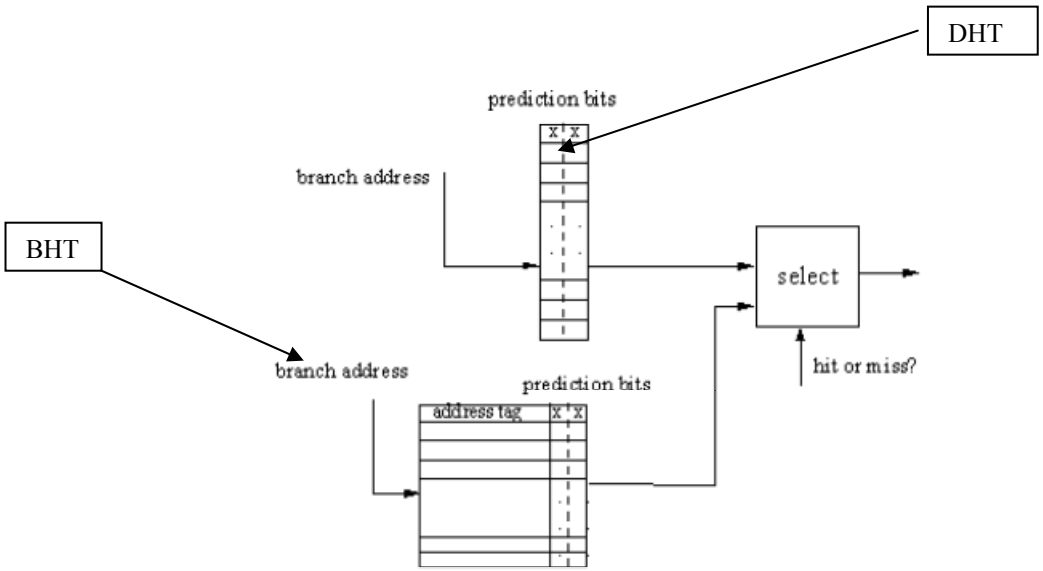


Figura 2.8: Esquema de Combinação DHT-BHT

Neste esquema o acesso é primeiramente feito na BHT. Se há um hit, então é usada a predição de desvio a partir da BHT. Caso contrário, a DHT é mapeada e a predição da DHT é usada [LEE 2003].

A principal vantagem deste esquema é poder fazer uma DHT muito maior que BHT por causa de suas baixas exigências de hardware.

Como ambos os preditores operam independentemente eles também são atualizados independentemente. O seletor (que consiste numa tabela de contadores de dois bits e é mapeada pelos k menos significativos bits do endereço de salto dado pela instrução) é atualizado de acordo com a tabela mostrada junto à Figura 2.9 a seguir.

Predictor 1	**Predictor 2**	**Update to Selector**
Correct	Correct	no change
Correct	Incorrect	Increment
Incorrect	Correct	Decrement
Incorrect	Incorrect	no change

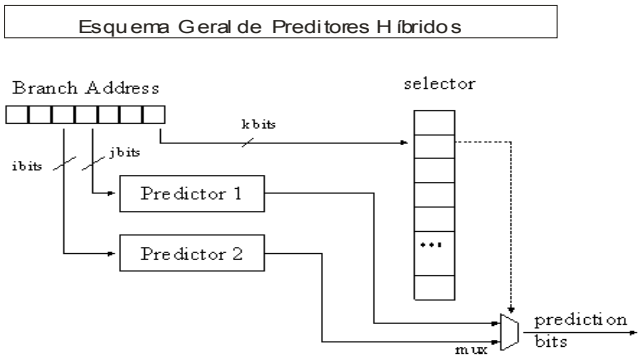


Figura 2.9: Esquema Geral de Preditores Híbridos

2.7 Memória Cache

Considerando que todos os telefones de uma cidade podem ser encontrados no guia telefônico, a agenda pessoal de telefones funciona como se fosse uma cache (telefones) que precisam ser examinadas por alguém. A memória pequena e rápida é chamada **cache** (do francês cacher, que significa esconder) e foi escolhida para designar o nível da hierarquia de memória situado entre o processador e a memória principal, na primeira máquina comercial que implementou esse nível extra de memória. Atualmente, apesar desta designação continuar a ser dominante para a palavra *cache*, o termo também é usado para designar qualquer memória gerenciada de modo a tirar vantagem da localidade de acesso. As memórias cache apareceram, primeiramente, nas máquinas desenvolvidas para pesquisa, no início da década de 1960, sendo implementadas em máquinas comerciais um pouco mais tarde, ainda na mesma década; virtualmente todas as máquinas comerciais desenvolvidas hoje, desde as mais rápidas até as mais lentas, incluem uma cache.

Historicamente, as CPUs sempre foram mais rápidas que as memórias. Enquanto as memórias melhoraram, as CPUs melhoraram muito, aumentando assim a diferença. Isso significa na prática que depois que a CPU faz uma requisição à memória, ela deve permanecer inativa por um tempo substancial enquanto estiver esperando que a memória responda. É comum que a CPU estabeleça uma leitura à memória durante um ciclo de barramento e não obtenha o dado até dois ou três ciclos mais tarde, mesmo que não haja estados de espera.

Realmente, o problema não é tecnológico, mas econômico. Os engenheiros sabem como construir memórias tão rápidas quanto as CPUs, mas são tão caras que equipar um computador com um megabyte ou mais está fora da questão (exceto, talvez para supercomputadores, onde o céu é o limite, e preço não é objeção). Assim, a escolha cai para ter uma pequena quantidade de memória rápida ou uma grande quantidade de memória lenta. O que iremos preferir é uma grande quantidade de memória rápida a um baixo preço.

São conhecidas técnicas bastante interessantes para combinar uma pequena quantidade de memória rápida com uma grande quantidade de memória lenta para obter (quase) a velocidade da memória rápida e a capacidade de grande memória a um preço moderado.

Quando um programa é executado em um computador, a maioria das referências é feita para um pequeno número de posições na memória. Tipicamente 90% do tempo de execução de um programa são gastos em apenas 10% do código. Esta propriedade é conhecida como **princípio de localidade.** Quando um programa tem como referência uma posição de memória, é provável que referencie a mesma posição de memória em breve, o que é conhecido como **localidade temporal.** De forma similar existe **localidade espacial**, onde uma posição de memória que está próxima a uma posição de memória recentemente acessada tem uma chance maior de ser referenciada do que uma posição localizada mais longe. Localidade temporal acontece porque os programas gastam muito do seu tempo em iterações ou em recursividade e, portanto, a mesma seção de código é visitada um número de vezes desproporcionalmente grande. Localidade espacial acontece porque dados tendem a serem armazenados em posições contínuas. Muito embora em torno de 10% do código seja responsável pela maioria do acesso a referências na memória, acessos dentro dos 10% tendem a serem agrupados. Portanto, para um dado intervalo de tempo, a maioria dos acessos à memória vem de um conjunto ainda menor de posições do que os 10% do tamanho do programa.

O acesso à memória é normalmente lento quando comparado com a velocidade da CPU e, portanto a memória se torna um gargalo significativo no desempenho do computador. Uma vez que a maioria das referências vem de um conjunto pequeno de posições, o princípio de localidade pode ser explorado a fim de melhorar o desempenho. Uma **memória cache** pequena mais rápida, na qual o conteúdo das posições mais acessadas é mantido, pode ser colocada entre a CPU e a memória. Quando um programa executa, a memória cache é pesquisada primeiro e a palavra referenciada é acessada na cache se estiver presente. Se a palavra referenciada não estiver presente, então uma posição livre é criada na cache e a palavra referenciada é trazida da memória principal para esta posição. A palavra é então acessada a partir da cache. Muito embora este processo gaste mais tempo do que simplesmente acessar a memória principal diretamente, o desempenho global é significativamente melhor se uma proporção grande dos acessos forem satisfeitos pela cache.

Sistemas de memória modernos podem ter diversos níveis de cache, chamados de nível 1 (L1), nível 2 (L2) e, em alguns casos, nível 3 (L3). Na maioria das vezes, a cache L1 é implementada dentro do chip da CPU. Ambos os processadores Intel Pentium e IBM- Motorola Power PC G3 têm 32 Kbytes de cache L1 na CPU.

Uma memória cache é mais rápida do que a memória principal por diversas razões. Componentes eletrônicos mais rápidos podem ser usados, o que pode resultar em um grande gasto em termos de dinheiro, tamanho e necessidade de energia. Considerando que a cache seja pequena, este aumento de custo é relativamente pequeno. Uma memória cache tem menos posições do que a memória principal, e como resultado tem uma árvore de decodificação de endereços mais curta, o que reduz o tempo de acesso. A cache é colocada física e logicamente mais próximo à CPU do que a memória principal, e esta situação evita atrasos de comunicação em um **barramento compartilhado** [MUR 2000]**.**

Uma situação típica é mostrada na Figura 2.10. Um computador simples sem memória cache é mostrado no lado esquerdo da figura. Este computador sem cache contém uma CPU com velocidade de 400MHz, mas se comunica com a memória principal através de um barramento de 66MHz, sendo que a memória tem uma velocidade ainda inferior, de 10MHz. Alguns ciclos do clock são necessários

normalmente para sincronizar a CPU com o barramento e, portanto a diferença em velocidade entre a memória principal e a CPU pode ser de um fator de 10 ou mais. Uma memória cache pode ser posicionada próxima à CPU como mostrado no lado direito da Figura 2.10, e desta forma a CPU vê acessos rápidos através de um caminho direto de 400 MHz até a cache.

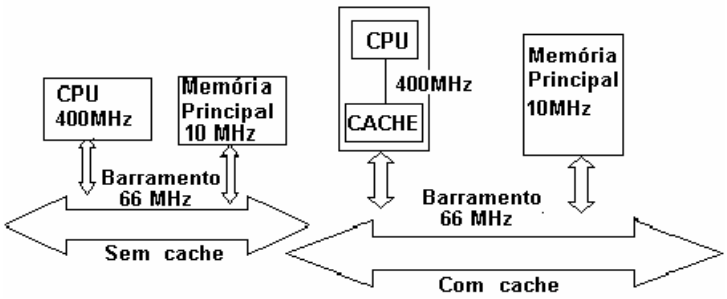


Figura 2.10: Posição da cache em um sistema computacional

2.7.1 Cache mapeada de forma associativa

Diversos mecanismos foram desenvolvidos para traduzir endereços da memória principal para endereços da memória cache. O usuário não precisa saber como esta tradução é feita. Isto traz a vantagem de que as melhorias na memória cache podem ser introduzidas em um computador sem modificações nos softwares aplicativos.

A escolha de mecanismos de mapeamento de cache afeta custo e desempenho, e não existe um único método que seja apropriado para todas as situações. A Figura 2.11 mostra um mapeamento associativo para um espaço de memória com 2^{32} palavras dividido em 2^{27} **blocos** de $2^5 = 32$ palavras do bloco. A memória principal não é particionada fisicamente desta forma, mas é a esta visão que a cache tem da memória principal. Blocos de cache, ou **linhas de cache**, como são conhecidas, tipicamente variam entre 8 e 64 bytes de tamanho. Os dados são colocados e retirados da cache em linhas completas, usando memória entrelaçada.

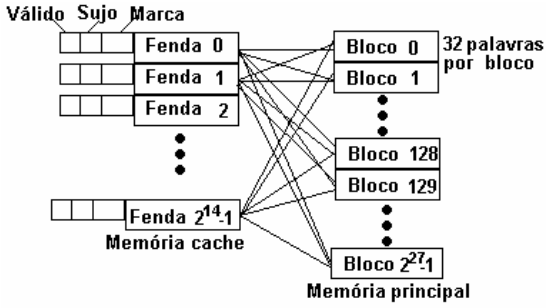


Figura 2.11: Memória cache mapeada de forma associativa

A cache para este exemplo consiste em 2^{14} **fendas**[1] nas quais blocos de memória são colocados. Existem mais blocos de memória principal do que fendas, e qualquer um dos 2^{27} blocos de memória principal pode ser mapeado em qualquer fenda da cache

[1] N.A – Foi utilizado o termo "fenda" como tradução para "slot" da versão original [MUR 2000].

(mas somente um bloco colocado em uma fenda a cada instante). Para manter controle de quais blocos dos 2^{27} está em cada fenda, um campo de **marca** de 27 bits é adicionado a cada fenda que contém um identificador no intervalo de 0 a 2^{27} - 1. O campo de marca guarda os 27 bits mais significativos dos 32 bits do endereço de memória presente na cache. Todas as marcas são armazenadas em uma memória especial onde podem ser pesquisadas em paralelo [MUR 2000]. Quando um bloco novo é armazenado na cache, sua marca é armazenada na posição correspondente da memória de marcas.

Quando um programa é carregado inicialmente na memória principal, a cache é limpa e, portanto, enquanto o programa está sendo executado, um bit **válido** é necessário para indicar se uma fenda contém ou não um bloco que pertence ao programa sendo executado. Existe também um bit **sujo** que controla se o bloco foi ou não modificado enquanto está na cache. Uma fenda que é modificada deve ser escrita de volta à memória principal antes que a fenda seja reutilizada para outro bloco [STA 2002].

Uma posição referenciada que é achada na cache resulta em um acerto (**hit**); caso contrário resulta em um erro (**miss**). Quando um programa é carregado inicialmente na memória, os bits válidos são todos zerados. A primeira instrução que é executada no programa consequentemente causará um miss, uma vez que não existe nenhuma parte do programa na cache neste momento. O bloco que causou o miss é localizado na memória principal e carregada na cache.

Se qualquer marca na memória de marcas da cache for igual ao campo de marcas da referência de memória, então a palavra é retirada da posição da fenda especificada no campo palavra. Se a palavra referenciada não for achada na cache, então o bloco da memória principal que contém a palavra é lido para dentro da cache e a palavra referenciada é retirada do mesmo. Os campos marcas, válido e sujo são atualizados e o programa continua sua execução.

Vejamos agora como um acesso à posição de memória $(A035F14)^{16}$ é mapeada na cache. Os 27 bits mais à esquerda do endereço formam o campo marca, e os cinco bits restantes formam o campo palavra como se vê abaixo.

Marca	Palavra
101 0000 0001 1010 1111 1000 0000	10100

Se a palavra endereçada está na cache, ela será achada na palavra $(14)_{16}$ de uma fenda com a marca $(501AF80)_{16,}$ onde marca é constituída usando-se os 27 bits mais significativos do endereço. Se a palavra endereçada não estiver na cache, então o bloco correspondente à marca $(501AF80)_{16}$ será trazido da memória principal para uma fenda disponível na cache e a referência à memória que causou o "cache miss" será respondida a partir da cache.

Muito embora este mecanismo de mapeamento seja poderoso o suficiente para atender a uma grande quantidade de situações de acesso à memória, existem dois problemas de implementação que limitam seu desempenho. Em primeiro lugar, o processo de decidir qual fenda deve ser liberada, quando um novo bloco é lido, pode ser complexo. Este processo exige uma quantidade significativa de hardware e introduz atrasos no acesso à memória. Um segundo problema que é, quando a cache é pesquisada, o campo de marca do endereço referenciado deve ser comparado com todos os 2^{14} campos de marca do cache [MUR 2000].

2.7.1.1 Política de reposição em caches mapeadas de forma associativa

Quando um novo bloco precisa ser colocado em uma cache mapeada de forma associativa, uma fenda disponível deve ser identificada. Se existem fendas livres, como por exemplo existe quando o programa inicia sua execução, a primeira fenda com um bit válido zero pode ser usada. Quando todos os bits válidos de todas as fendas das caches forem 1, contudo, uma das fendas ativas deve ser liberada para o novo bloco [STA 2002].

Quatro políticas de reposição que são comumente usadas são: a menos recentemente usada (**least recently used** – LRU), primeira a entrar primeira a sair (**first in first out** – FIFO), menos freqüentemente usada (**least frequently used** – LFU) e **aleatória.**

Para a política LRU, um selo temporal (timestamp) é acrescentado a cada fenda, que é atualizado quando a fenda é acessada. Quando uma fenda precisa ser liberada para um novo bloco, o conteúdo da fenda menos recentemente usada, como identificado pela idade do selo temporal correspondente, é descartada e o novo bloco é escrito na mesma.

A política LFU funciona de forma semelhante, quando uma fenda é necessária para um novo bloco, a menos freqüentemente usada é liberada. A política FIFO substitui fendas de forma cíclica (round-robin), uma após a outra na ordem de suas posições físicas na cache. A reposição aleatória simplesmente escolhe uma fenda aleatoriamente [MUR 2000].

Estudos mostram que a política LFU não é muito melhor que a política aleatória. A política LRU pode ser implementada de forma eficiente, e muitas vezes é escolhida por esta razão.

2.7.1.2 Vantagens e desvantagens da cache mapeada de forma associativa

A cache mapeada de forma associativa tem a vantagem de qualquer bloco de memória pode ser colocado em qualquer fenda. Isto significa que independente de quão irregulares são as referências a dados e ao programa, se uma fenda está disponível para um bloco, ele pode ser armazenado na cache. Isto resulta em um custo adicional de hardware significativo para se gerenciar a cache. Cada fenda deve ter uma marca de 27 bits que identifica sua posição na memória principal. Isso significa que, no exemplo anterior, a memória de marcas deve ter 27 x 2^{14} bits, e como descrito anteriormente, deve existir um mecanismo para buscar e comparar as marcas em paralelo. Memórias que podem ser pesquisadas pelo seu conteúdo em paralelo são chamadas de **associativas**, ou **endereçáveis por conteúdo**.

Restringindo-se onde cada bloco pode ser colocado na cache podemos dispensar o uso de uma memória associativa. Este tipo de cache é chamado de **cache mapeada diretamente** [MUR 2000].

2.7.2 Cache mapeada diretamente

A Figura 2.12 mostra um mecanismo de mapeamento direto para uma memória de 2^{32} palavras. Como anteriormente, a memória é dividida em 2^{27} blocos de $2^5 = 32$ palavras por bloco, e a cache consiste em 2^{14} fendas. Existem mais blocos de memória do que fendas da cache, e um total de $2^{27} / 2^{14} = 2^{13}$ blocos de memória principal podem ser mapeados em cada fenda da cache. Para se manter o controle de qual dos 2^{13} blocos

possíveis está em cada fenda, um campo de marca de 13 bits é acrescentado a cada fenda que armazena um identificador no intervalo de zero a $2^{13} - 1$.

Este mecanismo é chamado de "mapeamento direto" porque cada fenda da cache corresponde a um conjunto explícito de blocos da memória principal. Para uma cache mapeada diretamente, cada bloco da memória principal pode ser mapeado para somente uma fenda, mas cada fenda pode receber mais de um bloco.

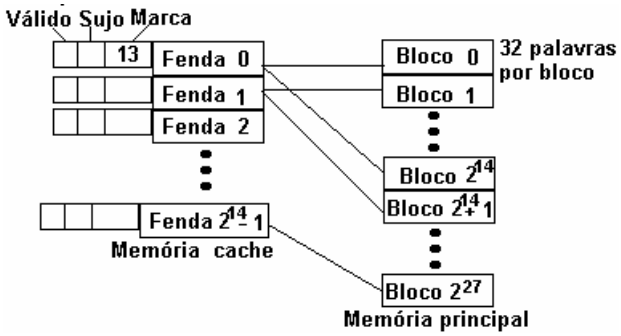


Figura 2.12: Memória cache mapeada diretamente

O mapeamento dos blocos da memória principal para fendas da cache é feito particionando-se um endereço em campos para a marca, a fenda e a palavra, como abaixo:

Marca	Fenda	Palavra
13 bits	14 bits	5 bits

Os 32 bits do endereço da memória principal são particionados em um campo de 13 bits de marca, seguido por um campo de 14 bits de fenda, seguido por um campo de cinco bits de palavras. Quando uma referência é feita a um endereço na memória principal, o campo fenda identifica em qual das 2^{14} fendas um bloco será encontrado se estiver na cache. Se o bit válido for 1, o campo marca do endereço referenciado é comparado com o campo marca da fenda. Se as marcas forem as mesmas então a palavra é retirada da fenda na posição especificada pelo campo palavra. Se o bit válido for 1, mas as marcas não forem as mesmas, então a fenda é escrita para a memória principal se o bit sujo for um, e o bloco de memória principal correspondente é lido para a fenda. Para um programa que começou sua execução recentemente, o bit válido será 0, e, portanto, um bloco é simplesmente escrito na fenda. O bit válido para o bloco é então setado para 1, e o programa continua sua execução.

Vejamos agora como um acesso à posição de memória $(A035F014)_{16}$ é mapeado na cache. O padrão de bits é particionado no formato mostrado anteriormente. Os 13 bits mais à esquerda formam o campo marca, os 14 bits seguintes formam o campo fenda e os cinco bits restantes, o campo palavra:

Marca	Fenda	Palavra
1 0100 0000 0110	10 1111 1000 0000	10100

Se a palavra endereçada estiver na cache, será encontrada na palavra $(14)_{16}$ da fenda $(2F80)_{16}$ a qual terá a marca $(1406)_{16}$.

2.7.2.1 Vantagens e desvantagens da cache mapeada diretamente

Cache mapeada diretamente é um mecanismo relativamente simples de implementar. A memória de marca do exemplo anterior seria de apenas 13 x $2^{14,}$ menos da metade da cache mapeado associativamente. Além disso, não há necessidade de uma busca associativa, uma vez que o campo fenda do endereço da memória principal da CPU é usada para "direcionar" a comparação da única fenda onde o bloco pode estar para saber se ele realmente está na cache [STA 2002].

A simplicidade tem seu custo. Considere o que acontece quando o programa referencia posições que estão separadas por 2^{19} palavras, que é o tamanho da cache. Este padrão pode acontecer naturalmente se uma matriz for armazenada na memória que será acessada por colunas. Cada referência vai resultar em um miss, o que fará com que o bloco inteiro seja lido para dentro da cache, embora somente uma única palavra se ajuste. Pior ainda, somente uma pequena fração da memória cache disponível será usada.

Pode parecer que qualquer programador que escreva um programa desta forma merece o desempenho ruim que resultará dele, mas na verdade cálculos matriciais e rápidos usam dimensões em potência de 2 (que permitem operações de deslocamento em vez de operações caras de multiplicação e divisão para indexação de vetores) e, portanto, o pior caso de acessar posições de memória que estão 2^{19} distantes uma da outra não é tão improvável. Para evitar essa situação, sem pagar o alto preço de uma implementação de uma cache completamente associativa, o mecanismo de **mapeamento associativo por conjunto** pode ser usado, que combina aspectos de mapeamento direto e de mapeamento associativo [MUR 2000]. Um mapeamento associativo por conjunto, também conhecido como **mapeamento direto do conjunto,** é descrito a seguir.

2.7.3 Cache mapeada de forma associativa por conjunto

O mecanismo de mapeamento associativo por conjunto combina a simplicidade do mapeamento direto com a flexibilidade do mapeamento associativo. Mapeamento associativo por conjunto é mais prático do que mapeamento completamente associativo porque a porção associativa é limitada apenas a algumas fendas que constituem um conjunto, como ilustrado na Figura 2.13. Para este exemplo, dois blocos fazem um conjunto e, por conseguinte, esta é uma cache associativa por conjunto de **dois caminhos.** Se existem quatro blocos por conjunto, chama-se cache associativa por conjunto de quatro caminhos.

Uma vez que existem 2^{14} fendas na cache, existem $2^{14}/2 = 2^{13}$ conjuntos. Quando um endereço é mapeado para um conjunto, o mecanismo de mapeamento direto é usado e então mapeamento associativo é usado dentro do conjunto. O formato de um endereço tem 13 bits no campo conjunto, que identifica o conjunto no qual a palavra endereçada pode ser encontrada se estiver na cache. Existem cinco bits para o campo palavra, como anteriormente, e existe um campo de marca de 14 bits que juntos completam os 32 bits do endereço mostrado abaixo:

Marca	Conjuntos	Palavra
14 bits	13 bits	5 bits

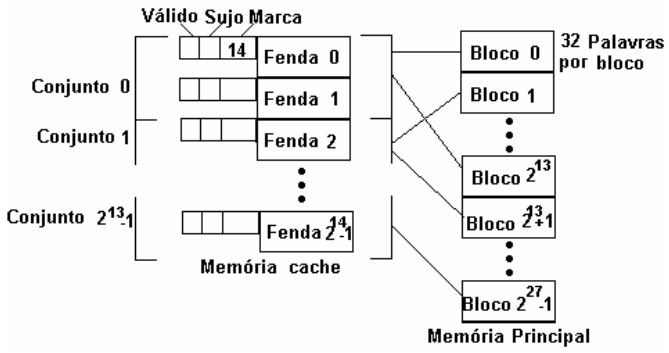


Figura 2.13: Memória cache mapeada de forma associativa por conjunto

Como um exemplo de cache associativa por conjunto vê um endereço na memória principal, considere mais uma vez o endereço $(A035F014)^{16}$. Os 14 bits mais à esquerda do campo marca, seguidos dos 13 bits do campo conjunto, seguidos dos 5 bits do campo palavra, são mostrados a seguir:

Marca	Conjunto	Palavra
10 1000 0000 1101	0 1111 1000 0000	1 0100

Como antes, o particionamento do campo endereço é conhecido somente pela cache, e o resto do computador desconhece qualquer tradução do endereço.

2.7.3.1 Vantagens e desvantagens da cache mapeada de forma associativa por conjunto

No exemplo anterior, a memória de marca é somente um pouco maior do que no caso de mapeamento direto, 13 x 2^{14} bits, e somente duas marcas precisam ser pesquisadas para cada referência à memória. A cache associativa por conjunto é usada quase que universalmente nos processadores modernos.

2.7.4 Desempenho do cache

Note que podemos trocar uma cache mapeada diretamente por um associativo, ou associativo por conjunto, sem quaisquer modificações no computador ou software. A única diferença entre os diferentes métodos é seu desempenho em tempo de execução [STA 2002].

Melhorar o desempenho em tempo de execução é a razão para usarmos uma memória cache, e existem várias questões que precisam ser resolvidas, por exemplo como fazer para que um bloco seja movido entre cache e a memória principal. Políticas de leitura e escrita em cache são resumidas na Figura 2.14. As políticas dependem do fato de a palavra pedida estar ou não na cache. Se uma operação de leitura em cache for feita e o dado referenciado estiver na cache, então há um “cache hit” e o dado referenciado é enviado imediatamente à CPU [MUR 2000]. Quando um cache miss acontece, o bloco completo que contém a palavra referenciada é lido da memória principal para a cache.

Em algumas organizações de cache, a palavra que causa o miss é imediatamente enviada à CPU tão logo seja lida para a cache, em lugar de esperar pelo preenchimento do resto da fenda, o que é conhecido como uma operação de leitura – direta (**load-through**). Para uma memória não intercalada, se a palavra ocorre na última posição do bloco, não existe um ganho de desempenho uma vez que a fenda toda é lida antes da leitura - direta. Para uma memória intercalada, a ordem de acesso pode ser organizada de tal forma que uma operação de leitura – direta resulte sempre em ganho de desempenho.

Para operações de escrita, se a palavra estiver na cache, então há duas cópias da palavra, uma na cache e outra na memória principal. Se ambas são atualizadas simultaneamente, o processo chama-se escrita – direta (**write through).** Se a escrita for adiada até que a linha de cache seja descarregada na memória, o processo é chamado de escrita – retorno (**write back).** Mesmo que o item de dado não esteja na cache quando a escrita acontecer, existe a opção de trazer o bloco contendo a palavra para dentro do cache e então atualizá-lo, em um processo conhecido como escrita –alocada (**write allocate),** ou atualizar a memória principal sem modificar a cache, conhecido como escrita – sem – alocação (**write no allocate**) [STA 2002].

Alguns computadores têm caches separados para instruções e dados, o que é uma variação de uma configuração conhecida como **arquitetura Harvard** (também conhecida como **split cache**). Como instruções nunca podem estar sujas (a não ser que escrevamos código automodificável, o que é raro hoje em dia), uma cache de instruções é mais simples do que uma cache de dados. Em favor desta configuração, observou-se que a maioria do tráfego de memória move-se para fora da memória principal, ao invés de em direção à mesma. Estatisticamente, existe somente uma operação de escrita para cada quatro operações de leitura da memória. Uma razão para isto é que as instruções de um programa em execução são lidas da memória principal e nunca escritas, exceto pelo carregador do sistema. Outra razão é que operações e dados tipicamente envolvem a leitura de dois operandos e a escrita de um resultado, o que significa que existem duas leituras para cada escrita. Uma cache que trata somente de leituras, enquanto escritas são enviadas diretamente à memória principal, pode ser eficiente. Além disso, um dos principais motivos para divisão da cache em dados e instruções é a possibilidade de paralelismo no acesso a ambos, evitando paradas no pipeline e consequentemente aumentando o desempenho da cache [MUR 2000].

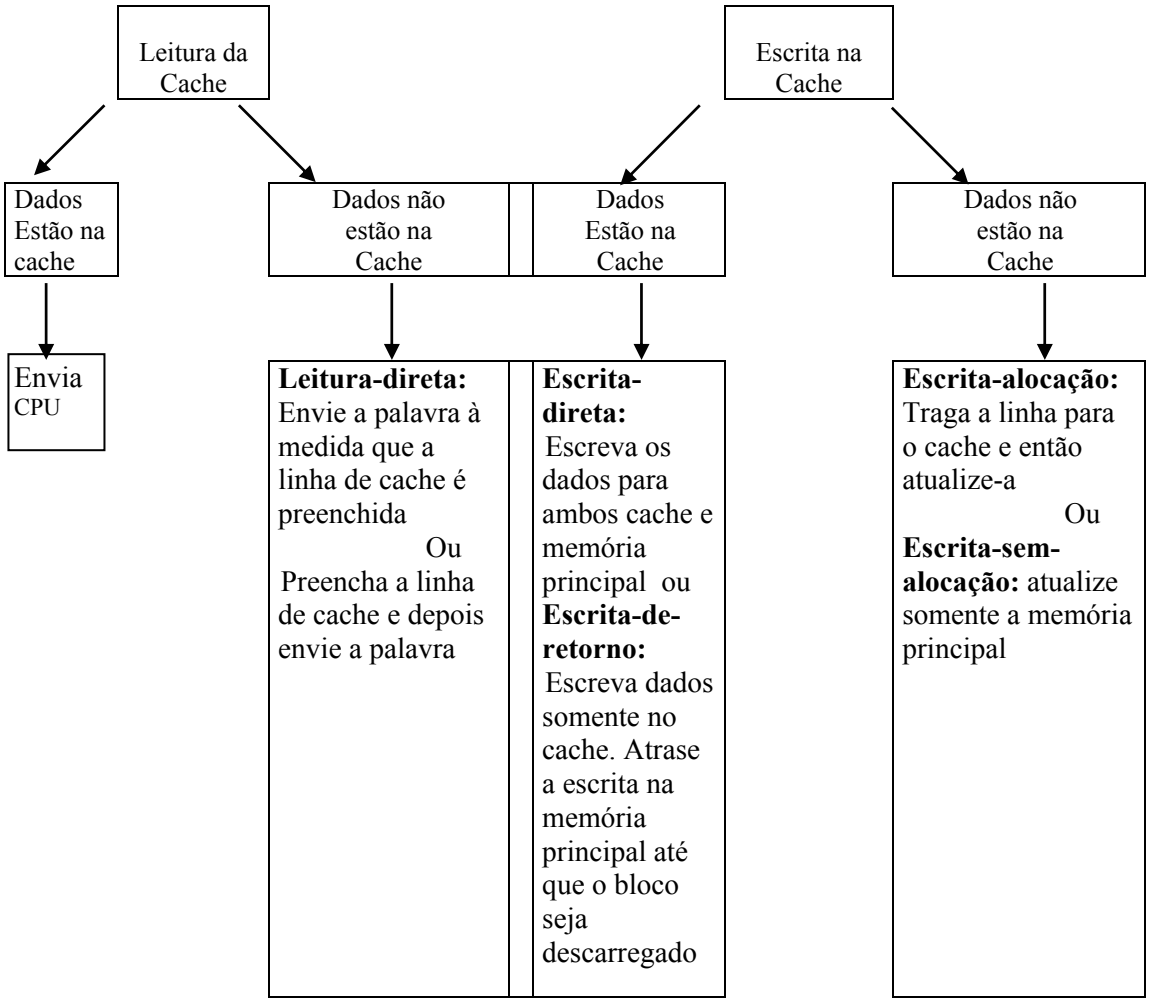


Figura 2.14: Políticas de leitura e escrita em cache

Não existe uma resposta simples sobre qual política de leitura e escrita é melhor. A organização de uma cache é otimizada para cada arquitetura de computador e a combinação de programas que o computador executa. Organização da cache e tamanho do mesmo são normalmente determinados pelos resultados de simulações que expõem a natureza do tráfego de memória [MUR 2000].

2.7.5 Taxas de acerto e tempo de acesso efetivo

Duas medidas que caracterizam o desempenho de uma memória cache são a taxa de acerto (**hit rate**) e o **tempo de acesso efetivo.** A taxa de acerto é calculada dividindo-se o número de vezes que palavras referenciadas são encontradas na cache pelo número total de referências à memória. O tempo de acesso efetivo é calculado dividindo-se o tempo total gasto acessando à memória (somando os tempos de acesso a cache e à memória principal) pelo número total de referências à memória.

As equações correspondentes são:

$$\text{Taxa de acerto} = \frac{\text{N}^{\text{o.}}\text{ de vezes que palavras referenciadas estão na cache}}{\text{Número total de acessos à memória}}$$

$$\text{Tempo de acesso efetivo} = \frac{(\text{n}^{\text{o}}\text{ hits})(\text{tempo por hit}) + (\text{n}^{\text{o}}\text{ miss})(\text{tempo por miss})}{\text{Número total de acessos à memória}}$$

Podemos calcular o impacto produzido pela cache no desempenho, medindo o impacto da taxa de acertos no tempo de acesso efetivo a memória:

$$\mathbf{Tce = Tc + (1-h)\,Tm}$$

onde:

Tce = tempo efetivo de acesso à memória cache, considerando o efeito dos misses.
Tm = tempo de acesso à memória principal
Tc = tempo de acesso à memória cache.
h = taxa de hits

Considerando Tc = 10 ns e Tm = 70 ns, teremos valores como:

Taxa de hits	0,7	0,8	0,9	1
Tce	31ns	24 ns	17 ns	10 ns

Percebemos que para uma taxa de acertos de 0,7 (acertos de 70% nas buscas na cache) teremos o triplo do tempo médio de acesso em relação à cache sem missess.

Mas podemos aprofundar esta análise considerando o conceito de CPI (Ciclos por instrução), a penalidade por faltas e o conceito de cache perfeita.

Na tabela acima foi apresentado como último valor a taxa de acertos 1. Este valor significa que não há misses na busca de dados e instruções na memória cache. Este conceito, chamado cache perfeita, permite analisar e comparar o desempenho da cache. O desempenho da cache é fator importante no desempenho do processador, pois cada vez que é buscado um dado ou instrução na cache e não é encontrado é produzida uma penalidade por faltas, ou seja, o processador perde um certo número de ciclos para busca do dados na memória principal e escrita na cache. Isto é normalmente quantificado em ciclos de clock.

Uma métrica de desempenho de processador muito utilizada é CPI = nº médio de ciclos de clock por instrução. Esta é uma média sobre todas as instruções executadas no programa [PAT 98].

$$\mathbf{CPI} = \frac{\text{nº de ciclos de clock do programa}}{\text{nº instruções do programa}}$$

Na maioria das caches organizadas segundo a técnica write – through, as penalidades para faltas de leitura e escrita são iguais (ambas correspondem ao tempo

para buscar um bloco na memória). Se admitirmos que as paradas são desprezíveis, podemos combinar as leituras e as escritas usando uma só taxa de faltas e uma única penalidade por faltas:

Ciclos de clock referentes à memória = Total de acessos à mem. x Taxa de Faltas x Penalidade por faltas parada

Pode-se reescrever esta equação da seguinte forma:

Ciclos de clock referentes à memória = no. total de Instruções x Faltas por x Penalidade por faltas parada Instrução

Em [PAT 98] é demonstrado que:

Tempo de processador com paradas	=	no. de Instruções x $CPI_{paradas}$ x Ciclo de Clock
Tempo de processador para Cache perfeita		no. de Instruções x $CPI_{perfeita}$ x Ciclo de Clock

Com a relação acima podemos comparar o desempenho de um processador em relação à cache perfeita. Em alguns roteiros do capitulo 5 serão utilizadas algumas das relações acima como forma de avaliação do desempenho da cache e fixação de conceitos.

2.7.6 Caches multiníveis

Na medida em que o tamanho dos circuitos integrados e a densidade de componentes nos mesmos aumentou, tornou-se possível incluir memória cache dentro do mesmo circuito integrado do processador. Uma vez que a comunicação dentro do chip é mais rápida que a comunicação entre chips, uma cache dentro da pastilha pode ser mais rápida do que uma cache fora do chip. Contudo, a tecnologia atual ainda não permite uma densidade de componentes tão alta, que seja possível colocar a cache completa na mesma pastilha que o processador. Por esta razão, **caches multiníveis** foram desenvolvidas nos quais o nível mais rápido da cache, L1, está no mesmo chip que o processador e os outros níveis da cache ficam localizados fora do processador. Caches de dados e instruções são mantidas separadas na cache L1. A cache L2 é **unificada**, o que significa que a mesma cache armazena os dados e instruções [MUR 2000].

Para se calcular a taxa de acerto e o tempo de acesso efetivo para a cache multinível, os hits e misses devem ser guardados para ambos as caches. Equações que representam a taxa de acerto global e o tempo de acesso efetivo, para uma cache de dois níveis, são mostradas a seguir. H_1 é a taxa de acerto para a cache dentro do chip, H_2 é a

taxa de acerto para a cache fora do chip e T_{EFF} é o tempo de acesso efetivo global. O método pode ser estendido para qualquer número de

$$H_1 = \frac{\text{Número de vezes que a palavra acessada está em L1}}{\text{Número total de acessos à cache L1}}$$

$$H_2 = \frac{\text{Número de vezes que a palavra acessada está em L2}}{\text{Número total de acesso à cache L2}}$$

$$T_{EFF} \quad \frac{\textbf{(No. de hits L1) (tempo de hit L1) + (No. de hits L2) (tempo de hit L2)+(No. de misses L2) (tempo de miss L2)}}{\text{Número total de acessos à memória}}$$

2.7.7 Gerenciamento de cache

O gerenciamento de uma memória cache apresenta um problema complexo ao programador do sistema. Se uma dada posição de memória representa uma porta de entrada e saída, o que pode acontecer em sistemas mapeados em memória, então provavelmente esta posição não deve aparecer na cache hora nenhuma. Se ela estiver na cache, o valor na porta de entrada e saída pode mudar, e esta mudança não será refletida no valor que está armazenado na cache. Isto é conhecido como dado "obsoleto": a cópia que estava na cache é "obsoleta" comparada com o valor na memória principal. Da mesma forma, em ambientes multiprocessados com memória compartilhada, onde mais de um processador pode acessar a mesma memória principal, o valor na cache e o valor na memória principal podem se tornar obsoletos por causa da atividade de uma ou mais CPUs. No pior caso, a cache em um ambiente multiprocessado deve implementar uma política "escrita – direta" para aquelas linhas que são mapeadas em posições de memória compartilhadas.

Por essas razões, dentre outras, a maioria das arquiteturas de processadores modernos permite que o programador do sistema tenha algum controle sobre a cache. Por exemplo, a cache do processador Motorola PPC 601 que normalmente usa uma política de "escrita – de – retorno" pode ser transformado em uma política "escrita – direta", para linhas individuais. Outras instruções permitem que linhas sejam especificadas como "cacheáveis", ou marcadas como inválidas, carregadas, ou descarregadas.

3 SimpleScalar

3.1 Origens

O conjunto de ferramentas SimpleScalar (*SimpleScalar Tool Set*) [BUR 97] foi desenvolvido na Universidade de Wisconsin-Madison por D. C. Burger e T. M. Austin. Proposto em linguagem C e código aberto possibilita a alteração de seu código permitindo o desenvolvimento de pesquisas baseadas em seus resultados. É largamente utilizado em centros de pesquisas de arquiteturas de computadores ao redor do mundo. Ele é utilizado para realizar simulações precisas, flexíveis e rápidas de arquiteturas superescalares.

O conjunto de ferramentas SimpleScalar é um sistema de infra-estrutura de software usado para construir e modelar aplicações para programas de análise de desempenho, modelando detalhadamente a microarquitetura e fazendo a co - verificação hardware e software.

Essa ferramenta implementa a arquitetura SimpleScalar, que é muito semelhante à arquitetura MIPS [PRI 95]. A ferramenta possui duas versões, uma *big-endian* e outra *little-endian*, para manter a portabilidade da mesma entre diversos sistemas. A semântica do conjunto de instruções (ISA) é um superconjunto da ISA do MIPS-IV, sendo chamada de PISA (*Portable Instruction Set Architecture*). O conjunto de instruções PISA (Portable Instruction Set Architecture) é um conjunto mantido primariamente para uso educacional. O GNU baseado no compilador cruzado GCC e bibliotecas pré-construídas também estão disponibilizados com este objetivo. O objetivo do PISA é particularmente o aproveitamento para o ensino de engenharia da computação onde as ferramentas podem ser utilizadas em uma larga faixa de plataformas.

As principais diferenças entre as duas arquiteturas MIPS e PISA são listadas a seguir:

• não existem slots de atraso; loads, stores e transferências de controle não executam a instrução sucessora. A técnica de "*delayed slots*" (slots de atraso) consiste em sempre executar a instrução sucessora (ou instruções sucessoras), no caso do MIPS, instruções de desvio e de acesso a memória. É apenas uma reordenação do código, para poder executar instruções após um desvio sem que o mesmo tenha sido resolvido. No caso do MIPS sempre uma instrução é executada (talvez um NOP se o compilador não achar nada).

• *loads* e *stores* suportam dois modos de endereçamento para todos tipos de dados em adição aos encontrados na arquitetura MIPS. São eles: indexado (registrador+registrador) e auto incremento/decremento.

• uma instrução de raiz quadrada, a qual implementa raiz quadrada em ponto flutuante de precisão simples e dupla.

• uma codificação estendida de instruções em 64-bits.

Como dito anteriormente, cada instrução possui tamanho fixo e igual a 64 bits, além de três formatos: registrador, usado para instruções de computação de um modo geral; imediato, que suporta a inclusão de uma constante de 16 bits; e, salto, que suporta a especificação de um endereço alvo de 24 bits para uma instrução de *jump*.

As principais vantagens dessa ferramenta são a alta flexibilidade, portabilidade, extensibilidade e desempenho. São incluídos seis simuladores orientados à execução de funções, isto é, as instruções são funcionalmente executadas, os quais serão mais detalhados logo abaixo. Além disso, a ferramenta disponibiliza binários pré-compilados (incluindo o SPEC95) e uma versão modificada do compilador GNU GCC (com seus utilitários associados), o qual permite a compilação de códigos fontes C (ou Fortran, através da ferramenta f2c) próprios do usuário. Um depurador e um visualizador de *pipeline* em modo texto também são providos pela ferramenta.

Usando as ferramentas SimpleScalar os usuários podem modelar aplicações para simular programas reais rodando sobre a faixa dos modernos processadores e sistemas com recursos parametrizáveis. O conjunto de ferramentas inclui desde simples simuladores com rápidos resultados até detalhados simuladores com modelos de processamento com execução fora de ordem e execução especulativa, "no blocking" caches e o estado da arte na predição de desvios. As ferramentas SimpleScalar têm sido usadas no ensino e pesquisa. Em 2000, mais de um terço de todos os artigos publicados na área de arquitetura de computadores em conferências usavam às ferramentas SimpleScalar para validar seus projetos. Adicionando aos simuladores, as ferramentas SimpleScalar incluem ferramentas para visualização de desempenho, recursos de debug e análise estatística.

3.2 Suporte

O conjunto de ferramentas SimpleScalar está disponibilizado para instalação no site www.SimpleScalar.com possibilitando uma visão geral do SimpleScalar 3.0. Nele são apresentadas as características dos simuladores que compõem o pacote de simulação e a organização modular do SimpleScalar que permite a utilização de diferentes ISAs.

Os simuladores SimpleScalar podem emular os conjuntos de instruções dos processadores Alpha [Digital 1992], da antiga DEC (Digital Equipament Corporation), hoje Compaq, a ARM, PISA (Portable ISA), derivada da ISA MIPS, e x86. O conjunto de ferramentas possibilita a definição da estrutura de processador o que permite que a maioria dos detalhes arquiteturais sejam separados e originados nas simulações. Todos os simuladores distribuídos com a corrente versão do SimpleScalar podem rodar programas oriundos de qualquer dos conjuntos de instruções listados acima. A emulação de conjuntos de instruções complexos (ex. x86) pode ser implementada com ou sem microcódigo, fazendo uso de ferramentas apropriadas para modelamento do conjunto de instruções CISC.

O SimpleScalar foi construído para rodar na maioria das plataformas de 32-bits e 64-bits baseadas nos Sistemas Operacionais UNIX e WindowsNT. A arquitetura de software do conjunto de ferramentas inclui um módulo de interface para utilização com telas gráficas, permitindo rápida e fácil portabilidade para o estudo de funcionalidades e desempenho.

3.3 Características Gerais

O SimpleScalar, em sua versão 3.0, possui algumas características que o distingue dos demais simuladores de arquitetura.

- **Expansível –** O código fonte (escrito na linguagem C [KER 86]) de todas as ferramentas está incluído na distribuição, o que permite ao usuário expandir o conjunto de instruções de uma ISA.
- **Portável –** É possível executá-lo na maioria das plataformas Unix disponíveis no mercado (a Tabela 3.1 exibe as plataformas nas quais o SimpleScalar foi testado).
- **Detalhado –** O SimpleScalar 3.0 inclui, na verdade, vários simuladores do tipo *execution-driven* (ver descrição no item 2.1). O simulador Sim-Outorder, por exemplo, suporta execução de desvios tomados errados, especulação de controle de dados, etc.
- **Desempenho –** O simulador *Sim-Fast,* um dos simuladores do pacote SimpleScalar 3.0, quando rodando em um IBM-PC com processador Pentium Pro 200-MHz possui desempenho de 4 MIPS (Milhões de Instruções Por Segundo). O simulador Sim-Outorder, por sua vez; tem desempenho 200 KIPS (Quilo Instruções Por Segundo).

Tabela 3.1: Portabilidade do SimpleScalar

Arquitetura	Sistema Operacional	Compilador
x86	Free BSD 2.2	Gcc
x86	Cyg Win32/Windows NT	Gcc
x86	Linux 1.3	Gcc
x86	Solaris 2	Gcc
SPARC	SunOS 4.1.3	Gcc
SPARC	Solaris 2	Gcc
RS6000	AIX 4.1.3	Gcc
RS6000	AIX 4.1.3	Xle
PA-RISC	HPUX	Gcc
Alpha	DEC Unix 3.2	Gcc
Alpha	DEC Unix 3.2	c89

3.4 Descrição do conjunto de simuladores

O pacote SimpleScalar é composto por oito simuladores e algumas ferramentas de auxílio a estes simuladores. A seguir, uma breve descrição dos simuladores é apresentada.

Sim-Fast

É o simulador mais simples, rápido e menos detalhado (simulação funcional). É ideal para treinamento pois exige poucos parâmetros de entrada para realizar a simulação. Este simulador retorna estatísticas como: taxa de instruções executadas por segundo, número total de instruções. Não verifica alinhamento de dados / instruções. Executa as instruções seqüencialmente.

Sim-Safe

Versão melhorada do Sim-Fast, permite alta velocidade de simulação e retorna os mesmos resultados do Sim-Fast. Acrescenta resultados como o total de acesso a dados (Loads / Stores) executados, verifica se há erros nas instruções e permite a limitação no total de instruções executadas, sendo possível fixar o tempo para finalização de simulação dentro de um período de aula, por exemplo. Verifica alinhamento e permissões nas referências à memória. Executa as instruções seqüencialmente.

Sim-Profile

O simulador Sim-Profile tem como função principal a determinação do perfil dos programas de avaliação de desempenho (benchmarks). Este simulador fornece estatísticas como o número de instruções executadas, tipos de instruções, número de vezes que foram executadas e distribuição de classes de instrução. A perfilação dos benchmarks a serem utilizados já é um bom motivo para sua aplicação uma vez que irá municiar o aluno ou usuário do conjunto de ferramentas com dados para todos os outros simuladores. Quando executado gera um grande volume de estatísticas (classes de instruções executadas, classes de endereço, acesso à memória, desvios e etc). Verifica alinhamento e permissões nas referências à memória. Executa as instruções seqüencialmente.

Sim-Cache

Sim-Cache é um simulador funcional de memória cache que possibilita o teste em cache de instrução, cache de dados e cache unificada, e ainda avalia caches até dois níveis. No relatório de saída podem ser obtidos parâmetros como: valores absolutos de acertos e erros na busca de dados e instruções na cache, taxas de acertos e erros, número de instruções testadas, tempo de simulação, acessos em cada nível de cache, velocidade de simulação (instruções por segundo).

Nos parâmetros de entrada é possível estabelecer a configuração para a simulação como mostrado na tabela abaixo:

–config	configurar o simulador via arquivo externo
-h	é possível imprimir auxílio na tela
-d	habilitar mensagem de debug
-redir:sim	redirecionar saída da simulação
-redir:prog	redirecionar saída do programa
-max:inst	limitar o número de instruções testadas
-cache:il1	configurar parâmetros como : número de conjuntos, tamanho de bloco, associatividade e política de reposição da cache de instruções
-cache:dl1	configurar parâmetros como : número de conjuntos, tamanho de bloco, associatividade e política de reposição da cache de dados.
-cache:ul1	configurar parâmetros como : número de conjuntos, tamanho de bloco, associatividade e política de reposição da cache unificada.

O formato do comando de configuração é mostrado abaixo:

```
      The cache config parameter <config> has the following format:

          <name>:<nsets>:<bsize>:<assoc>:<repl>

          <name>   - name of the cache being defined
          <nsets>  - number of sets in the cache
          <bsize>  - block size of the cache
          <assoc>  - associativity of the cache
          <repl>    - block replacement strategy, 'l'-LRU, 'f'-FIFO,
'r'-random

          Examples:   -cache:dl1 dl1:4096:32:1:l
                      -dtlb dtlb:128:4096:32:r
```

Sim-Cheetah

O Sim-Cheetah é um simulador de cache semelhante ao Sim-Cache[Rab 93]. Permite a simulação de várias configurações de cache em apenas uma execução do simulador. Isto simplifica a simulação de um benchmark. O desempenho da cache, nas simulações, pode ser avaliado em grande faixa de associatividade e número de conjuntos. Todos os tipos de mapeamento de cache citados na introdução podem ser testados, como mapeamento direto, mapeamento completamente associativo ou conjunto associativo. Das políticas de reposição efetivamente só poderá ser implementada a LRU (Menos recentemente utilizado) neste simulador.

O objetivo principal desta versão de simulador de cache é agilizar os resultados de simulação para cache de tal forma que possamos contar com resultados sem ter que parametrizar item por item.

Sim-Eio

O simulador Sim-Eio é usado para gerar *external event traces* (EIO traces) e *checkpoint* dos executáveis. Os *external event traces* capturam a execução de um programa e armazenam os dados em um simples arquivo para uma re-execução mais tarde. O simulador sim-eio também provê a funcionalidade de gerar *checkpoints* em qualquer ponto arbitrário de execução em conjunto com o EIO trace, permitindo que a execução de qualquer simulador do SimpleScalar comece a partir de *checkpoint.*

Sim-Bpred

O Sim-Bpred é um simulador de preditores de desvio e tem como função principal o teste de desempenho de preditores estáticos e dinâmicos como por exemplo preditor bimodal, 2 níveis, combinado e etc. O Sim-Bpred possuí muitos parâmetros de controle de entrada comuns aos outros simuladores do conjunto como por exemplo:

–config	configurar o simulador via arquivo externo
-h	é possível imprimir auxílio na tela
-d	habilitar mensagem de debug
-redir:sim	direcionar saída da simulação
-redir:prog	direcionar saída do programa
-max:inst	limitar o número de instruções testadas
-bpred	configurar parâmetros como técnicas específicas de predição de desvio (nottaken, taken, perfect, bimodal [Mcfarling93], 2-level adaptive predictor [Yeh91], combined –2 level e bimodal)

Podemos utilizar nas simulações os parâmetros tanto em predições estáticas como dinâmicas, conforme o descritor abaixo:

```
-bpred            bimod # branch predictor type {nottaken|taken|bimod|2lev|comb}
-bpred:bimod     2048 # bimodal predictor config (<table size>)
-bpred:2lev    1  1024  8  0  #  2-level  predictor  config  (<l1size>  <l2size>  <hist_size>
<xor>)
-bpred:comb      1024 # combining predictor config (<meta_table_size>)
-bpred:ras                  8 # return address stack size (0 for no return stack)
-bpred:btb       512 4 # BTB config (<num_sets> <associativity>)

Branch predictor configuration examples for 2-level predictor:
  Configurations:   N, M, W, X
    N   # entries in first level (# of shift register(s))
    W   width of shift register(s)
    M   # entries in 2nd level (# of counters, or other FSM)
    X   (yes-1/no-0) xor history and address for 2nd level index
Sample predictors:
  GAg     : 1, W, 2^W, 0
  GAp     : 1, W, M (M > 2^W), 0
  PAg     : N, W, 2^W, 0
  PAp     : N, W, M (M == 2^(N+W)), 0
  gshare  : 1, W, 2^W, 1
Predictor `comb' combines a bimodal and a 2-level predictor.
```

Além disto este simulador verifica alinhamento e permissões nas referências à memória e permite a execução das instruções seqüencialmente.

Sim-Outorder

Este simulador introduz a possibilidade de análise de tempo/ritmo das simulações, permitindo o teste das características funcionais e desempenho das configurações.

Enquanto os simuladores Sim-Cache e Sim-Cheetah simulam somente o estado da memória do sistema, o simulador Sim-Outorder simula o que efetivamente ocorre num processador pipeline superescalar, incluindo despacho de instruções fora de ordem, a latência das diferentes unidades de execução, os efeitos do uso de predição de desvio e etc. Em função disto o Sim-Outorder é mais lento nas simulações, por outro lado é o que gera mais informações sobre o que aconteceu no processador.

Por ser o simulador mais detalhado do conjunto, ele permite a configuração de todas as características funcionais implementadas nos outros simuladores com todos os parâmetros de entrada e saída tanto em cache, como em predição de desvio, possibilitando ainda a implementação e teste de parâmetros para avaliação de

desempenho como cache perfeita e preditor perfeito. Estas configurações serão úteis como objetivos a serem alcançados nas simulações das diversas configurações, possibilitando a análise de métricas de desempenho como IPC e CPI utilizadas como condutores para melhoria de configurações.

- Suporta a execução e o envio (*issue*) de instruções *out-of-order*, baseado no esquema Register Update [Soh 90];
- Faz uso dos caches simulados pelo Sim-Cache e dos preditores de desvio do Sim-Bpred possibilitando a configuração de opções como demonstrado anteriormente;
- Verifica alinhamento e permissões nas transferências à memória;
- Permite também a execução das instruções seqüencialmente;
- Permite configuração da fila de busca de instruções usando –fetch;
- Configuração da latência por erro de predição;
- Configuração da latência em ciclos por erro de busca na cache;
- Permite configuração de latência para os níveis (1 e 2) e tipos de cache: instruções, dados e unificada;
- Permite a determinação de despacho de instruções em ordem ou fora de ordem;
- Permite a configuração do número de ULA's para inteiros e ponto flutuante;
- Permite configuração de multiplicadores de inteiros e ponto flutuante;
- Permite configuração da largura de dados na memória;
- É possível configurar a latência de acesso à primeira palavra da memória.

4 O Simulador WinDLX

WinDLX (Windows Deluxe Simulator) [DLX 02] 1.3E é um simulador de processadores DLX [PAT 98], desenvolvido para o S.O. Windows da Microsoft. Este programa foi elaborado na Universidade de Tecnologia de Viena, e esta versão foi produzida em maio de 1992.

Este programa pode ser buscado gratuitamente na Internet e será descrito neste trabalho para que possamos realizar algumas simulações e comparar com o simulador SimpleScalar. Embora traga consigo as facilidades de uma interface amigável proporcionada pelas janelas, este simulador tem menos parâmetros a serem variados e conseqüentemente possibilita uma análise de comportamento muito mais restrita do que o conjunto de simuladores SimpleScalar.

O principal objetivo é estudar as características deste simulador e as arquiteturas DLX sem entrar em profundidade no funcionamento geral do DLX.

4.1 Interface

No momento que inicializamos o simulador, podemos ver 6 subjanelas dentro deste:

- Pipeline

Em primeiro lugar, se encontra esta janela que mostra o diagrama interno do processador DLX. Esta é composta de um conjunto de estágios (“Pipeline”) de 5 estágios. Os estágios vão desde a busca e carga da instrução que está armazenada até a execução e escrita: IF (Instruction Fetch ou busca da instrução) é o estágio que se encarrega de importar a instrução correspondente da memória de instrução; ID (Instruction Decode ou decodificação de instrução) onde a instrução é decodificada e os bancos de registradores são acessados para ler os operandos A e B (as entradas da ULA); EX (EXecution ou Execução), estágio que opera os operadores A e B carregados no ciclo anterior; MEM (MEMory access ou acesso a memória), estágio utilizado para ler e escrever na memória respectivamente no endereço calculado no estágio anterior; e WB (WriteBack ou escrita), neste último passo o resultado, proveniente da memória ou da ULA, é escrito nos registradores. Como esta operação (assim como a de ID) se realiza em meio ciclo, é possível armazenar a informação no estágio de WB e obter esta informação no estágio de ID, tudo dentro do mesmo ciclo evitando a necessidade de se realizar Forwarding.

- Diagrama de Ciclos de Clock

Esta janela chamada "Diagrama de Ciclos de clock" permite ver, de forma simples, a evolução cronológica da execução do código, a cada estágio do pipeline.

- Código

A janela de código representa à memória mediante três colunas: o endereço (simbólico ou numérico), a representação em linguagem de máquina (hexadecimal) da instrução e a instrução em assembler.

- Pontos de interrupção

Esta janela simplesmente nos informa dos distintos pontos de interrupção inseridos no código.

- Registradores

Nesta janela, tal como indica seu nome, se podem observar os valores que contêm os registradores do processador durante a execução. Mostram-se o nome do registrador e o valor que tem.

- Estatísticas

Esta janela nos mostra informação de diferentes aspectos de uma simulação como: a configuração de *hardware* que se utilizou na simulação (decidida pelo usuário), as bolhas que foram produzidas e suas causas (estruturais, de controle, por interrupção, *RAW*, *WAW*), os saltos que foram efetivamente realizados, o número de instruções de carga e armazenamento na memória, instruções com ponto flutuante e as mensagens mostradas por telas, além do número de ciclos simulados, instruções que passaram pelo estado ID e as instruções que se encontram no "*Pipeline*" neste momento.

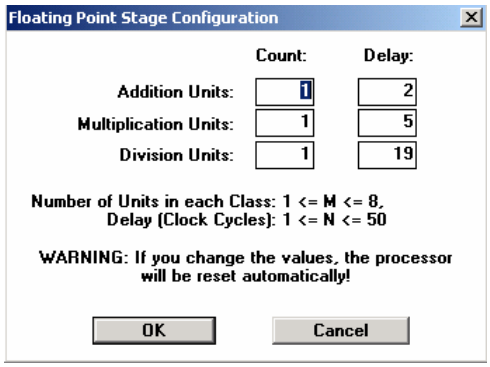
Toda a informação que encontraremos nesta janela será muito útil na hora de realizar comparações entre diferentes simulações trocando parâmetros de execução. Pode ser muito interessante comparar como afetam as trocas na configuração do DLX: habilitar ou desabilitar *Forwarding*, número de unidades de ponto flutuante e etc.

4.2 Opções e configuração do simulador

O simulador **WinDLX** se rege por parâmetros que podem ser variados, ao gosto do usuário, como é explicado exaustivamente no manual do mesmo.

O incremento do número das unidades aritmético-lógicas (soma, multiplicação e divisão para inteiros e ponto flutuante) pode ser útil para solucionar riscos estruturais, ou seja, para solucionar problemas de conflitos dos recursos, quando o hardware não pode suportar todas as combinações possíveis de instruções em execuções simultâneas. Teremos que considerar que a execução de uma multiplicação, soma ou divisão de ponto flutuante pode durar mais de um ciclo, e pode acontecer o caso que durante a utilização de uma destas unidades, seja necessário executar outra operação igual e se deva fazer uma bolha (*stall*) por falta de recursos. Todavia, incrementando o número de unidades aritmético-lógicas se soluciona este problema (até 8 de cada tipo permite o **WinDLX**)

Na “Floating Point Stage Configuration” (Configuração das unidades de ponto flutuante) se pode modificar a quantidade e duração (em ciclos de execução) das unidades “ponto flutuante” a soma (Additions Units), a multiplicação (Multiplication Units) e a divisão (Division Units) para operandos decimais. Estes parâmetros se encontram na barra de menu na Configuração → Floating Point Stage Configuration.



Pode-se supor que as unidades “ponto flutuante” realizam sua execução durante mais de um ciclo. Isto se pode configurar também no simulador. O WinDLX permite que estas operações demorem até 50 ciclos para completar sua execução.

Este simulador também tem a opção de que o usuário configure outros parâmetros: o tamanho da memória (desde 512/0x200 Bytes até 16777216/0x1000000 Bytes) com Configuration → Memory Size; a forma de mostrar os endereços da memória, absolutos (em hexadecimal) ou simbólicos (a partir da primeira instrução de código) através da Configuration → Symbolic Addresses; o tipo de contador de ciclos transcorridos, absoluto (tomando o ciclo inicial como 0 e os posteriores como 1, 2, etc.) ou relativo (tomando o ciclo atual como 0 e os anteriores como –1, -2, etc.) por meio da Configuration → Absolute Cycle Count; e também a opção de habilitar ou desabilitar Forwarding se pode fazer desde o menú Configuration → Enable Forwarding.

Na parte da configuração há diversas opções que podem ser realizadas antes e durante a execução de uma simulação. Temos que levar em conta o inconveniente de que ao realizar algumas destas se reinicia a simulação, e convém executar estas operações em primeiro lugar.

Uma destas opções é o Load Code or Data (Carregar código ou dados) que realiza a função de carga de um ou vários arquivos de código assemblador para uma simulação. Para acessar a ele temos que ir a barra de menu e selecionar File → Load Code or Data.

Também se pode fazer com que se reinicie o DLX (File → Reset DLX), com o que a simulação pára e voltaria ao princípio, ou que se reinicie tudo (File → Reset All), que pára a simulação, descarrega da memória o código assemblador e alguns dos parâmetros e a configuração voltam ao estado original.

Outra opção é a que permite ver, criar, apagar e modificar uma lista dos endereços simbólicos (etiquetas para as direções de saltos) e o valor atual destes endereços de memória. Também indica se as etiquetas são globais (G) ou locais (L), quer dizer, se podem ser acessadas desde qualquer módulo ou desde o interior do módulo. Esta opção se encontra na barra de menu em Memory → Symbols.

Memory → Display é um comando para mostrar o conteúdo da memória de diferentes formas. Se podem abrir até dez janelas deste comando. E o comando Memory → Change mostra e permite trocar o conteúdo da memória de um endereço dado.

Uma opção interessante é a de ver o que aconteceu internamente com o processamento de uma instrução. Para isto somente se deve ir à janela "Clock Cycle Diagram" e selecionar a linha da instrução que se quer examinar e se mostrará informação do que aconteceu em cada passo do Pipeline para essa instrução, como por exemplo se o estado terminou satisfatoriamente, se houve necessidade de soluções de Forwarding e para que dados, o emprego de ciclos de detenção (stalls), etc.

Além de todas estas opções temos as opções de execução do simulador. Se pode executar com um só ciclo (Execute → Single Cycle ou pressionando F7) e assim ver ciclo a ciclo como está se processando a instrução em cada etapa do Pipeline. Também se pode executar mais de um ciclo de cada vez (Execute → Multiple Cycles ou pressionando F8 e indicando o número de ciclos). Ainda se pode executar o resto dos ciclos até chegar ao final do código (Execute → Run ou pressionando F5) ou chegar a um estado concreto de uma instrução(Execute → Run to ... ou pressionando F4 e indicando a instrução e o estado onde se deve interromper).

Existem múltiplas opções que permitem trocar características da interface do simulador, como por exemplo, o tamanho do registrador cronológico do diagrama de ciclos de clock ou o detalhe da informação das estatísticas ou os âmbitos das estatísticas ou os registradores do processador, etc.

5 Roteiros Sugeridos

Neste capítulo serão apresentados os roteiros de simulações propostas, tendo os tópicos distribuídos em função dos parâmetros a serem estudados. Neste caso o primeiro roteiro utiliza o Sim-Profile e será utilizado para fazer um levantamento de classes de instruções e para que o aluno tenha um primeiro contato com o conjunto de simuladores. Os resultados desta simulação serão utilizados nos roteiros seguintes, sendo necessários para outras simulações, servindo como base auxiliar para avaliação.

A partir do segundo roteiro serão avaliados e verificados parâmetros ligados à memória cache, utilizando o simulador Sim-Cache, e serão induzidas conclusões no sentido de justificar os limites da cache e a necessidade de sua implementação, bem como sua influência no desempenho e funcionamento do processador. O maior número de roteiros está relacionado à cache, pois esta é a que permite maior quantidade de parâmetros a serem variados, sem falar em sua importância estratégica em qualquer processador. Nestes roteiros serão estudados os parâmetros, pela ordem: 2º. Número de conjuntos, 3º. Associatividade, 4º. Tamanho dos blocos e 5º. Política de reposição. No 6º. roteiro é proposta a utilização do Sim-Cheetah que é um simulador de cache capaz de testar várias configurações em uma só simulação, mediante a entrada de intervalos de simulação, possibilitando a variação ao mesmo tempo de no. de conjuntos, associatividade, tamanho dos blocos, mantendo fixa a política de reposição e possibilitando a análise em caches de dados, instruções e cache unificada.

No 7º. roteiro será avaliado o conceito de predição de desvio, utilizando o simulador Sim-Bpred que permite a análise de predições estáticas e dinâmicas, sendo possível com este simulador avaliar preditores taken, notaken, bimodal, combinada, dois níveis e híbridos, referenciando os resultados nas taxas de acertos para o caminho predito.

Do 8º. roteiro até o 10º. iremos utilizar o Sim-Outorder para análise de desempenho. Diferentemente dos simuladores funcionais utilizados até aqui, este simulador permite a verificação das funcionalidades vistas (como cache e predição de desvios por exemplo) , agregada a conceitos como: cache perfeita, preditor perfeito, IPC e CPI (métricas úteis na avaliação de desempenho). No 8º. roteiro será introduzido o simulador e seus parâmetros de funcionamento, bem como acompanhamento de desempenho do processador simulado com diversificação de cache e comparação com padrões como cache perfeita e sistema sem cache. No 9º. roteiro, seguindo a sistemática utilizada no roteiro anterior, o Sim-Outorder será utilizado para análise de predição de desvio e impacto no desempenho do processador simulado, via comparação de IPC e CPI, utilizando o conceito de preditor perfeito para mensurar a carga envolvida. No 10º.

Roteiro é proposta a implementação da simulação de um projeto com os diversos parâmetros que compõem um processador, onde será analisado o desempenho do mesmo mediante comparação de IPC e CPI com resultados dos projetos realizados nos roteiros 8º. e 9º, em que foram implementadas situações de cache perfeita e preditor perfeito respectivamente.

No 11º. roteiro utilizamos o simulador WinDLX para análise de parâmetros como pipeline e forwarding, mediante a implementação de um programa gerador de números primos e avaliação de resultados via janelas.

Todos os roteiros estão acompanhados dos resultados esperados para futura análise por parte do instrutor mas, não é pretendido com isto esgotar as diversas possibilidades de resultados que podem ser solicitados, ficando abertas as possibilidades para que o professor julgue e determine as perguntas em função das características das turmas.

Como sugestão de distribuição dos roteiros por cursos pode ser considerado o nível de aprofundamento do curso em termos de hardware e o número de créditos da disciplina para elaboração do plano de aula. No caso do curso de Administração com Ênfase em Análise de Sistemas, na disciplina de Arquitetura e Organização de Computadores, que tem 4 créditos / 80 horas-aula, poderiam ser implementados o roteiro 1 (com análise de perfis de benchmarks), roteiro 2 (Sim-Cache), roteiro 6 (Sim-Cheetah), roteiro 7 (Sim-Bpred) e roteiro 11 (WinDLX), permitindo que as demais aulas de laboratório sejam voltadas para análises com ênfase em Arquitetura de Computadores, como dispositivos de entrada e saída, análise de processadores comerciais, etc.. Para o curso de Sistemas de Informação, que possuí duas disciplinas de 4 créditos / 80 horas-aula (Organização de Processadores e Arquitetura de Computadores), pode-se implementar ao longo do primeiro semestre os 11 roteiros, deixando a análise voltada a aspectos de Arquitetura de Computadores para o segundo semestre. Para cursos com apelo em hardware como Engenharia da Computação e Ciência da Computação poderá ser utilizada a mesma fórmula de Sistemas de Informação com possível complementação de novos roteiros utilizando o Sim-Outorder.

Workbenchs Utilizados

SPEC (Standard Performance Evaluation Corporation) [SPEC] não é exatamente o nome de um benchmark. Na verdade, trata-se de um consórcio de 22 empresas que têm como objetivo "prover a indústria com um padrão realístico para medir a performance de sistemas de computadores avançados". Os benchmarks desenvolvidos por este consórcio de empresas são programas reais (o que os classifica como benchmarks de aplicação), que visam medir a performance da maior parte possível de componentes do sistema, principalmente o processador, arquitetura de memória e compilador.

Analisando estes benchmarks, pode-se dizer que medir a performance de computadores não é uma tarefa tão simples, e requer um estudo razoavelmente aprofundado das metodologias de análise existentes. O que mais dificulta no momento de fazer uma análise de performance de processadores é encontrar um benchmark que enfatize os tipos de programas que serão utilizados, além de conseguir interpretar corretamente as métricas utilizadas pelo benchmark. Muitos fabricantes, quando fazem

os testes de desempenho de seus computadores, geralmente fazem um "tuning" nos benchmarks e na respectiva compilação, ou seja, fazem pequenas otimizações no código e na compilação do mesmo. Algumas vezes isto pode afetar os resultados obtidos.

Para solucionar os problemas acima, tem-se feito um grande esforço no sentido de padronizar os benchmarks. As entidades que estão mais perto disso são o SPEC e o PARKBENCH. Ambos utilizam metodologias bem elaboradas e bastante aceitas, além de atualizarem constantemente seus programas de benchmark.

Quanto à unidade de medida, percebe-se que se tem dado mais valor ao tempo de execução total, chamado de "wall-clock time". Isto é bastante importante, visto que o tempo que mais interessa no momento da avaliação é o tempo real de execução, pois se o benchmark for bastante abrangente, o "wall-clock time" irá levar em consideração a performance de todos os componentes do sistema. O que se quer dizer com isso, é que se o sistema como um todo (e não de um componente em particular) tiver um bom desempenho, o tempo de execução do benchmark será baixo. É certo que vários benchmarks ainda têm a sua própria metodologia de análise de resultados, mas algumas já estão baseadas nesta unidade de medida.

Nestas simulações iremos utilizar dois benchmarks do conjunto SPEC95, para que possamos avaliar o comportamento do nosso processador simulado. Estes benchmarks são bem utilizados e conhecidos. Um deles é o COMPRESS, programa de compressão de arquivos que dispensa apresentações em função da grande utilização por todos aqueles que pretendem transportar arquivos em disquetes ou armazenar grandes quantidades de arquivos ocupando o mínimo de espaço possível. O outro é o Jogo Go bastante popular e simples, porém profundo, que teve origem na China, com nome de wei-chi, há cerca de 40 séculos. O principal fascínio deste jogo consiste na simplicidade de suas regras e na complexidade que o jogo alcança. É um jogo de disputa ou conquista de território e será útil na simulação de processadores como forma de teste para avaliação de desempenho.

Os benchmarks Go (descrito no apêndice) e Compress do SPEC foram escolhidos em função de suas características construtivas, sendo que um deles manipula principalmente dados e outro dá ênfase a manipulação de instruções, cobrindo assim amplo leque e mantendo maior imparcialidade no julgamento de desempenho do processador implementado, bem como as funcionalidades analisadas.

5.1 Roteiro I – O Sim-Profile e os perfis dos programas de teste

Objetivos

Neste roteiro será introduzido o primeiro simulador do conjunto SimpleScalar. Neste caso, iremos utilizar o simulador Sim-Profile para determinação dos perfis dos programas de avaliação de performance (benchmarks). Este simulador fornece estatísticas como o número de instruções executadas e distribuição de classes de instrução. A perfilação dos dois benchmarks que serão utilizados nesta aula servirá de base para todas as simulações a partir daqui.

Procedimentos

Para implementar nosso levantamento iremos rodar o simulador Sim-Profile junto aos benchmarks Go e o Compress e preencher os resultados na tabela abaixo. Lembre-se que o comando para início da simulação é

./Sim-Profile <argumentos> go.ss 50 9 go.in

onde:

- -iclass - Realiza a contagem de cada tipo de classe de instrução.
- -iprof - Realiza a contagem de cada tipo de instrução
- -bprof - Apresenta a contagem de cada tipo de desvio
- -amprof - Apresenta a contagem de cada modo de endereçamento.

Os argumentos poderão ser implementados um de cada vez, resultando quatro verificações e no final uma verificação geral. No caso desta simulação iremos dar ênfase à contagem e percentuais das classes de instruções, utilizando estes dados para preenchimento da tabela abaixo:

Tabela - Perfis das Classes de Instruções Go.

Classes de Instruções	Contagem de Instruções	% do Total
Load		
Store		
Desvio Incondicional		
Desvio Condicional		
Operação de Inteiros		
Operação de Ponto Flutuante		
Trap		

Utilizando o comando:

./Sim-Profile <argumentos> compress95.ss < bigtest.in

Repita a simulação para o Compress e complete a tabela usada na simulação anterior.

Perguntas e tarefas para a confecção do relatório

1. Utilize os resultados para construção de um gráfico de barras único, relacionando a classe da instrução com o percentual de vezes que ela foi acessada nos dois benchmarks.
2. Com os percentuais obtidos crie um gráfico tipo pizza para distribuir as instruções, para cada benchmark.
3. Utilizando os gráficos verifique o percentual de instrução de desvio para cada caso. No gráfico que relaciona os benchmarks aos tipos de instruções observe o número de instruções de desvio em cada caso e justifique com suas palavras a diferença (imagine a função de cada programa usado como benchmark).
4. Qual a relação entre os percentuais Load / Store encontrados para o Go e Compress ?
5. Justifique com base na natureza dos benchmarks a diferença entre Loads / Stores no Go e do Compress.

Resultados do Roteiro I

O aluno deverá, por intermédio desta simulação, perceber a grande diferença entre o percentual de instruções voltadas à manipulação de dados entre os dois benchmarks.

Através da análise dos resultados podemos perceber uma forte tendência no Compress em pouca carga (Load) para grande quantidade de armazenamento (Store). Neste caso temos uma relação de 10 vezes, enquanto no Go percebemos 3 cargas para um armazenamento. Certamente estes resultados estão relacionados aos algoritmos que constituem o Jogo Go e o aplicativo de compressão de arquivos Compress.

Enquanto no Go temos 28,67% de instruções Load / Store (dados), no Compress chegamos a um patamar de 64,44%. Isto irá afetar as avaliações em simuladores que dependam da manipulação entre dados e instruções.

No caso de desvios foi obtido que a quantidade de saltos incondicionais é três vezes maior no Go em relação ao Compress, enquanto que nos saltos condicionais o Go possuí 50% de saltos a mais. Isto irá afetar futuros resultados na predição de desvio. Serão obtidos, consequentemente, resultados proporcionais para os benchmarks analisados.

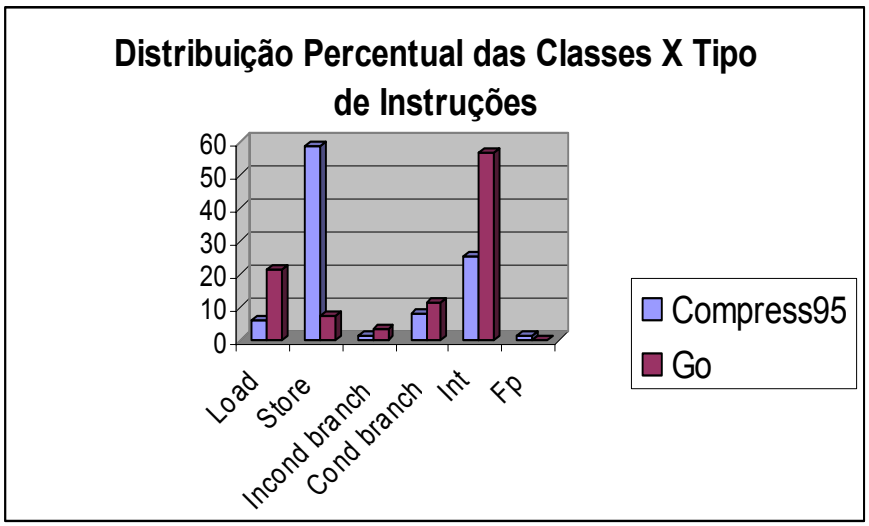
Os resultados de análise da distribuição de classes de instrução irão municiar os alunos com dados em relação aos benchmarks utilizados, para o roteiro futuro.

Como subproduto destas deduções o aluno irá manipular o Sim-Profile que possui características semelhantes aos demais simuladores da família SimpleScalar, mantendo desta forma o primeiro contato com o conjunto SimpleScalar de simuladores.

Tabelas e Gráficos:

Tabela 5.1: Perfis das classes de instruções do Go e Compress

Classe de Instruções	Total de Instruções Compress	Total de Instruções Go	Compress %	Go %
Load	185510	115800785	5,92	21,12
Store	1834570	41413575	58,52	7,55
Incond branch	34360	18177922	1,10	3,32
Cond branch	246080	62108771	7,85	11,33
Int	785263	310676323	25,05	56,67
Fp	49152	0	1,57	0
Trap	164	72	0,01	0



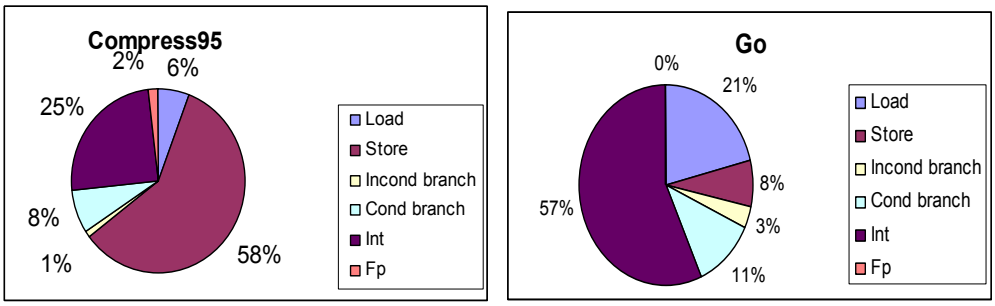


Figura 5.1: Gráficos Go e Compress do Sim-Profile

5.2 Roteiro II – Sim-Cache e os efeitos do aumento da cache pela variação do número de conjuntos.

Objetivos

O objetivo a partir desta aula é a realização da análise de cache, por intermédio das simulações, visando a avaliação da influência dos seus parâmetros de configuração. Para tanto iremos utilizar o simulador do conjunto de ferramentas SimpleScalar conhecido como Sim-Cache.

Nesta simulação será realizado um trabalho com mapeamento conjunto associativo e será variado o número de conjuntos, e portanto o tamanho total da cache, mantendo-se fixos a associatividade, o tamanho do bloco e a política de reposição. Observando os resultados e fazendo as necessárias comparações, chegaremos ao final deste trabalho com boas conclusões com relação à variação da taxa de misses, produzidos na cache, em função da variação do tamanho da cache, reiterando desta forma os conceitos estudados em aula. Observando os resultados e analisando as tabelas chegaremos ao final deste trabalho com boa familiarização com relação ao Sim-Cache,

Procedimentos

Através do Sim-Cache iremos implementar um modelo de processador com configuração de memória cache definida e rodar neste o benchmark Go.

Completar a tabela com os resultados, variando o número de conjuntos nos valores 64, 128,512,1024 e 2048 e mantendo fixas a associatividade em 2, o tamanho dos blocos em 32 e a política de reposição randômica, verificando através do simulador as taxas de erros em função da variação de parâmetros sugerida. Para isto deverão ser utilizados os resultados apresentados nos relatórios de simulação do SimpleScalar.

Elabore uma coluna com o tempo médio de acesso à cache, considerando a taxa de acertos para cada simulação, o tempo de acesso a cache é Tc = 1ns e o tempo de acesso à memória principal Tm = 15 ns.

Observe que estaremos avaliando os resultados das caches de instruções e dados separadas e de uma cache unificada.

Go			
Número de Conjuntos/Tamanho do Bloco/ Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache
Cache de Instruções IL1			
64/32/2/r			
128/32/2/r			
512/32/2/r			
1024/32/2/r			
2048/32/2/r			
Cache de Dados DL1			
64/32/2/r			
128/32/2/r			
512/32/2/r			
1024/32/2/r			
2048/32/2/r			
Cache Unificada IL1 = DL1			
64/32/2/r			
128/32/2/r			
512/32/2/r			
1024/32/2/r			
2048/32/2/r			

Agora utilize o benchmark Compress em tabela semelhante para avaliar e comparar os resultados.

Após tabular os dados, elabore dois gráficos relacionando as taxas de misses dos dois benchmarks e o no. de conjuntos (e portanto o tamanho da cache).
Perguntas e tarefas para a confecção do relatório:

1. O que ocorre com os misses à medida que o número de conjuntos é aumentado?
2. O que é possível concluir com relação ao tamanho da cache e a taxa de misses?
3. Este comportamento é comum para cache de dados, instruções e unificada?
4. Compare o valor do tempo médio de acesso à cache da tabela com o calculado através do conceito de cache perfeita.
5. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule o CPI real para o no. de conjuntos 64 e 2048. Descreva com suas palavras a comparação dos resultados obtidos.

Resultados do Roteiro II

Na Tabela e Figura 5.2 estão apresentados os resultados das simulações utilizando o Go.

Tabela 5.2: Comparativo de erros e taxa de misses com a variação do número de conjuntos no Go, utilizando o Sim-Cache

Go			
Conjuntos/Tamanho do bloco / Associatividade /Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
64/32/2/r	744136	0,0794	2,191
128/32/2/r	569468	0,0569	1,8535
512/32/2/r	271491	0,0271	1,4065
1024/32/2/r	159128	0,0159	1,2385
2048/32/2/r	72597	0,0073	1,1095
Cache de Dados DL1			
64/32/2/r	235841	0,0867	2,3005
128/32/2/r	146837	0,054	1,81
512/32/2/r	45113	0,0166	1,249
1024/32/2/r	27935	0,0103	1,1545
2048/32/2/r	18282	0,0067	1,1005
Cache Unificada IL1 = DL1			
64/32/2/r	1556008	0,1223	2,8345
128/32/2/r	1075061	0,0845	2,2675
512/32/2/r	477907	0,0376	1,564
1024/32/2/r	276282	0,0217	1,3255
2048/32/2/r	148833	0,0117	1,1755

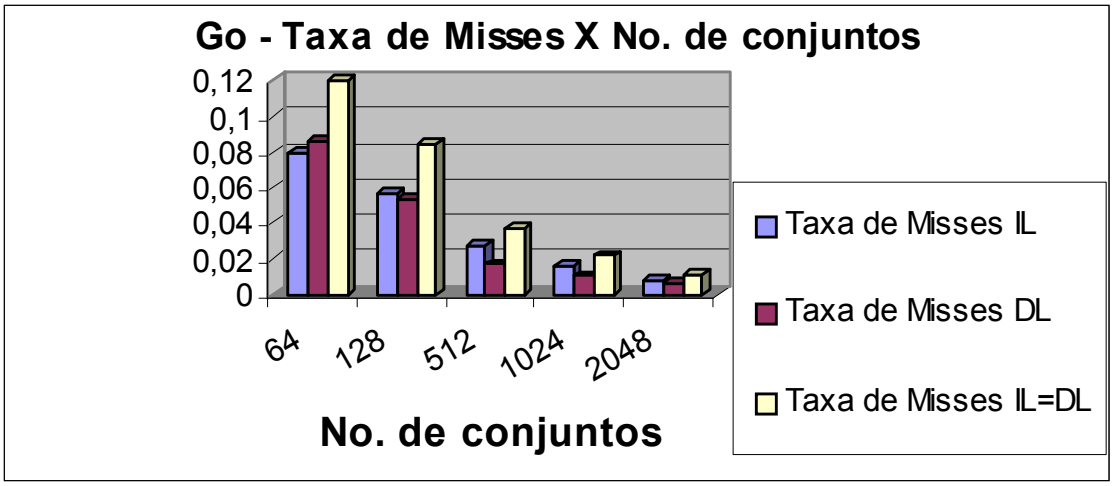


Figura 5.2: Taxa de misses em função do número de conjuntos – benchmark Go

Na Tabela 5.3 e na Figura 5.3 estão os resultados das simulações utilizando o Compress.

Tabela 5.3: Comparativo de erros e taxa de misses com a variação do número de conjuntos no Compress, utilizando o Sim-Cache

Compress			
Conjuntos/Tamanho do bloco / Associatividade /Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
64/32/2/r	45110	0,0144	1,216
128/32/2/r	28104	0,009	1,135
512/32/2/r	3344	0,0011	1,0165
1024/32/2/r	989	0,0003	1,0045
2048/32/2/r	942	0,0003	1,0045
Cache de Dados DL1			
64/32/2/r	229698	0,1099	2,6485
128/32/2/r	228901	0,1095	2,6425
512/32/2/r	228568	0,1094	2,641
1024/32/2/r	226960	0,1086	2,629
2048/32/2/r	201517	0,0964	2,446
Cache UnificadaIL1 = DL1			
64/32/2/r	308578	0,0591	1,8865
128/32/2/r	277721	0,0531	1,7965
512/32/2/r	247191	0,0473	1,7095
1024/32/2/r	242265	0,0464	1,696
2048/32/2/r	218140	0,0417	1,6255

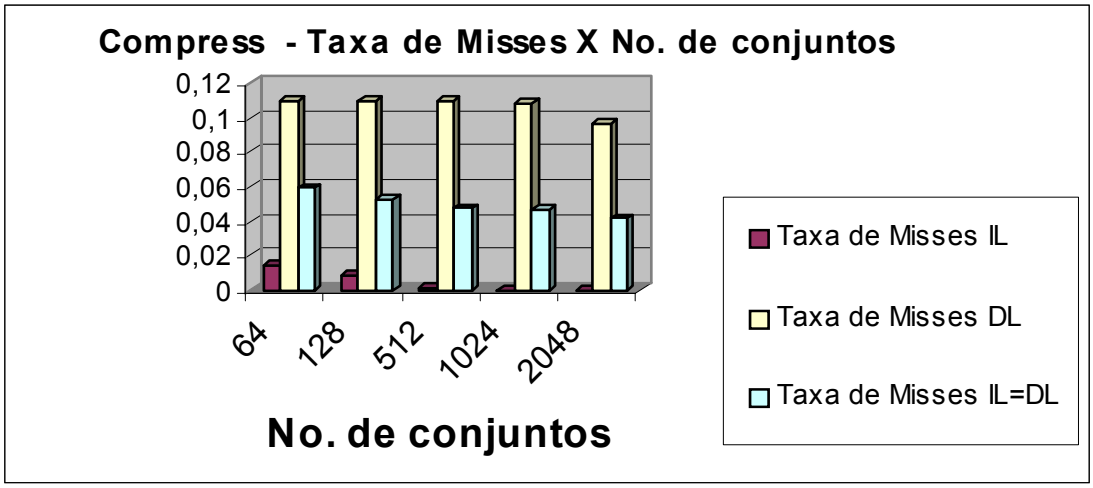


Figura 5.3: Taxa de Misses em função do número de conjuntos - benchmark Compress.

1. O que ocorre com a taxa de misses à medida que o número de conjuntos aumentado?

É perceptível que a medida que aumentamos o número de conjuntos e conseqüentemente o tamanho da cache ocorre uma queda na taxa de misses.

2. O que é possível concluir com relação ao tamanho da cache e a taxa de misses?

A taxa de misses e o tamanho da cache, para os tamanhos analisados, são inversamente proporcionais

3. Este comportamento é comum para cache de dados, instruções e unificada?

No Go a redução na taxa de misses ocorre tanto na cache de instruções, quanto na cache de dados e unificada. Já no Compress a redução na taxa de misses é mais acentuada na cache de instruções e unificada, mantendo-se praticamente estável na cache de dados.

4. Compare o valor do tempo médio de acesso à cache da tabela com o calculado através do conceito de cache perfeita.

O tempo médio de acesso à cache foi calculado com base na equação

Tméd = Tc+(1-h)Tm

onde: h = taxa de hits ; Tc = tempo de acesso à cache; Tm = tempo de acesso à memória principal. Percebemos a tendência à aproximação do tempo médio ao valor do tempo de acesso à cache perfeita à medida que aumentamos o número de conjuntos.

Go

64-32-2-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a Misses
Instruções	0,0794	15	1	1,191
Dados	0,0867	15	0,2867	0,372853
				1,563853
CPI sem parada				2
CPI com as paradas devidas a faltas				3,563853
Desempenho da cache perfeita em relação à cache				**1,781927**

2048-32-2-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,073	15	1	1,095
Dados	0,0067	15	0,2867	0,028813
				1,123813
CPI sem parada				2
CPI com as paradas devidas a faltas				3,123813
Desempenho da cache perfeita em relação à cache				**1,561907**

Compress

64-32-2-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0144	15	1	0,216
Dados	0,1099	15	0,6444	1,062293
				1,278293
CPI sem parada				2
CPI com as paradas devidas a faltas				3,278293
Desempenho da cache perfeita em relação à cache				**1,639147**

2048-32-2-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0003	15	1	0,0045
Dados	0,0964	15	0,6444	0,931802
				0,936302
CPI sem parada				2
CPI com as paradas devidas a faltas				2,936302
Desempenho da cache perfeita em relação à cache				**1,468151**

5. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule o CPI real para o no. de conjuntos 64 e 2048. Descreva com suas palavras a comparação dos resultados obtidos.

Considerando os valores do número de conjuntos 64 e 2048, que serviram de base para análise do desempenho, deverá ser observado que a razão da CPI com paradas e a CPI para a cache perfeita é 1,78 na configuração 64 e 1,56 vezes melhor para 2048 conjuntos no Go. No caso do Compress obtivemos razões de 1,63 com 64 conjuntos e 1,46 com 2048 conjuntos.

5.3 Roteiro III – O Sim-Cache e a influência do aumento da cache pela variação da associatividade.

Objetivos

Nesta aula iremos dar continuidade ao nosso estudo, avaliando a influência da associatividade e aumento da cache nos erros e acertos. E para isto iremos realizar simulações, fixando os demais parâmetros - número de conjuntos, tamanho do bloco e política de reposição, para que possamos avaliar o comportamento da cache com relação a erros e acertos que compõem o principal parâmetro avaliativo no projeto de caches. Para isto iremos utilizar a ferramenta Sim-Cache. Neste caso iremos alterar o tamanho da cache através da variação da associatividade e será verificado o desempenho proporcional a este fato, tanto para cache de instruções como cache de dados e cache unificada.

Procedimento

Utilizando o benchmark Go realize as simulações sugeridas na tabela abaixo, mantendo fixo o número de conjuntos em 512, o tamanho do bloco em 32 e a política de reposição randômica para que a avaliação desta tabela seja realizada somente sobre a variação do parâmetro associatividade.

Elabore uma coluna com o tempo médio de acesso à memória, considerando a taxa de acertos para cada simulação, o tempo de acesso a cache é Tc = 1ns e o tempo de acesso à memória principal Tm = 15 ns.

Deverão ser avaliados os resultados para as caches de instruções e dados separadas e para uma cache unificada.

Go			
Conjuntos/Tamanho do bloco / Associatividade /Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
512/32/1/r			
512/32/2/r			
512/32/4/r			
512/32/8/r			
512/32/16/r			
Cache de Dados DL1			
512/32/1/r			
512/32/2/r			
512/32/4/r			
512/32/8/r			
512/32/16/r			
Cache Unificada IL1= DL1			
512/32/1/r			
512/32/2/r			
512/32/4/r			
512/32/8/r			
512/32/16/r			

Agora utilize o benchmark Compress para avaliar e comparar os resultados, montando uma tabela semelhante

Após tabular os dados elabore um gráfico relacionando erros e associatividade; plotando os resultados através dos parâmetros apresentados após as simulações.
Perguntas e tarefas para a confecção do relatório:

1. O que você percebeu com relação aos níveis de associatividade, e os acertos e erros?
2. Utilizando a coluna do tempo médio de acesso à cache compare este valor com o calculado através do conceito de cache perfeita.
3. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule para a associatividade 1 e 16. Descreva com suas palavras a comparação dos resultados obtidos.
4. Faça uma pesquisa e explique com suas palavras os mapeamentos direto, completamente associativo e associativo por conjuntos.

Monte o relatório para entrega incluindo os gráficos plotados e suas conclusões com relação ao aumento do número de conjuntos e os erros na cache.

Resultados do Roteiro III

Tabela 5.4: Comparativo de erros e taxa de misses com a variação da associatividade no Go, utilizando o Sim-Cache

Go			
Conjuntos/Tamanho do bloco / Associatividade /Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
512/32/1/r	498667	0,0499	1,7485
512/32/2/r	271491	0,0271	1,4065
512/32/4/r	137152	0,0137	1,2055
512/32/8/r	65240	0,0065	1,0975
512/32/16/r	19255	0,0019	1,0285
Cache de Dados DL1			
512/32/1/r	132490	0,0487	1,7305
512/32/2/r	45113	0,0166	1,249
512/32/4/r	26343	0,0097	1,1455
512/32/8/r	17854	0,0066	1,099
512/32/16/r	14161	0,0052	1,078
Cache Unificada IL1= DL1			
512/32/1/r	934265	0,0735	2,1025
512/32/2/r	477907	0,0376	1,564
512/32/4/r	234691	0,0185	1,2775
512/32/8/r	124993	0,0098	1,147
512/32/16/r	51418	0,004	1,06

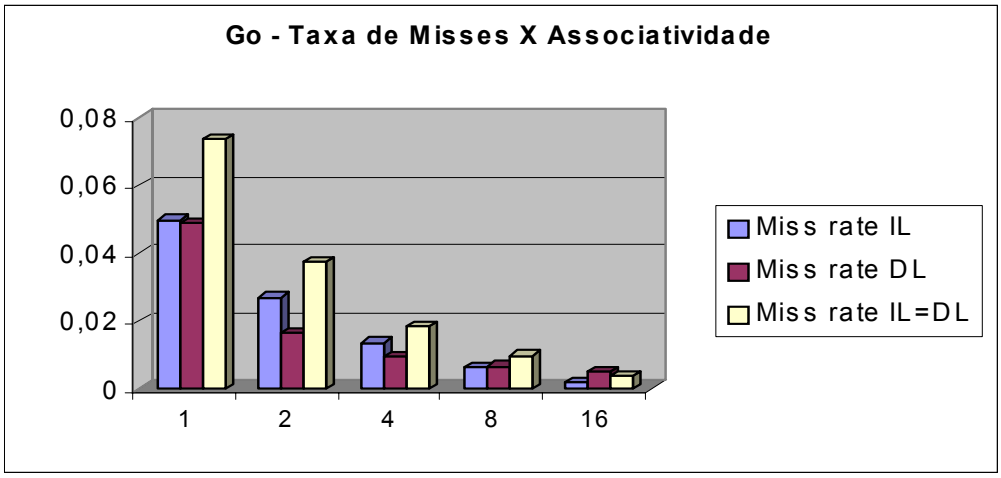


Figura 5.4: Taxa de misses x associatividade no GO

Tabela 5.5: Comparativo de erros e taxa de misses com a variação da associatividade no Compress, utilizando o Sim-Cache

Compress			
Conjuntos/Tamanho do bloco / Associatividade /Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
512/32/1/r	17839	0,0057	1,0855
512/32/2/r	3344	0,0011	1,0165
512/32/4/r	1057	0,0003	1,0045
512/32/8/r	978	0,0003	1,0045
512/32/16/r	940	0,0003	1,0045
Cache de Dados DL1			
512/32/1/r	229936	0,11	2,65
512/32/2/r	228568	0,1094	2,641
512/32/4/r	226013	0,1081	2,6215
512/32/8/r	193647	0,0926	2,389
512/32/16/r	53422	0,0256	1,384
Cache Unificada IL1= DL1			
512/32/1/r	274896	0,0526	1,789
512/32/2/r	247191	0,0473	1,7095
512/32/4/r	241506	0,0462	1,693
512/32/8/r	210844	0,0404	1,606
512/32/16/r	74964	0,0143	1,2145

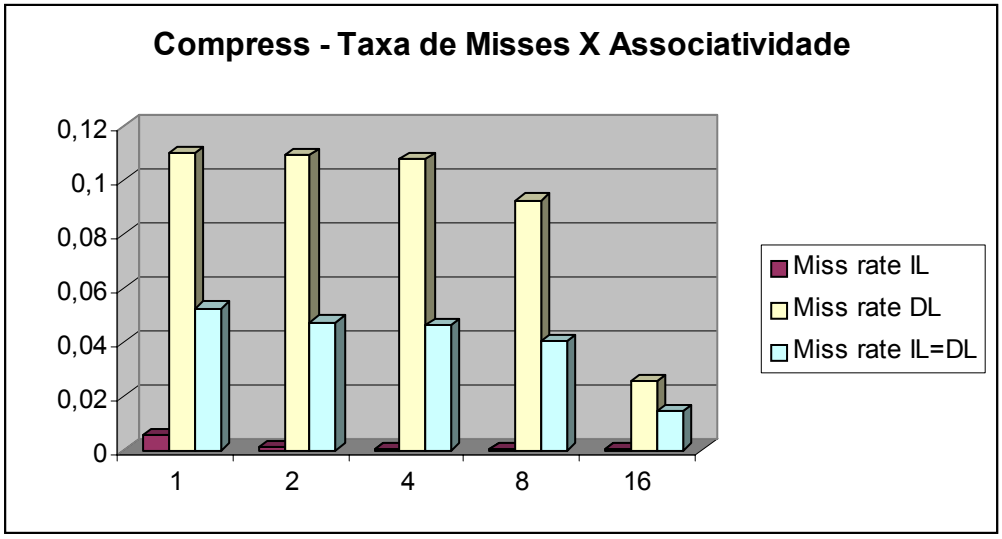


Figura 5.5: Taxa de misses x associatividade no Compress

1. O que você percebeu com relação aos níveis de associatividade, e os acertos e taxas de misses?

Percebe-se uma redução na taxa de misses à medida que aumentamos a associatividade. No Compress a taxa de misses se mantêm estável e alta na cache de dados, enquanto há uma leve redução na taxa para cache de instruções e unificada. O valor da taxa para cache de dados é muito maior neste caso em relação às caches de instruções e unificada em função do perfil do programa Compress.

2. Utilizando a coluna do tempo médio de acesso à memória compare este valor com o calculado através do conceito de cache perfeita.

O tempo médio de acesso foi calculado com base na equação

Tméd= Tc+(1-h)Tm

onde : h= no. de hits ; Tc=tempo de acesso à cache; Tm=tempo de acesso à memória principal. Percebemos a tendência à aproximação do tempo médio ao valor do tempo de acesso à cache à medida que aumentamos a associatividade.

No caso do Go o tempo médio de acesso se aproxima do valor de cache perfeita tanto em caches de instruções, dados ou unificada, mantendo certa uniformidade na redução. No caso do Compress o tempo de acesso médio à cache de dados é muito maior do que os valores para caches de instruções e unificada. Mas é mantida a tendência à redução à medida que é aumentada a associatividade e conseqüentemente o tamanho da cache.

3. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule para associatividades 1 e 16. Descreva com suas palavras a comparação dos resultados obtidos.

Go

512-32-1-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0499	15	1	0,7485
Dados	0,0487	15	0,2867	0,209434
				0,957934
CPI sem parada				2
CPI com as paradas devidas a faltas				2,957934
Desempenho da cache perfeita em relação à cache				**1,478967**

512-32-16-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores - Go	Ciclos devidos a misses
Instruções	0,0019	15	1	0,0285
Dados	0,0052	15	0,2867	0,022363
				0,050863
CPI sem parada				2
CPI com as paradas devidas a faltas				2,050863
Desempenho da cache perfeita em relação à cache				**1,025431**

Compress

512-32-1-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0057	15	1	0,0855
Dados	0,11	15	0,6444	1,06326
				1,14876
CPI sem parada				2
CPI com as paradas devidas a faltas				3,14876
Desempenho da cache perfeita em relação à cache				**1,57438**

512-32-16-r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0003	15	1	0,0045
Dados	0,0256	15	0,6444	0,24745
				0,25195
CPI sem parada				2
CPI com as paradas devidas a faltas				2,25195
Desempenho da cache perfeita em relação à cache				**1,125975**

No Go foram obtidos, para as configurações avaliadas 1 e 16 de associatividade valores 1,47 e 1,02 para desempenho da cache perfeita em relação à cache.

No Compress foi obtido valor 1,57 na relação de desempenho para associatividade 1 e 1,12 para associatividade 16.

5.4 Roteiro IV – Sim-Cache e o tamanho dos blocos no desempenho

Objetivos

Nesta experiência iremos realizar analisar simulações, dando ênfase ao comportamento da cache em função da variação do tamanho dos blocos e o conseqüente aumento do tamanho da cache. Para isto serão mantidos fixos os demais parâmetros: associatividade, número de conjuntos, política de reposição. Utilizaremos o simulador funcional Sim-Cache do conjunto de ferramentas SimpleScalar. Neste trabalho iremos realizar análise do comportamento de acertos e taxas de misses considerando as caches de instruções, dados e unificada.

Procedimentos

Implemente a simulação de um sistema de memória cache através do Sim-Cache e rode neste simulador o benchmark Go.

Complete a tabela abaixo, variando o tamanho da cache através do tamanho do bloco em 16, 32 e 64 e fixando o no. de conjuntos em 1024, a associatividade em 2 e a política de reposição LRU.

Elabore uma coluna com o tempo médio de acesso à cache, considerando a taxa de acertos para cada simulação, o tempo de acesso a cache é Tc = 1ns e o tempo de acesso à memória principal Tm = 15 ns.

Deverão ser avaliados os resultados para as caches de instruções e dados separadas e para uma cache unificada.

Go			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
1024/16/2/l			
1024/32/2/l			
1024/64/2/l			
Cache de Dados DL1			
1024/16/2/l			
1024/32/2/l			
1024/64/2/l			
Cache Unificada IL1= DL1			
1024/16/2/l			
1024/32/2/l			
1024/64/2/l			

Agora utilize o benchmark Compress para avaliar e comparar os resultados, utilizando para isto tabela semelhante a anterior:

Após tabular os dados, elabore um gráfico relacionando erros e tamanho do bloco.

Perguntas e tarefas para a confecção do relatório:

1. Descreva o que é o tamanho do bloco? Onde se aplica e qual sua influência na memória cache?
2. O que aconteceu com os erros à medida que houve variação no tamanho do bloco?
3. Utilizando a coluna do tempo médio de acesso à cache compare este valor com o calculado através do conceito de cache perfeita.
4. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule para os tamanhos de bloco 16 e 64. Descreva com suas palavras a comparação dos resultados obtidos.
5. Pesquise e explique com suas palavras em que medida afetará no desempenho blocos maiores ou menores.

Resultados Roteiro IV

Tabela 5.6: Comparativo de erros e taxa de misses com a variação do tamanho dos blocos no Go, utilizando o Sim-Cache

Go			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
1024/16/2/l	493504	0,0494	1,741
1024/32/2/l	159128	0,0159	1,2385
1024/64/2/l	40578	0,0041	1,0615
Cache de Dados DL1			
1024/16/2/l	65855	0,0242	1,363
1024/32/2/l	27935	0,0103	1,1545
1024/64/2/l	11015	0,0041	1,0615
Cache Unificada IL1 = DL1			
1024/16/2/l	763339	0,06	1,9
1024/32/2/l	276282	0,0217	1,3255
1024/64/2/l	98285	0,0077	1,1155

Tabela 5.7: Comparativo de erros e taxa de misses com a variação do tamanho dos blocos no Compress, utilizando o Sim-Cache

Go			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
1024/16/2/l	4922	0,0016	1,024
1024/32/2/l	989	0,0003	1,0045
1024/64/2/l	538	0,0002	1,003
Cache de Dados DL1			
1024/16/2/l	456119	0,2182	4,273
1024/32/2/l	226960	0,1086	2,629
1024/64/2/l	101141	0,0484	1,726
Cache Unificada IL1 = DL1			
1024/16/2/l	487877	0,0934	2,401
1024/32/2/l	242265	0,0464	1,696
1024/64/2/l	111397	0,0213	1,3195

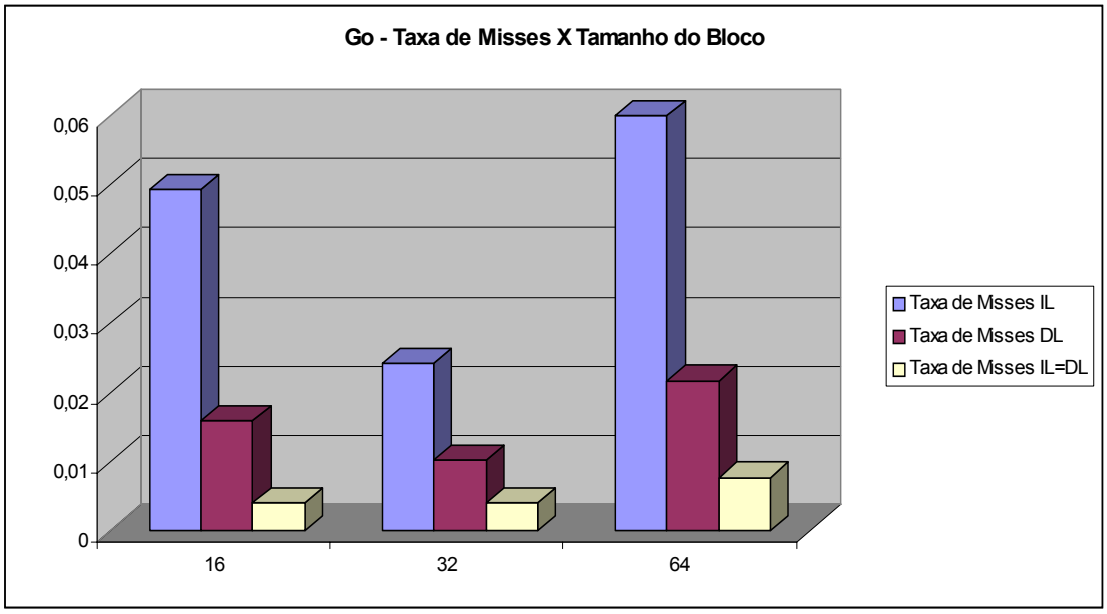


Figura 5.6: Taxas de misses nas caches unificada, dados e instruções em função do tamanho do bloco – benchmark Go

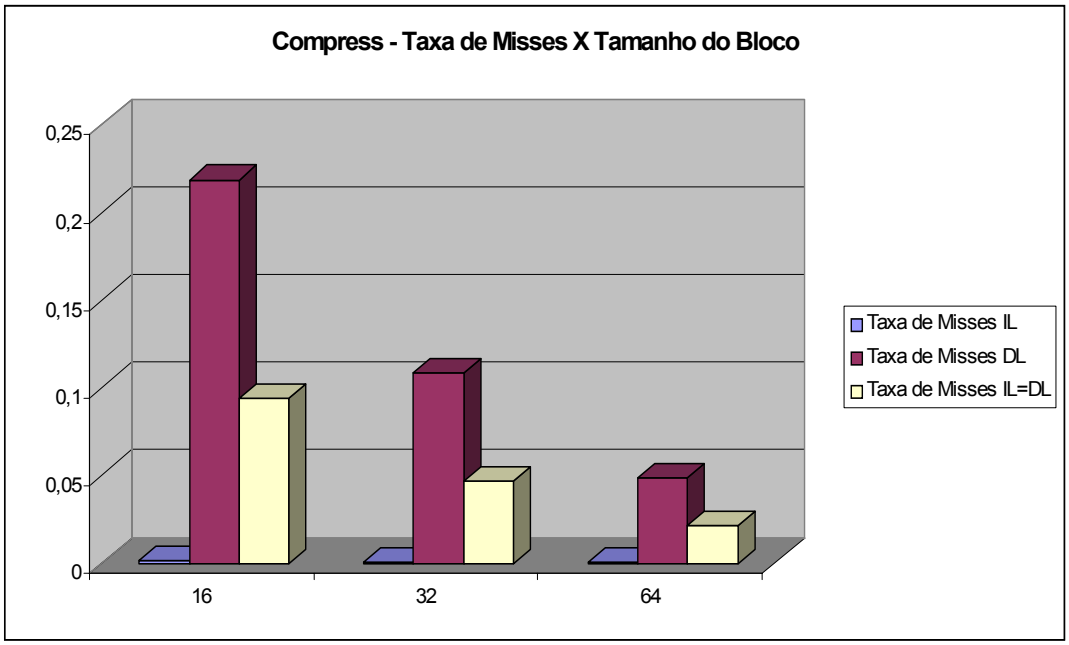


Figura 5.7: Taxas de misses na cache unificada, dados e instruções em função do tamanho do bloco – benchmark Compress.

1) Descreva o que é o tamanho do bloco? Onde se aplica e qual sua influência na memória cache?

Quando um bloco de dados é extraído da memória principal e colocado na memória cache, não é apenas a palavra pretendida que é armazenada na memória cache, mas um certo número de palavras adjacentes que são extraídas. Estas palavras adjacentes são denominadas “bloco” ou “linha” .

2) O que aconteceu com os erros à medida que houve variação no tamanho do Bloco?

A taxa de misses diminui, tanto no Compress como no Go, à medida que houve uma variação crescente no tamanho do bloco. Entretanto esta redução no Go é caracterizada pela descida mais acentuada nos erros de instruções e mais suave nos erros de dados. Já no Compress ocorre o oposto conforme podemos observar nos gráficos.

3) Utilizando a coluna do tempo médio de acesso à cache compare este valor com o calculado através do conceito de cache perfeita.

Percebemos a tendência à aproximação do tempo médio ao valor do tempo de acesso da cache, configurando a cache perfeita, à medida que aumentamos o número de conjuntos.

4) Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule para os tamanhos de bloco 16 e 64. Descreva com suas palavras a comparação dos resultados obtidos.

Go

1024/16/2/r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0494	15	1	0,741
Dados	0,0242	15	0,2867	0,1040721
				0,8450721
CPI sem parada				2
CPI com as paradas devidas a faltas				2,8450721
Desempenho da cache perfeita em relação à cache				**1,42253605**

1024/64/2/r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores - Go	Ciclos devidos a misses
Instruções	0,0041	15	1	0,0615
Dados	0,0041	15	0,2867	0,01763205
				0,07913205
CPI sem parada				2
CPI com as paradas devidas a faltas				2,07913205
Desempenho da cache perfeita em relação à cache				**1,039566025**

Compress

1024/16/2/r -	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0016	15	1	0,024
Dados	0,2182	15	0,6444	2,1091212
				2,1331212
CPI sem parada				2
CPI com as paradas devidas a faltas				4,1331212
Desempenho da cache perfeita em relação a cache				**2,0665606**

1024/64/2/r	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores - Go	Ciclos devidos a misses
Instruções	0,0002	15	1	0,003
Dados	0,0484	15	0,6444	0,4678344
				0,4708344
CPI sem parada				2
CPI com as paradas devidas a faltas				2,4708344
Desempenho da cache perfeita em relação à cache				**1,2354172**

5) Pesquise e explique com suas palavras em que medida afetará no desempenho blocos maiores ou menores.

À medida que ampliamos o tamanho do bloco, a taxa de acertos começa, primeiro por aumentar por causa do princípio da localidade: a probabilidade de que os dados na vizinhança das palavras referenciadas sejam muito provavelmente referenciados num futuro próximo. À medida que o tamanho do bloco cresce, mais dados úteis são trazidos para a cache. Contudo, a taxa de acerto começa a baixar, à medida que o bloco se torna cada vez maior e a probabilidade de usar os novos dados trazidos se torna menor do que a probabilidade de reutilizar os dados que foram substituídos. Dois efeitos específicos entram em jogo:

Blocos maiores reduzem o número de blocos que cabem na memória cache. Como cada extração de um bloco reescreve dados mais antigos na cache, um número pequeno de blocos resulta em que os dados são rapidamente reescritos depois de entrar na memória cache.

À medida que um bloco se torna maior, cada palavra adicional está mais distante da palavra pretendida, por isso é menos provável que venha a ser necessária num futuro próximo.

A relação entre o tamanho do bloco e a taxa de sucesso é complexa, esta é dependente das características de localidade de referências de cada programa. Nas simulações realizadas para tamanho de bloco 16, 32 e 64, tanto no benchmark Compress como no Go foi possível observar o número de acertos aumentar até certo ponto.

No Go o maior tamanho de bloco para a cache de 64 produz a menor razão dos tempos de execução, em torno de 1,03, enquanto para tamanho de bloco 16 é obtido 1,42. No Compress a diferença sobe chegando a 2,06 para tamanho de bloco 16 e a 1,23 para tamanho de bloco 64.

5.5 Roteiro V – Sim-Cache e a política de reposição

Objetivo

O objetivo desta simulação será analisar o desempenho da cache em função das possíveis políticas de reposição utilizadas. Neste caso iremos simular e verificar os resultados, fixando os demais parâmetros e avaliando as políticas para cache de dados, cache de instruções e cache unificada, para os dois benchmarks até aqui utilizados.

Procedimentos

Implementar através do Sim-Cache um sistema com as características de memória cache solicitadas na tabela e rodar neste o benchmark Go para verificar os resultados propostos.

Isto deverá ser realizado, variando a política de reposição da cache em LRU, FIFO e Randômica, fixando o número de conjuntos em 1024, tamanho do bloco em 32 e 64 e a associatividade 2 para o Go.ss.

Elabore uma coluna com o tempo médio de acesso à cache, considerando a taxa de acertos para cada simulação, o tempo de acesso a cache é Tc = 1ns e o tempo de acesso à memória principal Tm = 15 ns.

Deverão ser avaliados os resultados para as caches de instruções e dados separadas e para uma cache unificada.

Go			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	**Tempo Médio de acesso à cache (ns)**
Cache de Instruções IL1			
1024/32/2 – FIFO			
1024/32/2 – Randomico			
1024/32/2 – LRU			
1024/64/2 – Fifo			
1024/64/2 – Randômico			
1024/64/2 – LRU			
Cache de Dados DL1			
1024/32/2 – Fifo			
1024/32/2 – Randômico			
1024/32/2 – LRU			
1024/64/2 – Fifo			
1024/64/2 – Randômico			
1024/64/2 – LRU			
Cache Unificada IL1= DL1			
1024/32/2 – FIFO			
1024/32/2 – Randomico			
1024/32/2 – LRU			
1024/64/2 – Fifo			
1024/64/2 – Randômico			
1024/64/2 – LRU			

Agora utilize o benchmark Compress para avaliar e comparar os resultados, utilizando para isto uma tabela semelhante a anterior :

Após tabular os dados, elabore um gráfico relacionando erros e política empregada.
Perguntas e tarefas para a confecção do relatório:

1. Qual a influência da política de reposição nos erros e acertos da cache de dados? E de instruções?
2. Utilizando a coluna do tempo médio de acesso à memória compare este valor com o calculado através do conceito de cache perfeita.
3. Considere que a máquina simulada possui uma CPI de 2,0, sem qualquer parada, e que a penalidade por faltas é de 15 ciclos de clock para qualquer das faltas. Determine quanto a máquina vai rodar mais rápido se a equiparmos com uma cache perfeita que nunca gere faltas. Use os resultados obtidos na perfilação do roteiro I para Go e Compress e calcule para as políticas de reposição. Descreva com suas palavras a comparação dos resultados obtidos.
4. Pesquise e descreva as políticas de reposição empregadas no SimpleScalar.

Resultados Roteiro V

Para implementação desta experiência foram mantidos fixos o número de conjuntos em 1024 e a associatividade em 2, variando-se somente tamanho do bloco e política de reposição em todas as simulações.

Tabela 5.8: Comparativo de erros e taxa de misses com a política de reposição e tamanho dos blocos no Go, utilizando o Sim-Cache.

Go			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
1024/32/2 – FIFO	161686	0,0162	1,243
1024/32/2 – Randômico	159128	0,0159	1,2385
1024/32/2 – LRU	155118	0,0155	1,2325
1024/64/2 – Fifo	42604	0,0043	1,0645
1024/64/2 – Randômico	40578	0,0041	1,0615
1024/64/2 – LRU	41215	0,0041	1,0615
Cache de Dados DL1			
1024/32/2 – FIFO	27256	0,01	1,15
1024/32/2 – Randômico	27935	0,0103	1,1545
1024/32/2 – LRU	26513	0,0098	1,147
1024/64/2 – Fifo	10594	0,0039	1,0585
1024/64/2 – Randômico	11015	0,0041	1,0615
1024/64/2 – LRU	10341	0,0038	1,057
Cache Unificada IL1=DL1			
1024/32/2 – FIFO	271129	0,0213	1,3195
1024/32/2 – Randômico	276282	0,0217	1,3255
1024/32/2 – LRU	254306	0,02	1,3
1024/64/2 – Fifo	97852	0,0077	1,1155
1024/64/2 – Randômico	98285	0,0077	1,1155
1024/64/2 – LRU	91635	0,0072	1,108

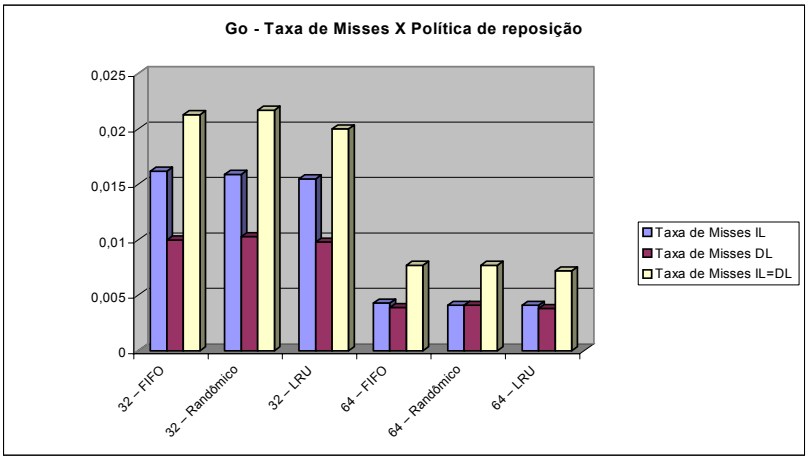


Figura 5.8: Misses em função da política de reposição – benchmark Go.

Tabela 5.9 Comparativo de erros e taxa de misses com a política de reposição e tamanho dos blocos no Compress, utilizando o Sim-Cache

Compress			
Conjuntos / Tamanho do bloco / Associatividade / Política de reposição	Misses	Taxa de Misses	Tempo Médio de acesso à cache (ns)
Cache de Instruções IL1			
1024/32/2 – FIFO	923	0,0003	1,0045
1024/32/2 – Randômico	989	0,0003	1,0045
1024/32/2 – LRU	923	0,0003	1,0045
1024/64/2 – Fifo	520	0,0002	1,003
1024/64/2 – Randômico	538	0,0002	1,003
1024/64/2 – LRU	520	0,0002	1,003
Cache de Dados DL1			
1024/32/2 – FIFO	228396	0,1093	2,6395
1024/32/2 – Randômico	226960	0,1086	2,629
1024/32/2 – LRU	228396	0,1093	2,6395
1024/64/2 – Fifo	113493	0,0543	1,8145
1024/64/2 – Randômico	101141	0,0484	1,726
1024/64/2 – LRU	113502	0,0543	1,8145
Cache Unificada IL1=DL1			
1024/32/2 – FIFO	242136	0,0463	1,6945
1024/32/2 – Randômico	242265	0,0462	1,693
1024/32/2 – LRU	241568	0,0462	1,693
1024/64/2 – Fifo	121491	0,0233	1,3495
1024/64/2 – Randômico	111397	0,0213	1,3195
1024/64/2 – LRU	121310	0,0232	1,348

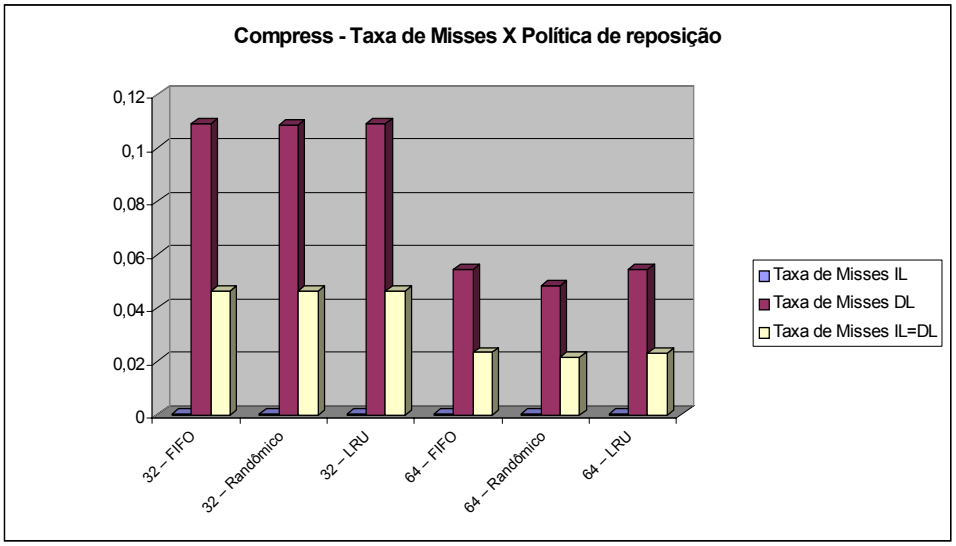


Figura 5.9: Misses em função da política de reposição – benchmark Compress.

1. Qual a influência da política de reposição nos erros e acertos da cache de dados? E de Instruções?

Pela análise dos gráficos percebemos basicamente que a política de reposição de blocos empregada pouco afetou no número de acertos e erros da cache tanto na cache de dados, instruções assim como unificada.

2. Utilizando a coluna do tempo médio de acesso a memória compare este valor com o calculado através do conceito de cache perfeita.

Foi elaborada uma coluna da tabela com o cálculo do tempo médio de acesso à cache, considerando a taxa de acertos para cada simulação e que o tempo de acesso à cache é Tc = 1ns e o tempo de acesso à memória principal Tm = 15 ns, comparando este valor com o calculado através do conceito de cache perfeita.

O tempo médio de acesso foi calculado com base na equação

Tméd= Tc+(1-h)Tm

onde : h= no. de hits ; Tc=tempo de acesso à cache; Tm=tempo de acesso a memória principal. Percebemos não ocorrer alteração do tempo médio de acesso da cache com a troca da política de reposição.

Go

1024/32/2/FIFO	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0162	15	1	0,243
Dados	0,01	15	0,2867	0,043005
				0,286005
CPI sem parada				2
CPI com as paradas devidas a faltas				2,286005
Desempenho da cache perfeita em relação à cache				**1,143003**

1024/32/2/Randômico	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0159	15	1	0,2385
Dados	0,0103	15	0,2867	0,044295
				0,282795
CPI sem parada				2
CPI com as paradas devidas a faltas				2,282795
Desempenho da cache perfeita em relação à cache				**1,141398**

1024/32/2/LRU	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a falta de instruções
Instruções	0,0155	15	1	0,2325
Dados	0,0098	15	0,2867	0,042145
				0,274645
CPI sem parada				2
CPI com as paradas devidas a faltas				2,274645
Desempenho da cache perfeita em relação à cache				**1,137322**

Compress

1024/32/2/FIFO	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0003	15	1	0,0045
Dados	0,1093	15	0,6444	1,056494
				1,060994
CPI sem parada				2
CPI com as paradas devidas a faltas				3,060994
Desempenho da cache perfeita em relação à cache				**1,530497**

1024/64/2/Randômico	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0003	15	1	0,0045
Dados	0,1086	15	0,6444	1,049728
				1,054228
CPI sem parada				2
CPI com as paradas devidas a faltas				3,054228
Desempenho da cache perfeita em relação à cache				**1,527114**

1024/64/2/LRU	Taxa de Misses	Penalidade por Falta	Freqüência de Loads/Stores	Ciclos devidos a misses
Instruções	0,0003	15	1	0,0045
Dados	0,1093	15	0,6444	1,056494
				1,060994
CPI sem parada				2
CPI com as paradas devidas a faltas				3,060994
Desempenho da cache perfeita em relação à cache				**1,530497**

Quanto ao desempenho no Go há uma pequena diferença em relação às políticas testadas, ficando em 1,143 para FIFO, 1,141 para Randômica e 1,137 para LRU, sendo estes valores em relação à cache perfeita.

No Compress os valores comparados de desempenho aumentam em relação ao GO, mas se mantêm quase constantes, obtendo-se 1,530 para FIFO, 1,527 para randômica e 1,530 para LRU. Percebemos que a política de reposição não produz alterações nestes valores.

5.6 Roteiro VI – Sim-Cheetah e análise da variação de diversos parâmetros de cache para alguns tamanhos fixos.

Objetivos

Nesta aula iremos utilizar as facilidades do multisimulador Sim-Cheetah, que permite a simulação de várias configurações de cache em apenas uma execução, para analisar o comportamento da cache e dos benchmarks quando é dada ênfase à fixação do tamanho da cache, obtendo assim resultados por faixas de valores dos parâmetros, sem necessidade de fixar um valor de cada vez para cada parâmetro. Neste caso serão variados o número de conjuntos, a associatividade e o tamanho do bloco, porém serão mantidos fixos o tamanho total da cache e a política de reposição.

Procedimentos

Através do Sim-Cheetah iremos implementar um modelo de processador com configuração de memória cache de tamanhos 8k, 16k e 32k. Para completar a tabela deverão ser utilizados os resultados apresentados nos relatórios de simulação do SimpleScalar. Você deverá realizar as simulações, utilizando o Sim-Cheetah, para simular e determinar a taxa de misses da cache e deverá configurar o simulador para produzir resultados para as seguintes condições:

Caches com variação do número de conjuntos entre 128 e 512, associatividade entre 1 e 4, tamanho de bloco (linhas) de 16, 32 e 64 bytes. A política de reposição é lru.

Os parâmetros que compõem a cache foram normalizados de forma a facilitar a confecção e comparação dos gráficos, ficando o valor para o número de conjuntos: 512 = 1 e consequentemente 256 = 0,5 e 128 = 0,25. Para a associatividade 4 = 1, 2 = 0,5 e 1 = 0,25 e para tamanho do bloco 64 ,32 e 16 eqüivalem respectivamente a 1,0,5 e 0,25.

Utilize os benchmarks da família SPEC, que têm sido testados nas simulações.

GO

Tam. da Cache	No. Conjuntos	Associat.	Tam. de Bloco	No. Conjuntos	Associat.	Tam. do Bloco	Taxa de miss
8192	128	1	64	0,25	0,25	1	
8192	128	2	32	0,25	0,5	0,5	
8192	256	1	32	0,5	0,25	0,5	
8192	128	4	16	0,25	1	0,25	
8192	256	2	16	0,5	0,5	0,25	
8192	512	1	16	1	0,25	0,25	
Tam. da Cache	**No. Conjuntos**	**Associat.**	**Tam. de Bloco**	**No. Conjuntos**	**Associat.**	**Tam. do Bloco**	**Taxa de miss**
16384	128	2	64	0,25	0,5	1	
16384	256	1	64	0,5	0,25	1	
16384	128	4	32	0,25	1	0,5	
16384	256	2	32	0,5	0,5	0,5	
16384	512	1	32	1	0,25	0,5	
16384	256	4	16	0,5	1	0,25	
16384	512	2	16	1	0,5	0,25	
Tam. Da Cache	**No. Conjuntos**	**Associat.**	**Tam. de Bloco**	**No. Conjuntos**	**Associat.**	**Tam. do Bloco**	**Taxa de miss**
32768	128	4	64	0,25	1	1	
32768	256	2	64	0,5	0,5	1	
32768	512	1	64	1	0,25	1	
32768	256	4	32	0,5	1	0,5	
32768	512	2	32	1	0,5	0,5	
32768	512	4	16	1	1	0,25	

Use o mesmo procedimento e tabela para o Compress

Perguntas e tarefas para a confecção do relatório

Você deverá elaborar um gráfico de barras plotando os resultados de cada simulação com a taxa de misses no eixo das abscissas e as taxas de associatividade, número de conjuntos e tamanho dos blocos da cache no eixo das ordenadas. Para manter a uniformidade nos valores considere taxas onde o maior valor de cada parâmetro é 1.

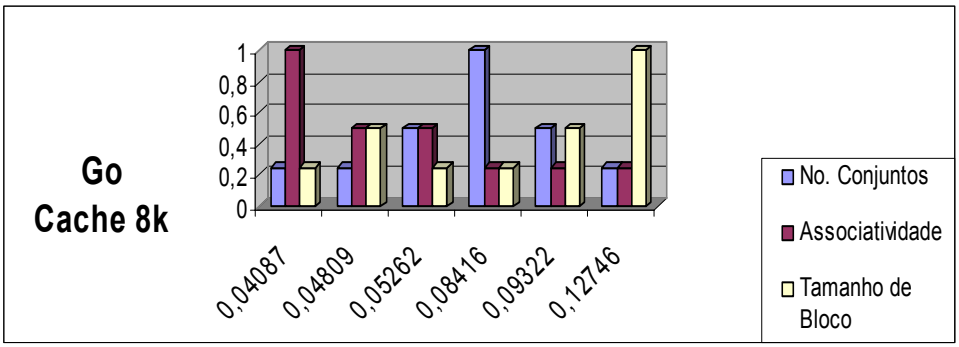
Observando o gráfico você poderá avaliar os benefícios, mantendo o tamanho da cache, com a variação da associatividade da cache, do tamanho do bloco e do número de conjuntos. Com estas observações responda as questões abaixo:

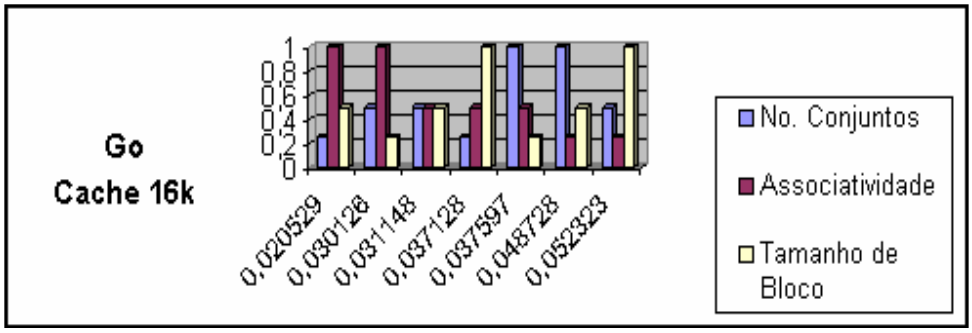
1. Qual o efeito do incremento da associatividade na taxa de misses da cache?
2. Qual o efeito do aumento no número de conjuntos na taxa de misses para a cache de 16k?
3. Para os tamanhos de cache analisados qual o parâmetro que ao ser aumentado produz a menor taxa de misses? Isto ocorre nos dois benchmarks?
4. Quais as vantagens em termos de desempenho e os possíveis efeitos negativos no aumento do tamanho da cache, aumento da associatividade e aumento no tamanho do bloco?

Resultados Roteiro VI

Tabela 5.10: Análise de diversos tamanhos de Cache com variação de parâmetros e resultando taxas de misses para o Go.

Go							
Tamanho Da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
8192	128	4	16	0,25	1	0,25	0,04087
8192	128	2	32	0,25	0,5	0,5	0,04809
8192	256	2	16	0,5	0,5	0,25	0,05262
8192	512	1	16	1	0,25	0,25	0,08416
8192	256	1	32	0,5	0,25	0,5	0,09322
8192	128	1	64	0,25	0,25	1	0,12746
Tamanho da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
16384	128	4	32	0,25	1	0,5	0,02053
16384	256	4	16	0,5	1	0,25	0,03013
16384	256	2	32	0,5	0,5	0,5	0,03115
16384	128	2	64	0,25	0,5	1	0,03713
16384	512	2	16	1	0,5	0,25	0,0376
16384	512	1	32	1	0,25	0,5	0,04873
16384	256	1	64	0,5	0,25	1	0,05232
Tamanho da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
32768	128	4	64	0,25	1	1	0,00949
32768	256	2	64	0,5	0,5	1	0,01116
32768	256	4	32	0,5	1	0,5	0,01477
32768	512	2	32	1	0,5	0,5	0,01592
32768	512	4	16	1	1	0,25	0,02363
32768	512	1	64	1	0,25	1	0,0273





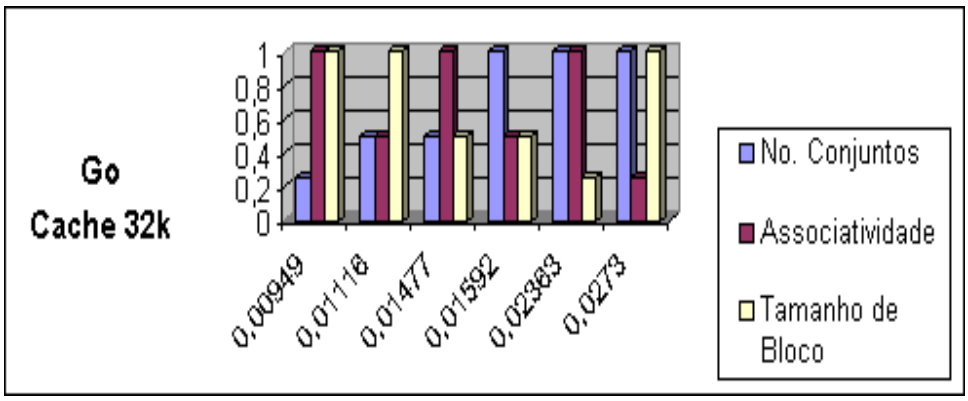


Figura 5.10: Gráficos dos diversos tamanhos de cache com parâmetros representados por barras e variação em função da taxas de misses para o Go.

Tabela 5.11: Análise de diversos tamanhos de Cache com variação de parâmetros e resultando taxas de misses para o Compress.

	COMPRESS						
Tamanho da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
8192	128	1	64	0,25	0,25	1	0,056042
8192	128	2	32	0,25	0,5	0,5	0,109333
8192	256	1	32	0,5	0,25	0,5	0,110294
8192	128	4	16	0,25	1	0,25	0,218226
8192	128	4	16	0,25	1	0,25	0,218226
8192	512	1	16	1	0,25	0,25	0,219241
Tamanho da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
16384	128	2	64	0,25	0,5	1	0,05482
16384	256	1	64	0,5	0,25	1	0,05571
16384	128	4	32	0,25	1	0,5	0,109274
16384	256	2	32	0,5	0,5	0,5	0,109284
16384	512	1	32	1	0,25	0,5	0,110009
16384	256	4	16	0,5	1	0,25	0,218036
16384	512	2	16	1	0,5	0,25	0,218149
Tamanho da Cache	No. Conjuntos	Associatividade	Tamanho de Bloco	No. Conjuntos	Associatividade	Tamanho de Bloco	Taxa de misses
32768	128	4	64	0,25	1	1	0,054804
32768	256	2	64	0,5	0,5	1	0,054811
32768	512	1	64	1	0,25	1	0,055426
32768	256	4	32	0,5	1	0,5	0,109272
32768	512	2	32	1	0,5	0,5	0,109274
32768	512	4	16	1	1	0,25	0,218134

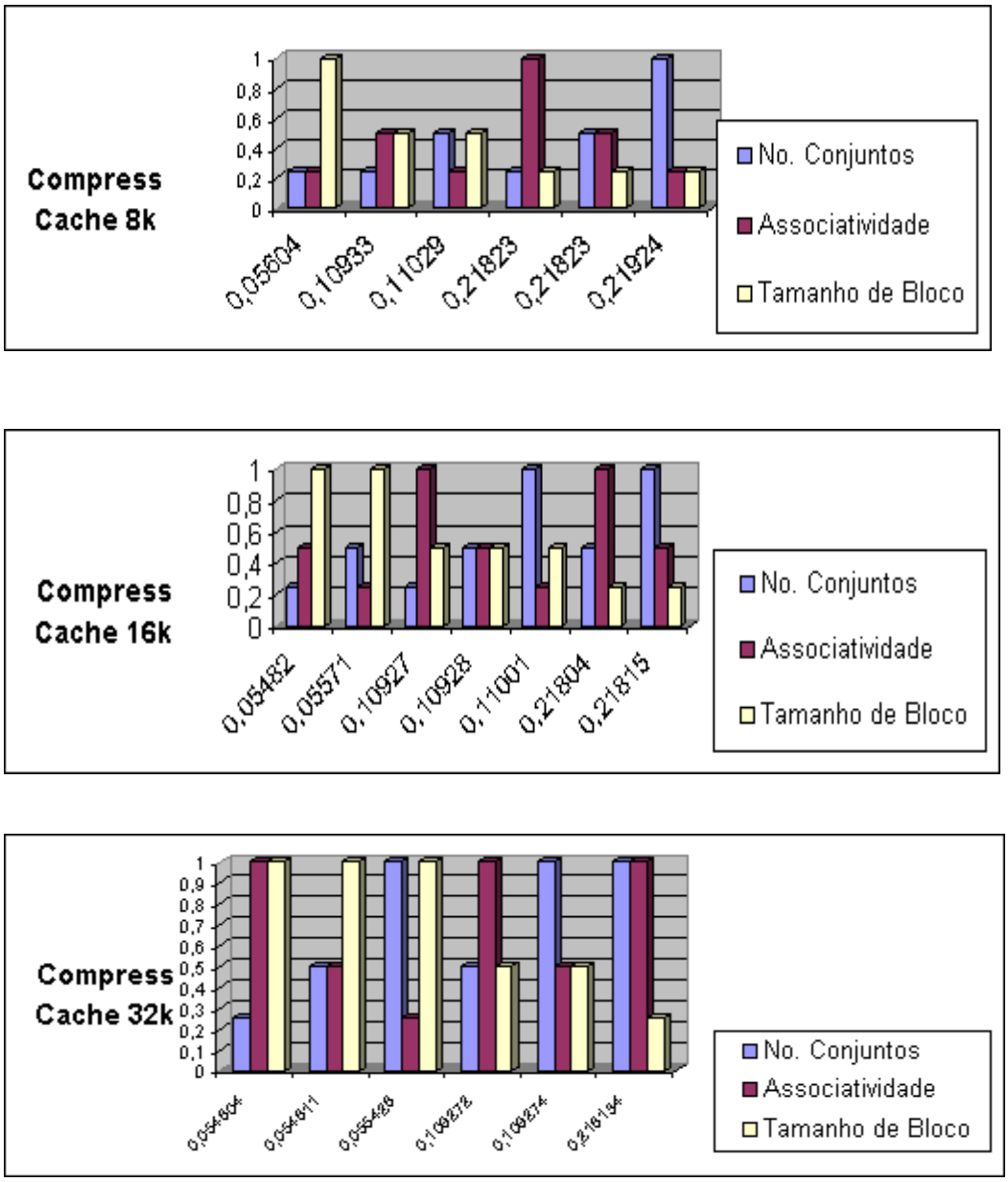


Figura 5.11: Gráficos dos diversos tamanhos de cache com parâmetros representados por barras e variação em função da taxas de misses para o Compress.

1. Qual o efeito do incremento da associatividade na taxa de misses da cache?

No Go para caches de tamanho: 8 e 16k, à medida que reduzimos a associatividade a taxa de misses aumenta. Para caches de tamanho 32k o número de conjuntos apresenta relação inversamente proporcional à taxa de misses. O tamanho de blocos apresenta relação linear somente na cache de 32k, sendo inversamente proporcional à taxa de misses.

No Compress há uma influência maior do tamanho de blocos na taxa de misses, mantendo-se esta inversamente proporcional para os tamanhos 8, 16 e 32k. Os demais parâmetros apresentam comportamento irregular em relação à taxa de misses.

2. Qual o efeito da variação de parâmetros na taxa de misses para a cache de 8,16 e 32k?

Foi verificado que não existe um comportamento padrão nos exemplos analisados. E em algumas situações aumentando a associatividade teremos maior desempenho, mas a medida que mudamos o programa em avaliação teremos melhores resultados com a alteração de parâmetros como o tamanho de blocos e número de conjuntos. Concluí-se que o próprio algoritmo reagirá de forma diferente em termos de desempenho possibilitando a criação de um ambiente mais favorável, dependendo do parâmetro modificado.

3. Para os tamanhos de cache analisados qual o parâmetro que ao ser aumentado produz a menor taxa de misses? Isto ocorre nos dois benchmarks?

A associatividade nas configurações analisadas é o parâmetro que gera a menor taxa de misses, considerando a variação de 1 a 4 analisada nos gráficos e simulações para o benchmark Go, obtemos uma redução de 0,127 para 0,040 na taxa de misses da cache de 8k, 0,052 para 0,020 em 16k e 0,027 para 0,009 em 32k. No caso do Compress o parâmetro de maior influência é o tamanho de blocos onde temos uma variação de 16 para 64 com a respectiva redução na taxa de misses de 0,21 para 0,05 nas caches de 8, 16 e 32k.

4. Quais as vantagens em termos de desempenho e os possíveis efeitos negativos no aumento do tamanho da cache, aumento da associatividade e variação no tamanho do bloco?

O aumento de tamanho da cache ou um aumento na associatividade produz como fator positivo uma redução na taxa de faltas, mas pode aumentar o tempo de acesso.

No caso de redução no tamanho do bloco com aumento dos outros parâmetros ocorre um aumento da taxa de faltas no Compress conforme podemos ver na Figura 5.11, diferente do que ocorre no Go onde percebe-se um comportamento irregular da taxa de misses relacionada ao tamanho dos blocos Figura 5.10.

5.7 Roteiro VII – Sim-Bpred e análise de preditores de desvio

Objetivos

Nesta aula iremos utilizar o simulador de preditores de desvio e os resultados do Sim-Profile produzidos no Roteiro I para avaliar os diversos tipos de predição de desvio possíveis de serem implementados através do Sim-Bpred, verificando quais os tipos com melhor eficiência para os workbenchs do conjunto SPEC analisados.

Utilizaremos nesta análise os preditores estudados na seção 2.6 ou seja preditores estáticos taken e nottaken; e os preditores dinâmicos de desvios utilizando contadores saturados (bimod), em 2 níveis (2lev) e híbridos (comb).

Procedimentos

Implemente as simulações abaixo utilizando o comando

./Sim-bpred –bpred <tipo> -redir:sim results/simbpredtipo go.ss 50 9 2stone9.in
ou **./Sim-bpred -h** para verificar opções

1ª. Etapa: Você deverá realizar as simulações utilizando o Sim-Bpred, variando a técnica de predição de desvio e preenchendo a tabela abaixo para o benchmark Go:

Go					
Sim-Bpred	Nottaken	Taken	2lev	Bimod	Comb
Bpred-hits					
Bpred-misses					
Bpred –rate misses					

2ª. Etapa: agora você irá utilizar o benchmark Compress, para preencher a tabela e comparar os resultados como as simulações da 1ª. Etapa.

Perguntas e tarefas para a confecção do relatório

Após tabular os dados elabore um gráfico de barras da taxa de misses em função do tipo de predição, relacionando os resultados com os parâmetros.

1. Que diferença encontramos nos resultados das simulações realizadas sobre o benchmark que tem maior número de desvios condicionais?
2. Com relação aos benchmarks apresentados, o que podemos dizer com relação aos tipos de predição e o desempenho em cada caso?
3. Pesquise e descreva a predição Bimodal.
4. Quais as vantagens e desvantagens da implementação da predição em Hardware e Software?

Resultados Roteiro VII

1ª. Etapa

Tabela 5.12: Comparativo de acertos, erros e taxa de misses para preditores de desvio no Go, utilizando o Sim-Bpred

Go					
Sim-Bpred	**Nottaken**	**Taken**	**2lev**	**bimod**	**comb**
Bpred-hits	46440076	52024539	60280416	64246104	65380411
Bpred-misses	33846617	28262154	20006277	16040589	14906282
Bpred –rate misses	0,4215719	0,352015421	0,249185466	0,199791	0,185663

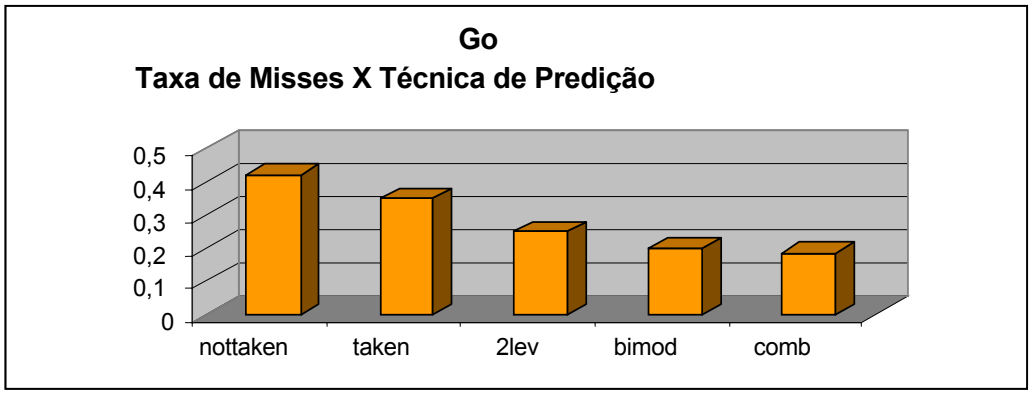


Figura 5.12: – Taxa de Misses em função das técnicas de predição de desvio Go

2ª. Etapa: agora você irá utilizar o benchmark Compress, para preencher a tabela e comparar os resultados com as simulações da 1ª. Etapa.

Tabela 5.13: Comparativo de acertos, erros e taxa de misses para preditores de desvio no Compress, utilizando o Sim-Bpred

Compress					
Sim-Bpred	**Nottaken**	**Taken**	**2lev**	**bimod**	**comb**
Bpred-hits	55873775	50771486	72884475	70562681	73125058
Bpred-misses	22358364	27460653	5347664	7669458	5107081
Bpred –rate misses	0,2857951	0,351014984	0,068356357	0,098035	0,065281

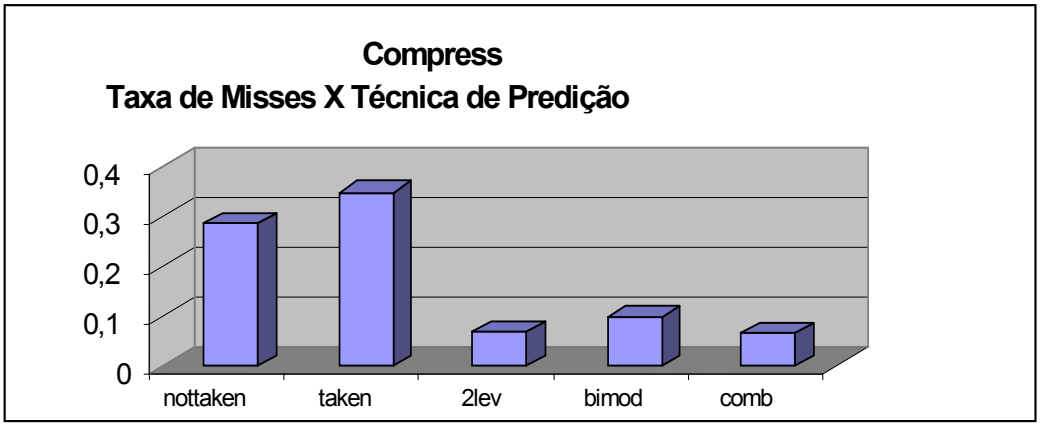


Figura 5.13: Taxa de Misses em função das técnicas de predição de desvio Compress

1. Que diferença encontramos nos resultados das simulações realizadas sobre o benchmark que tem maior número de desvios condicionais?

Recorrendo aos resultados do Roteiro I percebemos que o Compress tem 7,85% de desvios condicionais ou seja pouco mais que metade dos desvios condicionais do Go (11,33%), Como primeira conseqüência disto, percebe-se que é maior taxa de misses no Go em relação ao Compress.

2. Considerando os benchmarks apresentados, o que podemos dizer em relação aos diferentes tipos de predição e o desempenho em cada caso?

Tanto o Go como o Compress tem como melhor preditor em termos de acertos o preditor Combinado, contudo com relação ao segundo mais eficiente enquanto no Go temos o preditor Bimodal, no Compress temos o Preditor de 2 Níveis.

Foi percebida significativa redução na taxa de misses com a utilização de técnicas de predição dinâmica em relação às técnicas estáticas. Na comparação com o benchmark GO a redução chega a mais de 50% entre a predição estática Nottaken (0,4216) e predição dinâmica mais eficiente comb (0,1856). A faixa de valores entre as predições e estáticas está entre 0,3522 e 0,4216, enquanto que para as predições dinâmicas a variação está entre 0,1856 e 0,2492.

Estas faixas e diferenças são ampliadas quando utilizamos o benchmark Compress que no caso da predição estática está entre 0,2857 e 0,3510; enquanto nas predições dinâmicas a variação entre a técnica mais eficiente (comb) e a menos eficiente (bimod) é 0,0652 para 0,0980.

5.8 Roteiro VIII – Sim-Outorder e análise de desempenho de cache.

Objetivos

Nesta aula iremos introduzir o Sim-Outorder para análise de desempenho de cache com avaliação de tempo. Este é o simulador mais completo de todo o conjunto de ferramentas SimpleScalar, pois além de realizar simulações funcionais como Sim-Cache e Sim-Bpred, permite conceitos teóricos importantes como cache perfeita e preditor perfeito, muito úteis para que possamos ter referências de análise. Ele acrescenta ainda a avaliação de desempenho em função de tempo de execução, utilizando medidas de ciclos por instrução, permitindo a variação no número de unidades de execução tanto de inteiros como ponto flutuante e implementando a execução fora de ordem.

Procedimentos

Como todos os resultados que serão obtidos no uso do simulador Sim-Outorder, até que sejam mudados os parâmetros, dependem e só tem validade para arquitetura default, iremos dedicar algum tempo à análise destes parâmetros, para isto rode o comando ./**Sim-Outorder –h** para apresentação de parâmetros na tela e tente identificar o que significa cada um deles.

Tabela 5.14: Valores default para o Simulador Sim-Outorder.

Descrição Arquitetura	Default
Latência de erro(miss) na Predição de Salto	3
Largura da Instrução decodificada (No. de Instruções)	4
Despacho simples em Ordem	False
Largura da Instrução entregue (No. de Instruções).	4
ULA´s Para Inteiros	4
ULA´s Para Ponto Flutuante	4
Multiplicador de Inteiros	1
Multiplicador de Ponto Flutuante	1
Perfeita Predição de Desvio	Bimod
Cache de Instruções L1 – Mapeamento direto .	16K, 32 bytes
Cache de Dados L1 - Mapeamento Direto.	16K, 32 bytes
Latência para Acertos na cache L1 de dados e instruções	1
Latência para miss na cache L1 de dados e instruções	1
Cache L2	Sim
Latência de acesso à memória (primeira palavra, palavras restantes).	18 2
Largura dos dados na memória	8
Latência de erro (miss)TLB	30

Rode os benchmarks disponíveis com os parâmetros default e registre o número de instruções por ciclo (IPC) e as taxas de misses para as caches IL1, DL1 e cache unificada UL1 na tabela abaixo.

Agora execute a mesma simulação com o sistema de memória “perfect”. Fixando a latência de cada unidade de memória (L1 cache, L2 cache e TLB) em 1 ciclo, você pode simular o que acontece se o processador não espera pelo sistema de memória para entregar ou aceitar dados. Novamente registre o IPC e as taxas de misses para

caches IL1, DL1 e cache unificada UL1 na tabela. Verifique a atividade da memória utilizando as duas simulações e comparando as taxas de misses.

Sim-Outorder	**Cache**		
Go	Default	Perfect	None
IPC			
CPI			
Taxa de misses IL1			
Taxa de misses DL1			
Taxa de misses UL1			

Compress	Default	Perfect	None
IPC			
CPI			
Taxa de misses IL1			
Taxa de misses DL1			
Taxa de misses UL1			

Perguntas e tarefas para a confecção do relatório

1) Calcule a troca nas IPC (Instruções por ciclo) partindo da primeira execução para a segunda. Defina o percentual de troca:

Razão % = 100 X IPC novo / IPC anterior

2) Repita estas simulações para quando o sistema não contém caches. (Ou seja, il1 none, dl1 none, ul1 none, tlb none).

3) Quais destes experimentos informam a você sobre a importância do sistema de memória na criação de processadores mais rápidos? Você pode chegar a esta conclusão utilizando um simulador sem tempo determinado como o Sim-Cache? Explique sua resposta.

4) Calcule a velocidade da simulação com o número de instruções e o tempo. Compare o resultado obtido com a velocidade fornecida na saída do Sim-Outorder.

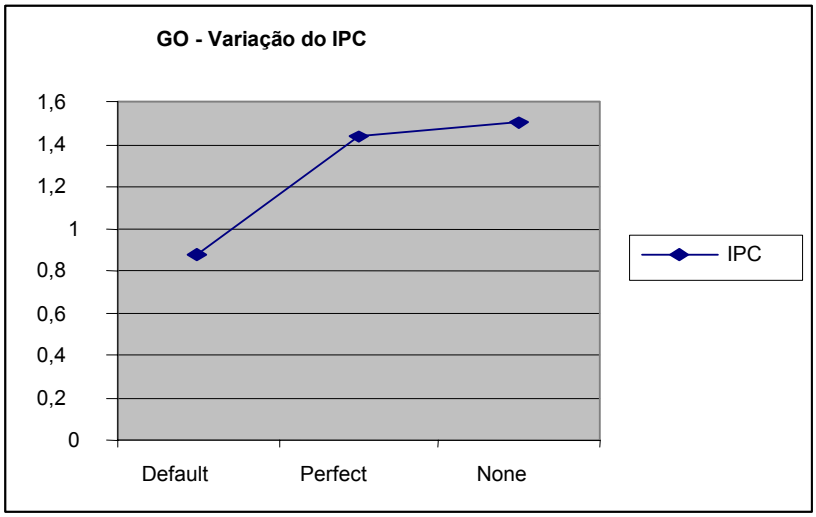
Resultados Roteiro VIII

1a. Etapa:

Tabela 5.15: Valores do IPC, CPI e taxas de misses nas caches de instruções IL1, cache de dados DL1 e Cache unificada UL1 para configurações default, perfeita e sem cache no Go & Compress, utilizando o Sim-Outorder.

Sim-Outorder	Cache		
Go	Default	Perfect	None
IPC	0,8789	1,4335	1,5041
CPI	1,1378	0,6976	0,6649
Taxa de misses IL1	0,0669	0,0701	
Taxa de misses DL1	0,0096	0,0095	
Taxa de misses UL1	0,0118	0,0114	

Compress	Default	Perfect	None
IPC	1,1261	1,1772	1,1783
CPI	0,888	0,8495	0,8487
Taxa de misses IL1	0	0	
Taxa de misses DL1	0,1818	0,1525	
Taxa de misses UL1	0,0106	0,0132	



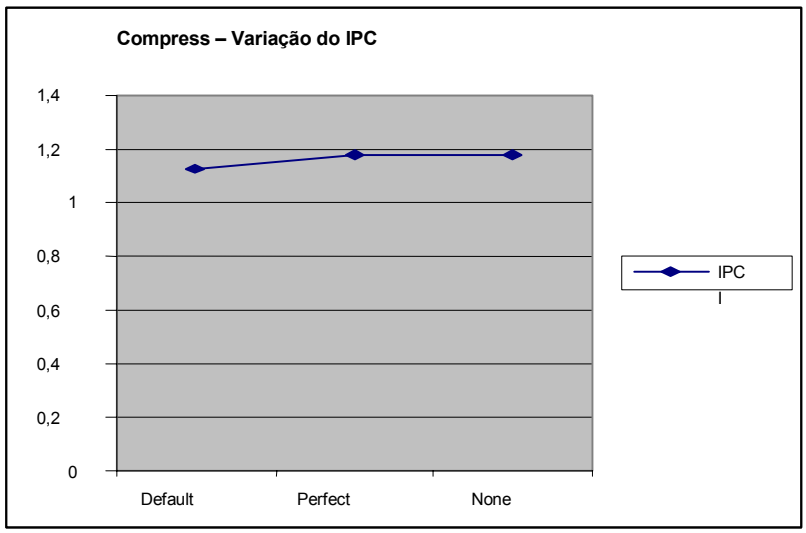


Figura 5.14: Variação do IPC e CPI em relação à cache perfeita, default e sem cache para Go e Compress, utilizando Sim-Outorder

O número de ciclos por instrução CPI para o Go é reduzido sensivelmente quando trocamos da cache default para a cache perfeita o que equivale dizer que o número de instruções por ciclo é praticamente dobrado.

No caso do Compress o aumento do número de Instruções por ciclo IPC do modo default para a cache perfeita é muito pequeno conforme observamos nos gráficos

1.Calcule a troca nas IPC´s (Instruções por ciclo) partindo da primeira execução para a segunda. Defina o percentual de troca.

Razão % = 100 X IPC novo / IPC anterior

IPC – Go
Razão % = 100 X 1,4335 / 0,8789
Razão = 163,10 %

Observa-se que o número de instruções por ciclo (IPC) tem um aumento de 63,1% na cache perfeita em relação aos valores default.

IPC – Compress
Razão % = 100 X 1,1772 / 1,1261
Razão = 104,54 %

Observa-se que o número de instruções por ciclo (IPC) tem um aumento de 4,54% na cache perfeita em relação aos valores default. Neste caso temos um menor acréscimo em função do algoritmo do Compress, o que indica pouca diferença na utilização por parte da aplicação da cache perfeita e cache default.

2. Repita estas simulações para quando o sistema não contém caches. (Ou seja, il1 none, dl1 none ul1 none, tlb none) Compare o valor do IPC entre a cache perfeita e sem cache.

Conforme a TABELA 5.15 quando o sistema não contém caches o IPC permanece praticamente estável em relação a cache perfeita, no caso do compress seu valor passa de 1,1772 na cache perfeita para 1,1783 indicando possível erro de arredondamento. Já no caso do Go os valores do IPC partem de 1,4335 na cache perfeita para 1,5041.

3. Quais destes experimentos informam a você sobre a importância do sistema de memória na criação de processadores mais rápidos. Você pode chegar a esta conclusão utilizando um simulador sem tempo determinado como o Sim-Cache?

Os experimentos que envolvem métricas de desempenho, como IPC e CPI, servem como base para a criação de processadores mais rápidos, pois com eles podemos refinar os resultados inicialmente produzidos em simuladores funcionais e ainda comparar estes resultados com sistemas de cache perfeita. Não é possível chegar a estas conclusões utilizando o Sim-Cache, pois nossa preocupação não são somente os acertos e erros, mas também o tempo necessário (latência) e o número de instruções para que isto ocorra.

5.9 Roteiro IX – Sim-Outorder e predição de desvio

Objetivos

Nesta simulação iremos utilizar o Sim-Outorder para avaliação e análise de desempenho das técnicas de predição de desvios, com a utilização dos benchmarks Compress e Go. Neste caso iremos manter a arquitetura default e variar as técnicas de predição de desvio para a arquitetura simulada com controle de tempo.

Procedimentos

O Sim-Outorder permite a simulação de seis diferentes tipos de predições de desvio: os mesmos cinco suportados pelo Sim-Bpred, e ainda como diferencial o preditor perfeito (que é similar ao cache perfeito, opcional para tornar possível a comparação entre os resultados obtidos nas simulações embora seja impossível ser implementado.) Utilizando o Sim-Outorder em vez do Sim-Bpred repita as simulações realizadas na experiência VII, armazenando os CPI´s e IPC´s para cada técnica de predição.

Use o preditor perfeito (deve ter taxa de predição 100%) e registre o número de ciclos e IPC.

Go

Go	taken	nottaken	2lev	comb	bimod	perfect
Bpred Taxa de hits						
Bpred-IPC						
Bpred-CPI						

Utilize tabela semelhante para o **Compress**

Perguntas e tarefas para a confecção do relatório

1. Qual a influência do tipo de predição de desvio no IPC do processador?
2. Por que no Sim-Bpred não foi possível ser também utilizado o preditor perfeito?
3. Considere o IPC do preditor perfeito como referência e calcule a razão em cada um dos outros preditores através da fórmula:

Razão % = 100 X IPC novo / IPC anterior

4. Monte uma tabela e compare os resultados obtidos das diversas técnicas de predição de desvios entre o Sim-Bpred – Roteiro VI com o Sim-Outorder. Os resultados são diferentes? Por quê?

Resultados Roteiro IX

Tabela 5.16: Variação do IPC, CPI e hits para preditores de desvio no Go, considerando o uso do Sim-Outorder e comparando com o preditor perfeito

Compress	taken	nottaken	2lev	comb	bimod	perfect
Bpred Taxa de hits	0,2858	0,2858	0,8456	0,9294	0,902	1
Bpred-IPC	0,7741	0,77	1,3912	1,5982	1,4974	1,6302
Bpred-CPI	1,2918	1,2987	0,7188	0,6257	0,6678	0,6134

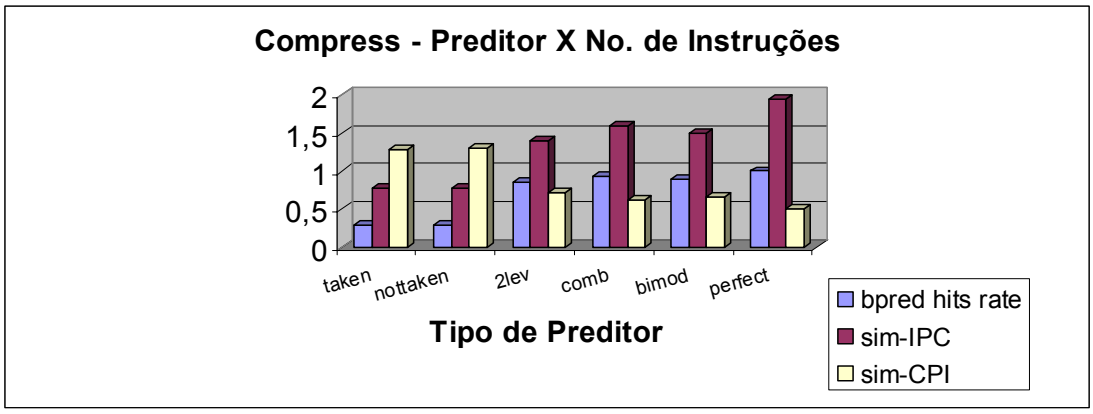


Figura 5.15: Métricas em função da técnica de predição de desvio. Compress

Tabela 5.17: Variação do IPC, CPI e hits para preditores de desvio no Compress, considerando o uso do Sim-Outorder e comparando com o preditor perfeito.

GO.ss	taken	nottaken	2lev	comb	bimod	Perfect
Bpred Taxa de hits	0,3522	0,3522	0,7189	0,8055	0,8002	1
Bpred-IPC	0,6774	0,6691	0,8376	0,8848	0,8789	1,0408
Bpred-CPI	1,4762	1,4946	1,1939	1,1301	1,1378	0,9608

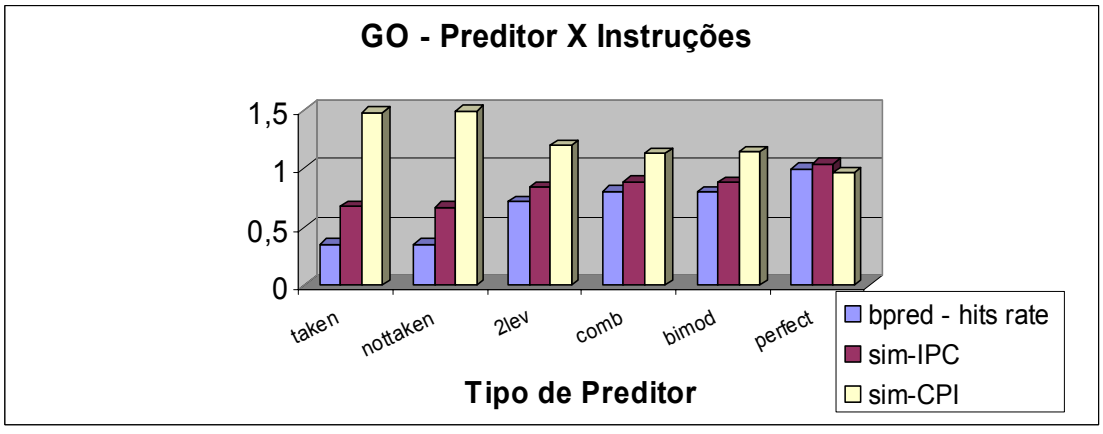


Figura 5.16: Métricas em função da técnica de predição de desvio. Go

1. Qual a influência do tipo de predição de desvio no IPC do processador?

Através da análise dos gráficos e comportamento nas tabelas percebemos que à medida que o tipo de preditor acerta mais as predições, o número de instruções por ciclo aumenta proporcionalmente, sendo o limite superior o preditor perfeito.

2. Por que no Sim-Bpred não foi possível ser também utilizado o preditor perfeito?

O Sim-Bpred é um simulador funcional não tendo sentido testar a funcionalidade de um preditor de desvios que acerta todas as predições. Neste caso o simulador mais adequado para este tipo de utilização é o Sim-Outorder uma vez que o mesmo testa as funcionalidades e o desempenho da configuração simulada, comparando ao preditor perfeito.

3. Considere o IPC do preditor perfeito como referência e calcule a razão em cada um dos outros preditores através da fórmula:

Razão % = 100 X IPC novo / IPC anterior

Go	taken	Nottaken	2lev	Comb	bimod	Perfect
sim-IPC	0,6774	0,6691	0,8376	0,8848	0,8789	1,0408
Razão %	65,085	64,287	80,477	85,012	84,445	100

Compress	taken	Nottaken	2lev	Comb	bimod	Perfect
sim-IPC	0,7741	0,77	1,3912	1,5982	1,4974	1,6302
Razão %	39,622	39,412	71,208	81,804	76,644	100

A análise das razões favorece a predição com técnica combinada, pelo número de predições certas e ainda proximidade com a ideal, tanto no benchmark Compress, quanto no Go, demonstrando desta forma, ser o modo mais eficiente estudado até aqui.

4. Monte uma tabela e compare os resultados obtidos das diversas técnicas de predição de desvios entre o Sim-Bpred – Roteiro VI com o Sim-Outorder. Os resultados são diferentes? Por quê?

Tabela 5.18: Comparativo entre resultados do Sim-Bpred e Sim-Outorder para predição de desvio utilizando o Go e Compress

Sim-Outorder X Sim-Bpred - Preditores de desvios

GO / Sim-Bpred	taken	nottaken	comb	bimod	2lev
Instruções executadas	548177449	548177449	548177449	548177449	548177449
Total de misses	28262154	33846617	41329242	16040589	20006277
Taxa de misses	0,0515566	0,06174391	0,0753939	0,0292617	0,036496
Branch address prediction rate	0,648	0,5784	0,8084	0,7943	0,7449
Branch direction prediction rate	0,648	0,5784	0,8143	0,8002	0,7508
GO / Sim-Outorder	**taken**	**nottaken**	**comb**	**bimod**	**2lev**
Instruções executadas	548177449	548177449	548177449	548177449	548177449
Total de misses	52008996	52008996	15619175	16040035	22565478
Taxa de misses	0,0948762	0,0948762	0,0284929	0,0292607	0,0411645
Branch address prediction rate	0,3522	0,3522	0,7974	0,7924	0,7077
Branch direction prediction rate	0,3522	0,3522	0,8055	0,8002	0,7189

Compress / Sim-Bpred	taken	nottaken	comb	bimod	2lev
Instruções executadas	413349611	413349611	413349611	413349611	413349611
Total de misses	22358364	27460653	5107081	7669458	5347664
Taxa de misses	0,0540907	0,06643445	0,0123554	0,0185544	0,0129374
Branch address prediction rate	0,7142	0,649	0,9347	0,902	0,9316
Branch direction prediction rate	0,7142	0,649	0,9347	0,902	0,9316
Compress / Sim-Outorder	**taken**	**nottaken**	**comb**	**bimod**	**2lev**
Instruções executadas	413349611	413349611	413349611	413349611	413349611
Total de misses	55873674	55873674	5526096	7669609	12076097
Taxa de misses	0,1351729	0,13517292	0,0133691	0,0185548	0,0292152
Branch address prediction rate	0,2858	0,2858	0,9293	0,902	0,8445
Branch direction prediction rate	0,2858	0,2858	0,9294	0,902	0,8456

Como vemos na Tabela 5.18 enquanto para as predições estáticas, tanto no Go quanto no Compress ocorre grande diferença entre as taxas de endereço previsto e de direção prevista do Sim-Bpred e Sim-Outorder.

Para os preditores dinâmicos os resultados são praticamente iguais. A diferença pode ser justificada pelo fato do Sim-Outorder ser um simulador completo, desta forma ele faz execução fora de ordem, usa caches e etc. o que pode influenciar indiretamente no desempenho do preditor de desvio. Já o Sim-Bpred faz uma verificação "funcional" dos preditores de desvio sem ser afetado por outros parâmetros ou pela arquitetura do processador.

5.10 Roteiro X – Sim-Outorder e análise de desempenho de uma arquitetura sugerida

Objetivos

A proposta desta aula é a simulação de uma arquitetura completa, conforme sugerido na tabela abaixo, para que possamos avaliar conceitos como número de ciclos por instrução, tempo de execução e ainda demonstrar a grande flexibilidade permitida por este simulador quando tratamos de projeto de novas arquiteturas. Também podemos analisar os resultados dos diversos parâmetros envolvidos e determinar com boa precisão o melhor desempenho. Isto será feito tanto com relação ao tempo, cache perfeita e preditor perfeito conceitos introduzidos nas simulações Roteiro VIII e Roteiro IX.

Procedimentos

Use o Sim-Outorder do conjunto de ferramentas SimpleScalar para simular a seguinte arquitetura. Os benchmarks utilizados serão o Go e o Compress.

Descrição Arquitetura	Default	Cache Perfect	Preditor Perfect	Arquitetura Sugerida
Latência de erro(miss) na predição de desvio	3	3	3	2 Ciclos
Largura da instrução decodificada (No. de Inst)	4	4	4	4 Inst.
Despacho simples em ordem	False	False	False	False
Largura da instrução entregue (No. de Inst.).	4	4	4	1 Instrução
ULA´s para inteiros	4	4	4	1 ULA
ULA´s para ponto flutuante	4	4	4	1 UPF
Multiplicador de inteiros	1	1	1	1 Int mul
Multiplicador de ponto flutuante	1	1	1	1 PF mul
Predição de desvio	Bimod	Bimod	Perfect	Bimod
Cache de instruções L1 Mapeamento direto .	16K, 32 bytes por linha.	16K, 32 bytes por linha..	16K, 32 bytes por linha.	16K, 64 bytes por linha.
Cache de dados L1 Mapeamento Direto.	16K, 32 bytes	16K, 32	16K, 32	16K, 64bytes
Latência para acertos na cache L1 de dados e instruções	1	1	1	1 Ciclo
Latência para misses na cache L1 de dados e instruções	1	1	1	1 Ciclo
Cache L2	Sim	Sim	Sim	Sim
Latência de acesso à memória (primeira palavra, palavras restantes).	18, 2	18, 2	18, 2	1,1 Ciclos
Largura dos dados na memória	8	8	8	8 bytes
Latência de erro (miss)TLB	30	1	30	1 Ciclo
Sim_IPC				
Sim_CPI				
Velocidade da simulação ($X10^6$ inst/s)				

Monte o relatório da simulação acima, comparando os resultados com os obtidos nas simulações anteriores como default, preditor perfeito e cache perfeita. Dobre o número de unidades funcionais da configuração sugerida, compare os resultados, tente avaliar as alterações nos resultados em função dos parâmetros alterados.

Resultados Roteiro X

Tabela 5.19: Variação entre IPC, CPI e velocidade de simulação para configurações Default, Cache Perfeita, Preditor Perfeito e Arquitetura sugerida no Go, utilizando o Sim-Outorder

GO

Descrição Arquitetura	Default	Cache Perfect	Preditor Perfect	Arquitetura Sugerida
Latência de erro(miss) na predição de desvio	3	3	3	2 Ciclos
Largura da instrução decodificada (No. De Instruções)	4	4	4	4 Instruções
Despacho simples em ordem	False	False	False	False
Largura da instrução entregue (No. De Instruções).	4	4	4	1 Instrução
ULA´s para inteiros	4	4	4	1 ULA
ULA´s para ponto flutuante	4	4	4	1 UPF
Multiplicador de inteiros	1	1	1	1 Int mul
Multiplicador de ponto flutuante	1	1	1	1 PF mul
Predição de desvio	Bimod	Bimod	Perfect	Bimod
Cache de instruções L1 Mapeamento direto	16K, 32 bytes por linha.	16K, 32 bytes por linha..	16K, 32 bytes por linha.	16K, 64 bytes por linha.
Cache de dados L1 Mapeamento Direto.	16K, 32 bytes por linha.	16K, 32 bytes por linha..	16K, 32 bytes por linha.	16K, 64 bytes por linha.
Latência para acertos na cache L1 de dados e instruções	1	1	1	1 Ciclo
Latência para misses na cache L1 de dados e instruções	1	1	1	1 Ciclo
Cache L2	Sim	Sim	Sim	Sim
Latência de acesso à memória (primeira palavra, palavras restantes).	18, 2	18, 2	18, 2	1, 1 Ciclos
Largura dos dados na memória	8	8	8	8 bytes
Latência de erro (miss)TLB	30	1	30	1 Ciclo
Sim_IPC	0,8789	1,4335	1,0408	0,6974
Sim_CPI	1,1378	0,6976	0,9608	1,4340
Velocidade da simulação ($X10^6$ inst/s)	0,265847	0,273268	0,348713	0,217963

Tabela 5.20: Variação entre IPC, CPI e Velocidade de simulação para configurações Default, Cache Perfeita, Preditor Perfeito e Arquitetura sugerida no Compress, utilizando o Sim-Outorder

Compress

Descrição Arquitetura	Default	Cache Perfect	Preditor Perfect	Arquitetura Sugerida
Sim_IPC	1,1261	1,1772	1,6302	0,7779
Sim_CPI	0,8880	0,8495	0,6134	1,2855
Velocidade da simulação ($X10^6$ inst/s)	0,287958	0,305555	0,239860	0,288259

Observamos que a arquitetura testada produziu grande impacto e redução no IPC tanto no Go, quanto no Compress. Pode-se analisar este fato como conseqüência do menor número de unidades de execução de inteiros e ponto flutuante. A velocidade da simulação reduziu a medida que a arquitetura sugerida tornou-se mais simples passando das 265847 para 217963 instruções por segundo no Go. No caso do Compress os valores da velocidade de simulação se mantiveram praticamente constantes para default e a arquitetura proposta em 287958 e 288259 instruções por segundo respectivamente. Houve diferença desprezível.

5.11 Roteiro XI – Simulador WinDLX e análise da arquitetura DLX

Objetivos

A proposta desta aula é a apresentação do simulador WinDLX e simulação de uma arquitetura DLX. Para isto será utilizado o programa gerador de tabela de números primos, em assembler, descrito abaixo, que deverá ser carregado no simulador WinDLX conforme descrito nos procedimentos.

Procedimentos

Preste a atenção nas manipulações que serão utilizadas neste simulador para que seja realizada uma comparação entre simuladores no relatório final deste trabalho.

O programa com que se vai trabalhar como exemplo gera uma tabela de números primos. Antes de iniciar com a simulação convém inicializar uma série de parâmetros que são os seguintes:

Unidades de soma de ponto flutuante (Addition Units):

Número de unidades (Count): 1

Ciclos de execução (Delay): 2

Unidades de multiplicação de ponto flutuante (Multiplication Units) :

Número de unidades (Count): 1

Ciclos de execução (Delay): 5

Unidades de divisão de ponto flutuante (Division Units):

Número de unidades (Count): 1

Ciclos de execução (Delay): 19

Devem estar ativadas as opções de:

Direção simbólica (Simbolic Addresses)
Conta absoluta de ciclos (Absolute Cycle Count)
Ativado o forwarding (Enable Forwarding)

Para poder começar a simulação em primeiro lugar deve-se carregar o código em assembler na memória principal (do simulador). A carga do código se faz desde: *File* → *Load Code or Data* e se seleciona o programa.

Agora o simulador está preparado para começar. Para iniciar a execução pressione "F7" ou *Execution* → *Single Cycle*, que executará um passo da simulação. Veja que a primeira linha da janela de código com a direção *$TEXT* se torna amarelo. Conforme vai sendo apertado "F7", você verá que as linhas sucessivas vão se colorindo. As cores mostram em que estado o *pipeline* se encontra da simulação.

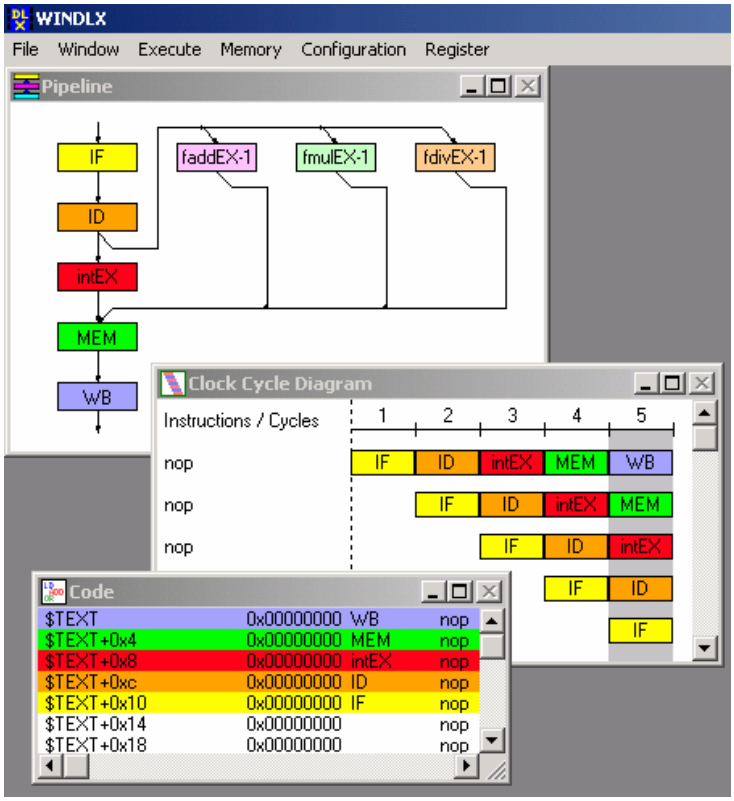


Figura 5.17: Janelas no WinDLX

Vá a linha a seguir que contém a instrução addi r1,r1,0X4 da janela de *Code Window* e selecione *Code→Set Breakpoint*. Isto irá indicar o lugar a partir do qual se deseja continuar a execução. Assim não terá que ir executando ciclo a ciclo. Assegure-se que na janela de *Breakpoint window* aparecerá a linha que foi indicada.

Pressionando agora a tecla "F5" ou *Execute → Run* continuará a simulação a partir do *Breakpoint* anteriormente posto. Se observar a janela de *Clock Cycle Diagram* (Diagrama de ciclos de clock) verá que se encontra no ciclo 11. E ainda aparecem linhas vermelhas e outras verdes. As vermelhas indicam a necessidade de uma parada e a causa está explicada na linha que indica. Se há um *R-Stall* significa que a parada foi produzida por RAW, tentativa de escrita antes de realizar uma leitura do registrador ou endereço de memória. E as linhas verdes simbolizam o uso de *forwarding* .

A partir de tudo isto, no ciclo 7 aparece a palavra *aborted*, que indica que foi abortado o processamento da instrução que estávamos processando. A explicação para isto é que no comando anterior há uma operação de salto condicional bnez. Mas o simulador não sabe se o salto é efetivo até que se tenha acabado a etapa ID. Para não desperdiçar tempo de processamento se adota uma técnica de predição de desvio.

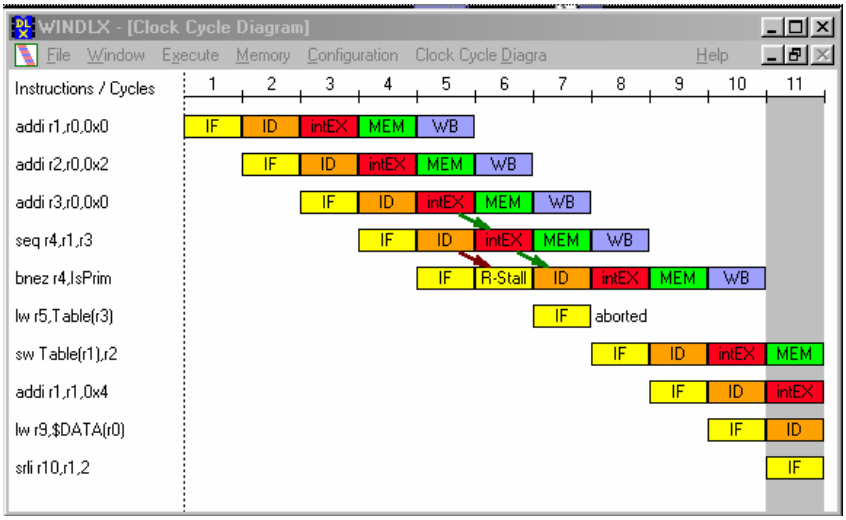


Figura 5.18: Janela de Diagrama de ciclo de clock para a simulação em andamento

Para avançar vários ciclos de uma vez, a partir dos *breakpoint*, aperte "F8" ou *Execute* → *Multiple Cycle*. Faça isto com o exemplo que estamos tratando e introduza o número de ciclos de 20 e verá a simulação até o ciclo 30.

Delete o breakpoint, faça a simulação rodar até 100 ciclos, observe o conteúdo da janela *Clock Cycle Diagram* e verifique que aparecem duas flechas entre os ciclos sessenta e três e sessenta e quatro. A explicação é um duplo forwarding uma vez que a instrução solicita dados das duas instruções anteriores. Neste caso temos uma instrução realimentando o registrador a partir da saída da unidade de execução e outra a partir da saída da unidade de memória.

Retire o forwarding, nas configurações, refaça a simulação e verifique o que ocorre com estas instruções e o número de ciclos.

Tecle f5 (run) com forwarding e sem forwarding e verifique o número total de ciclos de execução na janela de estatísticas.

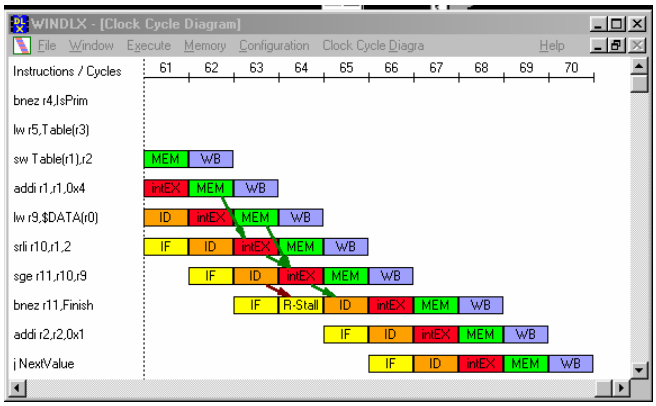


Figura 5.19: Janela de Diagrama de ciclo de clock para a simulação em andamento com duplo forwarding.

Monte o relatório da simulação descrevendo procedimentos e comparando os resultados e simulador com o outro tipo trabalhado anteriormente.

Resultados Roteiro XI

A utilização e análise do simulador WinDLX demonstrou que esta é uma ferramenta didática muito importante, e que tem como principal argumento favorável a utilização de telas gráficas. Permite manipular variáveis como unidades funcionais, cache, pipeline e etc.. Além de possibilitar a execução por passos, permite a fixação de breakpoint para verificação de execução. Possibilita também a visualização de passos como busca de instrução, decodificação, execução, busca dados na memória e escrita de resultados.

Obviamente que em termos de visualização este simulador apresenta melhores resultados que o SimpleScalar, mas quando tratamos da análise das funcionalidades de processadores percebemos que o conjunto SimpleScalar é muito mais detalhado e poderoso no sentido de permitir a manipulação de parâmetros que formam a base das funcionalidades, além de possibilitar a avaliação real de desempenho da arquitetura configurada. No caso de WinDLX percebe-se nitidamente uma intenção didática na sua criação e disponibilização. Já no SimpleScalar o objetivo é claramente a pesquisa e desenvolvimento de novas alternativas funcionais.

Mas enfim, as duas ferramentas trazem consigo o grande mérito do estimulo e contribuição ao estudo de arquitetura de computadores.

6 Conclusões

Após realizadas as simulações e produzidos os relatórios, percebemos que os conteúdos desta cadeira tornaram-se muito mais interessantes sob o ponto de vista dos alunos, que se mostraram motivados e demonstraram maior interesse nos conceitos ligados a processadores e simulações. Foi sugerido um aprofundamento maior via pesquisa em bibliografia básica e complementar da cadeira, dando ênfase à busca pela interpretação geral, e como um meio de melhoria no desempenho usando os parâmetros analisados como referência.

No decorrer das aulas houve um aprimoramento e adequação às condições existentes entre os roteiros propostos inicialmente e os efetivamente aplicados. Seja por questões como o tempo de realização até desequilíbrio natural dos conhecimentos entre o corpo discente e as plataformas empregadas, o que veio a acrescentar, pois criou-se um subproduto educacional, ampliando a abordagem e tornando ainda mais instigante a investigação e aprendizado. Observou-se ainda a necessidade de adaptação do conjunto das simulações aos conteúdos programáticos dos cursos em que sejam propostas e aplicadas as simulações, dando ênfase ao maior grau de complexidade para cursos de caracter mais voltado para projeto e análise de hardware.

O conjunto de simulações proposto neste trabalho teve aplicação em turmas de graduação de Análise de Sistemas e Sistemas de Informação nas disciplinas de Organização de Processadores e Arquitetura de Computadores. Foi percebida durante a aplicação deste conjunto um maior empenho e interesse por parte dos alunos tanto na identificação e análise de resultados e parâmetros dos simuladores, quanto na busca das origens do conjunto de simuladores. Muitos questionamentos ocorreram com relação as instituições onde são implementadas simulações com o SimpleScalar. Pode-se destacar enorme diferença entre o comportamento passivo verificado nas turmas em que os mesmos conteúdos foram abordados sem a utilização dos recursos de simulação e o comportamento ativo e positivo de participação e interesse constatado nas turmas em que foram implementadas as simulações. Nota-se que o perfil dos alunos de graduação nas áreas de Tecnologia da Informação deve ser fortemente dosado com aplicações práticas mantendo acesa a chama do interesse que irá auxiliar no percurso deste árido caminho, que poderá se tornar desestimulante dependendo da forma em que o professor conduzir os conteúdos.

Existe uma boa quantidade de trabalho que pode ser explorada, ainda que seja inviável neste trabalho em função de tempo, mas que poderá ser realizada, no sentido de ampliar a apresentação de conceitos e simulações, seja pela troca dos parâmetros analisados, pela abordagem ou ainda simplesmente pela utilização de simuladores com interfaces mais amistosas. Nos parece, inclusive, mais didático que alunos que se

propõem a obter nível aprofundado, na graduação, tenham convivência por algum tempo com interfaces mais rudes e realísticas, que não as interfaces gráficas. Não obstante foi introduzida uma simulação utilizando o WinDLX, com uma interface muito amistosa, embora tenhamos percebido a pouca maleabilidade do simulador em se tratando de parâmetros e tópicos especiais desenvolvidos com o SimpleScalar como predição de desvio, cache, execução fora de ordem e ainda a própria característica mais atraente dos sistemas de código aberto que é a possibilidade de adequação a futuras propostas com a implementação e compilação do novo código. Mas de qualquer forma foi muito válida a experiência, uma vez que apresentará ao grupo a existência de mais um simulador e repassará alguns conceitos que estarão sendo avaliados na aula teórica.

No caso do conjunto de simuladores SimpleScalar foi dada uma atenção superior à continuidade dos roteiros, desde o Sim-Profile que contribuiu com a identificação das classes de instruções envolvidas em cada benchmark. Neste caso pode-se verificar as instruções de carga e armazenamento de memória e as de desvio condicional, análises fundamentais para contextualização de cada roteiro, tanto nos simuladores como Sim-Cache e Sim-Cheetah que permitem a análise funcional da cache e por isto terão resultados que dependem da distribuição das classes de instruções em cada benchmark, como o simulador Sim-Bpred que apresentará resultados com relação à predição de desvio, que dependerão da quantidade de desvios condicionais do benchmark avaliado conforme visto no roteiro VII. No Sim-Outorder foram usados os conceitos e funcionalidades vistos nos roteiros anteriores possibilitando que colocássemos em funcionamento uma arquitetura modularizada e completa em termos de funções e parâmetros e que além da análise de tempo, permitiu uma visão global de projeto de processadores. No Sim-Cache por intermédio da análise de parâmetros e resultados no simulador vimos que para atingir maior desempenho precisamos ampliar o tamanho da cache, conclusão esta que já possuíamos intuitivamente. A novidade surgiu no fato de que o maior desempenho não estava ligado somente ao tamanho da cache , mas sim aos parâmetros que a compõem. Verificou-se que dobrando um parâmetro obtemos, em determinadas situações, desempenho superior a triplicar outro parâmetro. E que muitas vezes a melhoria de desempenho é efetiva para alguns algoritmos de aplicação, mas não para outros.

Embora na página destinada a apresentar o conjunto de simuladores Simplescalar haja disponibilização de todas as ferramentas necessárias para compilação dos fontes dos simuladores e benchmarks a tarefa de implementação não é trivial como relatado pelos próprios autores, é necessária habilidade no manuseio de Sistemas Operacionais de código aberto e mesmo na estruturação de uso e busca de resultados. Esta dificuldade é em função da grande quantidade de parâmetros e taxas produzidas como saídas das simulações. É necessária paciência para analisar os resultados produzidos com o objetivo de torná-los significativos.

O enfoque principal dado para análise de desempenho, neste trabalho, foi o IPC e CPI. Mas devemos ter em mente que a Vazão *(Throughput)* - taxa na qual os pedidos são atendidos; Utilização - fração do tempo em que o recurso permanece ocupado atendendo os pedidos; e o Tempo de resposta - tempo decorrido entre o pedido e a conclusão da realização do serviço são demonstrativos de desempenho importantes e que poderão fazer parte de futuros trabalhos utilizando o conjunto SimpleScalar. Neste caso o simulador Sim-Outorder além de permitir a manipulação de funcionalidades como cache e predição de desvio, ainda nos permite alterar o projeto do processador simulado com a alteração em parâmetros essenciais como latências, unidades funcionais, ordem de despacho de

instruções, multiplicadores e etc..Ele possibilita também a fixação de conceitos como cache perfeita e preditor perfeito para termos padrões na avaliação da vazão e tempo de resposta.

Neste trabalho os procedimentos foram conduzidos com objetivo didático conforme comentado inicialmente. Algumas propostas para futuros trabalhos seriam análise mais aprofundada, no sentido de maior quantidade de simulações e o aprimoramento nos tipos de variações dos parâmetros que exercem maior influência no desempenho da cache, a razão em termos de construção para este funcionamento. Outra possibilidade é a proposta de uma técnica de predição de desvio que traga um melhor desempenho e menor taxa de misses ao sistema empregado, sendo esta dinâmica ou híbrida.

REFERÊNCIAS

[AUS 97] AUSTIN, T. M. **A User and Hacker's Guide to the SimpleScalar Architectural Research Tool Set** (for tool set release 2.0). Slides presented at 30th Annual International Symposium on Microarchitecture, Jan. 1997. Disponível via WWW em http://www.simplescalar.com (Acessada em 03 de Março de 2004)

[AGE 87] AGERWALA, T.; COCKE, J. **High Performance Reduced Instruction Set Processors.** (Technical Report RC12434). [S.l. : s. n.], 1987.

[AND 67] ANDERSON, D.W.; SPARACIO, F. J.; TOMASULO, R. M. The IBM 360 Model 91: Machine Philosophy and Instruction Handling. **IBM Journal of Research and Development**, Armonte, v.11, n. 1, p. 8-24, Jan. 1967.

[BUR 96] BURGER, D. C.; AUSTIN, T. M.; BENNETT, S. **Evaluating Future Microprocessors**: the SimpleScalar Tool Set, Madison: University of Wisconsin, 1996. (CS TR #1308).

[BUR 97] BURGER, D.C.; AUSTIN, T. M. The SimpleScalar Tool Set, Version 2.0. **Computer Architecture News,** [S.l.], v. 25,n. 3, p. 13- 25, June 1997. Disponível via WWW em www.simplescalar.com (Acessada em 03 de Março de 2004)

[BHA 97] BHANDARKAR, D. RISC versus CISC: A Tale of Two Chips. **Computer Architecture News**, New York, v. 25, n. 1, p. 1-12, Mar. 1997.

[CME 94] CMELIK, B.; KEPPEL, D. Shade: a Fast Instruction-Set Simulator for Execution Profiling. In: ACM SIGMETRIC CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 1994.

Proceedings... [S.l.: s.n.], 1994.

[DIG 92] DIGITAL EQUIPMENT CORPORATION. **Alpha Architecture Handbook.** USA, 1992.

[GRO 90] GROHOSKI, G. F. Machine Organization of the IBM RISC System/6000 Processor. **IBM Journal of Research and Development,** Armonte, v. 34, n. 1, p. 37-58, Jan. 1990.

[HEN 86] HENNESSY, J. L.; RISC-Based Processors: Concepts and Prospects. In: NEW FRONTIERS IN COMPUTER ARCHITECTURE CONFERENCE, 1986. Proceedings... [S.l.: s.n.].1986.

[JOH 91] JOHNSON, M. **Superscalar Microprocessor Design**. Englewood Cliffs: Prentice- Hall, 1991.

[KEL 75] KELLER, R. M. Look-Ahead Processors. **ACM Computer Surveys,** New York, v. 7, n . 8, p. 177-195, Dec. 1975.

[KER 86] KERNIGHAN, B. W.; RITCHIE, D. M. **C:** – Linguagem de Programação. Rio de Janeiro: Campus, 1986.

[LEE 2003] LEE, B. **Dynamic Branch Prediction.** 2003. Notas de aula da disciplina High Performance Computer Architecture do Electrical and Computer Engineering da Oregon State University.

[MCF 93] MCFARLING, S. **Combining Branch Predictors**. [S.l]: Digital Western Research Laboratory, 1993. (WRL Technical Note TN-36).

[MOU 99] MOUDGILL, M.; WELLMAN, J. D.; MORENO, H. Environment for PowerPC Microarchietecture Exploration, **IEEE Micro**, Los Alamitas, May-June 1999.

[MUR 2000] MURDOCA, M.J.; HEURING, V. P. **Introdução à Arquitetura de Computadores.** Rio de Janeiro: Campus, 2000.

[MOU 2002]MOURE, Juan ; BENITEZ, Domingo. Enseñando Arquitectura de Computadores com KScalar. In: JORNADAS DE PARALELISMO, 13., 2002. [Proceedings ...]
[S.l.: s.n.], 2002.

[PAT 96] PATTERSON, D. A.; HENNESSY, J. L. **Computer Architecture:** a quantitative Approach. 2 nd ed. San Francisco: Morgan Kaufmann, 1996.

[PAT 2003] PATTERSON, D. A.; HENNESSY, J. L. **Computer Architecture:** a quantitative Approach. 3 rd ed. Amsterdam: Morgan Kaufmann, 2003.

[PAT 98] PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization & Design**: the Hardware/Software Interface. San Francisco: Morgan Kaufmann, 1998.

[PAT 2000] PATTERSON, D. A.; HENNESSY, J. L. **Organização e Projeto de Computadores**, a interface Hardware/ Software. Rio de janeiro: LTC, 2000.

[PAI 97] PAI, V. S.; RANGANATHAN, P.; ADVE, S. V. RSIM: Na Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors, **IEEE TCCA Newsletter**, Los Alamitas, Oct. 1997.

[PAL 97] PALACHARLA, S.; JOUPPI, N.; SMITH, E. **Complexity-Effective Superscalar Processors**. [S. l.: s. n.], 1997

[PAT 85] PATTERSON, D. A. Reduced Instruction Set Computers. **Communications of the ACM**, New York, v. 28, n. 1, p. 8-21, Jan. 1985.

[PRI 95] PRICE, C. **MIPS IV Instruction Set, revision 3.1**. Mountain View, CA, USA: MIPS Technologies, 1995.

[RAU 89] RAU, B. R. et al. Towle, The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions, and Trade-offs. **IEEE Computer**, Los Alamitas, p. 12-35, Jan. 1989.

[STA 2002] STALLINGS, W. **Arquitetura e Organização de Computadores**. São Paulo: Prentice Hall, 2002.

[SOU 99] SOUZA, A. F. de **Integer Performance Evaluation of the Dynamically Trace Scheduled VLIW Architecture.** 1999. Phd Thesis, Departament of Computer Science, University68 College, University of London, Lodon, UK.

[SUG 93] SUGUMAR, R. A.; ABRAHAM, S. G. Efficient Simulation of Caches under Optimal Replacement with Applications to Miss Characterization. In: - ACM SIGMETRICS CONFERENCE ON MEASUREMENTS AND MODELING OF COMPUTER SYSTEMS, 1993. Proceedings... [S. l: s. n.], 1993.

[SAI 93] SAINI, A. An Overview of the Intel Pentium Processor. In: COMPCON, 1993. Proceedings... [S.l.: s.n.], 1993.

[SAL 76] SALISBURY, A. B. **Microprogrammable Computer Architectures**: New York: Elsevier, 1976.

[SOH 90] SOHI, G. S. Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers. **IEEE Transactions on Computers**, Los Alamitas, vi 39, n.3, p. 349-359, Mar. 1990

[SUN 87] SUN MICROSYSTEMS. **The Sparc Architecture Manual – Version 7**. [S. l], 1987.

[TOM 67] TOMASULO, R. M. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. **IBM Journal of Research and Development**, Armonte, v. 11, n. 1, p. 25-33, Jan. 1967.

[UYE 2002] UYEMURA, J. **Sistemas Digitais**: uma abordagem integrada. [S.l.: s .n .], 2002.

[WEB 2001] WEBER, R. F. **Arquitetura de Computadores Pessoais**. Porto Alegre: Sagra-Luzzatto, 2001.

[YEH 91] YEH T.- Y.; PATT, Y. N. Two-Level Adaptive Branch Prediction. SIGMICRO Newsletter, New York, p. 51-61 , Nov. 1991. Trabalho apresentado no 24 Annual International Symposium on Microarchtecture, 1991