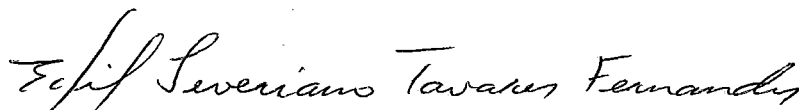


Efeito do Escalonamento Dinâmico no Desempenho de Processadores Super Escalares

Fernando Mauro Buleo Barbosa

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por :



Prof. Edil Severiano Tavares Fernandes, Ph.D.
(Presidente)



Prof. Cláudio Luís de Amorim, Ph.D.



Prof. Manuel Lois Anido, Ph.D.

Rio de Janeiro, RJ – Brasil

Fevereiro de 1993

BARBOSA, FERNANDO MAURO BULEO

Efeito do Escalonamento Dinâmico no Desempenho de Processadores Super Escalares [Rio de Janeiro] 1993.

XI, 76 p., 29,7cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1993)

Tese – Universidade Federal do Rio de Janeiro, COPPE.

1 - Arquiteturas Super Escalares

2 - Paralelismo de Baixo Nível

3 - Escalonamento Dinâmico

4 - Execução Especulativa

I. COPPE/UFRJ

II. Título (série).

A minha mãe
Edna
e a minha avó
Mercedes.

Agradecimentos

Ao professor Edil Severiano Tavares Fernandes, pela orientação competente e dedicada, pelo incentivo ao meu desenvolvimento acadêmico e pela sua amizade e confiança.

À professora Anna Dolejsi, pelo auxílio na aquisição de artigos e pela incansável disposição nos debates sobre algoritmos de despacho.

Ao Alberto Ferreira de Souza, pelo simulador do i860 e pelo suporte no uso do mesmo.

À Nahri, pela amizade e competência que nos tornaram companheiros inseparáveis de trabalho desde a graduação.

Ao Luís Carlos Quintela, pelo companheirismo.

A todos aqueles que de alguma forma contribuíram para a elaboração desta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Efeito do Escalonamento Dinâmico no Desempenho de Processadores Super Escalares

Fernando Mauro Buleo Barbosa

Fevereiro, 1993

Orientador : Edil Severiano Tavares Fernandes

Programa : Engenharia de Sistemas e Computação

Este trabalho descreve um modelo parametrizado de Arquitetura Super Escalar. Utilizando esse modelo básico, diversas configurações de máquina foram especificadas e simuladas. Através dos resultados produzidos por essas simulações, avaliou-se o efeito de importantes detalhes arquiteturais no volume do paralelismo de baixo nível que pode ser detectado e explorado por alguns algoritmos de escalonamento dinâmico de instruções.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Effect of Dynamic Scheduling on the Performance of Superscalar Processors

Fernando Mauro Buleo Barbosa

February, 1993

Thesis Supervisor : Edil Severiano Tavares Fernandes

Department : Computing and Systems Engineering

This work describes a parametrized model of Superscalar Architecture. Using this basic model, several machine configurations were specified and simulated. The results obtained from these simulations were used to evaluate the effect of important architectural details on the volume of low level paralelism that can be detected and explored by some dynamic instruction scheduling algorithms.

Índice

1	Introdução	1
1.1	Máquinas Super Escalares	2
1.2	Motivação	5
1.3	Metodologia Adotada	6
1.4	Organização do Texto	8
2	O Algoritmo de Tomasulo	9
2.1	O Escalonamento Associativo	10
2.1.1	As Reservation Stations	10
2.1.2	O Esquema de Rotulação	11
2.1.3	O Common Data Bus (CDB)	12
2.2	Tratamento de Dependências Verdadeiras	13
2.3	Tratamento de Dependências Falsas	16
3	Modelos de Simulação	20
3.1	A Máquina Básica	20
3.2	Modelos Prévios	22
3.3	Modelo Final	25

4	Método de Avaliação	29
4.1	A Máquina Referência	29
4.2	Programas Teste	29
4.3	A Simulação	31
5	Resultados	34
5.1	Impacto do Tamanho da Janela	35
5.2	Impacto da Utilização de Múltiplos <i>CDBs</i>	46
5.3	Influência das <i>Reservation Stations</i>	53
5.4	Impacto dos Desvios Condicionais	61
6	Conclusões	68

Lista de Figuras

2.1	Esquema Simplificado da Unidade de Ponto Flutuante do IBM-360/91	13
2.2	Efeito do Tratamento de Dependências Verdadeiras de Dados	15
2.3	Dependências de Saída (Adição terminando antes da Multiplicação) .	18
2.4	Dependências de Saída (Multiplicação terminando antes da Adição) .	18
4.1	Etapas do Processo de Simulação	32
5.1	Efeito do Despacho Múltiplo de Instruções na Taxa de Aceleração . .	38
5.2	Variação do Tempo Ocioso das Unidades <i>Core</i>	39
5.3	Ocupação de Grupos de 0 a 2 Unidades <i>Cores</i> no programa <i>BCDBin</i>	41
5.4	Ocupação de Grupos de 3 a 5 Unidades <i>Cores</i> no programa <i>BCDBin</i>	42
5.5	Ocupação de Grupos de 0 a 2 Unidades <i>Cores</i> no programa <i>Bubble</i> . .	43
5.6	Ocupação de Grupos de 3 a 5 Unidades <i>Cores</i> no programa <i>Bubble</i> . .	43
5.7	Efeitos dos <i>CDBs</i> e das Janelas de Instruções no Desempenho	52
5.8	Efeito dos <i>CDBs</i> nas Taxas de Aceleração	53
5.9	Tempo de Espera X Tempo de Ocupação	60
5.10	Efeito da Predição de Desvios nas Taxas de Aceleração	64

Lista de Tabelas

4.1	Total de Instruções por Unidade Funcional e Ciclos	31
5.1	Variação das Taxas de Aceleração com o Tamanho da Janela	37
5.2	Ocupação (%) das Cores para Janelas de Tamanho 1	40
5.3	Ocupação (%) das Cores para Janelas de Tamanho 2	40
5.4	Ocupação (%) das Cores para Janelas de Tamanho 4	40
5.5	Ocupação (%) das Cores para Janelas de Tamanho 8	41
5.6	Ocupação (%) das Cores para Janelas de Tamanho 16	41
5.7	Ocupação dos CDBs para Janelas de Tamanho 1	47
5.8	Ocupação dos CDBs para Janelas de Tamanho 2	47
5.9	Ocupação dos CDBs para Janelas de Tamanho 4	48
5.10	Ocupação dos CDBs para Janelas de Tamanho 8	48
5.11	Ocupação dos CDBs para Janelas de Tamanho 16	49
5.12	Ciclos (%) em que Houve Conflitos para Modelos com 1 CDB	49
5.13	Ciclos (%) em que Houve Conflitos para Modelos com 2 CDB	50
5.14	Ciclos (%) em que Houve Conflitos para Modelos com 3 CDB	50
5.15	Taxas de Aceleração Obtidas com 1 CDB	51
5.16	Taxas de Aceleração Obtidas com 2 CDBs	51

5.17	Taxas de Aceleração Obtidas com 3 <i>CDBs</i>	51
5.18	Ocupação (%) das <i>RSs</i> da <i>Core</i> para Janelas de Tamanho 1	54
5.19	Ocupação (%) das <i>RSs</i> da <i>Core</i> para Janelas de Tamanho 2	54
5.20	Ocupação (%) das <i>RSs</i> da <i>Core</i> para Janelas de Tamanho 4	55
5.21	Ocupação (%) das <i>RSs</i> da <i>Core</i> para Janelas de Tamanho 8	55
5.22	Ocupação (%) das <i>RSs</i> da <i>Core</i> para Janelas de Tamanho 16	56
5.23	Ocupação (%) da Unidade de Acesso à Memória	57
5.24	Ocupação (%) das <i>RSs</i> da <i>Memória</i> para Janelas de Tamanho 1	57
5.25	Ocupação (%) das <i>RSs</i> da <i>Memória</i> para Janelas de Tamanho 2	58
5.26	Ocupação (%) das <i>RSs</i> da <i>Memória</i> para Janelas de Tamanho 4	58
5.27	Ocupação (%) das <i>RSs</i> da <i>Memória</i> para Janelas de Tamanho 8	59
5.28	Ocupação (%) das <i>RSs</i> da <i>Memória</i> para Janelas de Tamanho 16	59
5.29	Influência dos Desvios Condicionais	62
5.30	Taxas de Aceleração Obtidas com Predição de Desvios	63
5.31	Tempo Ocioso (%) das Unidades <i>Core</i>	65
5.32	Ocupação (%) de Grupos de <i>Core</i> com Predição de Desvios	66

Capítulo 1

Introdução

Nos processadores convencionais, as instruções de máquina são executadas seqüencialmente, uma de cada vez, e a execução de uma instrução somente é iniciada quando a anterior estiver concluída. Contudo, o desempenho do processador pode ser aumentado consideravelmente se o algoritmo responsável pelo despacho de instruções for capaz de escalonar diversas instruções para serem executadas simultaneamente.

Arquiteturas Super Escalares, uma recente tendência na implementação de processadores, são exemplos típicos de máquinas que executam diversas instruções simultaneamente. A execução em paralelo de múltiplas instruções (i.e., a exploração do paralelismo de baixo nível) nessas arquiteturas é viabilizada pela presença de importantes detalhes de implementação. Unidades funcionais independentes, a presença de uma janela (no interior do processador) que armazena instruções provenientes de um mesmo programa de aplicação, e um mecanismo de previsão de desvios, são exemplos de detalhes arquiteturais usualmente incorporados pelos processadores Super Escalares.

Num processador Super Escalar, o algoritmo responsável pela escolha das instruções que serão iniciadas durante cada ciclo de máquina, pode ser implementado diretamente na sua unidade de controle. Nesse caso, o escalonamento é levado a cabo em tempo de execução, e por esse motivo ele é denominado algoritmo de *escalonamento dinâmico*.

Apesar do seu aspecto inovador, esse tipo de algoritmo de detecção

e extração do paralelismo de baixo nível, já havia sido implementado em duas máquinas comercializadas na década de 60: no modelo CDC-6600 da Control Data Corporation [THOR64] e no modelo 360/91 da IBM [TOMA67,FLYN67,DAND67,SAND67,BOLA67]. Esses dois processadores incorporam algoritmos de escalonamento que permitem a execução concorrente de várias instruções provenientes de um mesmo programa de aplicação. Adicionalmente, apesar das instruções serem executadas fora da ordem especificada originalmente pelo programador, o algoritmo de escalonamento garante a equivalência semântica do programa de aplicação.

Contudo, a tecnologia existente naquela época, tornava o custo de implementação de técnicas complexas como estas, muito elevado diante do desempenho obtido. Por esse motivo, nos anos subseqüentes, tais idéias deixaram de merecer a atenção dos pesquisadores, ficando relegadas a um segundo plano. Alternativamente, buscou-se aprimorar ou desenvolver outras técnicas que fossem economicamente mais viáveis. Idéias como *pipeline* e máquinas vetoriais foram amplamente discutidas e estudadas.

Com o decorrer do tempo, novas tecnologias foram desenvolvidas, tornando o custo do *hardware* cada vez mais baixo. Mais recentemente, com o advento da tecnologia *VLSI* e o conseqüente aumento na densidade dos circuitos, tornou-se possível encapsular em um único *chip*, recursos de *hardware* antes só encontrados em processadores de alto desempenho. Isto foi fundamental para o aparecimento de uma nova classe de máquinas que exploram os conceitos empregados nos processadores CDC-6600 e IBM-360/91 (i.e. as máquinas Super Escalares).

1.1 Máquinas Super Escalares

O termo **Super Escalar** é utilizado para descrever processadores que executam diversas instruções escalares concorrentemente. O termo também serve para diferenciar máquinas vetoriais deste tipo de arquitetura: em contraste com os processadores vetoriais, as máquinas super escalares escalonam (i.e., despacham) múltiplas operações **distintas** que serão executadas simultaneamente. Essa característica decorre da existência de diversas unidades funcionais independentes e do correspon-

dente algoritmo de despacho de instruções.

O escalonamento (durante cada ciclo de máquina) de múltiplas instruções que serão executadas em paralelo, a renomeação de registradores (*register renaming*) [KELL75] e a predição de desvios [JLEE84] são exemplos de detalhes arquiteturais que caracterizam os processadores super escalares.

Tendo em vista que a maioria das aplicações para microprocessadores são orientadas para processamento escalar, então podemos considerar as máquinas super escalares como representantes do próximo passo da evolução da arquitetura dos computadores.

O emprego do modelo de computação vetorial e a utilização de unidades funcionais implementadas segundo a técnica *pipeline*, foram importantes propostas que aumentaram substancialmente o desempenho dos processadores. A principal desvantagem dessas duas propostas consiste na limitada classe de programas de aplicação que pode ser efetivamente processada. Por exemplo, máquinas vetoriais são eficientes no tratamento daqueles problemas de computação científica em que é necessário operar com vetores constituídos por um grande número de componentes.

Por outro lado, para atingir o *throughput* máximo daqueles processadores que utilizam a técnica *pipeline* na implementação de suas unidades funcionais, torna-se necessário ativar instruções independentes continuamente. Desse modo, conforme discutido por Ramammorthy em [RAMA77], as instruções do programa de aplicação devem apresentar um elevado nível de paralelismo para que as unidades funcionais *pipelined* não sejam subutilizadas.

Nos experimentos realizados por Pleszkum (vide [PLES88]), foram verificados ganhos de 2 % a 4 % durante a execução de programas escalares (i.e., não vetoriais) em processadores com unidades funcionais *pipelined*. Diferentemente dessas propostas, arquiteturas super escalares não se destinam a um conjunto específico de problemas, mas sim para aplicações genéricas.

Embora ainda não exista uma classificação que seja amplamente usada (o que é bastante natural em áreas do conhecimento ainda incipientes), po-

demos utilizar o método de implementação do algoritmo de escalonamento de instruções, como critério para classificar as diferentes modalidades de máquinas super escalares. O algoritmo de escalonamento de um processador super escalar pode ser:

- $$\left\{ \begin{array}{l} \bullet \text{ Dinâmico} \\ \bullet \text{ Estático} \end{array} \right.$$

Dizemos que o algoritmo é dinâmico quando ele é implementado diretamente pelo *hardware* subjacente. Neste caso, a detecção e exploração de paralelismo entre instruções é feito durante a execução do programa. O processador i960 da Intel [HINT89], o modelo RS-6000 da IBM [GROH90,WARR90] e o processador MC88110 da Motorola [DIEF92], são exemplos de máquinas com algoritmo dinâmico de escalonamento.

No método estático, o escalonamento das instruções que serão executadas em paralelo durante cada ciclo de máquina, é realizado antes da execução do programa de aplicação. Em outras palavras, o algoritmo de escalonamento é implementado em *software*, por exemplo, pelo módulo de geração de código do compilador. Como exemplos de processadores que utilizam algoritmos estáticos, podemos incluir as máquinas VLIW [FISH84] e o processador i860 da Intel [INTE89].

Ao compararmos a eficácia dos dois métodos de implementação do algoritmo de escalonamento, podemos verificar que para aqueles programas de aplicação em que há pouca informação (em tempo de compilação) acerca do paralelismo das instruções, as máquinas com mecanismo dinâmico são mais eficientes: levando em conta que nessas situações o compilador terá que optar por uma decisão “segura” então o código gerado nem sempre será capaz de explorar ao máximo o paralelismo existente no programa de aplicação. Uma outra vantagem apresentada pelo escalonamento dinâmico refere-se a tendência, verificada nos últimos anos, de redução no custo do *hardware* e o aumento observado no custo do desenvolvimento do *software*.

Examinando a organização dos processadores super escalares da atualidade, verificamos que eles empregam soluções utilizadas em processadores anteriores. Clássicos exemplos dessas soluções incluem o *scoreboard* e a janela de instruções do CDC-6600 [THOR64], e as *reservation stations* do IBM-360/91 [TOMA67]. A arquitetura *decoupled* [SMIT82] e a organização da máquina HPS [PATT85], são

exemplos de outras soluções que foram adotadas durante o projeto de diversos processadores super escalares comercialmente disponíveis.

Trabalhos relacionados com o despacho de múltiplas instruções incluem: propostas de algoritmos de escalonamento [TJAD70], [TJAD73], [KELL75], [AUHT85] e [ACOS86]; experimentos avaliando o desempenho de processadores super escalares [WEIS84], [PLES88], [CHAN91] e [BUTL91]; um levantamento (*survey*) abordando as técnicas de seqüenciamento de instruções é apresentada por Krick e Dollas em [KRIC91].

Apesar do grande volume de estudos realizados, só mais recentemente é que alguns processadores super escalares com algoritmo de escalonamento dinâmico, tornaram-se comercialmente disponíveis. Esse é o caso dos processadores i960 da Intel (1989), RS-6000 da IBM (1990), e MC88110 da Motorola (1992).

1.2 Motivação

O nosso interesse em estudar as principais técnicas de exploração do paralelismo de baixo nível, motivou o desenvolvimento dos experimentos aqui apresentados. A principal contribuição desta tese refere-se a avaliação do efeito de alguns algoritmos dinâmicos de escalonamento de instruções no desempenho de processadores super escalares hipotéticos. Após termos avaliado a eficácia desses algoritmos, decidimos concentrar nossos esforços no estudo do algoritmo de Tomasulo. Uma breve descrição desse algoritmo segue-se.

Quando da especificação do modelo 360/91 da IBM [SAND67], os projetistas decidiram incluir unidades funcionais separadas para o processamento de operações em ponto flutuante. Essas unidades são controladas por um algoritmo de escalonamento —diretamente implementado no *hardware*— descrito por Tomasulo em [TOMA67] e que é conhecido como “Algoritmo de Tomasulo.” Através de um esquema de rotulação de registradores e de um barramento de dados comum (*Common Data Bus - CDB*) a todas as unidades funcionais, o algoritmo de Tomasulo é responsável pela ativação de operações em ponto flutuante que podem

ser executadas simultaneamente. Com o objetivo de reduzir o tempo de processamento, o algoritmo permite que operações iniciem (e terminem) fora de ordem pois ele preserva a equivalência semântica do programa de aplicação.

O conceito de dispositivos virtuais, denominados *reservation stations*, foi proposto e explorado por esse algoritmo de escalonamento: ao invés das instruções serem despachadas diretamente para os respectivos dispositivos funcionais, o algoritmo de Tomasulo transfere-as para esses dispositivos virtuais. É a partir da *reservation station* que uma operação com reais é executada.

As principais razões que motivaram o emprego do algoritmo de Tomasulo no nosso estudo seguem-se:

- sua eficácia na coordenação das atividades das máquinas super escalares;
- a simplicidade do algoritmo;
- a abrangência do algoritmo na exploração do paralelismo entre instruções.

1.3 Metodologia Adotada

No nosso trabalho, a estratégia adotada foi a de avaliar, através de simulações, a eficiência do algoritmo de Tomasulo no desempenho de arquiteturas constituídas de múltiplas unidades funcionais. Ao invés de definirmos uma nova arquitetura, decidimos utilizar uma família de máquinas derivada de um processador super escalar real, isto é, o processador i860 da Intel Corporation [INTE89]. Dessa forma, nosso modelo de máquina básica reconhece o repertório de instruções do i860. Os tipos de dados e de unidades funcionais desse processador também foram mantidos no nosso modelo.

O roteiro adotado durante nossos experimentos, pode ser grupado em três fases:

- (a) Especificação e avaliação de alguns algoritmos de escalonamento dinâmico;
- (b) Emprego de técnicas de balanceamento de arquiteturas super escalares;

(c) Avaliação do efeito de sofisticadas técnicas (despacho múltiplo e execução especulativa) no desempenho do nosso modelo básico de arquitetura super escalar.

Inicialmente especificamos alguns algoritmos de escalonamento dinâmico. Empregando esses algoritmos durante a interpretação de uma bateria de programas de teste (*benchmark*), observamos o efeito de diversos detalhes de implementação no desempenho do modelo básico. Como conseqüência dessas observações, decidimos adotar o algoritmo de Tomasulo, pois ele foi o que apresentou o melhor desempenho.

Com o objetivo de generalizar a atuação do algoritmo de Tomasulo, decidimos acrescentar no nosso modelo um outro tipo de unidade: o controlador de memória. Unidades desse tipo são responsáveis pela execução de instruções de transferência de dados entre os registradores e a memória principal (ou entre a memória *cache*). Empregado originalmente para controlar a unidade de ponto flutuante do IBM-360/91, o algoritmo de Tomasulo foi estendido a todos os tipos de unidade no nosso modelo básico.

Os programas de nossa bateria de testes (*benchmark*), codificados na linguagem C, foram traduzidos para o código de máquina do i860 por um compilador comercial. Em seguida, o código foi introduzido em um interpretador do processador i860, obtendo-se como resultado da interpretação os respectivos arquivos contendo *traces* da execução dos programas de teste. Finalmente, os *traces* foram processados por um simulador parametrizado do modelo de máquina.

Durante essas simulações, diversas medidas de desempenho foram realizadas. Através dessas medidas, foi possível avaliar o comportamento das diversas configurações do modelo de máquina, determinando-se dessa forma, a contribuição oferecida por cada dispositivo funcional no desempenho do processador.

Analisando os resultados obtidos especificamos uma configuração de máquina balanceada, usando como critério nesse caso, a relação entre o custo e o desempenho. Exemplos de medidas realizadas incluem: taxa de aceleração, nível de ocupação de cada recurso, porcentagem do tempo de execução em que um certo

número de recursos do mesmo tipo foi utilizado, ficou aguardando por operandos etc.

Nossos experimentos culminaram com a avaliação daquelas técnicas de exploração de paralelismo de baixo nível que foram desenvolvidas mais recentemente e que têm sido adotadas por arquiteturas super escalares comerciais, i.e., o despacho simultâneo de múltiplas instruções e a execução especulativa de instruções.

1.4 Organização do Texto

Essa tese está organizada em seis capítulos. O Capítulo 2 apresenta uma descrição do algoritmo de Tomasulo. O Capítulo 3 descreve as diversas alternativas de máquinas utilizadas pelos nossos experimentos. Os programas da bateria de testes (*benchmark*) e o método de simulação que utilizamos são tópicos abordados no Capítulo 4. No Capítulo 5 apresentamos os resultados desses experimentos e sua análise. As principais conclusões do trabalho estão relacionadas no Capítulo 6.

Capítulo 2

O Algoritmo de Tomasulo

As unidades de ponto flutuante do processador IBM-360/91 são controladas por um algoritmo de escalonamento dinâmico de instruções. Em homenagem ao seu criador, esse algoritmo é denominado “Algoritmo de Tomasulo” (vide [TOMA67]). Capaz de escalonar instruções que podem ser executadas em paralelo e fora da ordem original, o algoritmo de Tomasulo trata os conflitos no uso de recursos, de uma maneira muito eficiente. Este tipo de tratamento desempenha um papel importante na coordenação das atividades que ocorrem num ambiente com elevado nível de paralelismo, como é o caso das arquiteturas Super Escalares da atualidade.

O algoritmo define o conceito de **escalonamento associativo** de instruções. Graças à existência de um *Common Data Bus (CDB)*, de *reservation stations (RS)* e de um esquema de rotulação, a atividade de escalonamento de instruções permanece em operação mesmo na presença de dependência de dados entre instruções. O escalonamento só é interrompido quando não há mais *reservation stations* livres. Neste caso, o escalonamento será reativado tão logo seja liberada uma *reservation station*. A seguir, apresentamos uma breve descrição deste algoritmo.

2.1 O Escalonamento Associativo

2.1.1 As Reservation Stations

Em vez de despachar as instruções diretamente para as unidades funcionais, o algoritmo de Tomasulo transfere as instruções para unidades virtuais, denominadas *reservation stations*. As *reservation stations* atuam como *buffers*, armazenando os parâmetros das instruções (código de operação e operandos) além de algumas informações de controle. Cada *reservation station* é capaz de armazenar dados referentes a uma única instrução. Uma vez despachada (i.e., transferida) para uma *reservation station*, a instrução fica armazenada nesta unidade virtual, sendo descartada somente após o término da operação correspondente. Associada a cada unidade funcional existe uma ou mais *reservation stations*. A utilização de *reservation stations* permite reduzir o número de réplicas de unidades funcionais, tornando mais baixo o custo de implementação do projeto. Para ilustrar a utilização das *reservation stations* vamos considerar o seguinte fragmento de programa:

$$R_0 \leftarrow R_0 \times R_1$$

$$R_2 \leftarrow R_2 + R_0$$

$$R_3 \leftarrow R_3 + R_4$$

No trecho de programa, cada R_i é um dos registradores de um processador que possui unidades funcionais separadas para adição e multiplicação, uma unidade de cada tipo. Se a máquina não possuísse *reservation stations*, então as instruções seriam transferidas diretamente para as unidades funcionais.

As instruções do fragmento de programa seriam escalonadas da seguinte forma: a instrução de multiplicação iria para a unidade funcional de multiplicação e a primeira instrução de soma para a unidade de adição. Uma vez que só há uma unidade de adição, a segunda instrução de soma não poderia ser despachada pois não há unidade de adição disponível. Por essa razão, o processo de escalonamento seria interrompido. Se fosse possível despachar a terceira instrução, sua execução poderia ser iniciada imediatamente pois ela não apresenta dependências de

dados com nenhuma das instruções anteriores. Quando são freqüentes as situações em que a execução de instruções é atrasada pela ausência de recursos disponíveis, o desempenho pode ser comprometido gravemente.

Uma alternativa para resolver este problema seria aumentar o número de unidades de adição, garantindo assim, que na maioria dos casos existirá pelo menos uma unidade livre para receber a instrução. Contudo, a replicação de recursos sofisticados, como é o caso das unidades funcionais, aumenta o custo do projeto. A situação torna-se mais grave quando os recursos replicados são subutilizados. Isto pode ser observado no trecho de programa apresentado anteriormente. Uma vez incrementado o número de unidades de adição, o processo de escalonamento não seria interrompido por falta de unidades funcionais livres após termos despachado a primeira instrução de soma. Porém, por apresentar uma dependência de dados verdadeira (o registrador R_0 é usado como destino pela primeira instrução e como fonte pela segunda), a primeira soma só poderá ser executada após o término da multiplicação. Assim, teríamos uma unidade de adição ociosa, desempenhando durante algum tempo, a função de um *buffer*.

O emprego de *reservation stations* evita este desperdício, pois elas resultam da separação das funções de *buffer*, das funções de execução. Por essa razão, é importante ressaltar que para aumentar a taxa de instruções despachadas por ciclo de máquina (e conseqüentemente a taxa de aceleração dos programas de aplicação), não precisamos necessariamente aumentar o número de pares (*reservation stations, unidades funcionais*). A escolha (criteriosa) do número de *reservation stations* que ficarão associadas a cada tipo de unidade funcional é um importante parâmetro arquitetural: através dessa escolha é possível aumentar a taxa de instruções despachadas sem que tenhamos que prover um igual número de unidades funcionais.

2.1.2 O Esquema de Rotulação

Associado a cada registrador de ponto flutuante do IBM-360/91 existe um bit de ocupação e um campo de rótulo. O bit de ocupação indica se o valor armazenado no

registrador é válido ou não. O conteúdo do registrador torna-se inválido quando uma instrução escalonada anteriormente irá atualizá-lo. Neste caso, o bit de ocupação conterà o valor '1' e o campo de rótulo a identificação da *reservation station* armazenando tal instrução. Quando o conteúdo de um registrador for válido, o bit de ocupação é igual a '0'.

Toda vez que uma instrução for transferida para uma *reservation station*, os bits de ocupação dos registradores que atuarão como fonte da instrução são examinados. Se eles forem iguais a '0', o conteúdo do registrador é copiado na *reservation station*. Caso contrário, o algoritmo de Tomasulo copia o campo de rótulo na *reservation station*. Já o bit de ocupação do registrador que será utilizado para armazenar o resultado da operação (registrador destino) receberá o valor '1'. O rótulo (i.e., a identificação) da *reservation station* que armazena tal instrução é copiado para o campo de rótulo do registrador destino. A execução de uma instrução (armazenada na *reservation station*) é iniciada quando as seguintes condições forem satisfeitas:

- $$\left\{ \begin{array}{l} \bullet \text{ Existe uma unidade funcional livre;} \\ \bullet \text{ Os operandos fonte necessários já se encontram na} \\ \text{ } \textit{reservation station} \end{array} \right.$$

2.1.3 O Common Data Bus (CDB)

O *CDB* é um barramento que interconecta os componentes responsáveis pelas operações com reais do modelo 360/91: unidades funcionais, banco de registradores e *reservation stations*. Através desse barramento os resultados das operações são propagados para os outros componentes controlados pelo algoritmo.

Quando da conclusão de uma operação, a unidade funcional requisita o uso do *CDB*. Uma vez atendida, ela transmite (através do *CDB*) o resultado gerado em conjunto com o rótulo da *reservation station* que armazenava a instrução recém concluída. Em seguida, o banco de registradores e as *reservation stations* comparam este rótulo com o conteúdo de seus campos de rótulos. Se os rótulos forem iguais, o resultado transmitido através do *CDB* é transferido para o respectivo registrador e para os campos de operando das *reservation stations* que estavam aguardando por

CDB

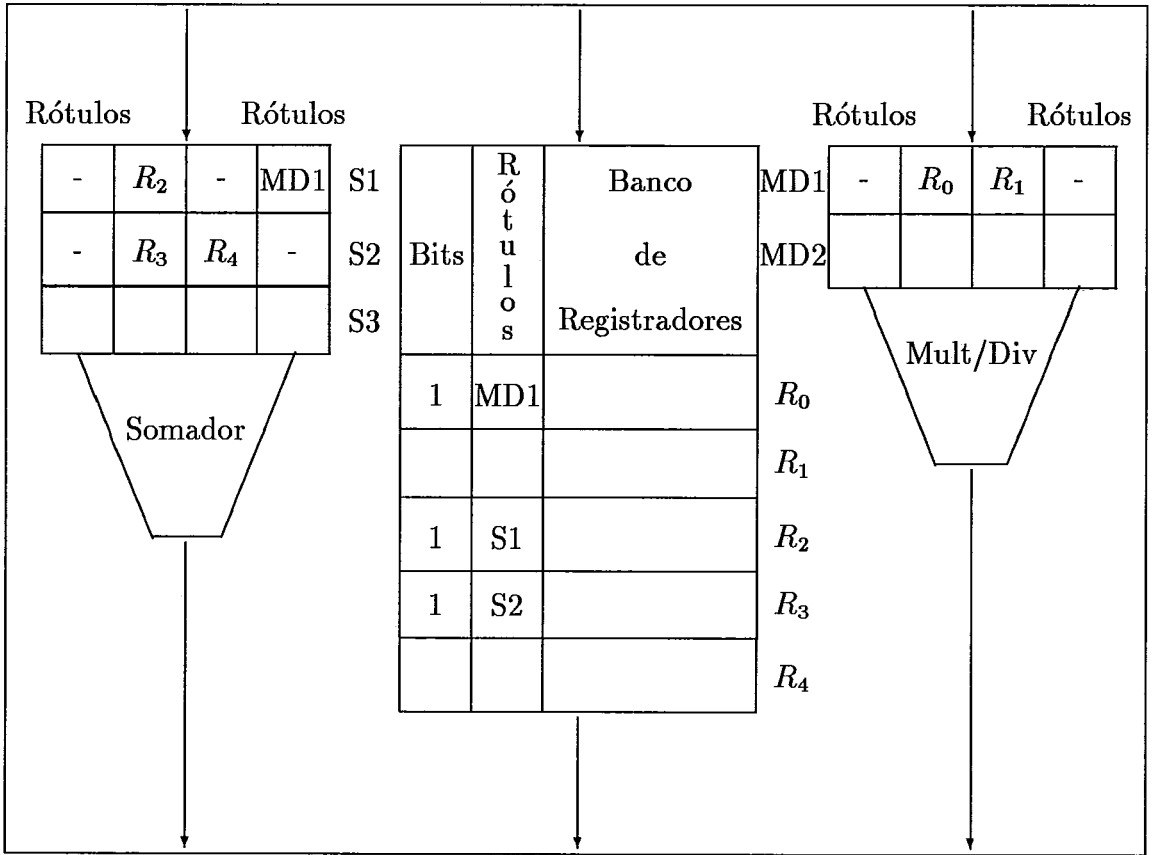


Figura 2.1: Esquema Simplificado da Unidade de Ponto Flutuante do IBM-360/91

esse resultado. O bit de ocupação do registrador cujo conteúdo foi alterado passa para '0', indicando que o valor armazenado nele é válido.

A Figura 2.1 apresenta um esquema simplificado da unidade de ponto flutuante do IBM-360/91. O conteúdo dos campos de registradores e das *reservation stations* representam o estado da máquina após o escalonamento do fragmento de programa apresentado na Seção 2.1.1 assumindo que, inicialmente, o conteúdo de todos os registradores eram válidos e todas as *reservation stations* estavam disponíveis.

2.2 Tratamento de Dependências Verdadeiras

Com o objetivo de ilustrar a atuação do algoritmo quando em presença de instruções apresentando dependência de dados verdadeira, usaremos novamente o trecho de programa apresentado na Seção 2.1.1 e da Figura 2.1.

A primeira *reservation station* associada à unidade de multiplicação/divisão (*reservation station MD1*) foi escalonada para receber a instrução de multiplicação. Conforme ilustrado na figura, a *reservation station MD1* armazena o conteúdo dos registradores R_0 e R_1 pois eles atuam como operando fonte dessa instrução e seus bits de ocupação indicavam que seus conteúdos eram válidos. Por isso, os correspondentes campos de rótulo de *MD1* estão vazios. R_0 foi especificado como registrador destino da primeira instrução. Por esse motivo, o correspondente campo de rótulo no banco de registradores recebeu a identificação da *reservation station MD1* enquanto que o correspondente bit de ocupação recebeu o valor '1'. Desta forma, ficou indicado que a instrução armazenada na *reservation station MD1* irá atualizar o conteúdo do registrador R_0 . Como todos os operandos fonte necessários já se encontram na *reservation station* e a unidade funcional de multiplicação/divisão está livre, a execução desta instrução pode ser iniciada logo após o despacho.

A segunda instrução foi transferida para a *reservation station S1*. Tendo em vista que R_2 atua como operando fonte e destino desta instrução, então o conteúdo de R_2 , por ser válido, foi transferido para um dos campos de operando de *S1*, enquanto que a identificação desta *reservation station* foi armazenada no campo de rótulo de R_2 , e o bit de ocupação de R_2 recebeu o valor '1'. Assim, ficou indicado que o conteúdo do registrador R_2 será atualizado pela instrução armazenada na *reservation station S1*. Com relação ao segundo operando da instrução (o registrador R_0), ao verificar que o bit de ocupação correspondente é igual a '1', o algoritmo transferiu para o segundo campo de rótulos de *S1* a identificação da *reservation station* responsável pelo cálculo do novo valor de R_0 (*reservation station MD1*). Esta identificação estava armazenada no campo de rótulo de R_0 . Desta forma, ficou indicado que *S1* obterá tal operando do *CDB*, quando ele for propagado pela instrução armazenada em *MD1*. A instrução armazenada em *S1* não poderá ser executada imediatamente após seu despacho, pois terá que aguardar na *reservation station* até que todos os operandos fonte estejam disponíveis.

A terceira instrução foi transferida para a *reservation station S2*. O conteúdo dos registradores que atuam como operando fonte (R_3 e R_4) foi transferido para *S2* pois ambos são válidos. Para indicar que a instrução armazenada em *S2* irá

<i>Eventos</i>	<i>Reservation Stations</i>				<i>Registradores</i>					
	<i>MD1</i>		<i>S1</i>		<i>R₀</i>		<i>R₁</i>		<i>R₂</i>	
	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>
Despacho Multiplicação	-	-	-	-	1	<i>MD1</i>	0	-	0	-
Despacho Adição	-	-	-	<i>MD1</i>	1	<i>MD1</i>	0	-	1	<i>S1</i>
Término Multiplicação	-	-	-	-	0	-	0	-	1	<i>S1</i>
Término Adição	-	-	-	-	0	-	0	-	0	-

Figura 2.2: Efeito do Tratamento de Dependências Verdadeiras de Dados

atualizar o conteúdo do registrador R_3 , o bit de ocupação correspondente recebeu o valor '1' e o campo de rótulo recebeu a identificação de $S2$. Como todos os operandos fonte estão disponíveis em $S2$ e a unidade funcional de adição está livre, esta instrução pode ser executada logo após ser despachada.

As instruções do exemplo ilustram o comportamento do escalonamento associativo na presença de uma dependência verdadeira: o registrador destino da primeira instrução é usado como operando pela segunda. Apesar desta situação adversa, o algoritmo continua escalonando as instruções subseqüentes.

Ao fim da execução da multiplicação, o novo valor de R_0 será propagado, junto com a identificação de $MD1$, para as *reservation stations* e para o registrador destino (no nosso exemplo, para $S1$ e R_0 respectivamente). Após comparar o rótulo propagado pelo *CDB* com o conteúdo dos campos de rótulo, o resultado da operação é transferido para o registrador destino (cujo bit de ocupação passa a valer '0') e para as *reservation stations* que estiverem esperando por esse resultado. No nosso exemplo, a partir desse momento, a primeira instrução de adição poderá ser iniciada.

A Figura 2.2 apresenta o estado dos campos de rótulo (*Rot*) de algumas *reservation stations* e registradores e de seus bits de ocupação (*Bit*), após os eventos relacionados com a execução das 2 primeiras instruções do fragmento de programa apresentado.

2.3 Tratamento de Dependências Falsas

O algoritmo de Tomasulo continua o processo de escalonamento mesmo que existam os outros dois tipos de dependências (dependência de saída e anti-dependência). Ao copiar a identificação da *reservation station* que armazena a instrução responsável pela avaliação do novo valor de um registrador para o seu campo de rótulo, o algoritmo garante que as instruções subseqüentes que tenham este registrador como operando irão utilizar o valor correto.

Dizemos que existe uma dependência de saída quando um mesmo registrador é especificado como destino por duas instruções. Para ilustrar o comportamento do algoritmo quando em presença de dependências de saída entre instruções, consideremos o seguinte fragmento de programa:

$$R_0 \leftarrow R_0 \times R_1$$

$$R_0 \leftarrow R_1 + R_2$$

Como o registrador R_0 é especificado como operando destino por ambas as instruções, dizemos que existe uma dependência de saída entre elas.

Vamos assumir que, inicialmente, todas as *reservation stations* estão livres e todos os registradores são válidos. A instrução de multiplicação é despachada para a *reservation station MD1*. Como os bits de ocupação dos operandos fonte (R_0 e R_1) indicam que tais registradores são válidos, o conteúdo deles é transferido para *MD1* durante o despacho. Para indicar que a instrução armazenada em *MD1* irá alterar o conteúdo de R_0 , o campo de rótulo deste registrador recebe a identificação de *MD1* e o seu bit de ocupação passa para '1'.

A instrução de adição é despachada para a *reservation station S1*. Por serem válidos (bit de ocupação igual a '0') os conteúdos dos registradores que atuam como operando fonte (R_1 e R_2) são transferidos para *S1*. O campo de rótulo do operando destino (R_0) é atualizado com a identificação de *S1* para indicar que a instrução desta *reservation station* irá alterar o conteúdo de R_0 . A identificação

armazenada anteriormente no campo de rótulo associado a R_0 ($MD1$) é destruída. O bit de ocupação, por sua vez, mantém o mesmo valor ('1').

Verificamos que durante o processo de escalonamento, o campo de rótulo dos registradores é continuamente atualizado (com a identificação da *reservation station* cuja instrução especifica esse registrador como destino da operação). A qualquer instante, o conteúdo desse campo de rótulo identifica a última instrução que especificou o registrador correspondente como destino. Em outras palavras, se diversas instruções (já despachadas) especificarem como destino um mesmo registrador, somente a última dessas instruções é que irá alterar o conteúdo desse registrador.

No nosso exemplo, ao término da execução da multiplicação, o resultado da operação e a identificação de $MD1$ serão propagados através do CDB . Porém, como o conteúdo do campo de rótulo de R_0 difere da identificação propagada pelo CDB , o resultado não será copiado para esse registrador. Quando a execução da instrução de adição chegar ao fim, o resultado propagado através do CDB será copiado para R_0 , pois o campo de rótulo deste registrador armazena a identificação de $S1$ (a menos que tenha sido despachada uma instrução posterior a adição que especifique R_0 como operando destino). O bit de ocupação correspondente passa a valer '0'.

Ainda que, devido aos tempos de latência das instruções, a execução da operação de multiplicação termine após a da operação de adição, somente esta poderá atualizar R_0 . Neste caso, quando o resultado da multiplicação for propagado, o valor do bit de ocupação de R_0 será '0', conforme a atualização feita ao término da execução da adição. Uma vez que o bit de ocupação estará indicando que o registrador não espera por atualizações, tal resultado não será copiado para R_0 .

Portanto, após a execução de um conjunto de instruções com dependências de saída, o valor armazenado no registrador destino independe da ordem de término das instruções.

As Figuras 2.3 e 2.4 apresentam os estados dos dispositivos funcionais após os eventos relacionados com a execução das instruções do fragmento de

<i>Eventos</i>	<i>Reservation Stations</i>				<i>Registadores</i>					
	<i>MD1</i>		<i>S1</i>		<i>R₀</i>		<i>R₁</i>		<i>R₂</i>	
	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>
Despacho Multiplicação	-	-	-	-	1	<i>MD1</i>	0	-	0	-
Despacho Adição	-	-	-	-	1	<i>S1</i>	0	-	0	-
Término Adição	-	-	-	-	0	-	0	-	0	-
Término Multiplicação	-	-	-	-	0	-	0	-	0	-

Figura 2.3: Dependências de Saída (Adição terminando antes da Multiplicação)

<i>Eventos</i>	<i>Reservation Stations</i>				<i>Registadores</i>					
	<i>MD1</i>		<i>S1</i>		<i>R₀</i>		<i>R₁</i>		<i>R₂</i>	
	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>	<i>Bit</i>	<i>Rot</i>
Despacho Multiplicação	-	-	-	-	1	<i>MD1</i>	0	-	0	-
Despacho Adição	-	-	-	-	1	<i>S1</i>	0	-	0	-
Término Multiplicação	-	-	-	-	1	<i>S1</i>	0	-	0	-
Término Adição	-	-	-	-	0	-	0	-	0	-

Figura 2.4: Dependências de Saída (Multiplicação terminando antes da Adição)

programa apresentado. Na Figura 2.3 está representada a situação em que o término da execução da instrução de adição ocorre antes do término da multiplicação. Já a Figura 2.4 representa a situação em que o término da execução da instrução de adição ocorre após o término da multiplicação.

Quando o registrador destino de uma instrução é utilizado como operando fonte por uma anterior dizemos que elas são anti-dependentes. Utilizaremos o fragmento de programa a seguir, para ilustrar o comportamento do algoritmo quando na presença de anti-dependências:

$$R_2 \leftarrow R_0 \times R_1$$

$$R_0 \leftarrow R_1 + R_3$$

Como R_0 é operando fonte da multiplicação e destino da adição, estas instruções são anti-dependentes. Dependendo do estado do bit de ocupação de R_0 quando a primeira dessas instruções for escalonada, diferentes ações são tomadas. Se o bit de ocupação indicar que o conteúdo é válido, o conteúdo do registrador é transferido para a *reservation station* no momento do escalonamento da instrução.

Por outro lado, se o bit de ocupação indicar que o conteúdo de R_0 não é válido, então a multiplicação possui uma dependência verdadeira de dados

com uma instrução anterior. Esta instrução poderia ser, por exemplo:

$$R_0 \leftarrow R_1 + R_4$$

Como vimos na seção anterior, neste caso o conteúdo do campo de rótulo de R_0 é transferido para a *reservation station* no momento em que a multiplicação for despachada. O valor do operando fonte representado por R_0 será copiado do *CDB* pela *reservation station* que armazena a multiplicação, quando do término da primeira instrução de adição. Assim, qualquer que seja a situação, a instrução fica imune a alterações no valor deste registrador por instruções posteriores.

Com a utilização deste esquema de rotulação, técnicas alternativas para o tratamento de falsas dependências, como por exemplo a renomeação de registradores, tornam-se desnecessárias.

Capítulo 3

Modelos de Simulação

Durante a realização deste trabalho diversos modelos de máquina foram especificados, simulados e avaliados. Através dos experimentos levados a cabo com esses modelos, foi possível observar melhor o funcionamento do algoritmo de Tomasulo, e avaliar a importância dos componentes (*reservation stations e CDB*) e das atividades (despacho e rotulação) por ele realizadas. Uma vez reconhecida a importância desses dispositivos e atividades, eles foram incorporados definitivamente nos demais modelos. O sistema de memória, nos modelos iniciais, limitava o desempenho da máquina e por esse motivo, decidimos empregar um esquema alternativo nos outros sistemas. Outros mecanismos propostos foram posteriormente incluídos e avaliados. Assim, passo a passo evoluiu-se até o modelo final de máquina Super Escalar. Estes modelos são descritos a seguir.

3.1 A Máquina Básica

Na modelagem de uma máquina Super Escalar é essencial determinar o conjunto de instruções e a mistura adequada de unidades funcionais. Tendo em vista que estávamos interessados em examinar o efeito do escalonamento dinâmico no desempenho de máquinas Super Escalares, decidimos então incluir no nosso modelo as características de uma arquitetura Super Escalar comercialmente disponível. Por essa razão é que foram incorporados no nosso modelo, o repertório de instruções e os tipos de unidades funcionais do processador i860 da Intel [INTE89].

Dentre as instruções do i860 somente as que são executadas no modo dual e as *pipelined* não foram incorporadas. As instruções do tipo dual são utilizadas em processadores cujos algoritmos de detecção de paralelismo são implementados no compilador (escalonamento estático). Como no algoritmo de Tomasulo esta tarefa é realizada pelo *hardware*, o modo dual foi ignorado. Conforme apontado por Pleszkun [PLES88] a utilização de técnicas de *pipeline* nas unidades funcionais de um processador não vetorial, não proporciona uma melhora superior a 4 % no desempenho. Por esse motivo as unidades funcionais do nosso modelo não utilizam tal tecnologia.

O conjunto de instruções do i860 opera sobre um banco de registradores inteiros e um banco de ponto flutuante. Nos nossos modelos, esses dois bancos de registradores possuem a mesma configuração como no i860. Os dados e as instruções são armazenadas em sistemas de memória que variam de modelo para modelo.

Excetuando a unidade *Core*, a funcionalidade das outras unidades do i860 foi mantida no nosso modelo. A unidade *Core* deixou de ser responsável pela execução das instruções de *Load/Store*. Para realizar estas operações, introduzimos um outro tipo de dispositivo funcional: a unidade de Acesso à Memória.

Desse modo, no nosso modelo de arquitetura, as seguintes funções são realizadas por cada tipo de unidade:

Core - Responsável pela execução de todas as operações lógicas e aritméticas com inteiros, instruções de desvio e de transferência de dados de um registrador inteiro para um outro de ponto flutuante.

Soma - Esta unidade executa as instruções de soma, subtração e comparação com reais, bem como a instrução de conversão de formato de ponto flutuante para inteiro.

Multiplicação - A unidade de multiplicação realiza as operações de multiplicação com reais e a avaliação do recíproco de um real (usada para implementar a divisão de reais).

Gráfica - Ela executa operações aritméticas sobre valores de tipo inteiro longo armazenados no banco de registradores de ponto flutuante. Executa, também, instruções para facilitar a implementação de algoritmos gráficos tridimensionais. Dentre estes algoritmos encontram-se o de sombreamento de *pixels* e o *Z-buffer* para a eliminação de superfícies encobertas. A operação de transferência de dados de registradores de ponto flutuante para os inteiros também é realizada por esta unidade.

Acesso à Memória - As instruções de transferência de dados entre o sistema de memória e os bancos de registradores, são executadas por este tipo de unidade funcional.

3.2 Modelos Prévios

Os primeiros experimentos levados a cabo estão descritos em [BAR92a]. No modelo utilizado, associamos a cada grupo de unidades funcionais de um determinado tipo da máquina básica, um conjunto de *reservation stations*. Graças a rede de conexão entre esses dois conjuntos de dispositivos, todas as unidades funcionais de um mesmo tipo tinham acesso a qualquer uma das *reservation stations* pertencentes ao conjunto associado à elas.

Para implementar o esquema de rotulação descrito no Capítulo 2, acrescentamos um campo de rótulo e um bit de ocupação nos registradores dos dois bancos. Além disso, foi incluído um *CDB* para propagar os resultados produzidos pelas unidades funcionais, para as *reservation stations* e registradores.

Nesses modelos, as instruções eram buscadas, decodificadas e escalonadas seqüencialmente. Assim, a busca de uma instrução só seria iniciada após o escalonamento da anterior. Como consequência da utilização do algoritmo de Tomasulo em todos os tipos de unidade funcional, o mecanismo de escalonamento de instruções foi alterado para tratar mais uma condição de parada. Além da inexistência de *reservation stations* disponíveis, o escalonamento de uma instrução de desvio condicional passou a ser uma condição de interrupção do mecanismo de des-

pacho.

Somente quando todos os dados necessários para a execução de uma instrução estiverem disponíveis na *reservation station* e quando uma unidade funcional estiver livre, é que a instrução pode ser iniciada. Havendo mais instruções prontas para serem executadas do que unidades funcionais disponíveis, a prioridade no atendimento é para aquelas que foram escalonadas há mais tempo. O algoritmo permite que uma instrução com todos os seus operandos disponíveis, seja executada antes de outra escalonada há mais tempo mas que não esteja pronta para ser executada. A única exceção é para a unidade de Acesso à Memória. Nessa unidade, as *reservation stations* são organizadas segundo uma fila. Assim, uma instrução só é executada quando todas as demais instruções de acesso à memória escalonadas anteriormente já tiverem começado a ser executadas. Através desse esquema garante-se a consistência dos dados na memória.

No nosso modelo, a memória é organizada em duas partições, uma para o armazenamento de instruções e outra para dados. Foi assumida a existência de um sistema de acesso a instruções extremamente rápido, como uma memória *cache*, por exemplo. Contudo, omitimos a descrição do comportamento desse tipo de memória. Assumimos que a busca de qualquer instrução levaria um mesmo intervalo de tempo t relativamente pequeno.

A memória de dados ficou organizada sob a forma de um ou mais bancos, cada um deles associado à uma unidade funcional de acesso à memória, sendo esta a única capaz de acessá-lo. Os bancos de memória são numerados em ordem crescente e os endereços de memória foram distribuídos através dos bancos com cada grupo de quatro *bytes* consecutivos armazenados num mesmo banco. Desta forma, é possível realizar simultaneamente tantos acessos à memória quantos forem os bancos.

Ao término da execução de uma instrução, é feito um pedido de acesso ao *CDB* para propagar o resultado da operação. Se mais de uma unidade requisitar o *CDB* simultaneamente, a prioridade no uso é para a unidade que começou a executar a sua instrução há mais tempo.

Nos modelos que foram descritos posteriormente [FER92a,FER92b], foram introduzidas modificações para aprimorar o modelo inicial e atingir um desempenho maior.

O sistema de memória de instruções foi alterado de modo que o comportamento de uma memória *cache* com as características da existente no i860 fosse reproduzido. Introduzimos intervalos de tempo distintos para o acesso a esse sistema caso ocorra um acerto ou uma falha.

Com o objetivo de investigar a contribuição (no desempenho do modelo) oferecida pelo *CDB* e pela presença de *reservation stations*, configurações incorporando (ou não) esses componentes foram especificadas. Desse modo, obtivemos as seguintes variações de arquiteturas derivadas do modelo básico:

- a) sem *CDB* e sem *reservation stations*: Como não existe nenhuma *reservation station*, as instruções são transferidas diretamente para um *buffer* nas unidades funcionais, capaz de armazenar, no máximo, uma instrução. Como não existe *CDB*, os resultados gerados são transferidos para os registradores de onde são lidos, como operandos, pelas unidades funcionais.
- b) com *CDB* e sem *reservation stations*: Essa variação possui um *CDB* conectando as unidades funcionais e os bancos de registradores da máquina. Através do *CDB*, os resultados produzidos pelas unidades funcionais são transferidos diretamente para um dos bancos de registradores.
- c) sem *CDB* e com *reservation stations*: Associado a cada conjunto de unidades funcionais de um determinado tipo existe pelo menos uma *reservation station*. O código de operação e os operandos de cada instrução são transferidos para uma *reservation station* antes dela ser executada. Como não há *CDB*, as *reservation stations* buscam os operandos no banco de registradores.
- d) com *CDB* e com *reservation stations*: Nesta variação, os dados relativos a cada instrução são transferidos para as *reservation stations*. Os resultados gerados pelas unidades funcionais podem ser transferidos para o banco de registradores e para as *reservation stations* simultaneamente.

Nas variações (a) e (c) (modelos sem *CDB*), um esquema do tipo *scoreboard* [THOR64] é responsável pelo tratamento das dependências. A inexistência, nesses modelos, de um *CDB* (e do correspondente esquema de propagação de rótulos e de valores) provoca uma condição adicional de interrupção no processo de escalonamento de instruções: toda vez que o registrador destino de uma instrução a ser escalonada já tiver sido designado para armazenar o resultado de uma instrução anterior, o processo é interrompido.

Para garantir a consistência dos dados armazenados na memória, empregamos a política *FCFS* (*First Come First Served*) na execução das instruções de *Load/Store*. Assim, nas variações (a) e (b) (modelos sem *reservation stations*), para que não tivéssemos uma condição adicional de interrupção no processo de escalonamento de instruções, foi permitido que as unidades funcionais de acesso à memória referenciassem qualquer um dos bancos de memória. Se cada unidade de acesso à memória estivesse ligada a somente um banco, o processo de escalonamento seria interrompido sempre que a unidade funcional ligada ao banco de memória que contém o endereço apontado pela instrução de *Load/Store* que se pretende despachar, estivesse ocupada.

3.3 Modelo Final

O modelo descrito a seguir foi o utilizado para avaliar o efeito do escalonamento dinâmico em processadores Super Escalares. Esse modelo é resultante da eliminação das principais deficiências que limitavam o desempenho dos modelos prévios. Adicionalmente, o modelo incorpora algumas técnicas modernas utilizadas em processadores atuais.

Nas configurações anteriores da máquina básica, o tempo de latência requerido pelas operações de *load* e *store* era relativamente longo quando comparado com o tempo de latência das instruções executadas pela *Core*. Por esse motivo, a memória se tornava um gargalo que penalizava o desempenho do processador, além de dificultar a avaliação do efeito dos demais dispositivos do processador e do seu algoritmo de escalonamento dinâmico. Ao introduzirmos um outro nível hierárquico

no subsistema de memória, nosso modelo ficou mais semelhante aos processadores Super Escalares atuais.

A capacidade de armazenamento da memória *cache* que foi incorporada no nosso modelo é igual a do processador i860, e a memória principal ficou organizada segundo um único banco.

O acesso ao sistema de memória de dados passou a ser feito através de uma única unidade funcional. Dessa forma, duas ou mais instruções de acesso à memória não podem ser executadas simultaneamente. Associada a esta única unidade funcional, um número variado de *reservation stations* ficam organizadas segundo uma fila. As instruções despachadas para estas *reservation stations* são enfileiradas de acordo com a ordem de escalonamento. Este é o critério usado para determinar a ordem de execução dessas instruções. Numa das variações desse modelo, decidimos incluir um método de remoção de ambigüidades [BAR92b]. Através desse método, a ordem de atendimento das operações de *load* e *store*, pode ser relaxada em algumas situações, deixando de obedecer ao critério *FCFS* (*First Come First Served*). Dessa forma, operações com a memória podem ser antecipadas nas seguintes situações:

1. um *Load* pode ser antecipado quando não houver nenhuma operação de *Store*, anterior a ela na fila, que acesse as mesmas posições de memória.
2. um *Store* pode ser antecipado quando não houver nenhuma operação de *Load* ou *Store*, anterior a ela na fila, referenciando à mesma posição de memória.

Portanto, para que uma instrução de *Load* seja antecipada, é necessário que os endereços de todas as instruções de *Store* anteriores a ela já tenham sido avaliados. Já as instruções de *Store* exigem também, a avaliação do endereço das operações de *Load* anteriores a ela. Caso exista mais de uma instrução capaz de ser antecipada será dada prioridade aquela despachada a mais tempo.

O algoritmo de escalonamento de instruções foi modificado de modo que múltiplas instruções pudessem ser buscadas, decodificadas e despachadas simultaneamente [BAR92b]. Baseado no trabalho [DWYE87], assumiu-se que o tempo

gasto na realização destas tarefas para múltiplas instruções simultaneamente, seria o mesmo que para uma instrução de cada vez. Além do mais, estávamos interessados em avaliar a eficiência desta política de escalonamento levando em consideração somente as limitações impostas pelo algoritmo de Tomasulo e pelas características dos componentes do *benchmark*.

Neste modelo, o “número n de *CDB's*” do processador passou a ser um parâmetro de simulação. Assim, tornou-se possível propagar os resultados de até n instruções que terminassem simultaneamente.

Também foi incluído um mecanismo de predição de desvios condicionais. Este método não está baseado em nenhuma proposta realista conhecida. Durante a simulação, ao se escalonar uma instrução de desvio condicional, é gerado um número aleatório que determinará se a predição deste desvio será correta ou não. A chance de acerto é passada como parâmetro de simulação através de um número denominado “taxa de acerto.” O motivo que nos levou a esta opção, foi a nossa preocupação em determinar um limite máximo teórico para o desempenho deste modelo ao utilizar algum método realista de predição de desvios. Este limite pode ser avaliado atribuindo-se o valor 100% à taxa de acerto.

Os tempos de latência de cada unidade funcional são expressos em termos de ciclos de máquina. As seguintes latências foram usadas durante nossos experimentos:

- decodificação e escalonamento de instruções: um ciclo cada;
- busca de instruções: um ciclo para *hit* ou dezesseis para *miss* na *cache* respectivamente;
- operações na unidade de adição de ponto flutuante: seis ciclos;
- operações na unidade de multiplicação de ponto flutuante: seis ou nove ciclos, para simples e dupla precisão respectivamente;
- operações de acesso a *cache* de dados: três ou dezoito ciclos, para *hit* ou *miss* respectivamente;
- operações na unidade funcional *core*: três ciclos;
- operações na unidade gráfica: seis ciclos

Assumiu-se que a busca de operandos para as instruções no banco de registradores seria feita durante a decodificação, que abrangeria as seguintes fases:

- $$\left\{ \begin{array}{l} \bullet \text{ decodificação;} \\ \bullet \text{ busca de operandos} \end{array} \right.$$

Para garantir o funcionamento correto do mecanismo de despacho associativo, os resultados propagados pelo *CDB* não podem alterar o banco de registradores durante a fase de busca de operandos. Portanto, quando resultados forem propagados durante o ciclo de decodificação, o banco de registradores será atualizado durante a primeira fase da decodificação.

O escalonamento de instruções também foi dividido em duas fases:

- $$\left\{ \begin{array}{l} \bullet \text{ despacho;} \\ \bullet \text{ avaliação do } CDB \end{array} \right.$$

Durante o despacho, os parâmetros para a execução das instruções (obtidos durante a decodificação) são transferidos para as *reservation stations* disponíveis. Na segunda fase do escalonamento, as *reservation stations* verificam se o *CDB* está propagando algum operando esperado por elas. Caso afirmativo, a *reservation station* armazena este operando no campo apropriado. Desta forma evitamos que instruções despachadas no mesmo ciclo em que algum de seus operandos foi propagado, fiquem esperando eternamente tais operandos.

Capítulo 4

Método de Avaliação

Este capítulo apresenta uma breve descrição do nosso meio ambiente de avaliação e da metodologia utilizada na condução de nossos experimentos. A bateria de programas de teste, a máquina referência e as características do nosso processo de simulação de Arquiteturas Super Escalares, serão discutidos a seguir.

4.1 A Máquina Referência

Para avaliar o modelo de máquina Super Escalar descrito no Capítulo 3, comparamos o seu desempenho com o de uma outra que servisse como referência. No modelo utilizado como referência as instruções são executadas seqüencialmente. Uma instrução só é buscada após a conclusão da anterior. Os tempos de latência das unidades funcionais da máquina Super Escalar foram mantidos na máquina referência. Comparando os desempenhos dos dois modelos, podemos verificar as vantagens oferecidas pelas técnicas de extração do paralelismo de baixo nível que foram exploradas pelos nossos experimentos.

4.2 Programas Teste

Para avaliar o efeito do escalonamento dinâmico no desempenho de máquinas super escalares, submetemos o nosso modelo de máquina super escalar a um conjunto de situações freqüentemente encontradas na prática. Assim, foi selecionado um

conjunto de oito programas, formando a bateria de teste (*benchmark*), com diferentes características. Os programas da bateria de teste são:

- Integral
- Decomposição LU
- Hu-Tucker
- Branch
- BCDBin
- Livermore Loop 24
- Quick-Sort
- Bubble-Sort

O programa **Integral** (Int) realiza o cálculo aproximado, segundo o método dos trapézios [CONT65], da integral da função $f(x) = x^2$ para $0 \leq x < 30$ e altura do trapézio igual a 0,1; o de **Decomposição LU** (LU) [FORS67] faz a decomposição de uma matriz de dimensão 10 x 10; o **Hu-Tucker** (Hu) [YOHE72] implementa um algoritmo de codificação binária alfabética com redundância mínima; o programa **Branch** (Brc) [DITZ87] realiza a contagem do total de números pares e ímpares contidos no intervalo [0,2559] e calcula o somatório de tais números; o **BCDBin** (Bcd) [AUHT85] realiza a conversão para formato binário de 40 números codificados em *BCD*; o **Livermore Loop 24** (Liv) [MCMA83] determina o índice do menor componente de um vetor de dimensão 700; o programa **Quick-Sort** (Qck) [KNUT73] classifica um vetor de 200 posições; finalmente, o programa **Bubble-Sort** [KNUT73] realiza a ordenação de um vetor de dimensão 35.

Codificados em linguagem C, os programas da bateria foram traduzidos para o código objeto do i860 por um compilador comercial. A Tabela 4.1 mostra o número de ciclos (na máquina referência) requerido por cada programa, o número de instruções do i860 executadas por cada tipo de unidade funcional, e o total de instruções executadas.

Examinando essa tabela, podemos constatar que somente os códigos dos programas Integral e Decomposição LU possuem instruções que são executadas pelas unidades funcionais de Multiplicação, Adição e Gráfica. Além disso, a razão entre a quantidade de instruções destes tipos e o total geral de instruções executadas, é bem maior no programa Integral do que no LU. O programa Branch, por sua vez, é o único componente da bateria cujas variáveis estão todas armazenadas em

Programa	Core	Mult	Ad	Mem	Gra	Total	Ciclos
Integral	8.083	1.196	2.991	2.995	300	15.565	109.410
LU	10.406	555	506	4.111	33	15.611	97.781
Hu-Tucker	9.544	0	0	4.605	0	14.149	86.399
Branch	19.206	0	0	0	0	19.206	116.563
BCDBin	16.184	0	0	2.501	0	18.649	113.142
Livermore	11.189	0	0	4.369	0	15.558	94.932
Quick-Sort	16.995	0	0	4.268	0	21.263	129.671
Bubble-Sort	10.282	0	0	5.808	0	16.090	97.314

Tabela 4.1: Total de Instruções por Unidade Funcional e Ciclos

registradores. Por esse motivo nenhuma instrução foi executada pela unidade de Acesso à Memória.

4.3 A Simulação

Uma vez traduzidos para o código objeto os programas de teste foram processados por um simulador do i860 e os correspondentes arquivos contendo os *traces* da execução foram produzidos. Cada arquivo de *trace* é composto por um conjunto de índices, cada um associado a uma instrução do arquivo de código seqüencial. Esses dois índices correspondem à seqüência de execução destas instruções. Desta forma, evitou-se que as informações relativas à uma instrução fossem repetidas desnecessariamente, como aconteceria se, em vez de índices, utilizássemos o código das instruções no *trace*.

Além da simulação, os arquivos contendo o código objeto do i860 passam por uma etapa de pré-processamento. Durante essa etapa, é gerado um arquivo, denominado arquivo de código seqüencial que armazena informações relacionadas com as instruções do programa.

O comportamento da máquina Super Escalar, ao executar cada um dos programas da bateria de teste, é reproduzido levando em consideração apenas as dependências de dados existentes entre as instruções do arquivo de código seqüencial quando despachadas na ordem indicada pelo *trace*, e de acordo com os tempos de latência. Durante a simulação destes programas no nosso modelo de máquina

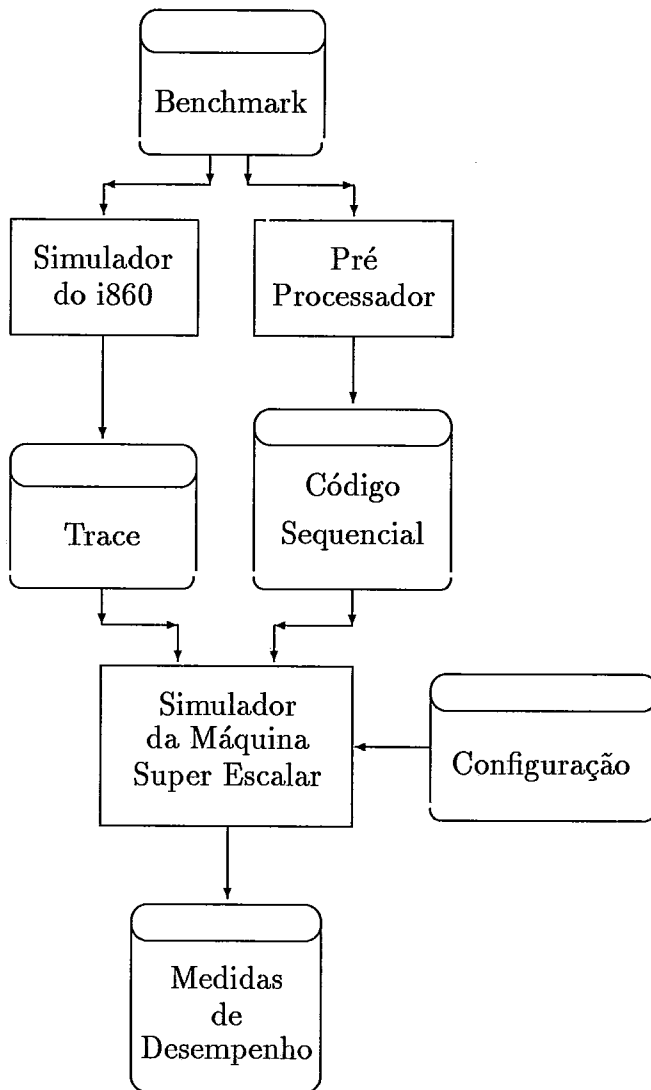


Figura 4.1: Etapas do Processo de Simulação

Super Escalar, as instruções não são efetivamente executadas. Esta execução ocorre somente uma vez para cada programa da bateria de teste e é realizada pelo simulador do i860.

Além do *trace*, o simulador lê um “arquivo de configuração” contendo os seguintes parâmetros:

- Total de *reservation stations* associadas a cada tipo de unidade funcional
- Total de unidades funcionais de cada tipo
- Número máximo de instruções que podem ser escalonadas simultaneamente (Tamanho da Janela de Instruções)
- Total de CDB's
- Taxa de acerto do método de predição de desvios

Ao término da simulação de um programa na máquina Super Escalar, um relatório é gerado. Os dados apresentados neste relatório incluem:

- Total de ciclos requeridos pelo programa na máquina Super Escalar
- Total de ciclos requeridos na máquina referência
- Taxa de aceleração (*speed-up*)
- Taxa de redução
- Tempo de ocupação de cada *reservation station*
- Tempo que um certo número de *reservation stations* permaneceram ocupadas
- Tempo de espera por operandos de cada *reservation station*
- Tempo de ocupação de cada unidade funcional
- Tempo em que um certo número de unidades funcionais ficaram ocupadas
- Tempo que um certo número de CDB's ficaram ocupados

A Figura 4.1 ilustra as diversas etapas do nosso modelo de simulação.

Realizando várias simulações com diferentes parâmetros e avaliando os resultados gerados no relatório, foi possível avaliar o efeito do escalonamento dinâmico no desempenho de máquinas super escalares.

