Register Renaming

Fernanda Alcântara Andaló
RA: 041462

fernanda.andalo@ic.unicamp.br

RESUMO

Register Renaming ou Renomeação de Registradores é uma técnica utilizada para evitar serialização desnecessária na execução de instruções. Neste trabalho, apresenta-se uma visão detalhada desta técnica, mostrando as principais implementações e atual utilização nos processadores.

1. INTRODUÇÃO

Processadores superescalares tentam explorar paralelismo de instruções para conseguir ganho em desempenho. Porém, esta melhoria é limitada pela granularidade do paralelismo presente no código de um programa [6]. Alguns mecanismos, como *Register Renaming* e especulação dinâmica, têm sido desenvolvidos para aumentar o número de instruções que um processador pode executar em paralelo.

A arquitetura do conjunto de instruções – *Instruction Set Architecture* (ISA) – provê um certo número de registradores para o programador [1]. Para se ter uma codificação de instruções compacta, a maioria dos processadores tem um número reduzido de registradores. Por exemplo, o conjunto de instruções $x86$ tem 8 registradores de inteiros, $x86$-64 tem 16, muitos RISC tem 32 e IA-64 tem 128. Este tipo de limitação gera problemas, como pode ser observado no trecho de código a seguir:

```
(1) r1 := r2 / r3
(2) r2 := r1 + r3
(3) r1 := r5 + r8
(4) r4 := r1 - r7
```

As quatro instruções apresentadas precisam ser executadas serialmente. A instrução (2) deve ser executada após (1), pois usa o resultado de (1); similarmente (4) precisa ser executada após (3). Tais dependências são chamadas de *dependências de fluxo*. Como (1) e (3) utilizam o mesmo registrador de resultado, (3) precisa ser executada após (1). Este é um exemplo de *dependência de saída*. Também, (3) precisa ser executada após (2), pois (2) utiliza o valor de r1. Se (3) terminar antes de (2), o valor em r1 utilizado por (2) seria sobrescrito, o que não é correto. Isso é conhecido como *anti-dependência* [6].

As duas últimas dependências são dependências falsas, pois a serialização é causada pelo uso do mesmo registrador de resultado em duas instruções diferentes. Esta serialização pode ser eliminada renomeando-se o registrador de resultado da operação (3), para que (3) seja executado em paralelo com (1) e (2). A técnica de *Register Renaming* tenta aplicar esta transformação dinamicamente, aumentando o paralelismo disponível.

Quando é possível, o compilador promove essa renomeação estaticamente. Entretanto, o compilador é limitado pelo número de registradores no conjunto de instruções. Por isso, muitos processadores de alto desempenho promovem a renomeação de registradores em *hardware* para extrair paralelismo adicional [10].

Mesmo com a utilização de um número maior de registradores disponíveis no conjunto de instruções, como no conjunto de instruções da arquitetura IA-64, ainda existem limitações [10]:

- É muito difícil para o compilador evitar a reutilização de registradores sem um grande aumento no tamanho do código. Em laços de repetição, por exemplo, iterações sucessivas teriam de usar diferentes registradores, o que implicaria na replicação de código.
- Um número grande de registradores requer muitos *bits* para especificação, fazendo com que o código aumente de tamanho.
- Muitos conjuntos de instruções já especificaram um número pequeno de registradores e isso não pode ser mudado.

O objetivo deste trabalho é apresentar a solução que a técnica de *Register Renaming* provê para os problemas apresentados, citando as implementações mais importantes e como elas fazem uso dos diferentes esquemas de *Register Renaming*.

O texto subseqüente está organizado da seguinte forma: a seção 2 discute algumas questões necessárias para a compreensão das próximas seções. A seção 3 apresenta os diferentes esquemas de *Register Renaming*. A seção 4 descreve e detalha as principais implementações da técnica em questão. A seção 5 apresenta um histórico dos processadores e compiladores que utilizam a técnica de *Register Renaming*, com ênfase na fase atual. Por fim, a seção 6 conclui este trabalho.

2. CONSIDERAÇÕES INICIAIS

2.1 Registradores físicos vs. Registradores de Arquitetura

Programas em linguagem de máquina lêem e escrevem em um conjunto limitado de registradores especificados pelo conjunto de instruções. Por exemplo, o conjunto de instruções do DEC Alpha especifica 32 registradores de inteiros, cada um de 64 *bits*, e 32 registradores de ponto-flutuante. Estes são os registradores da arquitetura. Programas escritos para processadores rodando o conjunto de instruções Alpha especificarão operações de leitura e escrita nesses 64 registradores.

Um processador particular que implementa este conjunto de instruções, o Alpha 21264, tem 80 registradores físicos de inteiro e 72 registradores físicos de ponto flutuante. Eles são, em um *chip* da Alpha 21264, 80 localidades fisicamente separadas que podem armazenar os resultados de operações com inteiros e 72 localidades que podem armazenar os resultados de operações de ponto flutuante. De fato, existem mais localidades do que as apresentadas, porém essas não são utilizadas para renomeação de registradores [10].

2.2 Register Renaming Parcial vs. Total

Para indicar a extensão da utilização de *Register Renaming* pelos processadores, existe uma classificação destes em renomeação parcial e total [7].

Renomeação parcial é restrita a um ou poucos tipos de instrução, por exemplo, apenas instruções de ponto flutuante. Renomeação total inclui todas as instruções que possuem registrador de destino.

2.3 Conflito de dados

Um conflito de dados é gerado toda vez que há dependência entre instruções e elas estão interligadas de forma que a ordem de execução em um *pipeline* mudaria a ordem de acesso aos operandos envolvidos na dependência. Por causa desta dependência, precisa-se preservar a ordem de programa, que é a ordem que as instruções seriam executadas se a execução fosse serial. O objetivo geral do *software* e *hardware* é explorar o paralelismo, preservando a ordem de programa somente quando uma mudança afeta a saída do programa. A detecção e prevenção de conflitos assegura que a ordem de programa necessária seja preservada [2].

Os conflitos de dados possíveis são:

- Leitura após escrita (RAW): A leitura de um registrador ou localidade de memória retorna um valor que lá foi colocado pela última escrita na ordem de programa, não qualquer outra escrita. Esta dependência requer que as instruções sejam executadas na ordem de programa. O conflito pode ser gerado pela seguinte situação: uma instrução j tenta ler uma fonte antes que a instrução i a escreva, então j obtém, incorretamente, o valor antigo.

- Escrita após escrita (WAW): Escritas sucessivas em um registrador ou localidade de memória particular devem deixar naquela localidade o valor gerado pela última escrita. Esta dependência é resolvida pelo cancelamento das escritas anteriores, se necessário. O conflito pode ser gerado pela seguinte situação: j tenta escrever um operando antes deste ser escrito por i. As escritas podem ser executadas na ordem errada, deixando no destino o valor escrito por i ao invés do valor escrito por j.

- Escrita após leitura (WAR): A leitura de um registrador ou localidade de memória deve retornar o último valor escrito naquela localidade, e não um escrito após a leitura. Este é o tipo de dependência que pode ser resolvida por renomeação. O conflito pode ser gerado pela seguinte situação: j tenta escrever em um destino antes que ele seja lido por i, obtendo incorretamente o novo valor.

3. ESQUEMAS DE REGISTER RENAMING

Register Renaming é uma técnica para remover dependências de dados falsas – WAR e WAW, vistas na seção anterior. Eliminando requisitos de precedência relacionados, a renomeação aumenta o número de instruções que estão disponíveis para execução paralela por ciclo. Isso resulta em um aumento no número de instruções executadas por ciclo – IPC [7].

O princípio da técnica é direto: se o processador encontra uma instrução que endereça um registrador de destino, ele temporariamente escreve o resultado da operação em um *buffer* de renomeação alocado dinamicamente, ao invés de escrever no próprio registrador de destino. A escolha do tipo de *buffer* de renomeação para ser usado em um processador tem impacto direto na implementação do processo de renomeação.

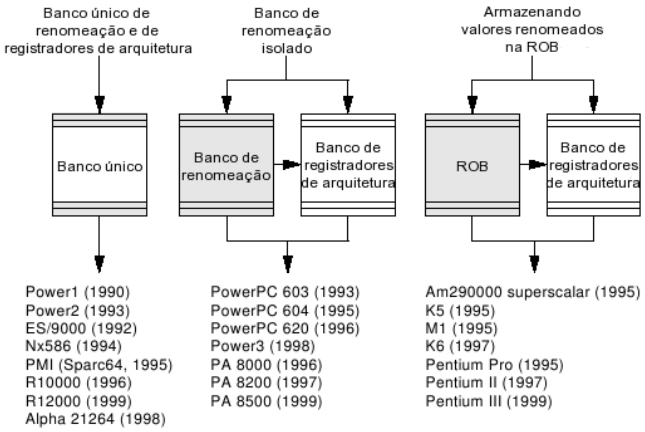

Figure 1: Tipos genéricos de *buffers* de renomeação. Caixas cinzas representam os *buffers* de renomeação.

As duas próximas subseções apresentam métodos que são soluções comuns para prover as localidades dos registradores de renomeação [5] (Figura 1).

3.1 Banco de registradores físicos

Neste primeiro método, os *buffers* de renomeação são implementados no mesmo banco de registradores físicos, cha-

mado de banco único de renomeação e de registradores da arquitetura. Então, ambos registradores da arquitetura e de renomeação são dinamicamente alocados a registradores particulares no mesmo banco de registradores físicos.

O banco de registradores físicos deve conter mais registradores do que os definidos pelo conjunto de instruções. Por meio de uma tabela de mapeamento, cada registrador da arquitetura é mapeado a um registrador físico no estágio de decodificação. O registrador de destino de uma instrução é mapeado a um registrador físico livre e os registradores de origem são associados ao último mapeamento associado a eles. Quando uma operação se encerra, o registrador físico alocado pela operação anterior com o mesmo registrador arquitetural de destino é liberado [5].

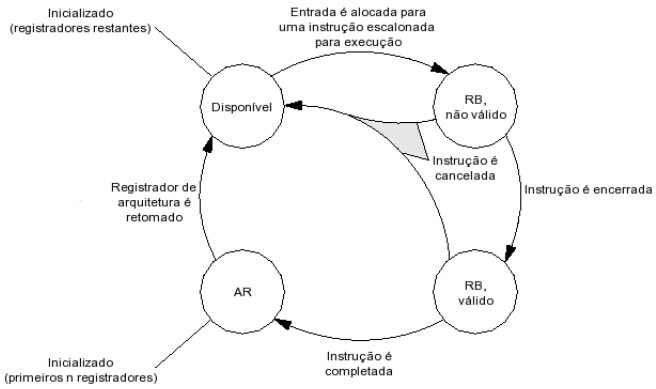

Figure 2: Diagrama de transição dos estados de um registrador particular do banco único. AR: registrador da arquitetura, RB: *buffer* de renomeação.

Para obter estes resultados, cada registrador físico do banco único deve estar em um dos quatro estágios possíveis. Estes estados refletem a utilização atual dos registradores físicos [7]. Os estágios são descritos a seguir (Figura 2).

- Estado disponível.
- Utilizado como um registrador da arquitetura (estado AR).
- Utilizado como *buffer* de renomeação – mas o registrador ainda não contém o resultado da instrução associada (Estado RB, não válido).
- Utilizado como *buffer* de renomeação – o registrador contém o resultado da instrução associada (Estado RB, válido).

Como parte da inicialização, os primeiros n registradores físicos são associados aos registradores da arquitetura, onde n é o número de registradores da arquitetura declarado pelo conjunto de instruções. Estes registradores estão no estado AR. O restante dos registradores físicos ficam disponíveis. Quando uma instrução escalonada para execução possui um registrador de destino, um novo *buffer* de renomeação é selecionado entre os disponíveis e alocado para o registrador de destino. Seu estado é mudado para RB, não válido, e seu *bit* de validade sofre um *reset*. Quando a instrução associada finaliza sua execução, o resultado produzido é escrito no *buffer* de renomeação alocado. Seu *bit* de validade é mudado e seu estado passa a ser RB, válido. Quando a instrução associada é completada, o *buffer* de renomeação alocado é declarado como o registrador da arquitetura que implementa o registrador de destino especificado. Seu estado, então, muda para AR para refletir esta situação. Finalmente, quando registradores da arquitetura antigos são retomados, seu estado muda para disponível.

O banco único é utilizado, por exemplo, nos modelos topo de linha dos *mainframes* ES/9000 da IBM, processadores Power e R1x000, e Alpha 21264. Outras alternativas separam os *buffers* de renomeação dos registradores da arquitetura. Uma possível variante, com um banco de renomeação isolado, é utilizada exclusivamente para implementar *buffers* de renomeação. O PowerPC 603/620 e os processadores PA8x00 são exemplos do uso do banco de renomeação isolado [7].

3.2 Reorder buffer (ROB)

Uma alternativa ao método apresentado é a utilização de *buffers* de reordenação (ROB). O ROB é muito utilizado atualmente para preservar a consistência seqüencial na execução das instruções. Quando utiliza-se o ROB, uma entrada é associada a cada instrução escalonada durante toda sua execução. É natural utilizar esta estratégia também para a renomeação. Isto pode ser feito basicamente estendendo-se as entradas com um campo adicional que possui o resultado da instrução. Exemplos de processadores que se utilizam do ROB são o superescalar Am29000, K5, K6, Pentium Pro, Pentium II e Pentium III.

Os resultados de cada instrução são mantidos no *buffer* de reordenação até o seu término. Então, o resultado é escrito no banco de registradores. Os operandos de origem que estão disponíveis quando a instrução é decodificada são lidos do banco de registradores ou da entrada correspondente no *buffer* de reordenação. Os operandos que ainda não estão disponíveis na decodificação são enviados das unidades de execução para as filas de entradas correspondentes (*reservation units*) quando são produzidos. Quando uma instrução termina, seu resultado é copiado do *buffer* de reordenação para o banco de registradores [5].

4. PRINCIPAIS IMPLEMENTAÇÕES

4.1 Algoritmo de Tomasulo

A primeira implementação de *Register Renaming* foi elaborada para a unidade de ponto flutuante do processador IBM 360/91 [9]. O processador IBM 360 possui arquitetura CISC, porém será descrito apenas para operações do tipo registrador-para-registrador.

Cada registrador é controlado por um *bit* que indica se ele está disponível ou não – *busy bit*. Se o *bit* está desligado, o registrador possui o valor que qualquer instrução que o referencia precisa. Se o *bit* estiver ligado, o registrador possui uma *tag* da instrução que produzirá o valor necessário.

Quando uma instrução entra no estágio de decodificação, ela recebe uma *tag*. A instrução lê cada registrador de origem,

obtendo um valor de entrada ou uma *tag*, dependendo do *busy bit*. A instrução escreve sua *tag* em seu registrador de destino e liga o *busy bit* deste registrador. A instrução e seus valores/*tags* de entrada são armazenados em uma das muitas *reservation stations*, dependendo da unidade funcional em que ela irá executar.

Em cada ciclo, uma unidade funcional busca as instruções armazenadas em sua *reservation station*. Uma instrução pode estar pronta, isto é, tem ambos os valores de entrada disponíveis; ou esperando, um ou mais de suas entradas é uma *tag*, e a instrução não pode executar até que outras instruções produzam a entrada. A unidade funcional irá selecionar uma instrução pronta e começar a executá-la.

Quando uma instrução estiver pronta, sua resposta e sua *tag* sofrem *broadcast* para todas instruções em espera. Cada instrução em espera compara a *tag* recebida com a *tag* que está esperando. Se as *tags* forem a mesma, a instrução sobrescreve sua *tag* com o resultado recebido. Isso pode mudar o status da instrução mude de "esperando" para "pronta". O resultado também é enviado para o banco de registradores. Se a *tag* no registrador de destino é a mesma da instrução, o valor é escrito no banco de registradores [6].

O Algoritmo de Tomasulo deve ser estendido para implementar interrupções precisas. Johnson [3] descreveu tal extensão, baseado no mecanismo de *future file*. Os registradores descritos acima correspondem ao *future file*. Também existe um banco de registradores duplicado chamado *in-order file*, e um ROB. Instruções escalonadas são descritas utilizando-se o *future file*. Adicionalmente, os resultados são escritos no ROB. O banco de reordenação é uma fila FIFO. Quando uma instrução é escalonada, ela entra na fila. Se a instrução no início da fila terminar, seu resultado é usado para atualizar o *in-order file*, e a fila prossegue [6].

Pode-se perceber que o ROB atualiza o *in-order file* com os resultados das instruções na ordem em que as instruções foram escalonadas. Quando o resultado de uma instrução é escrito no *in-order file*, qualquer exceção causada é assinalada. Então, quando uma instrução causa uma exceção, o *in-order file* contém o estato do registrador obtido na execução de todas as instruções anteriores à instrução que causou a exceção – exatamente o necessário para uma interrupção precisa.

Estes algoritmos são muito importantes pois foram os precursores dos algoritmos implementados para processores como o MIPS 10000, Alpha 21264, HP8000, Pentium II, PowerPC 604, entre outros.

A organização do Algoritmo de Tomasulo está presente na Figura 3 e logo abaixo tem-se um quadro-resumo dos estágios do algoritmo.

O estilo de renomeação de registradores utilizados em processadores como o MIPS R10000 e o Alpha 21264 é apresentado na Figura 4 [10]. Este estilo utiliza banco único de renomeação e de registradores da arquitetura.

O estilo de renomeação de registradores utilizados em processadores como o AMD K7 e K8 é apresentado na Figura 5. Este estilo utiliza ROB e *reservation units*.

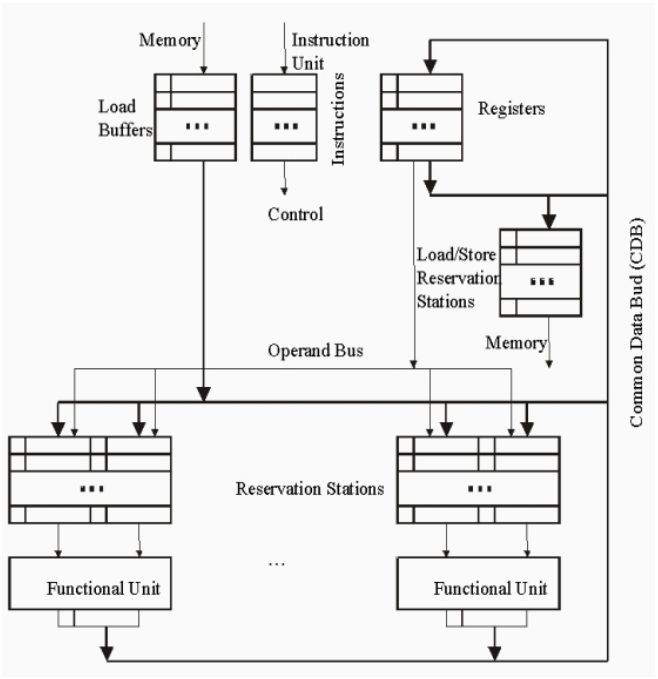

Figure 3: Organização da implementação do Algoritmo de Tomasulo.

O algoritmo de Tomasulo possui três estágios:

1. **Emissão** - retira instrução da Fila de Instruções
   Se a *reservation station* está livre, o algoritmo de Tomasulo emite a instrução e procura os operandos dos registradores, se possível.
2. **Execução** - opera nos operandos
   Quando ambos os operandos estão prontos, então envia a instrução para uma unidade funcional e a executa.
3. **Escrita do resultado** - finaliza a execução
   Escreve no caminho de dados comum para todas as unidades que esperam;
   Marca a *reservation station* como disponível.

Em ambos os esquemas, as instruções são inseridas nas filas na ordem de programa, mas são removidas fora de ordem.

O segundo estilo tem melhor latência de execução, pois o estágio de renomeação obtém os valores nos registradores diretamente, ao invés de determinar o número do registrador físico e depois utilizá-lo para obter o valor. Também, *reservation stations* têm menor latência, pois cada banco de registradores local é menor do que o banco único do primeiro estilo.

Os bancos de registradores físicos utilizados pelas *reservation stations* geralmente unem entradas não utilizadas em paralelo com as filas de instruções escalonadas que elas servem. Isso faz com que o banco de registradores seja maior em conjunto e mais complexo, precisando de mais energia.

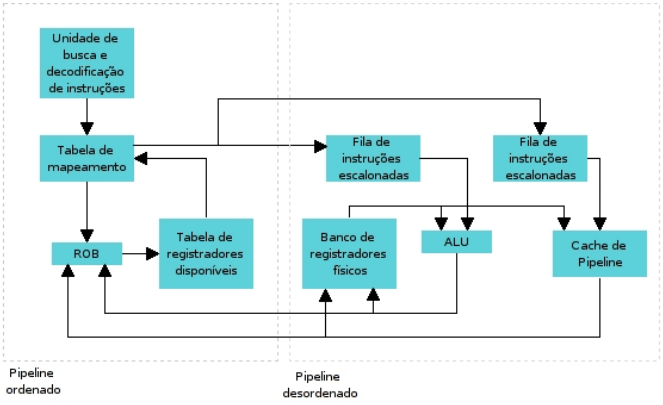

Figure 4: Estilo de renomeação utilizado no MIPS R10000 e Alpha 21264.

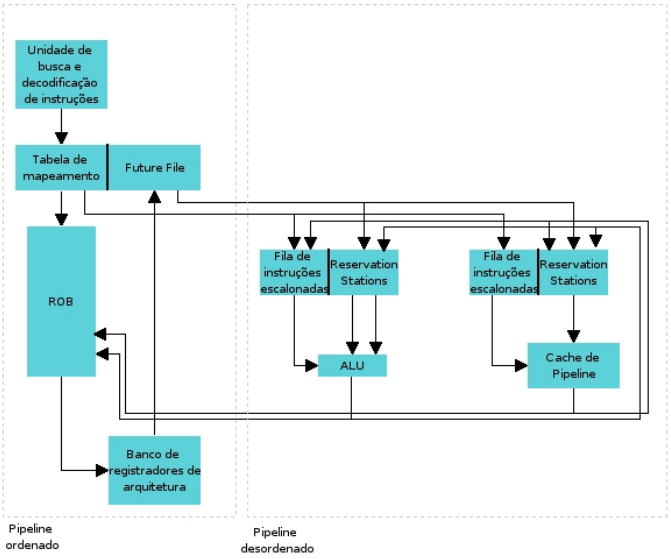

Figure 5: Estilo de renomeação utilizado no projeto do AMD K7 e K8.

Ainda, o segundo estilo possui várias localidades onde o valor do resultado pode ser armazenado (*reservation station*, banco de registradores da arquitetura, ROB,...), e o primeiro estilo tem apenas um (o banco único de registradores). Os resultados das unidades funcionais sofrem *broadcast* para todas as localidades de armazenamento, precisando alcançar um número bem maior de localidades do que no primeiro esquema, consumindo mais energia, área e tempo [10].

4.2 Implementação da arquitetura Metaflow – DRIS

A arquitetura Metaflow implementa execução desordenada e também especulação dinâmica. Muitos dos mecanismos necessários estão integrados na DRIS – *deferred-scheduling, register-renaming instruction shelf*). Nesta implementação, as instruções não são enviadas para *reservation stations* em cada unidade de execução, mas sim para uma estrutura central. Em cada ciclo, essa estrutura centralizada é analisada para se encontrar instruções que podem ser executadas que então são enviadas para unidade de execução apropriada.

A DRIS é uma fila muito similar ao ROB. Quando uma instrução é escalonada, um espaço é alocado no final da DRIS. Quando a instrução é completada, seu resultado é armazenado no espaço alocado. Se a instrução do topo da DRIS terminou sua execução, é tirada da DRIS e o banco de registradores é atualizado com o resultado da instrução.

Cada vez que uma instrução é adicionada à DRIS, a fila é analisada para se encontrar instruções cujos registradores de destino são iguais aos registradores de origem da nova instrução. Se nenhuma é achada, o valor requerido pode ser lido do banco de registradores. Se alguma for encontrada, o registrador de origem é associado com a instrução mais nova. Quando a instrução associada termina, o valor requerido pode ser lido da entrada na DRIS de tal instrução.

O mecanismo difere do *broadcast* do algoritmo de Tomasulo, pois apenas a *tag* (isto é, a entrada na DRIS) sofre *broadcast*. Isso indica que o resultado está disponível. Todas as instruções na DRIS que esperam pelo resultado, as marcam como disponíveis, e então podem ficar prontas. O real valor é lido da entrada da DRIS, quando a instrução é enviada para uma unidade de execução [6].

4.3 Implementação do ponto flutuante do processador RS/6000

O mecanismo de *Register Renaming* no RS/6000 é bem diferente dos discutidos anteriormente, tanto em motivação quanto em implementação. Um dos motivos para remover dependências era permitir que a unidade de ponto-flutuante executasse *loads* de pontos flutuante. Ao invés da unidade de ponto flutuante cuidar das falsas dependências de ponto flutuante, os desenvolvedores escolheram por renomear o registrador de destino do *load* para eliminar as dependências.

Como é implementado, a unidade de ponto flutuante tem mais registradores físicos do que registradores da arquitetura. Existe, então, uma tabela que mapeia os registradores da arquitetura a registradores físicos. Quando uma instrução de ponto flutuante, que não seja um *load*, é decodificada, seus registradores são renomeados de acordo com a tabela. Quando um *load* de ponto flutuante é encontrado, um registrador físico livre é removido da fila e usado como destino do *load*. A tabela é atualizada para refletir o fato de que todas referências subseqüentes ao registrador de destino irão utilizar o registrador físico recentemente alocado [6].

Outras implementações podem ser vistas em [6].

5. HISTÓRICO

Um precursor de *Register Renaming* foi desenvolvido por Tomasulo [9], em 1967, para instruções de ponto flutuante no processador IBM 360/91, um supercomputador escalar pioneiro tanto em *pipeline* como em escalonamento dinâmico de instruções. O processador 360/91 utilizava renomeação para preservar a consistência lógica e não para remover dependências falsas de dados.

Tjaden e Flynn [8] sugeriram o uso de *Register Renaming* para remover dependências falsas de dados, mas para um conjunto limitado de instruções. Porém, eles não utilizaram o nome *Register Renaming*. Keller [4] introduziu essa designação, em 1975, e estendeu a renomeação para todas as instruções que possuem um registrador de destino. Devido à complexidade da técnica, apenas duas décadas depois a técnica começou a ser muito utilizada em processadores superescalares.

Processadores superescalares mais antigos, como HP PA 7100, Sun SuperSparc, DEC Alpha 21064, MIPS R8000 e Intel Pentium, não faziam uso de renomeação. A técnica apareceu gradativamente, primeiro de maneira restrita chamada renomeação parcial (veja subseção 2.2), na década de 90, nos processadores IBM RS/6000 (Power1), Power2, PowerPC 601 e NextGen Nx586 (Figura 6). A renomeação total surgiu posteriormente, começando em 1992, primeiro nos *mainframes* IBM ES/9000 e depois no PowerPC 603. Subseqüentemente, *Register Renaming* passou a ser usado em praticamente todos os processadores superescalares, com exceção notável da linha Sun UltraSparc. Atualmente, *Register Renaming* é considerado uma característica indispensável em processadores superescalares orientados a desempenho [7].

Resumindo, primeiramente a implementação de *Register Renaming* era parcial, restrita a conjunto de instruções particulares. Alguns processadores são Power1 (RS/6000, 1999) que renomeia apenas *loads* de ponto flutuante; Power2 (1993), que estendeu a renomeação a todas outras instruções de ponto flutuante; PowerPC 601 (1993), que renomeia apenas o registrador de *link* e contador; e Nx586 (1994), que possuindo um núcleo de inteiros, renomeia apenas este tipo de instrução.

Após essa fase, os processadores começaram a implementar a renomeação total, compreendendo todas os tipos de instruções elegíveis. Essa linha de processadores superescalares começou com os processadores PowerPC 603 (1993), PA 7200 (1995), Pentium Pro (1995), R10000 (1996), K5 (1995) e MII (1997). As exceções mais notáveis são os processadores Alpha que precedem o Alpha 21264 e a linha Sun UltraSparc.

6. CONCLUSÃO

A figura 7 mostra um resumo dos processadores, o ano de maior produção e o tipo de *buffer* de renomeação.

A maioria dos processadores atuais empregam as seguintes alternativas básicas:

1. Uso de banco único de renomeação e registradores da arquitetura (R10000, R12000, M3);
2. Uso de banco de renomeação isolado e mapeamento de registradores (linha PA8x00, Power3); e
3. Renomeação no ROB e uso de tabelas de mapeamento (Pentium Pro, Pentium II, Pentium III).

Também é concebível usar alternativas diferentes para renomear instruções de inteiros e ponto flutuante, como é feito no K7. Este processador usa o ROB para renomear instruções de inteiro e um banco único para renomeação das instruções de ponto flutuante.

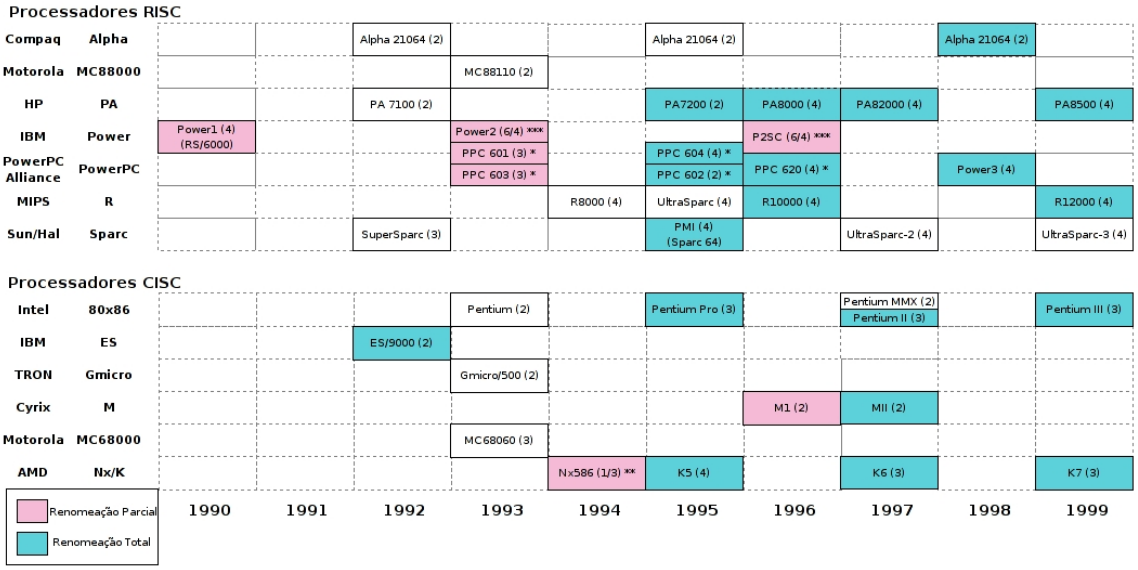

Figure 6: Cronologia de *Register Renaming* em processadores superescalares comerciais. * PPC designa PowerPC; ** O Nx586 tem um escalonamento escalar para instruções CISC, mas um núcleo escalar de 3 vias para instruções RISC convertidas; * A taxa de escalonamento do Power2 é 6 ao longo do caminho seqüencial, enquanto só 4 imediatamente após o *branch*.**

Percebe-se que *Register Renaming* é uma passo importante na evolução de processadores superescalares. A introdução desta técnica se tornou necessária para resolver os problemas de gargalo nos primeiros processadores. Como visto, *Register Renaming* é uma técnica complexa e seu entendimento requer a identificação do espaço de implementação: o escopo de *Register Renaming* e seu *layout*.

Tipo de registrador (ano de maior produção)	Tipo de buffer de renomeação	Número de buffers de renomeação FX	FP
RISC processors			
PowerPC 603 (1993)	Ren. reg. file	N/A	4
PowerPC 604 (1995)	Ren. reg. file	12	8
PowerPC 620 (1996)	Ren. reg. file	8	8
Power3 (1998)	Ren. reg. file	16	24
R10000 (1996)	Merged	32	32
R12000 (1998)	Merged	32	32
Alpha 21264 (1998)	Merged	48	41
PA 8000 (1986)	Ren. reg. file	56	56
PM1 (1996)	Merged	38	24
x86 (CISC) processors			
Pentium Pro (1995)	In the ROB	40	
Pentium II (1997)	In the ROB	40	
K5 (1995)	In the ROB	16	
K6 (1996)	In the ROB	24	
M3 (2000 expected)	Merged	32	N/A

Figure 7: Tipo e número de *buffers* de renomeação em alguns processadores superescalares recentes.

7. REFERÊNCIAS

[1] B. Bishop, T. Kelliher, and M. Irwin. The Design of a Register Renaming Unit. In *Proceedings of Ninth Great Lakes Symposium on VLSI*, page 34, 1999.

[2] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[3] M. Johnson. *Superscalar Microprocessor Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[4] R. M. Keller. Look-Ahead Processors. *Computing Surveys*, 7(4):177–195, Dec. 1975.

[5] T. Monreal, A. González, M. Valero, J. González, and V. Viñals. Dynamic register renaming through virtual-physical registers. *Journal of Instruction-Level Parallelism*, 2, May 2000.

[6] M. Moudgill, K. Pingali, and S. Vassiliadis. Register Renaming and Dynamic Speculation: an Alternative Approach. Technical report, Cornell University, 1993.

[7] D. Sima. The Design Space of Register Renaming Techniques. *IEEE Micro*, 20(5), 2000.

[8] G. S. Tjaden and M. J. Flynn. Detection and Parallel Execution of Independent Instructions. *IEEE Trans. Comput.*, C-19(10):889–895, Oct. 1970.

[9] R. M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal*, 11:25–33, January 1967.

[10] Wikipedia. Register Renaming — Wikipedia, the free encyclopedia, 2005. [Online; acesso em 1-Outubro-2005].