Uma Arquitetura Super Escalar com Múltiplos Fluxos de Instruções

Eliseu Monteiro Chaves Filho
Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ
Caixa Postal 68511, 21945-970 Rio de Janeiro, RJ
eliseu@cos.ufrj.br

Alberto Ferreira de Souza
Departamento de Informática, UFES
Av. Fernando Ferrari s/n, 29060-900 Vitória, ES
alberto@inf.ufes.br

Anna Dolejsi Santos
Departamento de Ciência da Computação, UFF
Pça. do Valonguinho s/n, 24210-130 Niterói, RJ
annads@dcc.uff.br

Rafael Santos
Instituto de Informática, UFRGS
Caixa Postal 15064, 91501-970 Porto Alegre, RS
rrsantos@inf.ufrgs.br

Resumo

Mecanismos sofisticados para escalonamento dinâmico de instruções, previsão dinâmica de desvios e execução especulativa são encontrados em arquiteturas super escalares destinadas à aplicações que exigem alto desempenho. No entanto, estas arquiteturas ainda apresentam um desempenho bem inferior ao que seria alcançado por uma arquitetura super escalar ideal. Este artigo apresenta resultados experimentais que identificam algumas das principais causas da discrepância entre os desempenhos de uma arquitetura super escalar real e de uma ideal. Também é apresentado um novo mecanismo, baseado em múltiplos fluxos de instruções, que tem como objetivo reduzir as limitações encontradas.

Abstract

In order to meet the performance demands of high-end applications, some superscalar architectures incorporate mechanisms for dynamic instruction scheduling, dynamic branch prediction and speculative execution. Nevertheless, the performance delivered by such architectures is still well behind the limit that would be achieved by an ideal architecture. This paper identifies some of the main factors leading to this performance gap and presents a new mechanism to reduce such shortcomings. This mechanism is based on the concept of multiple instruction streams.

1 Introdução

Arquiteturas super escalares [1,2] constituem atualmente o principal paradigma de exploração do paralelismo a nível de instrução. Processadores super escalares diferem nos mecanismos adotados para tratar as *dependências de dados* e as *dependências de controle* [3]. Alguns processadores, como por exemplo o PowerPC 604 [4] e 620 [5] e o MIPS R10000 [6], possuem mecanismos que realizam um *escalonamento dinâmico de instruções*, permitindo a execução de instruções subseqüentes enquanto a dependência é resolvida.

Em processadores como o Intel Pentium [7] e o DEC Alpha 21164 [8], as dependências de controle são resolvidas bloqueando-se o despacho de novas instruções até que a instrução de desvio seja finalizada. A busca de instruções continua, sendo direcionada por um mecanismo de *previsão de desvios*: quando a instrução de desvio é acessada, este mecanismo prevê o resultado do desvio (*tomado*, ou *não-tomado*) e a busca das próximas instruções prossegue ao longo do caminho previsto. Caso a previsão se revele incorreta, estas instruções são descartadas. Outros processadores, como por exemplo o Intel P6 [9] e o PowerPC 604 e 620, suportam *execução especulativa* de instruções. Nestes, o despacho de instruções não é bloqueado quando um desvio é encontrado. Ao contrário, as instruções ao longo do caminho previsto do desvio são despachadas e executadas condicionalmente. Os resultados produzidos por estas instruções alteram o estado da arquitetura apenas se a previsão for correta; se a previsão for incorreta, tais resultados são descartados.

Atualmente, mecanismos sofisticados de escalonamento dinâmico, previsão de desvios e execução especulativa, são encontrados em processadores super escalares destinados à aplicações que exigem alto desempenho. No entanto, verifica-se que o desempenho destes processadores ainda está bem abaixo daquele que seria alcançado por uma máquina ideal. Esta constatação motiva pesquisas no sentido de buscar-se novos mecanismos que levem a processadores super escalares com níveis mais elevados de desempenho.

Este artigo apresenta um mecanismo para reduzir uma das principais deficiências encontradas nas atuais arquiteturas super escalares. Inicialmente, é apresentada a motivação para este trabalho, comparando o desempenho de uma arquitetura super escalar típica (que será denominada *arquitetura real*) com o desempenho de uma arquitetura super escalar ideal. Em seguida, são identificadas algumas limitações na arquitetura real que provocam a diferença entre os desempenhos das arquiteturas real e ideal. Com base nestes resultados, sugere-se uma arquitetura que tem como objetivo minimizar estas deficiências.

2 Descrição das Arquiteturas Super Escalares

Na arquitetura super escalar real foram incluídas várias características encontradas em processadores super escalares comerciais. Na resolução de dependências de dados foi adotado um mecanismo de escalonamento dinâmico de instruções baseado no *algoritmo de Tomasulo* [10]. Dependências de controle são tratadas com um mecanismo de execução especulativa. Os mecanismos adotados no tratamento das dependências de dados e de controle são similares aos encontrados no PowerPC 604 e 620. As instruções são executadas em um *pipeline* super escalar com sete estágios: *Busca* (B), *Previsão* (P), *Decodificação* (D), *Despacho* (DP), *Início* (I), *Execução* (X) e *Resultado* (R).

O estágio B acessa instruções e as coloca no *buffer de busca*. O número máximo de instruções acessadas em um mesmo ciclo é denominado *largura de busca*. O estágio P verifica se alguma das instruções no *buffer* de busca é uma instrução de desvio. Ao detectar uma instrução deste tipo, este estágio faz uma previsão do resultado do desvio. Se o resultado é previsto como *tomado*, o estágio P calcula o endereço destino e redireciona o estágio B para este endereço. O resultado das instruções de desvio condicional é previsto dinamicamente, usando-se uma *tabela de história de desvios* (ou *branch history table*, [11]) idêntica à do Intel Pentium. O estágio P transfere as

instruções do *buffer* de busca para a *fila de instruções*. Se existir alguma instrução de desvio com resultado previsto *tomado*, as instruções posteriores no *buffer* de busca são descartadas durante a transferência (pois, segundo a previsão, tais instruções encontram-se no caminho incorreto).

O estágio D decodifica instruções armazenadas na fila de instruções, enquanto o estágio DP despacha instruções que já foram decodificadas. O número máximo de instruções despachadas em um mesmo ciclo é chamado *largura de despacho*. Instruções são despachadas para as *estações de reserva*, associadas às unidades funcionais. As estações de reserva armazenam o código de operação e os operandos das instruções a serem executadas na unidade funcional. No despacho de instruções, o estágio DP segue a operação típica do algoritmo de Tomasulo, descrita em [10]. O despacho obedece a ordem estática das instruções.

O estágio I controla o início da execução das instruções armazenadas nas estações de reserva. Este estágio escalona instruções dinamicamente: o início da execução de uma certa instrução depende apenas da disponibilidade de recursos (unidade funcional livre) e dos seus operandos, e independe da ordem estática da instrução em relação às demais. As unidades funcionais formam o estágio X. As unidades funcionais são homogêneas, ou seja, todas as unidades podem executar qualquer tipo de instrução.

O estágio R envia os resultados produzidos pelas instruções, através dos *barramentos de resultados*, para as estações de reserva e para o conjunto de registradores futuros. Após a escrita de resultados, o estágio I verifica quais as instruções que possuem todos os operandos disponíveis nas estações de reserva, e inicia a execução das instruções prontas de acordo com a disponiblidade das unidades funcionais.

Para suportar execução especulativa de instruções, foram incluídas na arquitetura uma *fila de reordenação* [12] e um segundo conjunto de registradores, denominados *registradores reais*. O primeiro conjunto de registradores, chamados *registradores futuros*, fornecem os operandos das instruções no momento do despacho e armazenam resultados de instruções executadas especulativamente. Os registradores reais guardam o estado correto da arquitetura. A fila de reordenação serve para garantir que registradores reais sejam atualizados na mesma ordem que as instruções foram despachadas.

Ao despachar uma instrução, o estágio DP também aloca uma entrada na fila de reordenação. Quando a instrução é completada, o estágio R envia o resultado para esta mesma entrada na fila de reordenação. O estágio R atualiza os registradores reais com o valor armazenado na entrada somente quando esta atinge a frente da file de reordenação. Note que, no momento que uma instrução de desvio alcança a frente da fila de reordenação, os registradores reais guardam um estado modificado apenas pelas instruções despachadas antes do desvio. Se o desvio foi previsto incorretamente, basta transferir o conteúdo dos registradores reais para os registradores futuros para cancelar as modificações das instruções executadas especulativamente.

O paralelismo a nível de instrução é fundamentalmente limitado pelas dependências de dados. No entanto, as dependências de controle limitam a eficiência do escalonamento dinâmico. Uma arquitetura ideal seria então aquela com um mecanismo de escalonamento dinâmico eficiente, e capaz de eliminar completamente as dependências de controle (supondo-se, adicionalmente, recursos ilimitados). Infelizmente esta arquitetura não é realizável, porque para tanto seria necessário o conhecimento prévio do resultado de cada instrução de desvio encontrada durante a execução do programa. No entanto, fica a pergunta de qual seria o limite máximo teórico de desempenho estabelecido pela arquitetura ideal, e o quão próximo deste limite encontram-se as arquiteturas reais.

Para responder esta questão, foi considerada uma arquitetura ideal com os mesmos estágios da arquitetura real, com exceção do estágio P. Ao acessar uma instrução de desvio, o estágio B determina o resultado correto (detalhes na Seção 3) e, no mesmo ciclo, passa a buscar as instruções no destino do desvio (mantido o valor da largura de busca). Desta forma, instruções são sempre acessadas ao longo dos caminhos corretos dos desvios, sendo sempre mantido um fluxo contínuo de instruções através do *pipeline*. As instruções acessadas são colocadas diretamente na fila de instruções, onde são decodificadas pelo estágio D. Na arquitetura ideal, foi mantido

o mesmo mecanismo de escalonamento dinâmico de instruções usado na arquitetura real. Os estágios DP, I e R operam da mesma forma na resolução de dependências de dados, com exceção das operações envolvendo a fila de reordenação e o conjunto de registradores reais. Como na arquitetura ideal não há necessidade de correções de estado, estas duas estruturas não existem.

3 O Ambiente Experimental

Os experimentos foram realizados com simuladores que reproduzem a operação das arquiteturas super escalares real e ideal. Estes simuladores são do tipo *trace-driven*, e usam um arquivo de história (*trace*) gerado por um simulador da arquitetura SPARC [13]. O simulador SPARC gera um arquivo com a história das instruções de desvio. Este arquivo contém, para cada instrução de desvio executada, um registro indicando o resultado do desvio e o endereço destino do desvio. O simulador da arquitetura ideal usa a história para determinar o endereço destino de um desvio: quando uma instrução de desvio é acessada, o registro correspondente é lido e o estágio B continua com o acesso de instruções a partir do endereço indicado no registro.

Como programas de teste, foram usados seis programas de computação inteira. Quatro deles pertencem ao conjunto SPECint92 [14] (`espresso`, `eqntott`, `compress`, `gcc`), e os dois outros são utilitários UNIX (`bison`, `gas`). Estes programas foram compilados com o GNU gcc 2.7.1 para o sistema operacional SunOS 4.1.4. Os arquivos de história usados nos experimentos cobrem a execução de 10 milhões de instruções de cada programa.

4 Resultados Experimentais

Os resultados apresentados nesta seção foram obtidos com configurações idênticas das arquiteturas real e ideal. Estas configurações apresentam os seguintes valores de parâmetros: largura de busca de oito instruções; *buffer* de busca e fila de instruções com 12 e 24 posições, respectivamente; largura de despacho de oito instruções; número de unidades funcionais variando entre duas e oito unidades, cada uma delas com oito estações de reserva; oito barramentos de resultados, e fila de reordenação com 32 posições.

Os gráficos na Figura 1 mostram o *speedup* das arquiteturas real e ideal, em função do número de unidades funcionais. O *speedup* foi calculado como a razão entre o número de ciclos gastos na execução pelo simulador da arquitetura SPARC e o número de ciclos consumidos pelo simulador de cada arquitetura super escalar.

Os gráficos revelam uma diferença significativa entre os desempenhos das duas arquiteturas. As curvas para a arquitetura ideal sugerem que aumentos adicionais no desempenho seriam obtidos em configurações com mais de oito unidades funcionais. Isto acontece porque, mesmo nas configurações com oito unidades funcionais, o despacho ainda é limitado pela disponibilidade de recursos. Na arquitetura ideal, a porcentagem de ciclos nos quais nenhuma instrução é despachada pela falta de recursos é 43% em configurações com duas unidades funcionais, diminuindo para 32% nas configurações com oito unidades.

O *speedup* apresentado pela arquitetura ideal deve-se ao elevado número de *instruções executadas por ciclo* (ipc). Para configurações da arquitetura ideal com oito unidades funcionais, o ipc médio é 3,0, enquanto na arquitetura real o ipc cai para 1,4. Os fatores fundamentais que limitam o número de instruções executadas em paralelo são (1) as dependências de dados e (2) a disponibilidade de recursos. O ipc da arquitetura ideal reflete estes dois fatores.

Um fator adicional que contribui para o menor desempenho da arquitetura real é a descontinuidade no despacho de instruções, provocada pelas dependências de

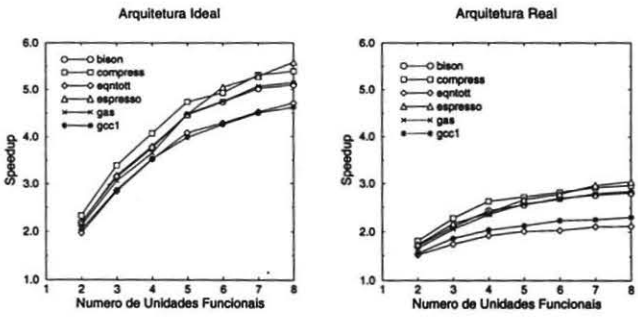


Figura 1: *Speedups das arquiteturas ideal e real.*

controle. Na arquitetura ideal, instruções são despachadas continuamente para as estações de reserva, mantendo a disponibilidade de instruções independentes que podem ser executadas em paralelo. No entanto, na arquitetura real, as dependências de controle quebram o fluxo, tornando freqüente a inexistência de instruções para serem executadas enquanto dependências de dados são resolvidas.

Este efeito pode ser avaliado através da proporção de *ciclos com despacho nulo*, i.e., ciclos nos quais nenhuma instrução é despachada para as estações de reserva. Ciclos com despacho nulo também ocorrem na arquitetura ideal, mas devido unicamente à falta de recursos. A Figura 2(a) mostra a porcentagem de ciclos com despacho nulo nas arquiteturas ideal e real, em função do número de unidades funcionais. Na arquitetura ideal, o número de ciclos com despacho nulo decresce à medida que são acrescentados mais recursos, indicando que a impossibilidade de despacho é dominada por este fator. No entanto, na arquitetura real, a proporção de ciclos com despacho nulo permanece praticamente constante quando são acrescentados mais recursos, o que indica a existência de um outro fator que limita mais severamente o despacho.

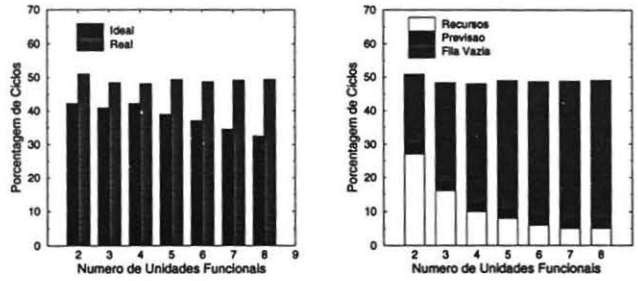


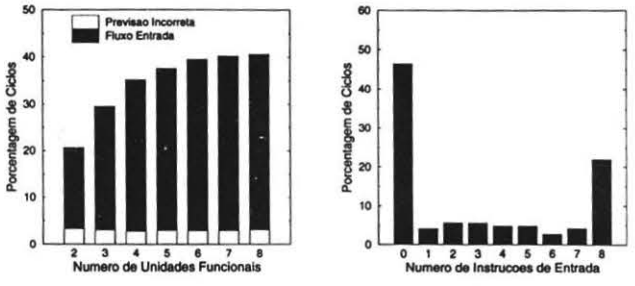
Figura 2: *Proporção de ciclos com despacho nulo e dos eventos que os provocam.*

Na arquitetura real, ciclos sem despacho de instruções podem acontecer devido aos seguintes fatores: (1) não há disponibilidade de recursos, (2) o despacho está

bloqueado porque uma instrução de desvio foi prevista incorretamente ou (3) a fila de instruções encontra-se vazia. A Figura 2(b) mostra a freqüência de ocorrência de cada um destes eventos na arquitetura real, em função do número de unidades funcionais. A contribuição para o despacho nulo proveniente da indisponibilidade de recursos diminui significativamente com o acréscimo de unidades funcionais, caindo de 27,16%, nas configurações com duas unidades funcionais, para 5,35% em configurações com oito unidades. A contribuição do bloqueio do despacho provocado pelas previsões incorretas é pequena, mantendo-se em torno de 3,0% para as configurações consideradas. O gráfico mostra que o esvaziamento da fila é o principal fator que contribui para ciclos com despacho nulo. A proporção dos ciclos nos quais a fila de instruções encontra-se vazia cresce de 20,63%, em configurações com duas unidades funcionais, para 40,95% nas configurações com oito unidades.

Um fator que resulta no esvaziamento da fila é a previsão incorreta de desvios. Como descrito na Seção 2, ao ser encontrada uma instrução de desvio prevista incorretamente, o despacho é bloqueado e o conteúdo da fila de instruções e do *buffer* de busca é descartado. Um segundo fator para o esvaziamento da fila seria um volume de entrada de instruções insuficiente, provocando um desbalanceamento entre a taxa com que o estágio P coloca instruções na fila, e a taxa com que o estágio DP retira instruções da fila para o despacho.

A Figura 3(a) mostra a contribuição destes dois eventos para o esvaziamento da fila de instruções. O gráfico revela que, na grande maioria dos casos, a fila encontra-se vazia devido à escassa entrada de instruções. Isto decorre das constantes quebras do fluxo de instruções, provocadas pelas dependências de controle. Conforme a descrição da arquitetura real na Seção 2, quando o resultado de uma instrução de desvio é previsto como *tomado*, apenas as instruções que se encontram antes do desvio (no *buffer* de busca) são transferidos para a fila de instruções. Adicionalmente, o estágio B é redirecionado para o destino previsto do desvio, e novas instruções são colocadas no *buffer* de busca somente dois ciclos após aquele no qual a instrução de desvio foi prevista.



(a) (b)

Figura 3: *Contribuição para esvaziamento da fila e perfil de entrada de instruções.*

O gráfico na Figura 3(b) mostra mais precisamente o impacto das dependências de controle sobre o volume de entrada de instruções. Ele apresenta a proporção de ciclos nos quais um certo número de instruções é transferido do *buffer* de busca para a fila de instruções. Na maioria dos ciclos, nenhuma instrução é transferida do *buffer* de busca para a fila de instruções. Em apenas 23% dos ciclos a largura de busca completa é transferida para a fila de instruções. Os valores abaixo da largura de busca resultam dos desvios previstos como *tomado*.

Os resultados aqui apresentados sugerem que a taxa de acerto não é o único fator que determina a eficácia de um mecanismo de previsão dinâmica de desvios.

Uma alta taxa de acertos não impede que o fluxo de entrada de instruções seja quebrado, nos casos em que o desvio é previsto como sendo *tomado*. Nos programas de teste utilizados, entre 61% e 69% dos desvios são previstos como *tomado*. A alta freqüência de instruções de desvio, aliada à proporção de desvios previstos como tomados, faz com que o volume de instruções entrando na fila de instruções seja pequeno, como indica o gráfico na Figura 3(b). O resultado final é o desbalanceamento entre a taxa de entrada e de saída da fila de instruções, e os excessivos ciclos com despacho nulo.

Aliado a uma alta taxa de acertos na previsão, é necessário que os mecanismos de busca e previsão sejam elaborados de forma a possibilitar um maior fluxo de entrada de instruções, mesmo na presença de desvios previstos como tomados. Na seção seguinte, é apresentado um mecanismo para atender este requisito.

5 Arquitetura com Múltiplos Fluxos de Instruções

Na arquitetura real, o volume de entrada de instruções é limitado por dois motivos. Primeiro, quando um desvio é previsto como sendo *tomado*, apenas as instruções anteriores são transferidas para a fila de instruções (o gráfico na Figura 3(b) mostra a distribuição destas transferências parciais). Segundo, quando um desvio é previsto como *tomado*, as instruções acessadas naquele ciclo devem ser simplesmente descartadas, e um ciclo adicional é necessário para acessar as instruções no destino do desvio. Assim, durante dois ciclos, o *buffer* de busca permanece vazio (o gráfico em 3(b) mostra que tal evento acontece com bastante freqüência). Isto acontece porque a busca de instruções e a previsão de desvios são realizadas em paralelo, por estágios diferentes do *pipeline*. Tal esquema é comum em processadores reais, para não aumentar o tamanho do ciclo do processador.

A operação desejável seria aquela onde, a cada ciclo, *largura de busca* instruções fossem sempre transferidas para a fila de instruções, mesmo quando um desvio fosse previsto como *tomado*. Além disso, novas instruções seriam colocadas no *buffer* de busca no ciclo imediatamente seguinte a uma previsão. Aliado a um mecanismo de previsão de desvios com alta taxa de acertos, este funcionamento resultaria em um fluxo de entrada na fila instruções com volume próximo ao observado na arquitetura ideal. Para obter-se esta operação, são necessárias as seguintes modificações na arquitetura real descrita na Seção 2:

(1) uma instrução de desvio passa a ser detectada no próprio estágio B e, no ciclo seguinte, é iniciada a busca das instruções nos dois possíveis caminhos do desvio. Para tanto, contadores de programa independentes são incluídos no estágio B, permitindo o acesso simultâneo das instruções seqüenciais e das instruções no destino do desvio. Com esta modificação, procura-se evitar os ciclos com *buffer* de busca vazio.

(2) quando um desvio é previsto como *tomado*, a transferência para a fila de instruções não é interrompida após a instrução de desvio. O estágio P continua a transferência, realizando um *encadeamento* das instruções do caminho previsto. Com esta modificação, procura-se aumentar o número médio de instruções transferidas para a fila de instruções a cada ciclo.

A detecção de instruções de desvio no estágio de busca pode ser implementada sem prejudicar o tamanho do ciclo do processador. Uma das possíveis maneiras é através de uma codificação criteriosa do código de operação das instruções, de forma que um único bit indique se a instrução é de desvio. O cálculo do endereço-destino, que (nos conjuntos de instruções RISC) requer apenas uma extensão do sinal seguida de uma soma, pode ser efetuado em paralelo com o incremento do contador de programa. Isto requer o acréscimo de um circuito dedicado, mas simples. O próprio cálculo do endereço-destino pode ser feito condicionalmente, em paralelo com a verificação do tipo da instrução.

Para que seja possível o encadeamento das instruções no destino de um desvio previsto *tomado*, é necessário que tais instruções já estejam disponíveis. Para tanto, foi introduzido um estágio adicional no *pipeline*, entre os estágios B e P, conforme mostra a Figura 4.

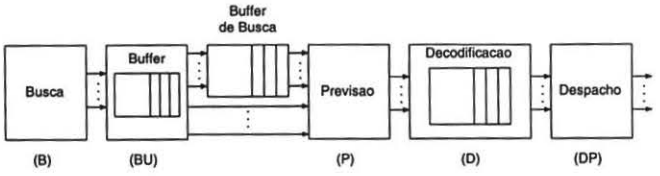


Figura 4: *Pipeline modificado para permitir encadeamento de instruções.*

O estágio denominado *Buffer* (BU) simplesmente introduz um ciclo de intervalo entre o acesso de uma instrução pelo estágio B e sua previsão no estágio P. Durante este ciclo, o estágio B acessa as instruções no destino do desvio e as coloca em uma fila no estágio BU. Para melhor entender a operação do mecanismo proposto, considere o cenário apresentado na Figura 5.

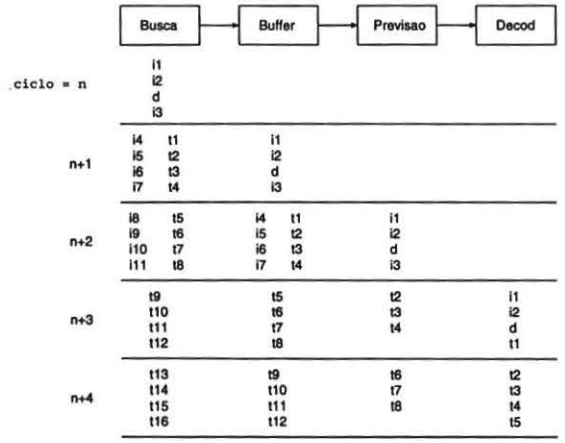


Figura 5: *Encadeamento das instruções no destino de um desvio.*

Neste exemplo, a largura de busca é quatro instruções. No ciclo n, o estágio B acessa uma instrução de desvio (d). No ciclo n+1, o estágio B continua buscando as instruções seqüenciais e inicia a busca das instruções no destino do desvio (t_1, t_2, ...). Ainda no ciclo n+1, a instrução de desvio passa para o estágio BU. No ciclo n+2, a instrução de desvio chega ao estágio P e as instruções nos dois caminhos são transferidas para o estágio BU. Suponha que o desvio é previsto como *tomado*. No ciclo n+3, o estágio P transfere para o estágio D as instruções i_1, i_2, d e, a partir do estágio BU, a instrução t_1, completando a largura de transferência de quatro instruções. As instruções t_2, t_3, t_4 passam para o estágio P. No ciclo n+4 estas

instruções, juntamente com a instrução t_5 encadeada a partir do estágio BU, são transferidas para o estágio P.

Através deste exemplo, verifica-se que o número médio de instruções transferidas para o estágio D, na presença de um desvio previsto *tomado*, é maior que no modo de operação original da arquitetura, descrito na Seção 2. Além disso, não ocorre a quebra no fluxo de instruções quando o desvio é previsto como *tomado*. As instruções no destino do desvio são acessadas antecipadamente, estando disponíveis quando o desvio é previsto.

Neste mecanismo, o estágio B sempre inicia a busca das instruções ao longo dos dois possíveis caminhos de um desvio. Cada um destes caminhos constitui um fluxo de instruções diferente. Por isso, a arquitetura com este mecanismo foi denominada *arquitetura com múltiplos fluxos de instruções* (ou, *arquiteturas multifluxo*).

A eficiência do mecanismo com múltiplos fluxos de instruções depende da taxa de acertos na previsão de desvios. A taxa de acertos seja alta, para que o encadeamento de instruções realmente consiga manter a continuidade da entrada de instruções. Um mecanismo de previsão dinâmica de desvios em dois níveis [15] pode ser empregado para assegurar a taxa de acertos adequada.

6 Conclusões e Trabalhos Futuros

Este trabalho mostrou que o desempenho de uma arquitetura super escalar que incorpora mecanismos sofisticados tais como escalonamento dinâmico e execução especulativa ainda é significativamente menor que o limite estabelecido por uma arquitetura ideal. Investigando os motivos que levam a esta diferença, verificou-se que em uma arquitetura real quase metade dos ciclos constituem ciclos com despacho nulo, nos quais nenhuma instrução é enviada para as unidades funcionais.

A principal causa dos ciclos com despacho nulo está no fato que o fluxo de entrada de instruções é constantemente interrompido e tem seu volume reduzido pelos desvios previstos como *tomado*. Em uma arquitetura com execução especulativa de instruções, esta não é a única origem dos ciclos com despacho nulo. Estes ciclos também são provocados pelo bloqueio do despacho e pelo descarte do conteúdo da fila de instruções, que acontecem quando é encontrado um desvio previsto incorretamente. No entanto, os resultados mostram que a limitação na entrada de instruções é o fator predominante.

Para reduzir estas limitações, foi apresentada uma arquitetura capaz de acessar múltiplos fluxos de instruções simultaneamente. Quando uma instrução de desvio é encontrada, o estágio de busca inicia imediatamente o acesso das instruções nos dois possíveis caminhos do desvio. Caso o desvio seja previsto como *tomado*, as instruções no destino do desvio, que foram acessadas antecipadamente, são encadeadas para os estágios seguintes do *pipeline*. Desta forma o fluxo de entrada de instruções não é descontinuado, e o número médio de instruções transferidas para despacho aumenta.

A arquitetura multifluxo apresentada pode ser estendida de duas formas. A primeira delas consiste em acrescentar um segundo *buffer* de busca, antes do estágio de previsão. Este *buffer* adicional armazenaria as instruções pertencentes ao fluxo não escolhido pelo mecanismo de previsão. No caso de uma previsão incorreta, tais instruções estariam imediatamente disponíveis. O ganho obtido com esta modificação depende da taxa de acertos do mecanismo de previsão. A segunda extensão consiste em executar, especulativamente, os dois fluxos originários de uma instrução de desvio. Na especulação ao longo de um único fluxo, ciclos são desperdiçados quando um desvio é previsto incorretamente. Executando os dois fluxos, as instruções corretas seriam executadas em paralelo com as instruções incorretas, reduzindo o desperdício de ciclos. As vantagens decorrentes desta extensão dependem do peso dos ciclos inúteis dentro do tempo total de execução.

Atualmente, está sendo construído um simulador da arquitetura multifluxo, com o qual será avaliada a eficácia do mecanismo apresentado. Após a avaliação do mecanismo básico, serão consideradas as extensões mencionadas. Trabalhos futuros também incluem a concepção em VLSI da unidade de instruções da arquitetura, permitindo uma avaliação mais precisa da viabilidade, custo e desempenho de um

processador com arquitetura multifluxo. Os resultados destes estudos serão publicados em outros artigos.

Agradecimentos

Este trabalho foi realizado no contexto do Projeto MULFLUX, patrocinado pelo CNPq através do programa ProTem III (Processo Institucional No. 680096/95.7). Os autores agradecem ao CNPq pelo suporte recebido para a realização desta pesquisa.

Referências

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

[2] Smith, J. E., Sohi, G., *The Microarchitecture of Superscalar Processors*, Proceedings of the IEEE 83(12), December 1995, pp. 1609-1624.

[3] Hennessy, J. L., D. A. Patterson, *Computer Architecture, A Quantitative Approach, 2nd Edition*, Morgan Kauffman, San Francisco, CA, 1995.

[4] Song, S. P., M. Denman, J. Chang, *The PowerPC 604 RISC Microprocessor*, IEEE Micro (14)5, October 1994, pp. 8-17.

[5] Diep, T. A., C. Nelson, J. P. Shen, *Performance Evaluation of the PowerPC 620 Microarchitecture*, Proc. of the 22th International Symposium on Computer Architecture, 1995, pp. 163-175.

[6] MIPS Inc., *The R10000 Microprocessor User's Manual*, 1995.

[7] Alpert, D., D. Avnon, *Architecture of the Pentium Microprocessor*, IEEE Micro (13)3, June 1995, pp. 11-21.

[8] Edmondson, J. H. *et al.*, *Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-Issue CMOS RISC Microprocessor*, Digital Technical Journal (7)1, 1995, pp. 119-135.

[9] Halfhill, T. R., *Intel's P6*, Byte (20)4, April 1995, pp. 42-58.

[10] Tomasulo, R. M., *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*, IBM Journal of Research and Development (11)1, January 1967, pp. 25-33.

[11] Lee, J. K. F., A. J. Smith, *Branch Prediction Strategies and the Branch Target Buffer Design*, IEEE Computer (17)1, September 1980, pp. 261-294.

[12] Smith, J. E., A. R. Pleszkun, *Implementing Precise Interrupts in Pipelined Processors*, IEEE Transactions on Computers (37)5, May 1988, pp. 562-573.

[13] Sun Microsystems, *The SPARC Architecture Manual, Version 7*, Mountain View, CA, 1987.

[14] SPEC Steering Committee, *SPEC INT92 V1.1 Technical Manual*, 1992.

[15] Yeh, T.-Y., Patt, Y., *Two-Level Adaptative Training Branch Prediction*, Proc. of the 24th Annual International Symposium on Microarchitecture, 1991, pp. 51-61.