Implementação de um Preditor de Desvio no MIPS 5 Estágios

Hector Perez Baranda, Jeronimo Costa Penha, Ricardo Ferreira
Universidade Federal de Viçosa, Viçosa - MG CEP 36570-900
Departamento de Informática
ticolotico@gmail.com, jeronimopenha@gmail.com, ricardo@ufv.br

Abstract

Apesar da predição de desvios ter um impacto importante no desempenho dos processadores, os livros de arquitetura de computadores não apresentam detalhes de sua implementação. Este trabalho propõe um preditor de desvios para o processador MIPS 5 estágios seguindo a sequência didática do livro "Organização e Projeto de Computadores" dos autores Hennessy e Patterson. Este artigo apresenta o conceito, a modelagem, a enumeração dos casos e a implementação simplificada com adição de apenas duas unidades e algumas conexões que são inseridas no caminho de dados do MIPS pipeline 5 estágios. A proposta serve de base para motivar o desenvolvimento de extensões e implementações de outros preditores, além de reforçar os conceitos sobre o tema.

1. Introdução

A predição de desvios tem um impacto direto no desempenho dos processadores superescalares. Por exemplo, o processador i7 da Intel tem um pipeline com 14 estágios e busca 4 instruções por ciclo, ou seja, no melhor caso poderá ter até 56 instruções em execução. Entretanto, devido às dependências de dados, às falhas de acessos à cache e à presença de desvios, o desempenho de 4 instruções por ciclo é reduzido para a faixa de 0,5 a 2,6 instruções por ciclo para um conjunto de 12 benchmarks com números inteiros do SPEC2006 em um processador I7 920 da Intel [1].

A sequência didática do livro clássico de Hennessy e Patterson [1] inclui o MIPS básico, o MIPS pipeline, a unidade de encaminhamento (*forward*), a unidade de *hazard* e finalmente as instruções de desvio, mas sem considerar a predição dinâmica no caminho de dados. Em cada etapa, o caminho de dados é detalhado, permitindo ao estudante a compreensão e a implementação do mesmo. Por exemplo, a unidade de controle é detalhada no projeto do MIPS básico com uma tabela ilustrando o processo de decodificação e controle, onde são enumeradas as instruções de *load, store, beq* e 6 instruções aritméticas. Em seguida, o conceito de pipeline é introduzido bem como a propagação dos sinais de controle pelos estágios. A unidade de encaminhamento é o próximo passo. Ela é inserida no caminho de dados juntamente com dois multiplexadores nas entradas da unidade lógica aritmética (ALU). As claúsulas condicionais para implementação da estrutura de controle da unidade de encaminhamento também são apresentadas. Posteriormente, a unidade de *hazard* para resolver a dependência de dados geradas pela instrução *load* é apresentada. Neste caso, se ocorrer dependência de dados, será inserida uma bolha na pipeline. Finalmente, a resolução de desvios com a abordagem estática de inserção de bolha no pipeline é adicionada ao caminho de dados. Na sequência, apenas os conceitos da predição dinâmica são comentados mas não são apresentados detalhes de implementação. A máquina de estados de um preditor de 2 bits é ilustrada mas não é apresentado nenhum conteúdo de como a máquina pode ser inserida no caminho de dados.

Este artigo segue a sequência e a metodologia do livro [1] ao inserir uma implementação da predição de desvios, logo após a apresentação da solução estática com inserção da bolha. Primeiro, iremos apresentar uma analogia com um envelope de uma carta que será usado para fixar o conceito da predição e o acesso às informações nas etapas de busca e decodificação. A analogia é apresentada na Seção 2. Na Seção 3, apresentamos as decisões que são tomadas e os recursos adicionados ao caminho de dados. O objetivo é mostrar o primeiro passo para a implementação do preditor com uma unidade de predição simplificada e uma tabela de histórico dos desvios. A Seção 4 apresenta, caso a caso, as situações que são resolvidas no estágio de decodificação que envolvem o cadastro da instrução de desvio e a validação da predição. Nesta seção também finalizamos o caminho de dados com o preditor completo. Na Seção 5 apresentamos a validação da proposta que foi adicionada a um simulador didático do MIPS [2]. Na Seção 6 são apresentadas sugestões das extensões ao preditor básico e outras abordagens para ensino da predição. A Seção 7 sugere um exercício de implementação de um preditor de 2

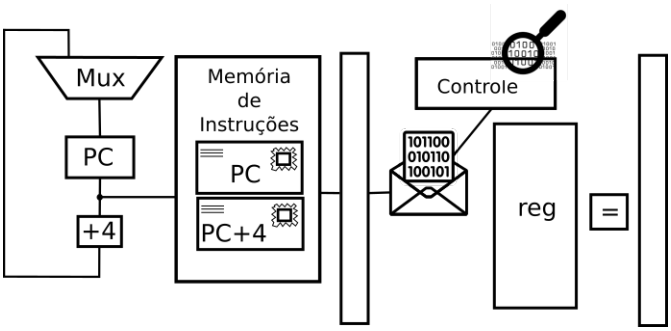

Figura 1. Etapas de busca e decodificação: o endereço e o conteúdo do envelope

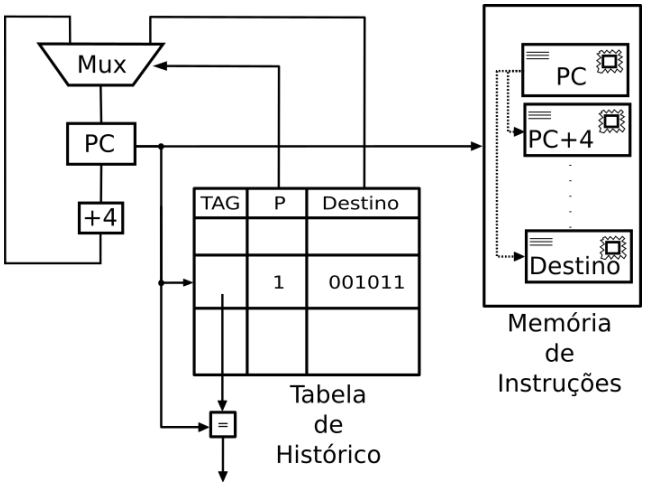

Figura 2. A tabela de histórico das predições e o registrador PC

bits. Finalmente, as considerações finais são apresentadas na Seção 8.

2. Busca e Decodificação

Diferente do encaminhamento que é resolvido no terceiro estágio e o *hazard* do *load* que é resolvido no segundo estágio, o desvio é o mais crítico pois tem que ser resolvido na primeira etapa do pipeline. No estágio de busca não sabemos o tipo da instrução, portanto não sabemos se é uma instrução de desvio e também não sabemos se o desvio ocorrerá ou não.

A Figura 1 ilustra as informações disponíveis nos estágios de busca e decodificação através de uma analogia simples com uma carta em um envelope. Suponha que toda instrução é uma carta. No estágio de busca só temos acesso ao endereço (PC) do envelope, uma vez que o envelope está lacrado e não sabemos o conteúdo (instrução que está dentro). Portanto, a predição deve ser realizada usando apenas o endereço. Na Seção 3 iremos detalhar como o endereço PC é utilizado para resolver a predição no estágio de busca.

No estágio de decodificação temos acesso às informações dentro do envelope, ou seja, abrimos o envelope. Sabemos qual é a instução. Neste momento, a unidade de controle realiza a decodificação e o processo de execução da instrução é iniciado com a leitura dos registradores. No caso específico da instrução de desvio, o cálculo do destino e a avaliação da condição também são resolvidos no estágio de decodificação. Porém se desvio for resolvido apenas neste estágio, sempre será necessário inserir uma bolha quando o desvio é tomado. Esta é a solução estática que é apresentada pelo livro [1].

3. Predição no Estágio de Busca

As técnicas de predição de desvios envolvem uma tabela com a predição e o destino [3, 4]. Uma retrospectiva da evolução das abordagens foi apresentada em [5]. Este artigo tem o objetivo principal de ser didático e completo ao apresentar uma implementação detalhada. Inicialmente, um preditor bem simples será apresentado. O preditor pode ser facilmente estendido para incorporar as outras abordagens [5].

O primeiro passo consiste em responder duas questões. A primeira questão é se desvia ou não, ou seja, o desvio será tomado ou não tomado. Para esta pergunta é necessário armazenar apenas 1 bit, caso o desvio seja tomado ou não tomado. A segunda questão é para onde vamos, ou seja, qual é o destino para buscar a próxima instrução, será PC+4 ou o destino que está codificado na instrução. O destino já pode ser pré-calculado. Caso o desvio ocorra, então o destino pré-calculado já está armazenado na tabela. Esta tabela é normalmente implementada por um módulo de memória indexado pelo registrador PC.

A Figura 2 apresenta de uma forma simplificada a tabela de predição. A coluna P tem a predição que irá controlar o multiplexador na entrada do registrador PC. Caso o desvio seja "não tomado", o multiplexador irá selecionar a entrada PC+4 (próxima instrução). Caso a predição seja verdadeira, então o multiplexador irá selecionar a entrada que vem da tabela de histórico de predição. Na próxima seção iremos detalhar como a tabela é preenchida e atualizada.

A implementação da tabela também envolve um conceito importante em computação, que é o mesmo usado pela função *hash* em software e pela memória cache em hardware. Não precisamos ter uma entrada para cada operação de desvio, senão o custo seria proibitivo. A tabela armazena apenas um histórico dos últimos desvios. A implementação mais simples é uma cache de mapeamento direto indexada pelo endereço do PC. Os últimos bits são usados para inde-

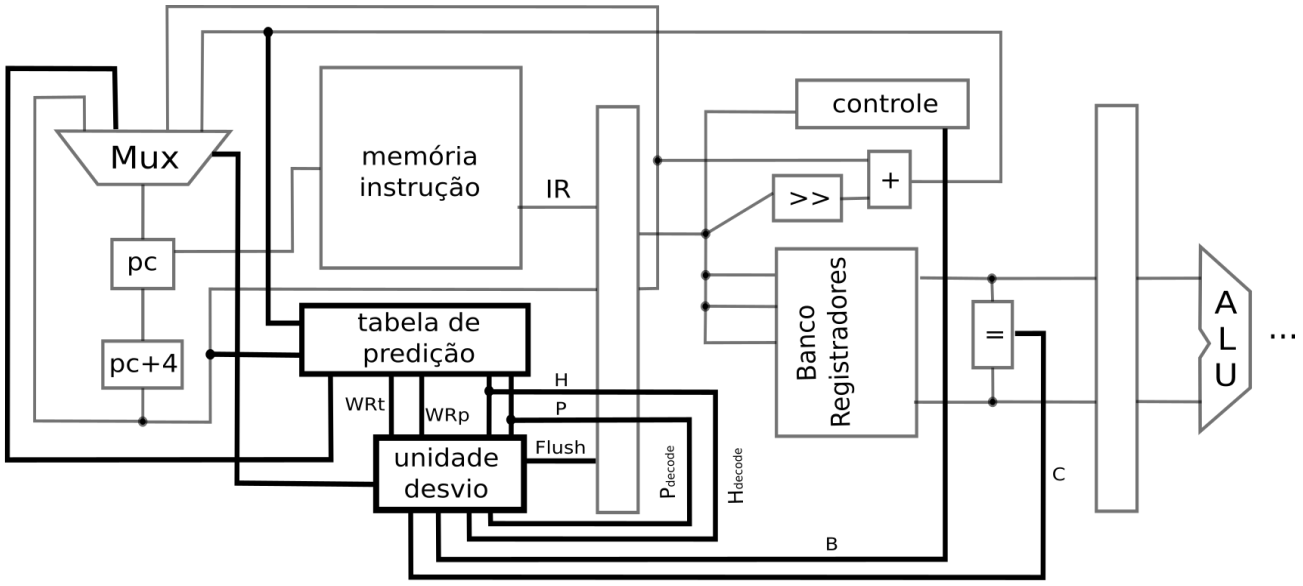

Figura 3. Tabela de Histórico e Unidade de Desvio inseridos nos Estágios de Busca e Decodificação do MIPS

xar a linha da tabela e o prefixo é usado para comparar com o campo de *tag* para saber se a instrução está presente na tabela.

Resumindo, o valor do PC indexa a tabela de predição, caso não esteja na tabela, a instrução não é considerada um desvio e o PC será alimentado com PC+4 pelo multiplexador. Caso exista uma entrada para o PC na tabela, o bit de predição controlará o multiplexador e o campo *target* da tabela alimentará o destino do desvio caso seja tomado. A maioria dos livros e artigos apresenta a implementação de desvios neste nível. Este artigo irá detalhar mais, enumerar os casos e com poucas alterações mostrar que é possível inserir um preditor dinâmico no MIPS pipeline proposto em [1], além de especificar a tabela de controle da unidade de desvio.

4. Predição no Estágio de Decoficação

Para detalhar a implementação, iremos enumerar e tabular os casos. Inicialmente iremos revisitar o estágio de busca com a proposta do preditor completo, incluindo todos os sinais na Seção 4.1. Em seguida, completaremos a análise e a enumeração dos casos no estágio de busca na Seção 4.2 e no estágio de decodificação nas Seções 4.3 e 4.4.

4.1. Novo Caminho de Dados

A Figura 3 ilustra as modificações para inserção da tabela de histórico, o multiplexador com quatro opções, a unidade de desvio e os sinais de controle adicionais. A primeira modificação é indexar a tabela de histórico e predição com PC+4 no lugar de PC. Esta modificação irá facilitar o próximo passo, pois o estágio de decoficação recebe o valor PC+4. Na Seção 4.4 explicaremos melhor a vantagem de usar o PC+4. A segunda modificação é incluir um multiplexador com quatro opções: $PC + 4$, $Target_{Tabela}$, $PC + 4_{decode}$ e $Target_{decode}$. A terceira modificação é a unidade de desvio que será implementada por uma tabela semelhante à unidade de controle, onde todos os casos serão enumerados nas próximas seções.

Os sinais adicionais são: P, H, P_{decode}, H_{decode}, C e B. Cada sinal tem uma função. P é a predição da tabela no estágio de busca e P_{decode} é a predição que já foi propagada para estágio de decodificação onde será validada. Importante ressaltar que são dois momentos da instrução de desvio. No estágio de busca, ocorre uma especulação, a predição já foi inserida na tabela e é apenas utilizada. No estágio de decodificação que a maior parte do trabalho é realizado incluindo a inserção na tabela (primeira vez que o desvio ocorre), atualização da tabela caso ocorra erro na predição e inserção de uma bolha para retornar ao fluxo correto. Continuando com o detalhamento dos sinais, temos o H que indica se a instrução está na tabela de histórico e portanto é um desvio. H_{decode} é o sinal propagado para o estágio de decodificação. C é o resultado da comparação que irá validar o desvio. Neste artigo iremos considerar a simplificação de tratar apenas a instrução *beq*. O sinal B vem da unidade de desvio e indica se a instrução no

estágio de decodificação é um desvio. Além disso, existem alguns sinais adicionais da unidade de desvio para a tabela de histórico que iremos detalhar posteriormente.

4.2. Predição

Nesta seção iremos tratar os dois casos do estágio de busca. O primeiro caso é o mais simples, a instrução não está na tabela de histórico. Então o sinal H será falso e a unidade de desvio irá selecionar o multiplexador para a opção $PC + 4$. A Figura 4(a) ilustra este cenário de forma simplificada. Entretanto, a instrução pode ser uma instrução de desvio que ainda não consta na tabela. Pode ser porque é a primeira vez que é buscada ou porque foi substituída da tabela por conflito devido ao uso do mapeamento direto na implementação da tabela de predição.

O segundo caso é quando a instrução é um desvio e está na tabela de histórico. Podemos observar que existem duas situações que são desvio não tomado ($PC + 4$) e desvio tomado ($Target_{tabela}$) ilustrados nas Figuras 4(b) e 4(c). O sinal H será verdadeiro e P será falso ou verdadeiro se a predição for não tomado ou tomado, respectivamente. A Figura 4(d) mostra os três casos que podem ocorrer estágio busca.

A seguir, iremos começar a projetar a unidade de desvio. Primeiro, a Tabela 1 ilustra um esboço enumerando as três situações que podem ocorrer no estágio de busca.

Tabela 1. Unidade de Desvio simplificada considerando somente a Busca

H	P	outros sinais	Mux	Caso
0	X	...	0	Não Consta Tabela
1	0	...	0	Predição não tomada
1	1	...	1	Predição tomada

4.3. Cadastro da Instrução

Iremos supor apenas que uma instrução *beq* ocorra por vez e que não ocorram dois desvios *beq* consecutivos. Quando uma instrução não é encontrada na tabela não quer dizer que não seja um desvio. Porém o cadastro só será realizado no estágio de decodificação quando a instrução é detectada pela unidade de controle, ou seja, quando abrimos o envelope para fazer a decodificação. O sinal B irá acionar a unidade de desvio indicando que um *beq* está no estágio de decodificação para o cadastro.

Agora iremos detalhar todos os sinais e preencher a Tabela 2. Adicionamos ainda três colunas à direita para facilitar a explanação e uma coluna com o número da linha à esquerda. A coluna *busca* informa qual a instrução que está presente no estágio de busca e a coluna *decode* qual a instrução que está na decodificação. Esta informação é apenas para facilitar a explicação de cada situação. No hardware, os sinais H e B que irão fornecer esta informação. Pois se H é verdadeiro no estágio de busca então uma instrução de desvio está presente. Porém quando a instrução ainda não foi detectada, não saberemos até ela passar para o próximo estágio. Esta situação é ilustrada pelo símbolo *??* na primeira linha da Tabela 2. A última coluna da tabela apresenta uma descrição complementar. O termo $inst$ é usado para qualquer insttrução que não seja um *beq*. Nas colunas de sinais binários, usamos X para representar uma condição *don't care*.

As três primeiras linhas contêm as situações já descritas na Seção 4.2. A Tabela 2 tem três partes. As seis primeiras colunas são os sinais de entrada da unidade de desvio, onde H e P correspondem ao sinal de *hit* (desvio está presente) e de predição provinientes da tabela de histórico. As colunas H_d e P_d correspondem as estes sinais no próximo ciclo de relógio quando já foram propagados para o estágio de decodificação. Os sinais C e B são as saídas do comparador C que é responsável pela avaliação da condição de desvio e a saída b da unidade de controle para indicar que é um *beq*. Ambos são sinais gerados no estágio de decodificação, como ilustrado na Figura 3.

A segunda parte da Tabela 2 são as saídas da undidade de predição, ou seja, como a unidade comanda os recursos de hardware (multiplexador que alimenta o PC, tabela de histórico para cadastro e atualização e o *flush* na barreira para gerar a bolha). A saída Mux controla o multiplexador da entrada do registrador PC, onde o valor 0 irá carregar $PC + 4$, o valor 1 irá carregar a predição $Target_{tabela}$, o valor 2 irá carregar o $PC + 4_{decode}$ e o valor 3 irá carregar $Target_{decode}$. Os valores 0 e 1 são os momentos quando executa uma instrução que não é desvio ou quando executa uma predição, respectivamente. Já os valores 2 e 3 são as situações que ocorreu uma falha na especulação e a predição deve ser cancelada e mudar o fluxo. O valor 3 também ocorre durante o cadastro do *beq*. As saídas WR são usadas para atualizar a tabela de histórico. A saída f (flush) serve para inserir uma bolha no pipeline no caso de falha na predição ou durante o cadastro do *beq* no caso do desvio tomado. Como já mencionada, a terceira parte da tabela é utilizada para facilitar o seu preenchimento e compreensão.

A quarta e a quinta linha da tabela enumeram os dois casos de cadastro quando a instrução beq executa pela primeira vez. Primeiro, podemos observar que o sinal H_d é 0, que significa que o estágio de busca não detectou o beq que não está cadastrado ainda. Porém, o sinal B está ativo no estágio de decodificação onde a instrução está sendo processada. Neste caso temos duas situações. Se o desvio é

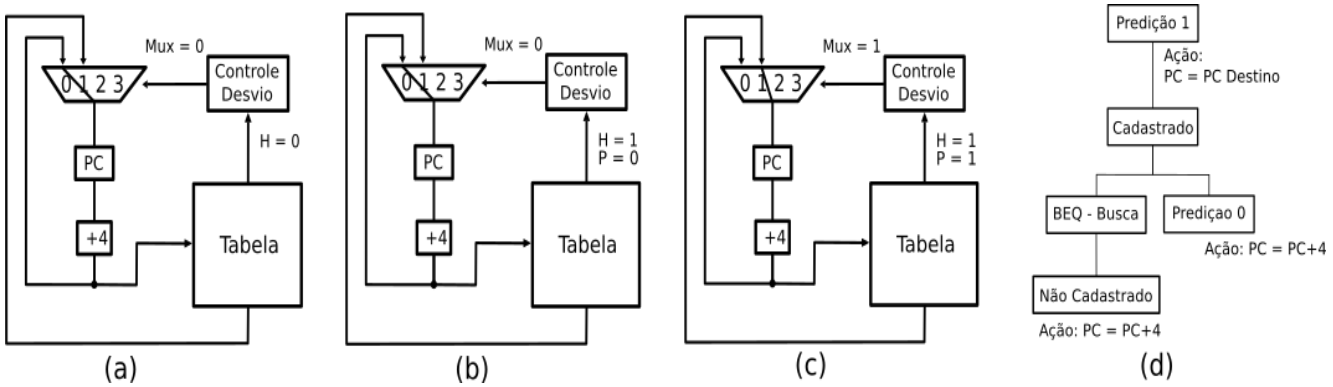

Figura 4. Predição de desvios no estágio de busca: (a) Não consta na tabela; (b) Predição de não desviar; (c) Predição de Desviar; (d) Árvore dos casos do estágio de busca.

Tabela 2. Descrição detalhada da tabela para especificação do comportamento da Unidade de Desvio

	Sinais de Entradas						Saídas				Comentários		
linha	H	P	H_d	P_d	C	B	Mux	WR_p	WR_t	flush	Busca	Decode	Descrição
1	0	X	0	X	X	0	0	0	0	0	??	inst	Não cadastrada
2	1	0	0	X	X	0	0	0	0	0	beq	inst	Predição NT
3	1	1	0	X	X	0	1	0	0	0	beq	inst	Predição T
4	0	X	0	X	0	1	0	1	1	0	inst	beq	Cadastro NT
5	0	X	0	X	1	1	3	1	1	1	inst	beq	Cadastro T
6	0	X	1	0	0	1	0	0	0	0	inst	beq	Acerto Predição NT
7	0	X	1	1	1	1	0	0	0	0	inst	beq	Acerto Predição T
8	0	X	1	0	1	1	3	1	0	1	inst	beq	Erro Predição NT
9	0	X	1	1	0	1	2	1	0	1	inst	beq	Erro Predição T

não tomado, podemos cadastrar e continuar a execução sem perda. Caso o desvio seja tomado ($c = 1$), teremos que inserir uma bolha, observe que o f é ativado e o Mux é programado para receber o $Target_{decode}$. Além disso, temos que cadastrar o beq na tabela de histórico. O sinal WR_p é usado para escrever a predição na tabela. O sinal WR_t é usado para cadastrar o $target_{decode}$ e o *tag* do $PC + 4$ (indexador) do beq. Uma vez cadastrada, a instrução beq permanecerá na tabela. Pode ocorrer uma colisão na tabela, caso já exista uma outra instrução que tenha o mesmo sufixo e neste caso será substituída. Os outros casos serão explicados na próxima Seção.

4.4. Validação

Agora iremos enumerar os casos que a instrução beq já teve sua predição executada no estágio de busca e está agora passando pelo estágio de decodificação para ser validada. Lembrando que não estamos considerando dois beq consecutivos.

Inicialmente nas linhas 6 e 7 iremos apresentar os casos de sucesso da predição. O primeiro caso é quando a predição é "não tomada" ($P_d = 0$) e a execução comprovou que o desvio não foi tomado ($C = 0$). Neste caso não alteramos nada e a execução prossegue como ilustrado na linha 6. O segundo caso é quando o desvio é "tomado" ($P_d = 1$ e $C = 1$). Nenhuma ação é necessária. Importante destacar que o estágio de busca já está carregando no $PC+8$ no caso do desvio "não tomado" ou $target_{tabela}+4$ no caso do desvio "tomado". Ou seja, a predição já foi feita, estamos apenas validando sem prejuízo para a execução.

Existem dois casos que a especulação falha que estão ilustrados nas linhas 8 e 9 da Tabela 2. Suponha que a predição seja "não tomado" ($P_d = 0$) mas a execução foi realizada considerando o desvio "tomado" ($C = 1$). Neste caso várias ações serão necessárias. Primeiro um *flush* ou uma bolha no pipeline, o PC será carregado com o valor $target_{decode}$ (mux=3) e a predição será atualizada ($WR_p = 1$) para o valor de C. Finalmente, o último caso é quando a predição é um desvio tomado ($P_d = 1$) e a execução foi um desvio não tomado ($c = 0$). Novamente é inserida uma bolha, a predição é atualizada com o valor

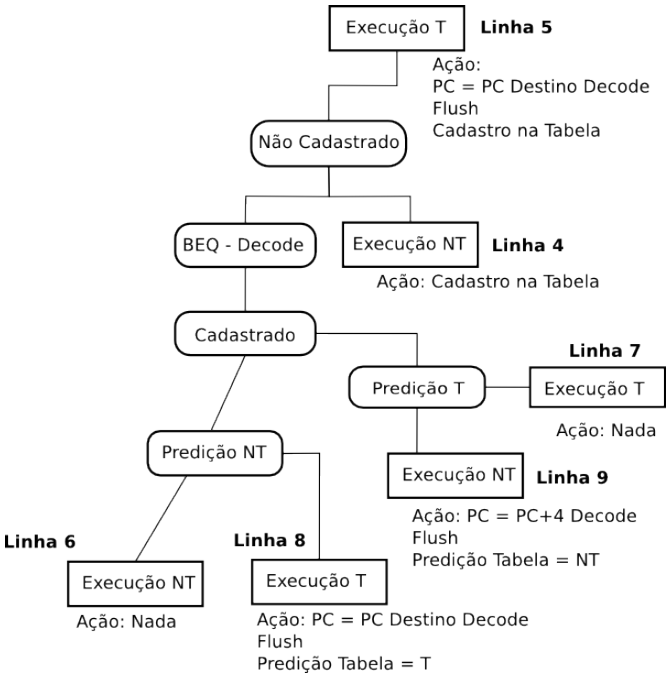

Figura 5. Árvore com os casos do estágio decodificação

de c e agora o PC será carregado com o valor $PC + 4_d$ que se encontra no estágio de decodificação, pois o estágio de busca já perdeu esta informação.

A Figura 5 ilustra todas as situações quando o *beq* está no estágio de decodificação e as ações executadas em cada caso para reforçar o siginficado de cada linha da Tabela 2.

A Tabela 2 enumera as nove situações e pode ser diretamente traduzida para um conjunto de equações lógicas ou descrita em alto nível em uma linguagem de programação ou em uma linguagem de descrição de hardware (como Verilog ou VHDL).

5. Implementação do Preditor

Neste trabalho também apresentamos a validação do preditor implementado em um ambiente de simulação, dando continuidade a uma versão didática do MIPS apresentada em [2] que segue a sequência de projetos do livro texto "Organização e Projeto de Computadores" de Patterson e Hennssy [1]. O projeto pode ser facilmente implementado em outros ambientes de simulação e prototipação com SystemC, C, Verilog, VHDL, etc. Para simular foi usada a extensão MIPS-FPGA [2] do simulador Hades [6] que é um simulador de eventos com interface gráfica. A finalidade inicial do Hades é a simulação de circuitos digitais em vários níveis (portas, RTL, etc) [7], mas outros trabalhos já foram implementados como extensão: fluxo de dados [8, 9]. simulador RaVi de coerência em cache e MIPS [10], uma abordagem mais ampla para projeto de simuladores e ensino [7], dentre outros.

Uma implementação do preditor foi adicionada ao caminhos de dados apresentado em [2]. A implementação foi realizada através da inclusão de dois novos componentes: a tabela de predição e a unidade de desvio. O projeto é simples e segue a estrutura do livro de Patterson e Hennessy.

O componente tabela de histórico é implementado como uma cache de mapeamento direto com os três campos: tag, predição e endereço de destino ($target$), que já foi apresentado na Seção 2. A tabela foi implementada como uma extensão de um módulo de memória genérica com 256 linhas. Como a interface gráfica herdada permite visualizar graficamente o conteúdo com números hexadecimais, optamos por usar múltiplos de 4 bits para facilitar a exibição dos resultados.

A tabela recebe um endereço de 12 bits na entrada derivado dos 12 bits menos significativos do valor de PC+4. Esta opção foi herdada do simulador [2] para simplificar, pois permite programas de teste com até 4 mil instruções que é um valor muito razoável para fins didáticos. Como a tabela tem 256 linhas usa os 8 bits menos significativos para encontrar a linha que pode conter a predição, restando os 4 bits mais significativos que serão usados como *tag* do endereço. Um campo de 32 bits é usado para armazenar o endereço destino (*target*) e um terceiro campo de 1 bit é usado para armazenar a predição. Caso a comparação do tag com o prefixo seja verdadeira, o sinal H (ou *hit*) será enviado para a unidade de desvio que irá repassar para os estágios de busca e decodificação os valores de H, bem como o valor de P que é o campo de predição. Além disso, uma saída fornece o endereço de 32 bits do destino do desvio (*target*).

A unidade de predição, como já foi apresentado na Seção 4, é a implementação direta da Tabela 2 como uma unidade de controle. No simulador pode ser codificada com claúsulas *if-else* na linguagem Java.

Iremos apresentar a linha 9 da Tabela 2 como exemplo do trecho de código em Java para implementação da unidade de controle de desvio como componente a ser adicionado ao projeto do MIPS FPGA [2]. O trecho é ilustrado na Figura 6. A linha 9 descreve a situação onde ocorre uma falha na predição, o *beq* já está no estado de decodificação. Portanto se faz necessário alterar o valor do bit de predição na tabela, inserir uma bolha ativando o sinal de *flush* e escolher a opção 2 para o multiplexador atualizar o valor do PC que atualmente é igual a *Target*+4 para PC+4.

Finalmente a Figura 7 apresenta o caminho de dados completo que o aluno visualiza e simula, assim como o acesso ao *Github* onde se encontra o repositório do projeto que está disponível para a comunidade. Observe que

```
...
else if (beq.is_1() &&
Hd.is_1() && Pd.is_1(){
   if ( cmp.is_0()){
        Mux = 2;  WR_P = HIGH;
        flush = HIGH;
        time = simulator.getSimTime()+delay;
        Schedule(port_Mux, Mux, time);
        Schedule(port_flush, flush, time);
        Schedule(port_Wrdata, WR_P, time);
   }
}
...
```

Figura 6. Trecho de Código em Java para Unidade de Controle correspondente a linha 9 da Tabela 2

.

o aluno pode alterar o projeto acrescentando ou retirando componentes e fios e derivar novos projetos.

6. Outros Preditores e Ferramentas

Como o trabalho pretende apresentar de uma forma didática um preditor de desvio, como já mencionamos optamos por uma versão simples com 1 bit de predição. Porém este preditor falha em laços aninhados, onde a predição erra na primeira vez que passa pelo laço e na última vez. Para solucionar esse problema pode-se acrescentar mais um bit na tabela de históricos, como está explicado no livro texto de Patterson & Hennessy e na literatura da área [1, 3].

Para códigos com padrões de desvio (ou saltos) mais complexos, existem outros tipos de preditores dinâmicos que podemos mencionar, os quais permitem atingir taxas de acerto superiores à 80%, como por exemplo o preditor que utiliza dois níveis de informação dos históricos dos saltos para fazer a predição [4].

Uma atividade de ensino é fazer a implementação com um histórico global de predição de saltos ou local. O histórico global guarda as últimas predições de todo o programa, enquanto o local só guarda as últimas predições da instrução de salto em avaliação, são também conhecidos como preditores de correlação [11] .

No caso de ensino e pesquisa com predição de desvios, a maioria das ferramentas são para execução de códigos e medições de desempenho, sem detalhar a implementação. Por exemplo, temos o simulador GEM5 [12] que é uma plataforma modular para a pesquisa de arquitetura computadores, que permite a descrição da arquitetura tanto no nível de sistema como no nível do processador e sua microarquitetura. O GEM5 tem três tipos de preditores de desvios: 2BitLocal, Hibrido e Bi-mode. Além disso, tem a opção de acrescentar mais preditores, como por exemplo a técnica *Agree* [13], que trabalha de forma similar ao preditor bi-mode [12].

O ambiente SimpleScalar é um simulador de arquitetura *open source* desenvolvido por Todd Austin na University of Wisconsin Madison [14]. O SimpleScalar é um conjunto de ferramentas que permite modelar uma arquitetura virtual de um computador com CPU, cache e hierarquia de memória permitindo aos estudantes e pesquisadores explorarem várias combinações e configurações para executar códigos completos. Entre suas características, o SimpleScalar tem uma funcionalidade que pode simular diferentes esquemas de predição de desvios e com vários recursos para exibir os resultados dos testes. Porém, assim como GEM5, o SimpleScalar não detalha a implementação, servindo para testar o desempenho dos preditores no nível de instruções apenas.

A abordagem de preditores de desvios empregada pela ferramenta SimpleScalar pode ser estática ou dinâmica. As estáticas são: não tomado, tomado e perfeito, As abordagens dinâmicas são: bimodal usando um buffer de destino de saltos (BTB) com contadores de 2 bits, preditor adaptativo de 2 niveis, preditor híbrido combinando os preditores bimodal e adaptativo de 2 niveis [14].

O MARS é um ambiente de desenvolvimento interativo leve (IDE) para programação em linguagem assembler para MIPS, destinado ao uso educacional também baseado no livro texto de Patterson & Hennessy. A partir da verção 3.8 liberada em Janeiro de 2010, o MARS inclui o simulador de predição de desvios usando a Tabela de Históricos de desvios (BHT) [15].

Outra opção é o software Valgrind é um framework para a construção de ferramentas para análise dinâmica. O Valgrind é composto por um conjunto de ferramentas, como o Cachegrind, que simula o comportamento da cache e também dos preditores de desvios para processadores convencionais. A predição é feita usando uma tabela de 16K posições e contadores de 2 bit para predição. O preditor utiliza correlação global e local, e como resultado, as predições para qualquer desvio dependem tanto de sua própria história quanto do comportamento dos desvios anteriores [16].

Resumindo, as ferramentas disponíveis para ensino e pesquisa de preditores atuam no nível de instrução, sem detalhes de implementação. Este artigo apresenta uma metodologia simples e direta para incorporar, no nível de transferência de registradores com apenas dois componentes, um preditor de desvios ao caminho de dados do MIPS 5 estágios.

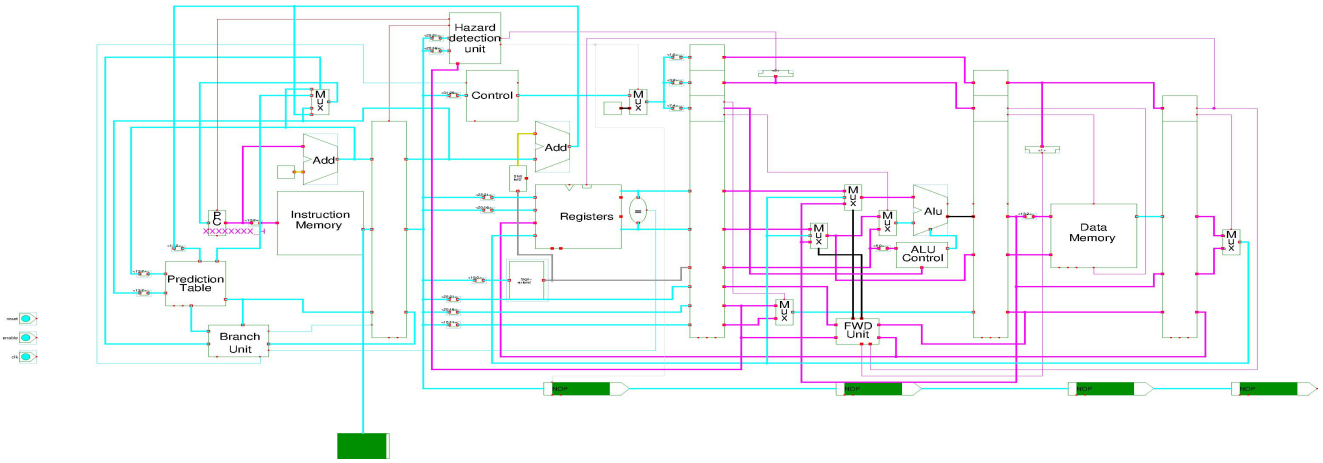

Figura 7. Caminho de Dados do MIPS com a inclusão da Unidade de Predição de desvio e a tabela de histórico no estágio de busca. O projeto e o ambiente de simulação estão disponíveis no link *https://github.com/ComputerArchitectureUFV/ufv_mipsfpga_predictor_1bit.git* **.**

7 Proposta de um Exercício: Implementar um preditor de 2 bits

Seguindo a sequência do livro texto de Patterson & Hennessy, este trabalho propõe como sugestão de exercício a implementação de um preditor com 2 bits cuja a vantagem é ter uma inércia para trocar a predição. Apenas a máquina de estados é ilustrada sem detalhes de como interligar ao projeto, como ilustrado na Figura 8(a).

A solução para a implementação pode ser guiada com pequenas modificações no caminho de dados apresentadas na Figura 8(b). A proposta é acrescentar mais uma unidade (máquina) e fazer pequenas modificações na tabela de histórico. A unidade adicionada irá implementar a máquina de estados para atualizar a predição. Sempre que a instrução estiver no estágio de decoficação, a predição é atualizada seguindo a máquina de estados. Ou seja, os sinais C, B e H_d são suficientes para gerar a lógica de controle da máquina que está detalhada na Figura 8(c).

A tabela de histórico agora passa a incorporar 2 bits de predição. Se usarmos o bit mais significativo para diferenciar entre não tomado e tomado, este bit irá gerar o sinal P que é a saída da tabela de predição no estágio de busca. Portanto com uma pequena alteração é possível estender o projeto básico para um preditor de 2 bits. Outras tarefas que podem ser incorporadas são os projeto com histórico local ou global, preditor de dois níveis, etc.

8. Considerações Finais

Este artigo ilustra que com o projeto de apenas dois componentes e poucas modificações, é possível incorporar um preditor dinâmico ao caminho de dados do MIPS pipeline com 5 estágios. Embora seja o preditor mais básico, este preditor pode ser facilmente estendido e ilustra com detalhes as situações que o projetista deve resolver, usando regularidade e recursos que podem ser facilmente implementados em hardware.

A metodologia de projeto segue a linha de desenvolvimento do livro texto de Patterson & Hennessy e pode ser incorporada na sequência didática. Ao implementar com detalhes, o estudante passa a ter uma visão concreta dos recursos necessáriod, exercita a modelagem e implementação de um recurso que só pode ser mapeado de forma extremamente eficiente em hardware para que tenha efeito de melhorar o desempenho,

Trabalhos futuros podem incorporar exemplos de outros preditores e códigos de teste de benchmarks para validação da execução com medições de desempenho. Outra sugestão é acrescentar o projeto de caches ao MIPS básico que pode envolver duas etapas. A primeira é a implementação da cache em si, onde várias versões podem ser avaliadas. A segunda parte é o acoplamento de uma cache de dados e de instruções que pode gerar um bloqueio do pipeline que deve ser resolvido em hardware.

9. Agradecimentos

Este projeto foi financiado pelas agências FAPEMIG (Universal 2014), CNPq e CAPES e a empresas Intel/Brazil e Gapso.

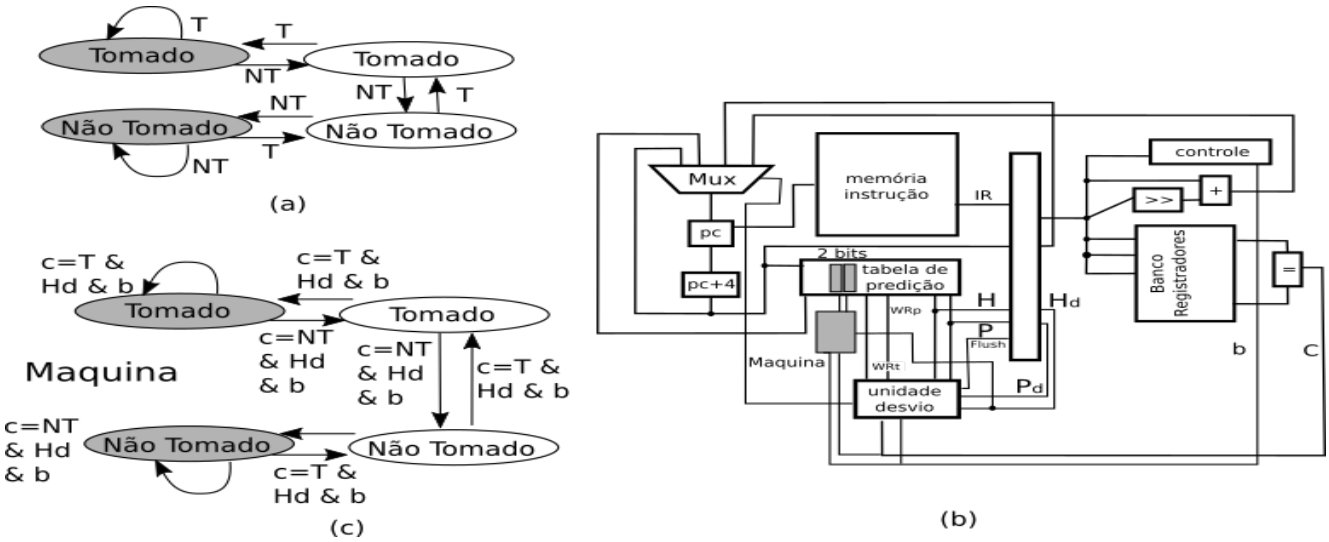

Figura 8. (a) Máquina de Estados adaptada do Livro [1]; (b) Detalhe do Estágio de Busca com a unidade de atualização de 2-bits; (c) Máquina de Estado com os sinais de controle

Referências

[1] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface*. Morgan Kaufmann, 2013.

[2] J. Penha, G. Fontes, and R. Ferreira. Mipsfpga - um simulador mips incremental com validação em fpga. *International Journal of Computer Architecture Education*, 5(1):19–25, 2016.

[3] J. E. Smith. A study of branch prediction strategies. In *Proceedings of the IEEE Annual Symposium on Computer Architecture*, 1981.

[4] T. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *ACM SIGARCH Computer Architecture News*, volume 20, pages 124–134. ACM, 1992.

[5] O. Mutlu, R. Belgard, T. R. Gross, J. L. Hennessy, Y. N. Patt, et al. Common bonds: Mips, hps, two-level branch prediction, and compressed code risc processor. *IEEE Micro*, 36(4):70–85, 2016.

[6] N. Hendrich. A java-based framework for simulation and teaching: Hades—the hamburg design system. In *Microelectronics Education*. Springer, 2000.

[7] R. Ferreira, J. Nacif, et al. Be a simulator developer and go beyond in computing engineering. In *IEEE Frontiers in Education Conference*, 2015.

[8] R. Ferreira, J. Cardoso, and H. C. Neto. Data-driven regular reconfigurable arrays: design space exploration and mapping. In *Embedded Computer Systems Architectures, Modeling and Simulation*, 2005.

[9] R. Ferreira, J. Cardoso, and H. C. Neto. An environment for exploring data-driven architectures. In *Field-Programmable Logic and Applications*, 2004.

[10] P. Marwedel, K. Cong, and S. Schwenk. Ravi: Interactive visualization of information system dynamics using a java-based schematic editor. 2002.

[11] S. McFarling. Combining branch predictors. Technical report, Tech. Report TN-36, Digital Western Research Laboratory, 1993.

[12] N. Binkert, S. Sardashti, R. Sen, K. Sewell, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, May 2011.

[13] G Glenn Henry and Terry Parks. Static branch predictor using opcode of instruction preceding conditional branch, July 16 2002. US Patent 6,421,774.

[14] D. Burger and T. M. Austin. The simplescalar tool set. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[15] K. Vollmar and P. Sanderson. Mars. *Proc. of SIGCSE technical symposium on Computer science education*, 2006.

[16] Valgrind TM Develomers. 5. cachegrind: a cache and branch-prediction profiler.