Extensões ao Modelo VHDL do MiniMIPS

Giuliano T Bertoncello Lucas M Koeb
Roberto A Hexsel (orientador)
Depto de Informática – Universidade Federal do Paraná
CEP 81531-990 – Curitiba – PR – Brasil
{gtb06,lmk08,roberto}@inf.ufpr.br

Resumo

Com o objetivo de disponibilizar uma implementação de um processador MIPS descrita em VHDL adequada à utilização em projetos maiores, propomos melhorias e extensões a um projeto existente – MiniMIPS. Alguns problemas encontrados no projeto original foram corrigidos e certos detalhes da implementação, em especial no circuito de predição de desvios, foram alterados para melhorar o desempenho geral do processador. O conjunto de instruções foi estendido com instruções importantes que não faziam parte do projeto e aparecem com grande freqüência nos códigos gerados por compiladores. Pretendemos empregar este trabalho no projeto de um multiprocessador que foi desenvolvido em trabalho de Mestrado no Departamento de Informática da UFPR.

1. Introdução

A motivação inicial para este projeto de Iniciação Científica surgiu quando os autores cursaram a disciplina de Arquiteturas Avançadas de Computadores do curso de Bacharelado em Ciência da Computação da UFPR. Nos pareceu interessante a realização de um trabalho extracurricular para colocar em prática a teoria vista naquela disciplina, durante o primeiro semestre de 2011. Com a idéia de desenvolver um trabalho de projetar um processador modelado em VHDL, descobrimos um projeto semelhante já existente, que é o MiniMIPS [2]. Decidimos então que seria mais interessante refinar o projeto existente, para utilizá-lo em um projeto de maior complexidade desenvolvido em trabalho de Mestrado na UFPR, que é o Multiprocessador Minimalista com Caches Coerentes (MMCC) [7, 8].

O MiniMIPS é um modelo descrito em VHDL de um processador segmentado de cinco estágios que implementa uma parte do conjunto de instruções Mips32 [5, 6] e está disponível através do portal `OpenCores.org`. O código VHDL do MiniMIPS é dividido em cinco módulos: circuito de dados (*datapath*), bloco de registradores, unidade de gerenciamento de memória (coprocessador 0), controle do barramento e previsor de desvios. A Figura 1 mostra um diagrama de blocos com as relações entre os módulos do MiniMIPS e os componentes de um hipotético microprocessador baseado no MiniMIPS.

Os estágios do circuito de dados são: ⟨PF/EI⟩ busca, composto pelas unidades de *prefetch* e *instruction extraction*; ⟨DI⟩ decodificação, que contém uma tabela de microcódigo com as definições dos sinais de controle para cada instrução; ⟨EX⟩ execução; ⟨MEM⟩ acesso à memória; e ⟨WB⟩ *writeback* com o controle do banco de registradores e que realiza adiantamentos para evitar riscos de dados.

Após uma série de testes e análise do código VHDL, verificamos que alguns dos módulos do processador apresentavam erros. A unidade de predição de desvios não funcionava corretamente para grande parte dos laços usados nos testes, e a instrução no *branch delay slot*, que deveria ser executada em todos os desvios, não era executada. Além destes problemas, diversas instruções importantes do conjunto de instruções Mips32 [5] não constavam do projeto inicial. As instruções de *load/store byte*, *load/store half* e divisão de inteiros não faziam parte do projeto original embora sejam usadas com freqüência em código gerado por compiladores, pois são utilizadas para manipulação de *strings*.

O projeto *MIPS Segmentado* [4] contém uma implementação parcial do conjunto de instruções Mips32, com circuito de dados segmentado e detecção e tratamento de riscos, mas sem previsão de desvios. [3] apresenta um estudo de caso no qual um processador MIPS de 32 bits é sintetizado a partir de uma descrição codificada em VHDL, e é acoplado a um controlador de vídeo 'padrão' VGA. [1] descreve uma implementação em VHDL do conjunto de instruções Mips32 segmentado, com exceções precisas, coprocessador-0 que trata das exceções e gerenciamento de memória, e memória cache. O texto contém detalhes da implementação de algumas unidades funcionais do circuito de dados.

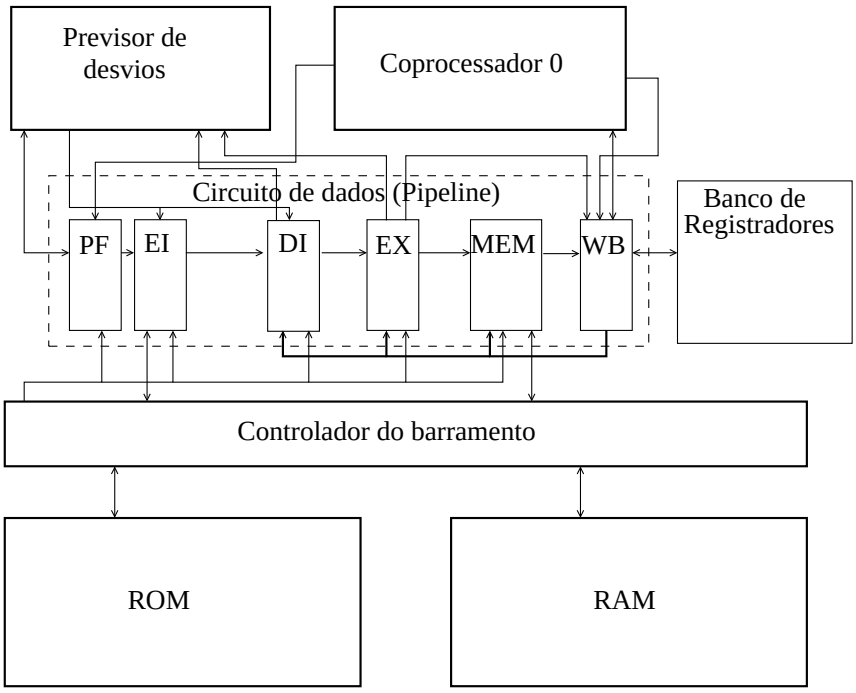

Figura 1. Módulos do MiniMIPS e os estágios do circuito de dados.

2. Correções

Para facilitar a programação, geração e carga do código, tornando o MiniMIPS adequado à utilização no projeto de um multiprocessador, os problemas já citados no projeto deveriam ser corrigidos. O problema com a unidade de predição de desvio poderia ser contornado com a remoção da unidade, mas isto teria um impacto negativo no desempenho do processador. O problema com a instrução do *branch delay slot* pode causar erros de execução, já que o compilador poderia alocar instruções nestas posições na tentativa de gerar código otimizado. Além da correção destes dois problemas, a extensão do conjunto de instruções do MiniMIPS é essencial para possibilitar o uso de um compilador para testes com programas escritos em linguagens como C.

2.1 Predição de desvios

Devido à necessidade de utilizar bloqueios *(stalls)* para fazer com que dependências de dados e de controle sejam respeitadas, desvios trazem um impacto negativo no desempenho geral do processador. Uma forma de diminuir as penalidades é tentar prever o comportamento dos desvios. Existem dois métodos para realizar estas previsões. O primeiro, que se baseia no código do programa, consiste em técnicas para re-arranjar as instruções, reduzindo os riscos e número de bloqueios. Chamada de "previsão estática de desvios", este método é utilizado em tempo de compilação. O segundo, que é o método utilizado no MiniMIPS, é implementado em hardware e prevê dinamicamente o comportamento dos desvios com base no resultado das últimas execuções de cada desvio.

A predição de desvios do MiniMIPS apresentou um comportamento errático para alguns dos testes executados: a execução de laços era interrompida muito antes do que seria o correto. Executando simulações e investigando o código dos módulos do processador descobrimos que o erro era causado pela unidade de predição, que não estava preparada para as situações em que a instrução de desvio condicional dependia dos dados gerados pela instrução imediatamente anterior a ela. Algumas alterações na unidade de predição de desvio e alterações nos sinais de controle do *datapath* foram suficientes para corrigir este problema.

O funcionamento da predição de desvios do MiniMIPS tem como base uma tabela que contém, em cada posição, o resultado de um desvio executado anteriormente. Toda vez que uma instrução de desvio é executada, há uma verificação se a instrução pertence a esta tabela. Instruções que estão ausentes são adicionadas e o status da predição indica um desvio não-tomado, resultando em uma previsão que mantém o fluxo de execução sempre que a instrução é executada pela primeira vez. Se a instrução de desvio já foi executada anteriormente e está na tabela, temos o resultado obtido na última execução, e neste caso a unidade de predição de desvio faz com que este seja também o resultado da nova execução. Após estas verificações, a

instrução de desvio chega ao estágio de execução, que realiza a comparação entre seus operandos gerando o resultado efetivo do desvio. Este resultado é comparado com o utilizado pela unidade de predição, que verifica se a predição foi correta. No caso de um acerto na predição o fluxo de execução continua inalterado, de acordo com a previsão. Caso contrário, quando a previsão é errada, a tabela é atualizada, as instruções que foram buscadas incorretamente são anuladas e o fluxo de execução é alterado para a instrução correta.

2.2 Branch Delay Slot

O endereço seguinte à qualquer instrução de desvio é chamado de *branch delay slot*, e de acordo com o documento que define o conjunto de instruções Mips32 [5] instruções nestes endereços devem sempre ser executadas, independentemente do resultado de desvios condicionais. É importante que a definição do conjunto de instruções seja seguida, pois freqüentemente instruções são reordenadas pelo compilador na busca de melhor desempenho e o *slot* após os desvios pode ser preenchido com uma instrução que executa trabalho útil. No projeto original do MiniMIPS estas instruções não eram executadas em nenhum caso, o que além de estar em desacordo com a definição do conjunto de instruções pode gerar sérios erros na execução, se o compilador utilizar estas posições ao re-ordenar o código para obter melhor desempenho e aumentar a utilização dos recursos do processador. Com algumas mudanças no módulo de busca de instruções e na unidade de predição de desvios este problema foi corrigido.

3. Novas Instruções Implementadas

Algumas instruções utilizadas pelo compilador GCC não estavam implementadas no modelo original. Dentre estas, as instruções de divisão de inteiros e de leitura e escrita de *bytes* na memória nos pareceram as mais importantes. Estas instruções são utilizadas pelo compilador principalmente para o tratamento de *strings*, e isso dificulta a compilação de programas de teste. Para solucionar este problema, os módulos do *datapath* e controle do barramento do processador foram alterados para conter a implementação destas instruções, de acordo com o conjunto de instruções Mips32.

Para as instruções de divisão de inteiros (`div, divu`), foi suficiente a definição dos campos de identificação das instruções e a adição dos sinais de controle correspondentes na tabela de microcódigo do estágio DI, além de adicionar as operações de divisão na Unidade de Lógica e Aritmética (ULA). A tabela de microcódigo define os valores dos sinais de controle para a execução de cada instrução. Supomos que as bibliotecas distribuídas com o compilador GHDL tratam os operadores de divisão corretamente, quando utilizadas para síntese do processador em FPGAs.

As modificações necessárias para a implementação das instruções de acesso a memória (`sb, lb, lbu, sh, lh, lhu`), abrangeram os estágios do módulo do *datapath*, e também o módulo do controle do barramento. Foi necessária a adição de novos sinais de controle para a interação com a memória RAM, e também a modificação da interface original com o processador, que não possibilitava acesso de leitura e escrita em *bytes*.

Os *bytes* que serão escritos das instruções de *store byte/half* são selecionados de acordo com um novo sinal de controle. Os *bytes* que serão lidos pelas instruções de *load byte/half* são tratados no controlador do barramento por um circuito adicional que recebe uma palavra da memória RAM e escolhe a fração adequada. Implementamos as instruções desta forma para possibilitar uma melhor integração deste processador ao MMCC, que é a extensão deste trabalho, como discutido adiante.

4. Melhoramentos ao Projeto Original

O módulo de Predição de Desvios do MiniMIPS é composto por uma tabela com capacidade para o registro de apenas três endereços de desvios condicionais, com a política de substituição FIFO. Cada elemento da tabela contém: (a) um *bit* válido, (b) o endereço da instrução de desvio, (c)) o resultado do último desvio, e (d) o endereço de destino.

Esta tabela é pequena, e para os programas de testes com laços aninhados em alguns poucos níveis, esta estrutura causa perda de desempenho por causa das substituições muito freqüentes, adicionando *stalls* desnecessários. Modificamos a tabela, aumentando o número de registros para 32 posições, com mapeamento direto dos endereços das instruções de desvio no índice da tabela, de acordo com os 5 *bits* menos significativos do endereço da instrução de desvio, como mostra a Figura 2.

5. Conclusão e Trabalhos Futuros

A aplicação prática neste trabalho de conteúdos estudados nas disciplinas da área de Arquitetura de Computadores nos permitiu uma melhor compreensão global sobre o assunto, principalmente devido as interações entre os componentes do processador, e o código *assembly* gerado pelo compilador – uma alteração no processador pode resultar numa série de consequências, exigindo um planejamento detalhado sobre quaisquer melhorias desejadas.

Com novas instruções que implementamos, executamos testes com código gerado pelo GCC, e para a geração da maior parte dos programas de teste, utilizamos o montador

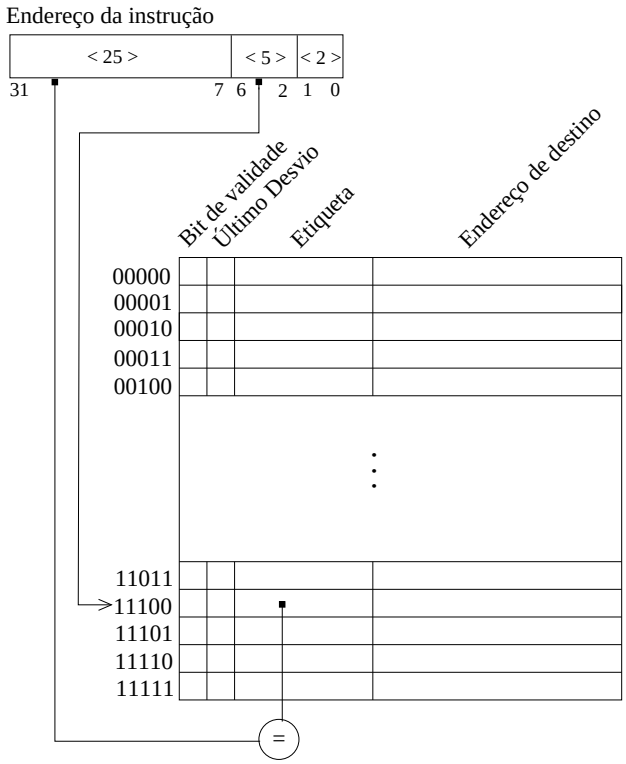

Figura 2. Nova tabela de predição de desvios.

disponibilizado com o MiniMIPS. Todos os testes produziram resultados corretos. É necessário adaptar o ligador do GCC para alocar código e instruções segundo o mapeamento de endereços do código do processador, e esta tarefa será iniciada em breve. Então, será possível efetuarmos uma avaliação mais extensa sobre os ganhos de desempenho obtidos com as melhorias introduzidas no processador. Outra tarefa ainda pendente consiste em carregar o processador em um FPGA e então testá-lo com alguns *benchmarks* para então avaliar seu desempenho de forma mais ampla.

As modificações introduzidas no processador podem parecer simples, ou até mesmo simplórias. Algumas escolhas, como a do previsor de desvios com um único bit de previsão, decorrem de limitações dos FPGAs de que dispomos. A implementação do MMCC com 4 processadores MiniMIPS ocupa praticamente todos os recursos do circuito Xilinx Virtex4 XV4VSX25-12FF668 [7, 8]. Quando dispusermos de FPGAs de maior capacidade para efetuarmos testes, a implementação do previsor de desvios empregará uma tabela maior, e contadores de dois bits, desde que a avaliação de custo e desempenho do MMCC com o MiniMIPS estendido assim o justifique.

A próxima etapa deste projeto é adaptar o código do processador para usá-lo no multiprocessador desenvolvido em um trabalho de Mestrado do PPGInf da UFPR [7]. As tarefas pendentes consistem principalmente do projeto de um barramento que suporte operações com bytes e *half-words* para a comunicação entre os vários núcleos processadores. Estes núcleos serão compostos pelo MiniMIPS estendido, como descrito neste trabalho.

Referências

[1] C. Brej. A MIPS R3000 microprocessor on an FPGA. Relatório técnico, CiteULike, Dez 2008.

[2] S. Hangouët, S. Jan, L.-M. Mouton, and O. Schneider. MiniMIPS. Relatório técnico, OpenCores.org, 2009.

[3] R. M. Kubde, D. B. Bhoyar, and R. S. Khedikar. Design of 32 bit (MIPS) RISC processor using FPGA. In *Intl Conf and Workshop on Emerging Trends in Technology*, ICWET '10, pages 936–939, 2010.

[4] E. Luján. MIPS segmentado. Relatório Técnico, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina, dez. 2008, OpenCores.org, 2010.

[5] MIPS. *MIPS32 Architecture For Programmers – Introduction to the MIPS32 Architecture*, volume 1. MIPS Technologies, Jul 2005.

[6] D. A. Patterson and J. L. Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufmann, 4th edition, 2009. ISBN 978-0-12-374493-7.

[7] J. Tortato Jr and R. A. Hexsel. MPSoC minimalista com caches coerentes implementado num FPGA. In *WSCAD-SSC'09: X Workshop em Sistemas Computacionais de Alto D esempenho*, pages 1–8, Out 2009.

[8] J. Tortato Jr and R. A. Hexsel. A minimalist cache coherent MPSoC designed for FPGAs. *Int J. High Performance Systems Architecture*, 3(2-3):67–76, 2011.