Simulador MIPS (Pipeline) NewPipe

Rafaelo Pinheiro da Rosa[1], Vinícius Berne da Costa[1], Eduardo Kluwe Veiga[1], Charles Rodrigo Ribas Almeida[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Bagé – RS – Brasil

Rafaelo_pinheiro@yahoo.com.br, bernevini@gmail.com, edu@bsd.com.br, charlesdp9@bol.com.br

Abstract. *This article describes the implementation of the MIPS architecture in C language, explains in detail how each block was done, and showing that the resources of the language used to represent each component of the architecture. It also describes how to represent the clock, the other signals to control the representation of hazards and implementation of caches of two and four channels. Shows some changes that occurred in the path data pipeline for the implementation of some MIPS instructions.*

Resumo. *Este artigo descreve a implementação da arquitetura MIPS em linguagem C, explicando detalhadamente como cada bloco foi feito, e mostrando quais os recursos da linguagem utilizados para representar cada componente da arquitetura. Descreve também a forma de representação do clock, dos demais sinais de controle, a representação dos Hazards e a implementação das Caches de duas e quatro vias. Mostra algumas mudanças que ocorreram no caminho de dados pipeline para a execução de algumas instruções MIPS.*

1. Introdução

O Simulador MIPS (Pipeline) NewPipe foi implementado para possibilitar a visualização da execução de um programa sobre a arquitetura MIPS. A linguagem de programação utilizada para a implementação foi C. O Simulador fornece recursos para exibição de resultados no final da execução completa e ciclo a ciclo, possuindo suporte para monitoração de valores para sinais de controle, todos os registradores da arquitetura (banco de registradores), conteúdo da memória de dados contendo a faixa de endereços e o conteúdo das Caches de duas e quatro vias. Os componentes da arquitetura foram descritos através de funções e procedimentos. Foram utilizadas estruturas para a representação de todos os campos de uma instrução, dos sinais de controle e dos registradores de pipeline. Todos os Hazards são detectados pelo Simulador, embora não sejam impressos, na tela, as situações em que ocorrem.

2. Utilização de Estruturas

Para facilitar a implementação do Simulador, foram utilizadas várias estruturas, cada uma com vários campos. Como as instruções MIPS são do mesmo tamanho (32 bits ou 4 Bytes), os tipos das instruções (R, I ou J) possuem alguns campos diferentes. A maneira mais convincente encontrada para representar uma instrução MIPS em C foi criar uma estrutura do tipo Instrução que possuísse todos os campos possíveis encontradas em instruções dos tipos R, I ou J, além de outros. O código em C para a representação de uma instrução:

```
1	typedef struct instrucao {
2		int op, funct, tipo, rs1, rt1, rd1, sh1, imediato, off, end1, extensao;
3		char rs[6], rt[6], rd[6], end[11], offset[11], sh[6], im[11];
4		int validade;
5	}Instrucao;
```

A estrutura contém 7 vetores do tipo char que são utilizadas em uma função que é executada como se fosse um tradutor, fazendo com que esses vetores recebam os campos da instrução MIPS. Também possuem 12 campos do tipo int, sendo 7 deles uma representação inteira dos vetores do tipo char feitas pela função que converte os caracteres em inteiros para facilitar a execução e o entendimento do usuário. Contém um campo extensão, também do tipo int, relacionado com a unidade de extensão de sinal encontrada no mapa da arquitetura mostrada no anexo. Esse campo foi colocado nessa estrutura para ajudar no momento em a função de extensão de sinal for chamada. Outro campo dentro desta struct foi a validade. Quando o bit de validade for 0 significa que a instrução buscada na memória não é validade, ou seja, não precisa ser executada.
Para representar os sinais de controle, também foi utilizado o recurso de estruturas, contendo todos os sinais de controle da arquitetura MIPS e mais dois sinais que foram colocados para facilitar as execuções de algumas instruções MIPS como jal, beq e bne.

```
1       typedef struct sinais {
2               int origpc, escrevereg, branch, regdst, origalu0, origalu1, lemem;
3               int escrevemem, mempararеg, opalu, inversor, link;
4       }Sinais;
```

Foi escolhido essa forma de representação para ajudar na implementação do código, assim, foi possível inicializar todos os sinais em uma função mandando apenas uma variável do tipo Sinais. Foi criado o sinal inversor para ajudar na implementação de das instruções de desvio condicional como beq e bne. Esse sinal escolhe qual das entradas em um mux 2:1 será selecionada. Ou ele seleciona o bit zero que vem de uma unidade de comparação ligada ao banco de registradores, ou seleciona seu valor invertido. Também foi criado o sinal link, para a execução do jal. Quando esse bit estiver ativo, ele salva o conteúdo de PC + 4 no registrador $ra (R[31]).
Foram criadas estruturas para os quatro registradores de pipeline do MIPS. Todos os dados que são passados para esses registradores na arquitetura são representados dentro dessas estruturas, além de alguns outros campos. Um deles, presentes em todos os registradores de pipeline é o exit que será explicado mais a frente. Esses registradores também armazenam os sinais que serão utilizados em cada um dos estágios da arquitetura (IF, ID, EX, MEM, WB). Foram utilizadas estruturas para essas representações, pois cada um dos registradores no caminho de dados do pipeline armazena mais de um dado. Cada dado encontra-se no seu campo e implementamos esses campos usando a Estrutura.
Outra situação em que ocorre a implementação por meio de estruturas é no caso das Caches. Como o simulador possui duas memórias, uma para as instruções e outra para os dados, foram criadas duas Caches, uma de duas e outra de quatro vias, para a representação dos dois tipos de memória. Cada estrutura de Cache relacionada com a memória de instruções possui quatro variáveis do tipo inteiro, uma que representa o bit de modificação, uma que representa o bit de referência, uma que representa o bit de validade e uma variável que representa a tag. Já nas caches relacionadas com a memória de dados, as estruturas possuem cinco variáveis do tipo inteiro, sendo quatro dessas variáveis iguais às variáveis contidas nas Caches de instruções e uma outra variável que representa o dado.

3. Utilizações de Funções

Uma das primeiras funções implementadas foi a função filtro. Ela é executada como se fosse um tradutor recebendo uma linha contendo a instrução MIPS do arquivo texto. Ela analisa toda a Instrução e a traduz para um formato numérico compreendido pelo MIPS (no nosso caso usamos a notação decimal). A função recebe como parâmetro um vetor do tipo char, uma variável do tipo Instrução, uma variável n (tamanho do vetor) do tipo int, e uma variável cont também do tipo int. A função valor_registrador retorna o número decimal que representa o registrador dentro da arquitetura MIPS.
Os cinco estágios do pipeline foram implementados dentro de funções, pois foi uma forma

característica de representar o que esta acontecendo dentro de cada um independente do outro. As chamadas dessas cinco funções são feitas de forma inversa para dar ilusão do paralelismo do pipeline como mostra o código abaixo:

```
1       while (exit==0) {
2               exit=WriteBack(&memory);
3               Memory(&execute,&memory);
4               Execute(&decode,&execute);
5               instruction_decode(&fetch,&decode,reg,&sc);
6               instruction_fetch(&fetch,&decode);
7               //Hazard(&fetch, &decode, &execute, &memory);
8               print(&fetch,&decode,&execute,&memory);
9       }
```

Quando é feita uma leitura da memória de instruções e a posição é inválida, um sinal exit (fora dos sinais de controle) é ativado significando que aquele estágio não precisa ser mais executado. Quando o sinal chegar no último estágio, a chamada das funções termina. A função que representa a unidade de detecção de Hazard (linha 7) foi implementada seguindo a terceira edição do livro Organização e Projeto de Computadores dos autores Patterson e Hennessy, com alguns acréscimos para a execução de casos gerais. Ela é feita através de uma função que recebe os quatro registradores de pipeline representados por estruturas, comparando-os. Caso seja detectado algum **** Hazard, a função retorna um sinal (também fora dos sinais de controle) para selecionar a entrada com valor 0 do mux 2:1, a outra entrada contendo uma variável do tipo Sinais não seria selecionada.

A unidade de extensão de sinal foi feita através de uma função que recebe uma variável do tipo Instrução e uma variável do tipo int. Todas as instruções que saem da memória de instruções passam por essa função. Algumas dessas instruções passam por um deslocamento de 2 bits à esquerda. Esse deslocamento é feito dentro da função execute.

A ULA (Unidade Lógica e Aritmética) assim como o controle da ULA também foram implementadas como funções, sendo que a função da ULA dependia da função do controle da ULA, embora as duas tenham sido feitas com o mesmo recurso da linguagem, sendo utilizados nesse caso comandos de escolhas para determinar que operação deve ser realizada por cada uma delas. O controle da ULA retorna um valor inteiro que irá selecionar que operação deve ser feita pela ULA.

Os Mux, por serem elementos de estado, foram implementados através de funções embora tenham sido inicialmente implementados de forma errada através de vetores durante o seminário de andamento.

4. Utilização de vetores

Foram criadas duas memórias, implementadas através de vetores, com o mesmo tamanho (mesmo número de posições, ou seja, 256). Uma delas é a memória de instruções que guarda todas as instruções traduzidas ou “filtradas” pela função filtro, e a outra é a memória de dados que guarda todos os dados e endereços utilizados pelas instruções load word e store word. O banco de registradores também foi implementado como um vetor de 32 posições indo da posição 0 até a posição 31 representando todos os registradores que a arquitetura MIPS suporta. Foi feito dessa forma, pois foi a maneira mais fácil que encontramos para representar o banco de registradores.

As duas Caches de instruções de duas e quatro vias, foram feitas através de matrizes, assim como as duas Caches de dados. As Caches de instrução e de dados que possuem duas vias possuem número de linhas igual a quatro e número de colunas igual a dois. Já as Caches de quatro vias possuem número de linhas igual a dois e número de colunas igual a quatro.

5. Alterações no caminho de dados pipeline

Algumas mudanças foram feitas em relação ao mapa da arquitetura da figura 6.41 do livro Organização e Projeto de Computadores [Patterson and Hennessy 2005]. Uma delas foi no Mux que seleciona qual das entradas que vai para o PC. Foram acrescentados mais duas entradas à esse Mux; uma com o endereço, para a execução do Jump, que vem do registrador pipeline IF_ID, a outra mudança foi numa entrada para o Jump Register que vem do segundo dado lido no banco de registradores. Outras mudanças ocorreram nos dois Mux que selecionam qual dos dados que vão entrar na ULA. No primeiro Mux, foi colocada mais uma entrada relacionada com a instrução Lui. Essa entrada vem de uma outra unidade de deslocamento de 16 bits à esquerda que também foi colocada no segundo estágio (Instruction Decode) do caminho de dados pipeline, ligada por um fio à unidade de extensão de sinal. Sem essa unidade de deslocamento, a execução da instrução Lui resultava em erros. No segundo Mux, apareceu uma entrada com o valor zero. Esse valor está relacionado com a execução da Lui, ou seja, os valores que irão entrar na ULA, caso a execução seja da Lui, será a entrada que vem do deslocamento de 16 bits à esquerda no primeiro Mux e a entrada com o valor zero no segundo Mux. Outra mudança no segundo Mux foi a adição de mais duas entradas. Uma para o campo Shamt da instrução MIPS para a execução das instruções sll e srl. A outra foi para a extensão para a execução das instruções do tipo imediato. Assim, o primeiro Mux recebeu uma nova entrada e o segundo recebeu três novas entradas.

5.1. Alteração nos sinais de controle

Como já mencionamos, os sinais de controle da arquitetura MIPS receberam dois novos sinais. Um deles denominado inversor, serve para selecionar uma das duas entradas no mux 2:1. Uma das entradas vem direto de um comparador ligado ao banco de registradores, a outra passa ainda por um inversor. Esse novo sinal serve para dar suporte ao Simulador para a execução de instruções de desvio condicional beq e bne.

O outro sinal criado foi para a execução da instrução jal (jump-and-link). Esse sinal denominado link, atribui para o registrador de posição 31 ($ra) o valor do pc incrementado. Contudo, essa atribuição só é feita se esse sinal estiver ativado.

6. Forma de representação do clock

A forma de representação do clock foi pela utilização da estrutura de repetição while.

7. Conclusão

O trabalho foi um grande desafio para o nosso grupo, tivemos muitas dificuldades, mas com elas aprendemos muita coisa, entre elas trabalhar em grupo, pesquisar sobre assuntos que não sabíamos e lembrar de outros que estavam esquecidos.

Através do esforço realizado amadurecemos e nos motivamos para seguir em frente na busca de novos conhecimentos.

Referências Bibliográficas

Patterson, David A. and Hennessy, John L. (2005) “Organização e Projeto de Computadores”. Rio de Janeiro: Editora CAMPUS, 2005.