Scoreboarding

Gustavo Cruz, Otávio Kochi, Vinícius Menossi

Universidade Estadual de Maringá - Centro de Tecnologia - Departamento de Informática

Maringá, 11 de novembro de 2020

***Resumo*—Ao longo dos anos, o processador evoluiu e, junto a ele, as técnicas e algoritmos implementados. O tema deste trabalho é o algoritmo Scoreboarding. Ele é uma aplicação de escalonamento dinâmico, que visa paralelizar a execução de múltiplas instruções que o processador recebe, podendo ser visto como uma evolução direta de um pipeline convencional. Este trabalho contempla uma implementação de um simulador do Scoreboarding.**

***Index Terms*—Scoreboarding, Escalonamento Dinâmico, Artigo, LaTeX.**

I. Introdução

DESDE que o processador surgiu, procura-se encontrar métodos que aumentem a eficiência da execução de instruções. Uma destas técnicas, é a de escalonamento dinâmico, que é transformar a execução de instruções em uma linha de montagem, chamada de ***pipeline***, onde existe uma divisão em etapas menores das instruções. Isto na prática significa aumentar a velocidade de execução, a final, ao invés de demorar 'n' ciclos de **clock** para concluir uma instrução, temos uma instrução sendo executada à cada ciclo quando todos os estágios do pipeline estiverem ocupados. Ao passar do tempo, surgiram técnicas para aprimorar a eficiência de um pipeline, uma delas, foi o **Scoreboarding**, que permite que múltiplas instruções executassem em única estágio, ao contrário do pipeline convencional, onde apenas uma instrução por vez está em cada estágio.

II. Scoreboarding

O Scoreboarding trabalha com múltiplas instruções executando em um único estágio. Isto só é possível de ser aplicado em processadores com múltiplas unidades funcionais. Com isto, é possível fazer a leitura, execução e escrita de diversas instruções em único ciclo. O Scoreboarding traz ainda diversas técnicas para controlar a execução e garantir que não haja problemas durante a execução superescalar.

III. Implementação

A implementação foi dividida em diversos arquivos e estes são citados e explicados a seguir.

A. Arquivo Structs

O arquivo *structs* é onde ocorre as definições das estruturas utilizadas durante todo o programa, bem como os *enums* e as constantes. Para o trabalho, foram utilizadas quatro constantes, dentre elas o **NILL** que foi definido com o valor um negativo, **units_n** que foi definido com o valor cinco, representando a quantidade de unidades funcionais que o *Scoreboarding* possui, **inst_n** que foi definido com o valor sessenta e quatro, representa a quantidade de instruções possíveis apresentadas no *Green Card* do '**MIPS 32**', porém não foram utilizadas todas as instruções possíveis e pôr fim a última constante definida foi o **register_n** possuindo o valor trinta e dois que representa a quantidade de registradores que o *Scoreboarding* possui. Foram utilizados quatro enums, um para os registradores, um para as unidades de função, um para o formato das instruções e outro paras operações utilizadas. Foram definidas 8 tipos de *structs*: ***Instruction_R*** *struct* criada para armazenar instruções do tipo R; ***Instruction_I*** *struct* criada para armazenar instruções do tipo I; ***FunctionUnity*** *struct* criada para simular uma unidade funcional; ***Scoreboarding*** *struct* que possui um vetor com cinco elementos do tipo *FunctionUnity*; ***Pipeline***, *struct* criada para armazenar em que ciclo de clock as instruções foram executadas em determinado estagio do Pipeline; ***Instruction*** *struct* criada para armazenar as instruções, possuindo também uma struct *Pipeline*; ***instConfig*** *struct* para armazenar os valores de latência do arquivo de configuração passado; ***RegisterMemory*** *struct* criada para armazenar os valores do registradores assim como se está sendo utilizado por alguma unidade funcional ou não;

B. Arquivo Principal

O arquivo principal é onde a ordem de execução das mútiplas partes do algoritmo está implementada, bem como é o arquivo responsável pela inclusão de todas as dependências de bibliotecas e estruturas necessárias para o código. O arquivo começa com as inclusões das bibliotecas padrões da linguagem C para a manipulação de *strings*, adição de valores *booleanos*, inclusão de funções matemáticas, estruturas de dados usadas (inclui-se aqui estruturas para as instruções, memória, registradores, unidades funcionais, pipeline e scoreboarding. Além disso, conta ainda com diversos *Enums*, que visam deixar a implementação com valores inteiros) e por último a adição das bibliotecas contendo o código dos arquivos para a conversão do programa de entrada (converter.h), leitura do arquivo de configurações de latências das operações (configFile.h) e da lógica do processamento (processor.h) do Scoreboarding. Após as inclusões, é feito uma coleta dos parâmetros passados para o programa, ou seja, o nome do programa de entrada, o nome do programa de saída, o nome do arquivo de configurações de latências e a quantidade de instruções que serão executadas (representado por tamanho da memória alocada internamente). A execução só prossegue se todos os parâmetros fornecidos estiverem corretos. Com isto, são feitas as chamadas para as

funções responsáveis pela conversão do arquivo de entrada, leitura do arquivo de configurações e, por fim, execução do Scoreboarding.

C. Arquivo Converter

O arquivo para converter as instruções de entrada tem como objetivo traduzir as instruções recebidas em formato de texto para o código correspondente no '**MIPS 32**', para isso, ela lê linha a linha do arquivo de entrada e divide cada um dos parâmetros de entrada das instruções, isto é, a operação e os operadores. Com esta divisão feita e salva, o *converter* identifica qual o tipo da instrução e a salva em sua respectiva estrutura (instruções do tipo R, são armazenadas em uma estrutura para elas, enquanto instruções do tipo I, são armazenadas em outra). Por fim, estas instruções, já em suas respectivas estruturas, são usadas (parâmetro por parâmetro) na conversão para um número inteiro de trinta e dois *bits* que represente a instrução e este número é salvo na estrutura *'Instructions'*.

D. Arquivo ConfigFile

O arquivo *ConfigFile* tem como objetivo obter os dados passados através do arquivo de configuração e armazená-los na estrutura *InstConfig*, para isso, lemos cada linha por vez do arquivo e pegamos a primeira parte que corresponde à instrução, após isso, armazenamos o número inteiro que indica a quantidade de ciclos de *clock* que a instrução necessita para realizar a execução. Tendo o nome da instrução e a quantidade de ciclos necessários, passamos essas informações para o vetor que a estrutura *InstConfig* possui, armazenando em seus devidos lugares.

E. Arquivo Mapping

Este aquivo é responsavel por mapear os registradores e as instruções utilizadas na escrita do arquivo de entrada para o código corresponde no '**MIPS 32**'. Houve duas exceções, nas instruções li e move, pois estas são pseudo-instruções, e, para sanar este problema foram escolhidos dois códigos não utilizados nesse simulador, sendo eles respectivamente o nove (código do addiu) e o trinta e nove (código do nor).

F. Arquivo Processor

Este é o arquivo responsável pela lógica do Scoreboarding, e, por isso, é o maior arquivo do projeto. Ele possui um total de sete variáveis globais que são usadas ao longo de toda a execução, sendo elas: uma memória para os registradores (registerMemory); uma variável para o scoreboarding que guardará o *status* de todas as unidades funcionais durante toda a execução (scoreboarding); uma variável de tamanho dinâmico à quantidade de instruções a serem executadas, onde é guardado todos os dados que dizem respeito às próprias instruções; uma variável do tamanho da memória de registradores para o auxílio durante a etapa de execução; e uma variável contendo as latências para cada tipo de instrução (provindas do arquivo fornecido pelo usuário). Com o auxílio destas variáveis globais, e após a inicialização de todas que forem possíveis, temos um comando de repetição (que vai ser verdadeiro enquanto ainda houver pelo menos uma instrução a ser executada) que chama seis funções para simular o Scoreboarding. O Pipeline é executado em ordem inversa para simular o paralelismo.

A primeira função chamada é a responsável pela escrita dos resultados obtidos, lá é feita a escrita dos resultados (na memória dos registradores) que já foram calculados, caso não haja nenhuma dependência do tipo **WAR**. A unidade funcional responsável pela instrução que acabará de finalizar é marcada para ser 'limpa' e, então, a instrução é marcada como concluída.

Após isso, temos a chamada da função responsável pela etapa de execução, onde é feita a operação correspondente à instrução, levando em conta, ainda, a latência que cada uma leva. Além disso, a execução é realizada para todas as instruções que forem possíveis naquele ciclo de *clock*.

A próxima função chamada é a de leitura de operandos, onde é feita a leitura de todas as instruções que possam ser lidas no *clock* atual, caso não exista nenhuma dependência do tipo **RAW**. Aqui é onde a leitura dos operandos é feita para dentro da estrutura da instrução (o que evita que uma escrita em um dos operandos altere o valor antes da execução da instrução. Aqui é onde os marcadores 'qj', 'qk', 'rj' e 'rk' são atualizados após a leitura ser realizada.

Em seguida, são chamadas duas funções: uma para traduzir a instrução do número de trinta e dois bits na estrutura de uma instrução, isso é, com campos tipados para cada um dos parâmetros da instrução; outra que trata a lógica da emissão da instrução. Na função de emissão é verificado se a instrução na posição **PC** pode ser emitida (em outras palavras, se ela não tem dependência do tipo **WAW**), com isso, toda a lógica de atualização da unidade funcional responsável pela instrução e inicialização de dados de ambos são feitas. Somente se a etapa de emissão tiver sucesso, PC é incrementado.

Após isso, é feita a chamada de uma função responsável pela limpeza de unidades funcionais que desocuparam durante a escrita. Esta função é necessária tendo em vista a sequência invertida dos estágios do pipeline, uma vez que se a limpeza fosse feita no começo, a mesma unidade poderia finalizar uma instrução e já começar outra no mesmo ciclo de *clock*.

Por último, é chamada a função responsável pela impressão do resultado, que irá escrever no arquivo de saída passado como parâmetro de entrada, o estado do scoreboarding em cada ciclo.

REFERÊNCIAS

[1] ZHANG, Zhao. Scoreboarding and Tomasulo Algorithm. 2005. Disponível em: http://users.utcluj.ro/ sebestyen/_Word_docs/Cursuri/SSC_course_5_Scoreboard_ex.pdf. Acesso em: 27 out. 2020.

[2] PATTERSON, David A.; HENNESSY, John L.. Computer Architecture, Fifth Edition: A Quantitative Approach. 5. ed. 2011.

[3] ELSEVIER. MIPS Reference Data. Disponível em: https://inst.eecs.berkeley.edu/ cs61c/resources/MIPS_Green_Sheet.pdf. Acesso em: 27 out. 2020.