Processador com dois núcleos com ambos superescalares

Gabriel T. H. Santos RA:107774, Thiago I. Yasunaka RA:103069

***Resumo*—Em uma execução de pipeline o desempenho e efetividade são muito valorizados e para isto há vários estudos para ocorrer estes aprimoramentos, para isto um dos quais já são usados é o método de *scoreboarding*, tal método será abordado neste projeto, juntamente com a utilização de *threads*. A técnica de *scoreboarding* consiste em um escalonador dinâmico de instruções que permite executar fora de ordem cada instrução desde que exista recursos disponíveis e que não haja dependência de dados. Além do *scoreboarding*, haverá também a implementação de um simulador de *threads* em um processador, que são alternativas para melhorar o desempenho dele. De forma simples, as *threads* funcionam como um processo de execução concorrente, isto é, a capacidade do processador executar diversos processos de forma simultânea, podendo ou não ter memória compartilhada entre eles.**

I. Introdução

ESTE projeto baseia-se na implementação de um simulador da técnica de *scoreboarding* em um processador de dois núcleos, simulando de forma completa os status de cada componente presente nela e em cada ciclo de *clock*. O *scoreboarding* foi usado pela primeira vez no computador *CDC 6600* com o objetivo de organizar dinamicamente um *pipeline* e consequentemente, melhorar o desempenho de um processador. No entanto, neste modelo proposto, a execução fora de ordem das instruções acabaram sendo frequentes. Para que não tenha conflito de dados, o *scoreboarding* verifica minuciosamente cada etapa da sua execução antes de prosseguir para o próximo ciclo de *clock*.

Com o intuito de melhorar ainda mais o desempenho de um processador, foi estudado uma maneira de executar diversos processos simultaneamente, dessa forma, o processador seria capaz de realizar operações de forma concorrente, podendo ou não compartilhar a mesma memórias entre eles. A proposta feita foi a criação de *threads*, como foi dito acima, ela permite executar várias tarefas simultaneamente. De maneira informal, é como se existissem vários processadores dentro de um único processador.

10 de Dezembro de 2020

II. Scoreboarding

Para o processamento, existem alguns pré-requisitos que foram definidos antes da implementação, as figuras 1 e 2 representam a arquitetura do *scoreboarding* que será seguido, sendo mais detalhada sobre os processos que ocorre internamente na figura 2. Portanto, para o *scoreboarding* temos quatro etapas importantes para o caminho desde a instrução até a escrita no registrador, das quais são listadas abaixo:

- Emissão *(Issue)*
- Leitura de Operandos *(Read Operand)*
- Execução *(Execution)*
- Escrita dos Resultados *(Write Results)*

A implementação abordará apenas instruções inteiras, diante disso teremos apenas estas unidades funcionais (UF):

- *1 UF para soma/subtração*
- *2 UF para multiplicação*
- *1 UF para divisão*
- *1 UF para operações lógicas*

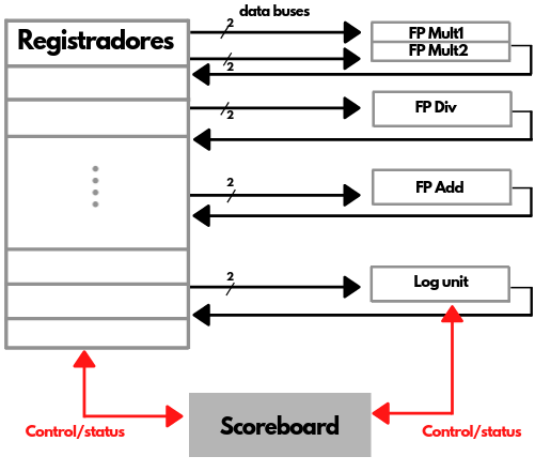

Figura 1: Arquitetura *Scoreboarding*

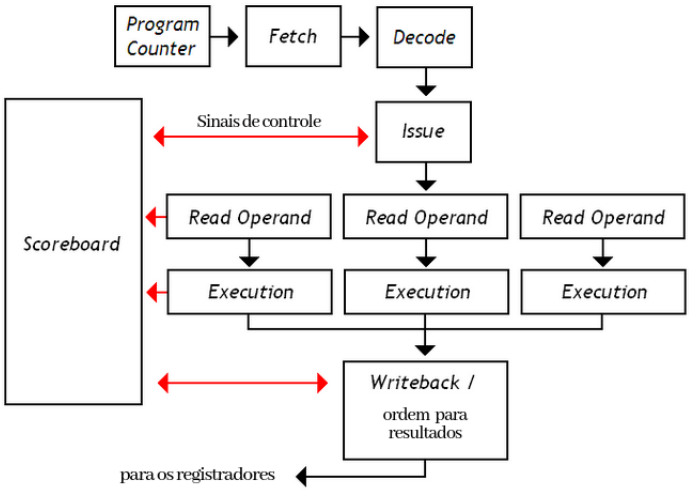

Figura 2: Arquitetura detalhada do *Scoreboarding*

III. Arquitetura Implementada

Pode ser visto a seguir, na figura 3, a representação do diagrama da arquitetura a ser seguida neste projeto para a implementação já com o processador de dois núcleos.

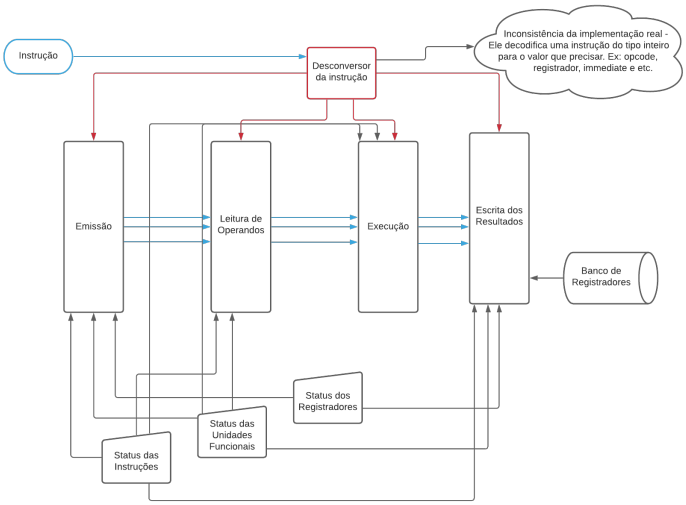

Figura 3: Arquitetura Implementada

Analisando a figura 3, observa-se que existem algumas inconsistências em relação a verdadeira implementação de um *scoreboarding*, é fácil observar que existe um componente destacado em vermelho chamado "desconversor da instrução". Este item no diagrama faz a desconversão de uma instrução (representado por um número inteiro, ou seja, utilizando 32 *bits*) para qualquer parâmetro que ela necessitar. Suponha que exista uma instrução `addi $s1 $s0 10`, utilizando este desconversor é possível resgatar qualquer um dos quatro parâmetros desta instrução, para isso basta utilizar uma função específica para realizar a operação. Um meio para contornar esta divergência, porem não abordado seria executar as operações de forma invertida, ou seja, da escrita dos resultados para a emissão, juntamente com *structs* auxiliares.

A arquitetura consiste em quatro etapas principais: Emissão, leitura de operandos, execução e escrita dos resultados. Além destas quatro etapas necessárias para a simulação do *scoreboarding*, é preciso outros quatro componentes que controlam os estados das instruções e dos registradores, para isso foi definido diversas *structs* (escritas em linguagem C) que representam elas, sendo: status das instruções, status das unidades funcionais, status dos registradores e o banco de registradores, este último utilizado para armazenar os valores que cada registrador possui.

A figura 3 representa o processamento que apenas um núcleo do processador faz, porém, o projeto final trabalha com dois núcleos de processamento simultâneos. Para isso, utilizou-se a biblioteca em C chamada *pthreads*, com ela é possível executar diversos fluxos de códigos de forma paralela, por exemplo, executar mais de uma função ao mesmo tempo. A figura 4 abaixo mostra um diagrama visual para exemplificar um processador utilizando dois núcleos com *scoreboarding* implementado.

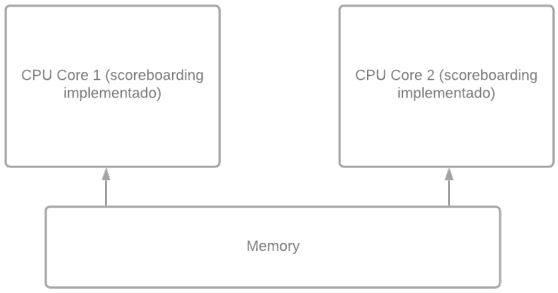

Figura 4: Arquitetura com dois núcleos

A figura 4 mostra uma arquitetura desenvolvida com dois núcleos, por sua vez, estes núcleos são interligados por uma memória compartilhada. Como dito acima, a implementação das *threads* foi codificada utilizando a biblioteca *pthread* na linguagem C, o que representaria essa memória compartilhada no código seriam as variáveis globais. As variáveis globais no programa implementado são somente as variáveis recebidas por parâmetros, sendo elas: caminho para o arquivo de configuração, caminho dos programas 1 e 2, arquivo de saída para os programas 1 e 2 e a quantidade de instruções para os programas 1 e 2.

A ideia por trás de simular dois núcleos simultaneamente são definidos logo que o programa inicia. A partir dos parâmetros recebidos ocorrem a validação deles e após isso é chamado uma função `execute_pthread` em que será responsável para as execuções seguintes, logo então é criado dois fluxos utilizando a função `pthread_create()`, nela são passados os argumentos necessários e a função a ser executada, no caso, a execução do *scoreboarding*. O fim da execução das *threads* é realizado pela função `pthread_join()`.

IV. Uma solução

A inicialização do *scoreboarding* ocorre a partir do arquivo *pthreads.c*, nela são introduzidas as criações das *threads* para a execução concorrente e além disso, durante a inicialização é organizado toda a configuração e início da execução do *pipeline*.
A partir disso, referencia-se para chamada do processador (*processador.c*), onde ficará a execução do escalonador e criação de todo o fluxo do *scoreboarding*.
Para a implementação na linguagem *C*, foram definidos *structs* bases para cada etapa da arquitetura, isto quer dizer que cada componente do *scoreboarding* recebeu uma estrutura própria para melhor representá-la. Foram estas *structs*:

- *functional_unit_status_table*
- *instruction_status_linked*
- *register_database*
- *register_result_status_table*
- *config*
- *I*, *R*

. A organização arquitetural utilizada dentro da execução do processador foi definida em separar-se em quatro funções principais, sendo elas nomeadas com a mesma nomenclatura das quatro etapas presentes no *scoreboarding*. Por sua vez, por meio de passagem de parâmetros por referência, cada uma

destas funções trabalha e implementa a lógica das execuções do *scoreboarding* separadamente. Durante o desenvolvimento foram utilizadas diversas funções auxiliares que foram armazenadas dentro do diretório */utils*, neste diretório existem funções que são utilizadas mais de uma vez. Este modelo foi necessário a fim de organização e remoção de código duplicado. Existem funções reutilizadas em outros arquivos também, como no caso do *get's* e *set's* nos arquivos do diretório */components* do projeto. Um resumo da organização do projeto é definido na lista abaixo:

- */components*: Definição das tabelas presentes no *scoreboarding* e as unidades(*mult1, mult2, add, log, divide*) que ela possui, sendo utilizado também um *empty* para representar a não utilização.
- */config*: Integração com o arquivo de configuração que é recebido por parâmetro para a memória.
- */examples*: Exemplos utilizados para os testes durante o desenvolvimento,
- */operations*: Implementação das operações que o MIPS (de acordo com a documentação do projeto) realiza e integração com o banco de registradores.
- */prints*: *Prints* para a escrita no arquivo de saída para cada *clock* e também análise dos dados resultantes.
- */types*: Tipagem e definição das constantes a ser utilizado no projeto como um todo, por exemplo: Registradores, operações (*addi*, *add*, *sub*, *div*, *etc.*) e tipos. Os valores das constantes seguiu a documentação oficial do *green-card* do *MIPS*, salvo as exceções para as operações LI e MOVE, que receberam os valores 28 e 30, e do tipo I e R, ambos respectivamente.
- */utils*: Este diretório contém diversas funções auxiliares que foram utilizados em todo o projeto, necessário para evitar possíveis duplicações de código.
- *conversor.h*: Faz todas as conversões e desconversões necessárias para o conjunto de instruções e atribui à memória do programa.
- *main.c*: Verifica os parametros passados, caso haja algum de modo incorreto há o retorno e a não continuação do programa, caso contrario, aqui também se faz a chamada da execução das *threads* e então a partir dele, como já comentado, o *scoreboarding*.
- *processador.c*: O *scoreboarding* e toda a sua implementação depende desse arquivo. Aqui é implementado toda a lógica necessária, as inicializações dos componentes necessários para a simulação do *scoreboarding*, principalmente por causa das *threads*, houve a necessidade de se implementar ao mesmo arquivo. Como também grande parte das funções auxiliares são utilizadas em funções que pertencem a este arquivo. Sendo as suas principais funções de cada etapa da arquitetura (*executeIssue()*, *readOperands()*, *executeOperands()*, *writeResult()*) desenvolvidas dentro de um laço de repetição *while* para simular as operações enquanto não há o término de todas as escritas de resultado.

.

V. Análise e Discussão

Para validar a implementação do simulador de *scoreboarding* com a integração das *threads*, foram executadas diversos testes no decorrer do desenvolvimento por etapas e também por funções, utilizando também *debugger* para ver de fato quais e onde cada campo estava sendo modificado. A partir disso, foram criadas arquivos para o conjunto de instruções, com *mnemonios* com base no material visto em aula e também pela *internet* para confirmar, resolver os problemas e também possíveis incompatibilidades no algoritmo. Houve então resultados compatíveis com os esperados, sem erros de leitura ou preenchimento das tabelas, a ordem gravadas com os *clocks* sendo executadas corretamente e as verificações de dependências novamente corretas, e ao final as escritas nos registradores com os valores esperados, contando com a operação por ela desenvolvida.

Um ponto a ser destacado e que não está presente na documentação é a passagem do parâmetro `-n` e `-m` na compilação do algoritmo, isto foi definido em algumas aulas anteriores à data de entrega. Estes dois parâmetros representam a quantidade de instruções que o conjunto de instruções possui, para o programa1 e para o programa2, respectivamente. Dessa forma, um exemplo completo de execução pode ser visto abaixo (Essa execução desconsidera qualquer vínculo com o *makefile*):

1) Vá a raiz do projeto e execute os comandos abaixo;
2) `gcc main.c -D_REENTRANT -lpthread;`
3) `./a.out -n <num of instr program1> -m <num of instr program2> -c <config.txt> -o <outputProgram1.txt> -q <outputProgram2.txt> -p <program1.txt> -r <program2.txt>;`
4) As saídas dos programas deverão estar nos arquivos `<outputProgram1.txt>` e `<outputProgram2.txt>` ;

Cada argumento durante a passagem dos parâmetros possuem um significado para a execução do programa, ela é mais detalhada nos tópicos abaixo, a ordem que é passada não é relevante:

1) `-c`: Caminho para o arquivo de configuração, esta configuração é utilizada tanto no programa 1 quanto no programa2;
2) `-n`: Quantidade de instruções para o programa 1;
3) `-m`: Quantidade de instruções para o programa 2;
4) `-o`: Arquivo de saída do simulador para o programa 1;
5) `-q`: Arquivo de saída do simulador para o programa 2;
6) `-p`: Caminho para o código que será executado que representa o programa 1;
7) `-r`: Caminho para o código que será executado que representa o programa 2;

VI. Conclusão

A implementação do simulador de um processador de dois núcleos, com cada um destes utilizando o método de *scoreboarding* e sendo implementado com a integração das *threads*, pode-se dizer que da organização do algoritmo até a saída desejada pode ser dita como trabalho realizado com sucesso. Alguns aspectos de desenvolvimento podem ser incompatíveis,

isto é, detalhes mínimos do que um *scoreboarding* faz na realidade, como por exemplo a execução em paralelo de cada etapa do pipeline, em que mantivemos uma ordem para estes e de certas *structs* auxiliares não abordadas para a passagem do início ao fim de cada, escolhendo apenas um desconversor neste processo. Apesar destas possíveis desavenças, o simulador faz o seu papel com muito êxito.

REFERÊNCIAS

[1] W. Stallings, "Arquitetura E Organização De Computadores", 1987.

[2] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", 1993.

[3] https://www.embarcados.com.br/instrucao-slt-no-mips/

[4] https://en.wikibooks.org/wiki/MIPS_Assembly/Pseudoinstructions

[5] https://www.ic.unicamp.br/~pannain/mc542/aulas/ch3_arq.pdf

[6] http://users.utcluj.ro/~sebestyen/_Word_docs/Cursuri/SSC_course_5_Scoreboard_ex.pdf

[7] http://www2.dcc.ufmg.br/disciplinas/aeds3_turmaN/pthreads.pdf