Relatório: Modelo de Pipeline para Microprocessador RISCO

Gabriel Igor[1], Luiz Eduardo[1] e Mateus Santiago[1]

[1]Estudantes do Bacheralado em Tecnologia da Informação na Universidade Federal do Rio Grande do Norte (UFRN), Brasil.

Abstract— This report proposes a pipeline model for the RISCO microprocessor, explaining its development and operation through diagrams.

I. INTRODUÇÃO

Pipelining é uma técnica de implementação em que várias instruções são sobrepostas na execução. Por exemplo, ao lavar roupas sujas, o procedimento sem pipelining seria colocar a trouxa de roupas sujas na lavadora; em seguida colocar a trouxa na secadora; passar as roupas e ao terminar de passar, guardar as roupas. Ao fim deste procedimento, reiniciamos o processo com outra trouxa de roupas.

A técnica com pipeline leva um tempo bem menor, pois assim que a primeira trouxa é inserida na secadora, uma outra já é inserida na lavadora. Esse procedimento se repete para cada uma das etapas do processo. Ou seja, os estágios do procedimento (lavar, secar, passar e guardar) podem ser executados simultaneamente. Desde que haja recursos separados para cada estágio, pode-se usar pipeline para as tarefas.

O motivo desta técnica ser mais rápida é que tudo está trabalhando em paralelo, de modo que mais trouxas são terminadas por hora. O pipelining melhora a vazão do sistema de lavanderia sem melhorar o tempo para concluir uma única trouxa. Logo, a técnica de pipelining não diminuiria o tempo para concluir uma única trouxa de roupas, porém, ao se utilizar um conjunto de trouxas para lavar, a melhoria na vazão diminui o tempo total para concluir o trabalho.

Entretanto, a técnica de pipelining apresenta algumas desvantagens. Existem situações em que a próxima instrução não pode ser executada no ciclo de *clock* seguinte. Esses eventos são chamados de *hazards* ou conflitos e podem ser de três tipos: estruturais, de dados e de controle. Existem estratégias de soluções para cada tipo de *hazard*, que devem ser implementadas com cuidado. Além disso, quanto maior o número de estágios em um pipeline, melhor será a eficiência do processador, aproveitando os componentes ociosos, entretanto, a complexidade para implementar e tratar conflitos também aumenta. Vale salientar também que existe um limite de estágios em que uma instrução pode ser dividida, pois microinstruções muito curtas tornam a aplicação do pipeline inviável e aumentam o custo do processo.

Levando-se em conta os fatores já mencionados, este relatório tem como principal objetivo apresentar uma possível implementação de pipeline para o microprocessador RISCO.

II. IMPLEMENTAÇÕES DE PIPELINE

A implementação do pipeline pode ser realizada de diversas formas, alterando-se a quantidade de estágios, a divisão de micro-instruções por estágio e as estratégias para solucionar conflitos estruturais, de dados e de controle. A decisão sobre como essas alterações devem ser implementadas são de responsabilidade do desenvolvedor.

Alguns exemplos de implementações podem ser visualizadas abaixo.

2.1. Pipeline de 2 estágios

Este modelo de pipeline geraria dois conjuntos de micro-instruções. Um exemplo seria:

- Busca de instrução e decodificação de instrução
- Leitura dos registradores, execução, acesso à memória e escrita do resultado

Esse modelo teria como principais vantagens o seu baixo custo e a sua simplicidade, facilitando a sua implementação. Entretanto, por possuir apenas dois estágios, não causaria um impacto tão grande na eficiência do processador.

2.2. Pipeline de 4 estágios

Este modelo dividiria a instrução em 4 conjuntos de micro-instruções, como por exemplo:

- Busca de instrução
- Decodificação de instrução e leitura dos registradores
- Execução
- Acesso à memória e escrita do resultado

Este modelo apresenta características opostas ao pipeline de 2 estágios. Ele teria como vantagem a sua maior eficiência, devido a sua maior quantidade de estágios, entretanto, sua implementação seria mais complicada e a necessidade de se adotar estratégias adequadas para lidar com conflitos acarretaria em um maior custo de sua produção.

2.3. Pipeline de 3 estágios

O pipeline com 3 estágios apresenta um comportamento intermediário entre os dois modelos citados anteriormente. Apresenta uma boa eficiência e não possui uma alta complexidade, quando comparado com o modelo de 4 estágios, balanceando-se o custo e o benefício do processo. Por tais razões, este modelo foi o escolhido para o desenvolvimento do pipeline para o microprocessador RISCO, dividindo-se a instrução da seguinte forma:

- Busca de instrução
- Decodificação de instrução, leitura dos registradores e execução
- Acesso à memória e escrita do resultado

III. MODELO PROPOSTO

A implementação do microprocessador RISCO foi feita usando-se a biblioteca SystemC da linguagem C++. O modelo Parte Operativa/Parte de Controle (PO/PC) para este tipo de processador necessita de alguns componentes principais.

3.1. Componentes da PO por estágio

3.1.1. Busca de Instrução

No primeiro estágio, os seguintes componentes são usados:

- Contador de Programa
- Memória de Instruções
- Somador
- Multiplexador 1

3.1.2. Decodificação da instrução, leitura dos registradores e execução

No segundo estágio, os seguintes componentes são usados:

- Decodificador de instruções
- Verificador de conflitos de dados
- K_Constants (Extensor de sinal)
- Banco de Registradores
- Multiplexadores 2 e 3
- ULA

3.1.3. Acesso à memória e escrita do resultado

No terceiro estágio, os seguintes componentes são usados:

- Multiplexadores 4, 5, 6 e 1
- Memória de dados
- Banco de Registradores

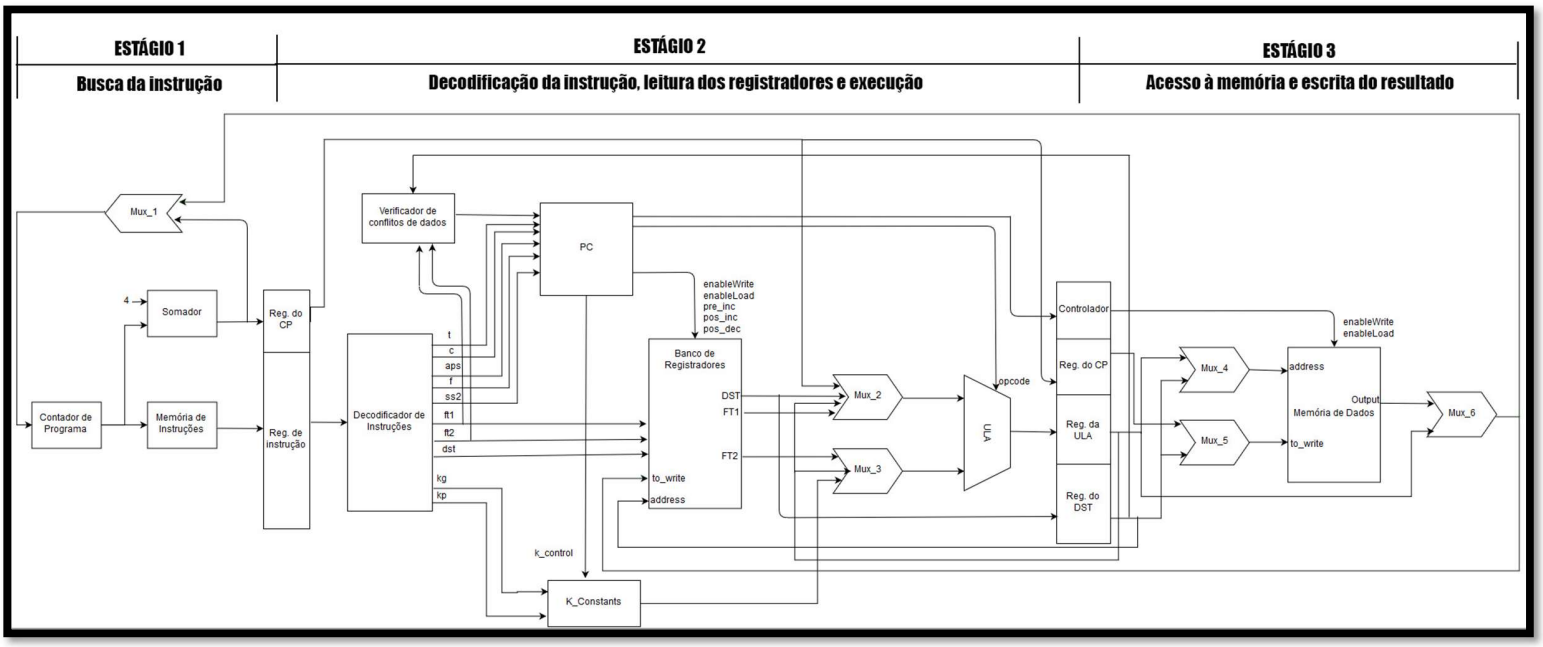

Figura 1: Processador RISCO com pipeline de 3 estágios

Para uma melhor compreensão do desenvolvimento do projeto, a figura 1 ilustra como os componentes citados se comunicam entre si e a divisão dos estágios. Entre os estágios estão presentes os registradores dos barramentos que possibilitam a aplicação da técnica de pipelining.

Vale salientar que para facilitar a interpretação no diagrama, os fios que saem da PC para os multiplexadores foram omitidos. Entretanto, esses fios estão implementados no código do processador.

Na figura acima, pode-se observar que entre os estágios 1 e 2 estão presentes dois registradores:

- Registrador do CP: Guarda o valor do contador do programa após passar o somador;
- Registrador de instrução: De acordo com o valor do contador de programa, uma determinada instrução é selecionada na memória de instruções e encaminhada para esse registrador.

Já entre os estágios 2 e 3 estão presentes os seguintes registradores:

- Controlador: Guarda os sinais de controle que são encaminhados para os componentes do processador presentes no estágio 3.
- Registrador do CP: Armazena mesmo sinal do registrador de CP entre os estágios 1 e 2, como explicado anteriormente.
- Registrador da ULA: Armazena resultado gerado pela ULA após operação lógico-aritmética.
- Registrador do DST: Armazena o valor do registrador de destino, vindo do banco de registradores.

Esses registradores mencionados compõe os barramentos necessários para a implementação da técnica de pipelining e auxiliam o tratamento dos conflitos.

3.2. Parte de controle

A parte de controle irá gerar todos os sinais de controle necessários para a execução das instruções. Como já citado anteriormente, cada um dos multiplexadores presentes no processador recebem um sinal vindo da PC, entretanto, esses sinais foram apenas omitidos, apesar de estarem presentes na implementação do código. Os

componentes, seus sinais de controle e suas respectivas funções podem ser vistos abaixo:

3.2.1 Banco de Registradores

- enableBankByPC: Permite acesso ao banco de registradores ou não.
- writeLoadInRegisterByPC: Operação de escrita ou leitura no banco de registradores.
- preIncByPC: Segundo operando (FT2) será pré-incrementado ou não.
- posIncByPC: Segundo operando (FT2) será pós-incrementado ou não.
- posDecByPC: Segundo operando (FT2) será pós-decrementado ou não.

3.2.2 Memória de dados

- enableMemory: Permite acesso a memória de dados ou não.
- writeMemory: Se é uma operação de escrita na memória de dados.

3.2.3 ULA

- opCodeToUla: Sinal que define qual operação lógico-aritmética será realizada.
- apsToActivateUla: Indica se a palavra de status do processador deve ser atualizada ao fim da instrução.

3.2.4 Barramentos

Os barramentos usados para implementação do pipeline possuem uma série de registradores já citados anteriormente. Esses registradores vão receber dois sinais da PC.

- pause: Pausa a transmissão do sinal, deixando de atualizar o valor no registrador
- bubble: Indica uma possível ocorrência de uma bolha. Caso seja verdadeiro, os valores nos registradores são zerados. É usado principalmente em instruções de jump e sub-rotina.

3.3. Conflitos (*Hazards)*

3.3.1. Conflitos estruturais

Os conflitos estruturais ocorrem em situações que o hardware não pode admitir a combinação de instruções que queremos executar em um mesmo ciclo de clock.

Para se evitar os conflitos estruturais, ao invés de se utilizar uma única memória para dados e instruções, foram usadas componentes distintas: uma memória de instruções, presente no estágio 1 e uma outra memória para os dados, presente no estágio 3.

Além disso, dependendo das micro-instruções a serem executadas em cada estágio, poderia surgir uma situação em que o banco de registradores fosse usando tanto para escrita quanto para leitura. Dessa forma, para permitir esse "paralelismo", no início do clock é feita a escrita para só depois ser realizada a leitura do conteúdo dos registradores.

3.3.2. Conflitos de dados

Os conflitos de dados ocorrem quando o pipeline precisa ser interrompido porque uma etapa precisa esperar até que outra seja concluída. Eles surgem quando uma instrução depende de uma anterior que ainda está no pipeline.

Vamos usar como exemplo as seguintes instruções:

- ADD $0, $2, $4
- SUB $5, $0, $6

A instrução SUB necessita usar o resultado da instrução ADD. Para solucionar tal problema, no nosso modelo foi feita uma realimentação da ULA. Assim após ter seu resultado armazenado no registrador da ULA do pipeline, esse resultado é encaminhado para os Muxes 2 e 3 antes mesmo de ser escrito.

Além disso, o valor do registrador de DST presente no pipeline é encaminhado para o Verificador de conflitos de dados. No verificador são checadas duas condições:

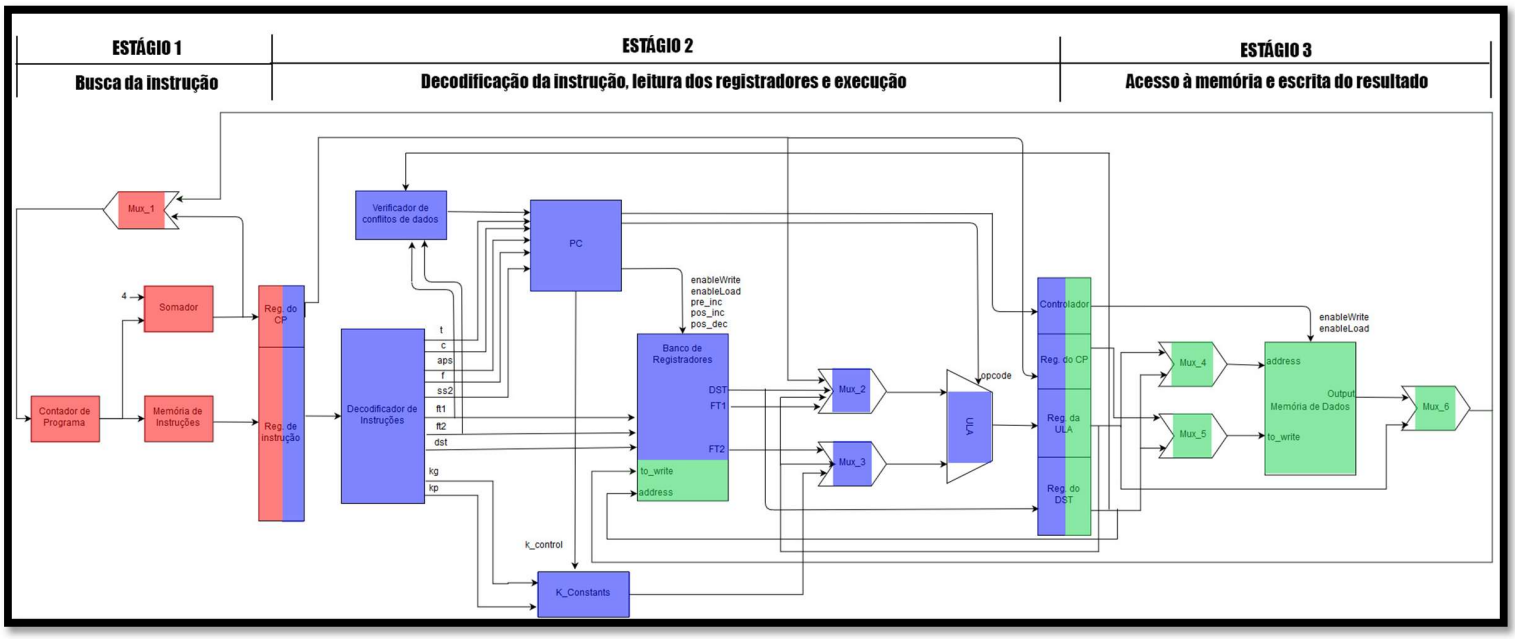

Figura 2: Exemplo de diferentes instruções no processador

- Se o valor de DST da instrução anterior é igual ao valor do primeiro operando da instrução atual.
- Se o valor de DST da instrução anterior é igual ao valor do segundo operando da instrução atual.

Caso qualquer uma das condições seja verdadeira, é emitido um sinal para a Parte de Controle (PC), confirmando que existe um conflito de dados. A PC por sua vez, tomará as devidas providências para enviar os sinais corretos para os multiplexadores e permitir que o processador continue funcionando normalmente

3.3.3. Conflitos de controle

Conflitos de controle surgem da necessidade de tomar uma decisão com base nos resultados de uma instrução, enquanto outras estão sendo executadas. Ou seja, surgem a partir de instruções de desvio.

Para se evitar conflitos de controle, são gerados atrasados de 2 ciclos caso a condição seja verdadeira, sendo necessário zerar os registradores dos barramentos, através dos sinais de pause e bubble.

IV. EXEMPLO

Para exemplificar o funcionamento do pipeline, vamos observar a Figura 2. Cada cor na figura representa uma instrução diferente sendo executada em um determinado estágio. Este exemplo trata-se de uma situação onde não existem conflitos de dados ou controle ocorrendo, ou seja, uma situação ideal.

No entanto, caso ocorressem qualquer um dos conflitos mencionados, as devidas ações seriam executadas. Um conflito de dados, como já mencionado anteriormente, implicaria na geração de um sinal do verificador de conflitos de dados, normalizando a situação, enquanto que um conflito de controle acarretaria em um atraso de dois ciclos e os registradores seriam zerados.

V. CONCLUSÕES

O desenvolvimento deste trabalho nos possibilitou ter um melhor entendimento sobre o funcionamento de um processador, como um pipeline deve ser implementado e os cuidados que se deve ter durante a execução das instruções para não haver conflitos.

Além disso, a disciplina nos deu a possibilidade de conhecer e nos familiarizar com a biblioteca SystemC da linguagem C++. Fazendo-se uma comparação com a disciplina de Circuitos Lógicos, acredito que a utilização do SystemC ao invés do VHDL facilitou a resolução dos problemas propostos pelos trabalhos da disciplina. No entanto, foram enfrentadas dificuldades principalmente a dois aspectos:

- Formas de conexão: Implementar maneiras diferentes de conectar os componentes usando, por exemplo, barramentos, foi um pouco complicado, levando-se em consideração que desde Circuitos estamos habituados apenas a realizar o port mapping entre os diferentes componentes. Talvez exemplos de códigos facilitassem a compreensão nesse aspecto.
- Parte de controle: A implementação da parte de controle torna-se mais complexa a partir do momento que a quantidade de componentes, e consequentemente sinais, vão aumentando muito.

Devido a esses fatores mencionados, consequentemente, a execução de simulações durante as unidades da disciplina muitas vezes foi difícil.

De maneira geral, lidar com um projeto tão grande foi uma boa experiência para os alunos, necessitando de uma grande atenção durante cada etapa de desenvolvimento. Além disso, a divisão do trabalho durante as unidades, ao invés de ser passado somente durante a terceira unidade da disciplina, foi uma boa decisão, de forma que os grupos conseguiram se manter em dia para cumprir os requisitos do projeto e não se sentirem tão sobrecarregados durante a terceira unidade.

IV. REFERÊNCIAS

[1] PATTERSON, D.A. & HENNESSY, J. L. Organização e Projeto de Computadores – A Interface Hardware/Software. 3ª ed. Campus, 2005.