Universidade de Brasília
Departamento de Ciências da Computação
Organização e Arquitetura de Computadores (CIC0099)

Relatório:

Pipeline Risc-V

David Herbert de Souza Brito Mat: 20/0057405
Gustavo Almeida Valentim Mat: 20/2014468

Professor:
Ricardo Pezzuol Jacobi

27 de Julho de 2023

Conteúdo

1 Introdução

O presente relatório descreve o projeto desenvolvido para a disciplina Organização e Arquitetura de Computadores, que tem como objetivo a implementação de uma versão do processador RISC-V Pipeline em VHDL. O projeto busca a construção de uma arquitetura de pipeline básica do processador RISC-V, com ênfase na execução eficiente de instruções e no aumento do desempenho.

O processador RISC-V é uma arquitetura de conjunto de instruções reduzido (RISC) baseada em uma especificação aberta, o que a torna uma opção atraente para diversos projetos de hardware. A implementação em pipeline, por sua vez, é uma técnica avançada para o aumento do desempenho dos processadores, permitindo a execução paralela de instruções e otimizando o tempo de processamento.

A proposta deste projeto é proporcionar aos estudantes uma experiência prática no desenvolvimento de um processador RISC-V em VHDL, permitindo a aplicação dos conceitos teóricos abordados em sala de aula de forma concreta. Além disso, a implementação em pipeline permite explorar os desafios e benefícios da execução paralela de instruções, bem como a importância da arquitetura de computadores no contexto de sistemas computacionais modernos.

Neste relatório, serão detalhadas as etapas do projeto, desde a concepção da arquitetura até a implementação em VHDL. Serão abordados aspectos relacionados à organização dos estágios do pipeline, às instruções suportadas pelo processador, bem como aos controles e sinais utilizados para garantir o correto funcionamento do processador.

Ademais, para facilitar o acompanhamento e contribuição, o código-fonte do projeto está disponível no **GitHub**, onde interessados podem acessar os arquivos, sugestões e colaborar com o desenvolvimento contínuo do projeto.

Com isso, espera-se que este projeto proporcione uma oportunidade valiosa de aprendizado e aprofundamento no campo da arquitetura de computadores, permitindo aos estudantes uma compreensão mais sólida dos princípios fundamentais que regem o funcionamento dos processadores modernos.

2 Descrição do Projeto

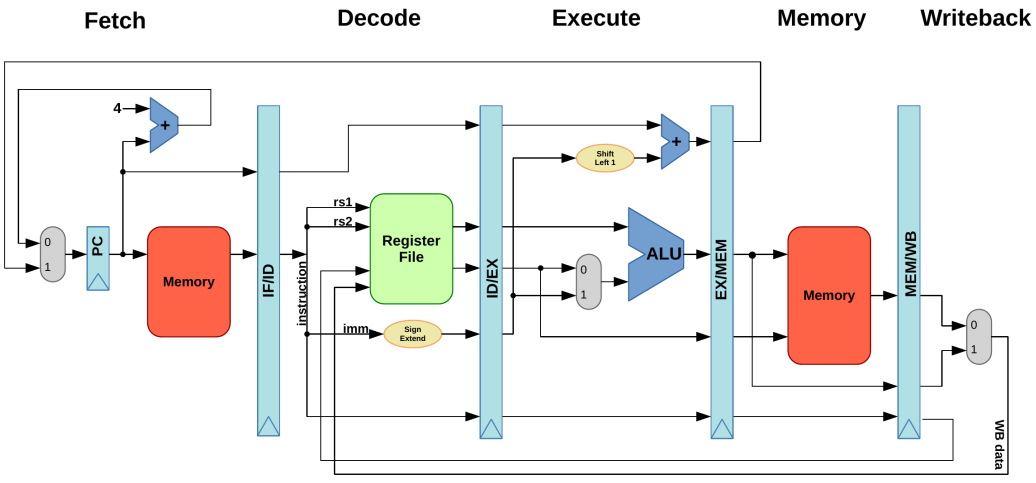

Figura 1: Arquitetura do Pipeline Risc-V

O RISC-V Pipeline é uma arquitetura de processador baseada na ISA (Arquitetura de Conjunto de Instruções) RISC-V, que visa melhorar o desempenho por meio da execução de múltiplas instruções simultaneamente, dividindo o processo de execução em estágios indepen-

dentes. Esse conceito de pipeline permite otimizar o tempo de processamento, tornando o processador mais eficiente.

2.1 Estágios do Pipeline

- Estágio de Busca de Instrução (IF): Nesse estágio, a instrução é buscada na memória de programa (memória ROM) usando o contador de programa (PC). A instrução encontrada será decodificada e encaminhada para o próximo estágio.
- Estágio de Decodificação de Instrução (ID): Nesse estágio, a instrução é decodificada, e os operandos necessários para a sua execução são buscados, como os registradores de origem (rs1, rs2), o registrador de destino (rd) ou os imediatos.
- Estágio de Execução (EX): Nesse estágio, a instrução é efetivamente executada. Isso inclui operações aritméticas, lógicas, de deslocamento e controle de fluxo, como somar, multiplicar, deslocar, comparar e realizar operações de saltos condicionais.
- Estágio de Acesso à Memória (MEM): Nesse estágio, ocorre o acesso à memória para leitura ou escrita de dados, dependendo da instrução em execução, como carregar um valor da memória ou armazenar um resultado de volta na memória.
- Estágio de Escrita de Resultado (WB): Nesse estágio, o resultado final da instrução é escrito de volta no banco de registradores, atualizando os valores dos registradores de destino.

2.2 Descrição do Projeto

A implementação VHDL do projeto consiste na descrição de cada módulo do processador e na interligação entre eles por meio de sinais. A parte operativa do RISC-V é baseada em dados de 32 bits, incluindo memória, registradores, ULA e instruções. Para facilitar a organização, optou-se por utilizar memórias com 8 bits de endereço, o que permite um espaço de endereçamento de 256 palavras para programa e dados.

2.3 Principais Módulos

- PC (Contador de Programa): Um registrador de 32 bits responsável por armazenar o endereço da próxima instrução a ser buscada na memória de instruções.
- Memória de Instruções (MI): Uma memória ROM que armazena o código a ser executado pelo processador. Cada endereço da memória contém uma instrução de 32 bits.
- Memória de Dados (MD): Uma memória RAM que armazena dados que podem ser lidos e/ou escritos pelo processador.
- Banco de Registradores (XREG): Consiste em 32 registradores de 32 bits, sendo o registrador índice zero, XREG[0], uma constante que sempre retorna zero e não pode ser escrito.
- Unidade Lógico-Aritmética (ULA): Realiza operações lógicas e aritméticas em dados de 32 bits, fornecendo resultados com 32 bits e um sinal de saída ZERO para indicar que o resultado é zero. As operações suportadas pela ULA incluem adição, subtração, operações lógicas como AND, OR, XOR, deslocamentos (shifts) e comparações.

O projeto busca proporcionar uma experiência prática aos estudantes no desenvolvimento de um processador RISC-V em VHDL, permitindo a aplicação dos conceitos teóricos em um projeto concreto e incentivando a compreensão dos princípios fundamentais da arquitetura de computadores.

3 Implementação detalhada do RISC-V Pipeline

O funcionamento do nosso pipeline é baseado em um fluxo de dados eficiente que permite a execução simultânea de várias instruções, dividindo o processo de execução em estágios independentes. Primeiramente, todas as instruções são armazenadas em um arquivo de texto chamado "ROM_inst.txt". A partir desse arquivo, a memória de instruções é carregada, armazenando todas as instruções em um vetor.

O processo de execução inicia-se com o contador de programa (PC), que é responsável por apontar para a próxima instrução a ser buscada na memória de instruções. O PC normalmente é incrementado de 4 em 4 a cada batida de clock, permitindo que a próxima instrução seja buscada sequencialmente. Isso não se aplica caso a instrução seja de salto.

Na fase de Decodificação (ID), os sinais de controle são extraídos pelo módulo de Controle, que utiliza o Opcode e outros sinais da instrução para determinar as operações a serem realizadas. Para exemplificar esse processo, consideremos as instruções JAL (Jump and Link) e JALR (Jump and Link Register).

No caso da instrução JAL, o Controle identifica o opcode correspondente (0x6F), e com base nessa informação, define os sinais de controle apropriados. O sinal "jal"é configurado para indicar que o PC será atualizado devido ao salto (Jump), permitindo que a próxima instrução seja buscada em outro endereço. O sinal "regWrite"é ativado para que o endereço de retorno (PC+4) seja escrito de volta no registrador de destino (rd). Além disso, o sinal "aluSrc"é indiferente para essa instrução, pois ela não necessita da ULA para ser executada, e o sinal "branch"é desativado pois não é uma instrução de salto condicional. Já o sinal "resultSrc"é o seletor do mux do write back (WB), o qual informa qual saída será escrita no banco de registradores.

Já no caso da instrução JALR, o Controle identifica o opcode e a funct3 (correspondentes a 0x67 e 0x0, respectivamente). Com base nessas informações, os sinais de controle adequados são configurados. O sinal "jal"é novamente utilizado para indicar que o PC será atualizado devido ao salto (Jump), enquanto o sinal "aluSrc"é ativado para indicar que o valor imediato será utilizado na ULA. O sinal "regWrite"é ativado para que o endereço de retorno (PC+4) seja escrito de volta no registrador de destino (rd), o sinal "branch"é desativado pois não é uma instrução de salto condicional. A ULA, nesse caso, realizará uma soma do valor do registrador com o valor imediato. Já o sinal "resultSrc"é o seletor do mux do write back (WB), o qual informa qual saída será escrita no banco de registradores.

Essas informações de controle são transmitidas de estágio para estágio por meio dos registradores internos do pipeline. Cada estágio realiza sua respectiva operação e envia os resultados para o próximo estágio até chegar ao último estágio (Write Back - WB). No WB, o resultado final é escrito de volta no banco de registradores.

Dessa forma, o pipeline executa as instruções de forma rápida e eficiente, aproveitando ao máximo a capacidade de processamento do processador RISC-V. No entanto, é importante ressaltar que nosso pipeline é uma versão básica e não incorpora melhorias avançadas, como forwarding e hazard detection, que podem ser implementadas para otimizar ainda mais o desempenho do processador.

Nossa CPU é projetada para executar um pipeline básico, dividindo o processamento de instruções em cinco estágios: Fetch (busca), Decode (decodificação), Execute (execução), Memory (acesso à memória) e Write Back (escrita de volta). Cada estágio realiza uma parte específica do processamento, permitindo a execução simultânea de múltiplas instruções e melhorando o desempenho geral da CPU.

O funcionamento do pipeline começa na fase de Fetch (IF), onde a próxima instrução é buscada na memória de instruções com base no valor do contador de programa (PC).

Em seguida, na fase de Decode (ID), a instrução é decodificada, e seus sinais de controle são extraídos pelo módulo de Controle. Esses sinais indicam as operações a serem realizadas em etapas posteriores. Além disso, os registradores de origem e destino para a instrução são

identificados durante essa fase.

No estágio de Execute (EX), tem o somador que calcula o endereço de salto do jal, e o controle a parte ALU (Unidade Lógica e Aritmética) realiza a operação da instrução, como adição, subtração, lógica, etc. Os operandos necessários para a operação são obtidos dos registradores, e o resultado é calculado. O controle da ALU é determinado com base no tipo de operação presente na instrução.

O estágio de Memory (MEM) é responsável por acessar a memória quando necessário. Isso ocorre, por exemplo, em instruções de load ou store, onde dados são lidos ou escritos na memória principal. O endereço de memória a ser acessado é calculado com base no resultado da ALU, e o dado é lido ou escrito na memória.

Por fim, no estágio de Write Back (WB), o resultado final da instrução é escrito de volta no banco de registradores, substituindo o valor anterior no registrador de destino (se a instrução tiver um destino).

Ao combinar todos esses componentes e estágios, nossa CPU é capaz de executar uma sequência de instruções de maneira eficiente e simultânea, aumentando o desempenho geral da execução dos programas. No entanto, é importante destacar que essa implementação é uma versão básica de um pipeline e que existem muitas melhorias e otimizações adicionais que podem ser feitas para aprimorar ainda mais o desempenho da CPU.

4 Resultados e Conclusões

Abaixo analisou-se os nossos resultados e conclusões finais, mostrando a implementação do JALR:

```
1    Address    Code       Basic                    Source
2
3    0x00000000 0x00200293 addi x5,x0,0x00000002 3       addi t0, zero, 2
4    0x00000004 0x00438393 addi x7,x7,0x00000004 4       addi t2, t2, 4
5    0x00000008 0x00000013 addi x0,x0,0x00000000 5       nop
6    0x0000000c 0x00000013 addi x0,x0,0x00000000 6       nop
7    0x00000010 0x00000013 addi x0,x0,0x00000000 7       nop
8    0x00000014 0xfff302e7 jalr x5,x6,0xffffffff 8       jalr t0, t1, -1
```

Figura 2: Código exemplo RISC-V

A fim de explicar o código de uma forma sucinta, considerou-se apenas as principais partes da simulação para melhor entendimento.

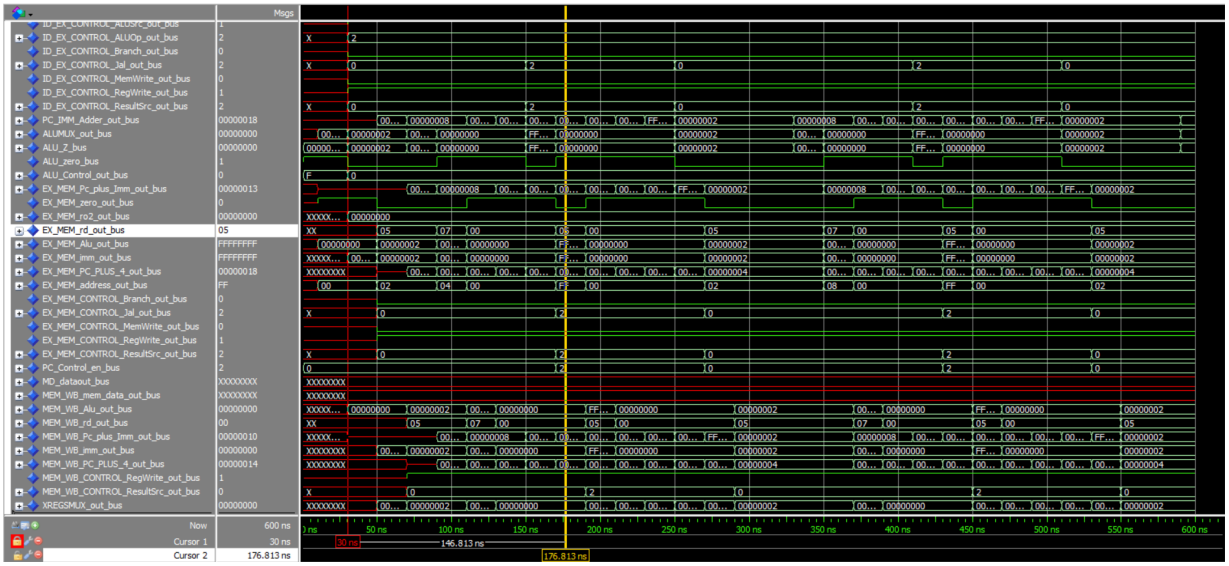

Figura 3: Registrador de destino

Na Figura 3 é mostrado o registrador que se encontra o endereço de retorno (PC+4), o qual é o último valor de PC antes de obter o endereço de salto fornecido pelo Jalr. Isso evidência que o endereço de retorno foi armazenado no registrador de destino correto, como pode se observar na Figura 2.

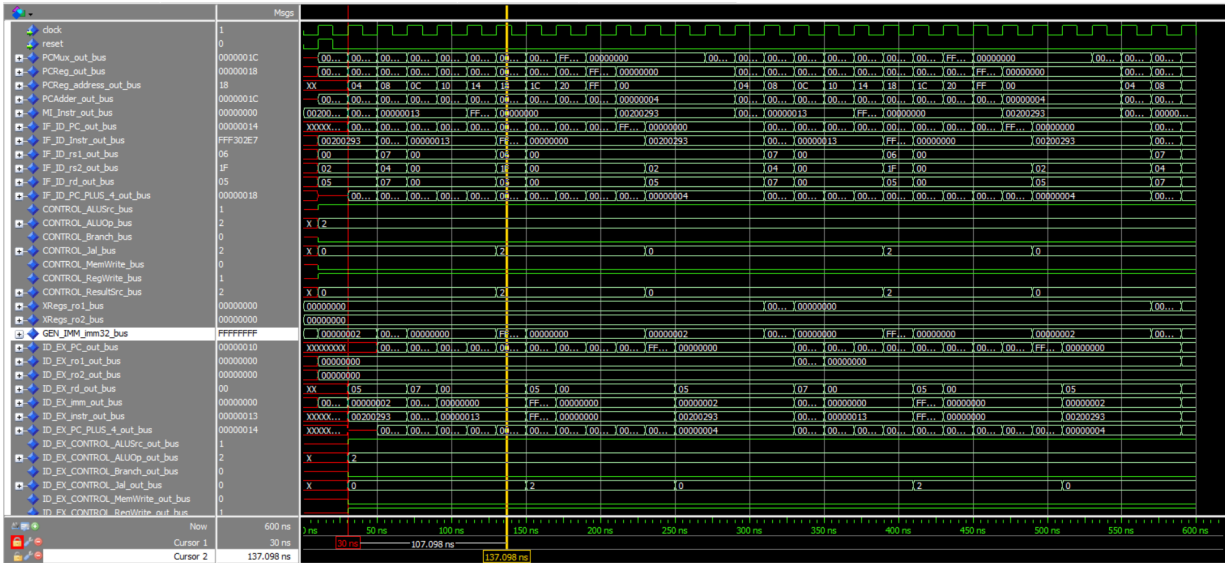

Figura 4: Gera o imediato para o endereço de salto

Já na Figura 4 é mostrado o dado gerado pelo imediato, o qual será usado como endereço para qual PC irá saltar. Como pode ser visto na figura, o imediato foi gerado corretamente, uma vez que o imediato fornecido na instrução (Figura 2) é igual ao gerado.

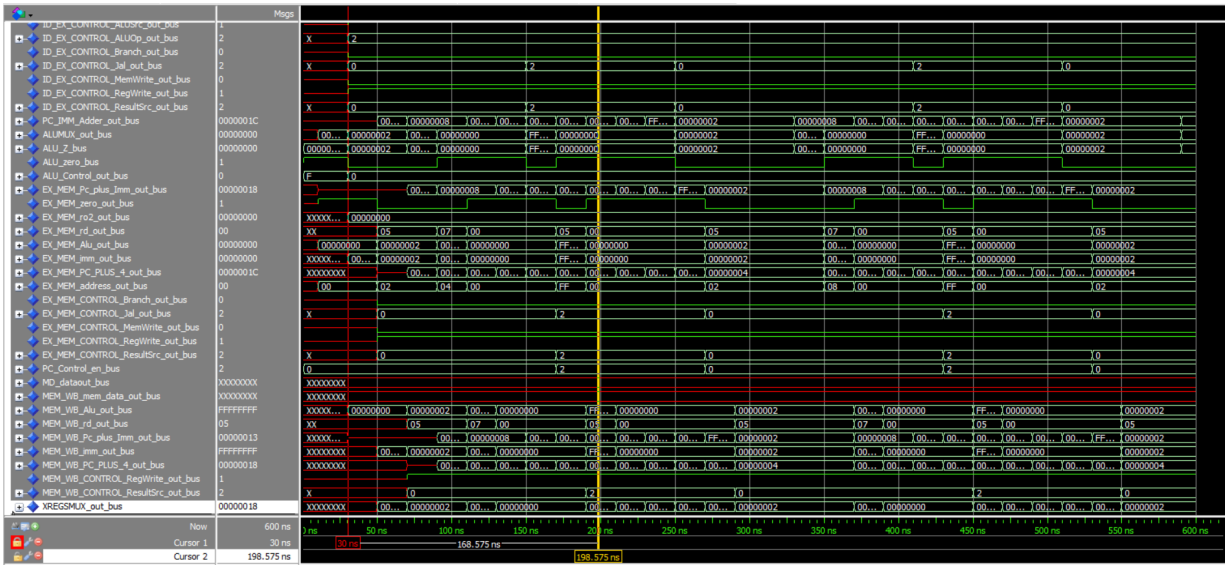

Figura 5: Endereço do PC+4 armazenado no registrador de destino

Na Figura 5, podemos observar o fluxo de valores no pipeline durante o estágio de Write Back (WB). Quando a instrução JALR é executada, o valor de retorno (PC+4) é armazenado. Nesse exemplo específico, o sinal XREGS_MUX exibe o valor $(00000018)_{16}$, indicando que esse valor foi escrito no registrador de destino (rd). Esse processo ilustra a etapa final do pipeline em que os resultados das instruções são efetivamente armazenados nos registradores adequados.

Figura 6: Endereço do PC após Jalr

Ademais na Figura 6, pode-se observar o valor do endereço de salto no PC, o qual é coerente com o valor informado na instrução. O que evidência que o salto foi realizado, pois o endereço de salto visto no PC é o esperado, uma vez que o imediato fornecido é o $(-1)_{10}$ ou $(FF)_{16}$ (Figura 2).

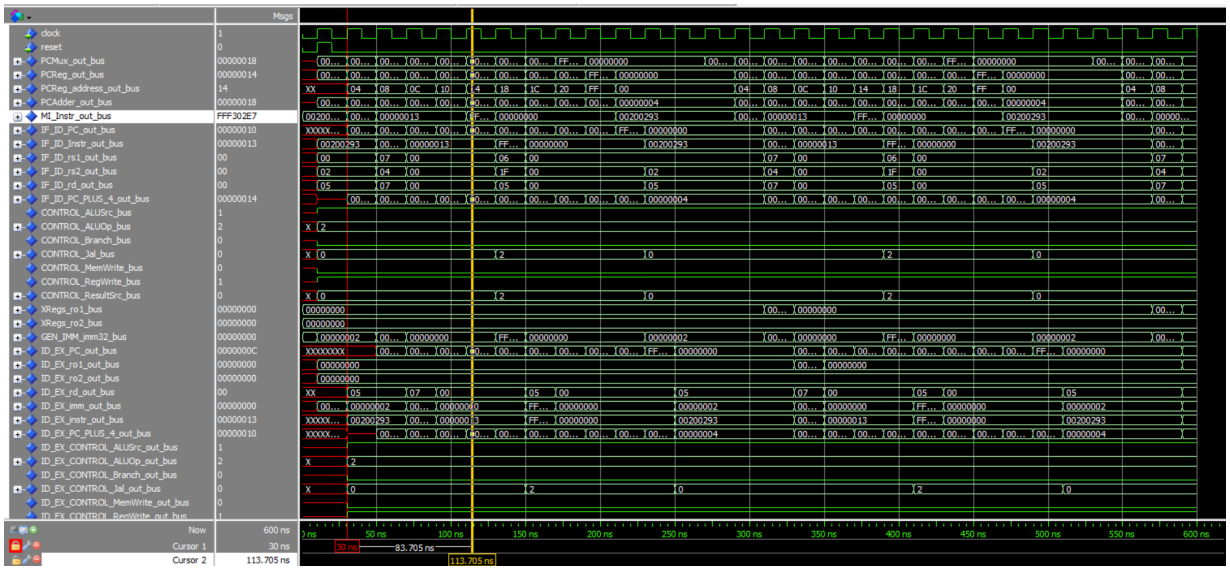

Figura 7: Código da instrução Jalr na memória de instruções

Essa 7 serve apenas para mostrar que as instruções estão sendo bem interpretadas pela ROM, o código $(FFF302E7)_{16}$ representa o JALR, como pode ser observado na Figura 2.

5 Considerações Finais

Neste projeto, foi desenvolvida uma versão do processador RISC-V Pipeline em VHDL, com o objetivo de implementar uma arquitetura básica de pipeline e explorar o funcionamento eficiente do processador através da execução simultânea de múltiplas instruções. Ao longo do desenvolvimento, todas as funções essenciais do pipeline foram implementadas e testadas com sucesso.

A arquitetura do RISC-V Pipeline consiste em cinco estágios principais: Busca de Instrução (IF), Decodificação de Instrução (ID), Execução (EX), Acesso à Memória (MEM) e Escrita de Resultado (WB). Cada estágio foi cuidadosamente projetado e interligado por meio de sinais para garantir o correto fluxo de instruções e a execução eficiente do programa.

Durante os testes realizados, o pipeline demonstrou um desempenho satisfatório, executando corretamente as instruções fornecidas. Além disso, foi possível verificar o funcionamento adequado dos saltos condicionais e das operações aritméticas e lógicas, bem como o acesso correto à memória.

É importante ressaltar que o pipeline também foi capaz de lidar com instruções "nop"(no operation), que não realizam nenhuma operação e são utilizadas para preencher ciclos de clock ociosos. Dessa forma, o processador demonstrou sua flexibilidade em lidar com diferentes tipos de instruções.

Apesar dos resultados positivos alcançados com a implementação do pipeline básico, é importante mencionar que algumas melhorias e otimizações ainda podem ser realizadas. Não foram implementadas funcionalidades avançadas, como forwarding e hazard detection, que poderiam aprimorar ainda mais o desempenho do pipeline.

Em conclusão, o projeto proporcionou uma valiosa experiência prática aos estudantes no desenvolvimento de um processador RISC-V em VHDL. A implementação do pipeline básico permitiu aplicar os conceitos teóricos aprendidos em sala de aula em um projeto concreto, contribuindo para a compreensão dos princípios fundamentais da arquitetura de computadores e do funcionamento de um processador em pipeline.