



## MC851 Projeto em Computação

# Relatório Entrega 2

**Resumo:** planejamento e desenvolvimento de um processador RISC-V de 32 bits, com pipeline, que suporte o conjunto RV32I.

***Equipe "RISC-VI":***

RA 169374, Daniel Paulo Garcia

RA 182783, Lucca Costa Piccolotto Jordão

RA 185447, Paulo Barreira Pacitti

RA 198435, Guilherme Tavares Shimamoto

RA 216116, Gabriel Braga Proença

RA 221859, Mariana Megumi Izumizawa

## 1. Introdução

O *design* e desenvolvimento de um sistema computacional pode ser muito complexo, e muito caro também. Graças a tecnologias de HDL (*Hardware Description Language*) e *chips* FPGA (*Field Programmable Gate Arrays*), podemos prototipar *chips* de forma barata e rápida. Mesmo com experiências com RISC-V em MC404 (Organização Básica de Computadores e Linguagem de Montagem), com FPGA e HDL em MC613 (Laboratório de Circuitos Digitais) e MC732 (Projeto de Sistemas Computacionais), construir um SoC (*System-on-a-Chip*), a equipe entende o projeto como um desafio bastante complexo e têm estudado e revisado conceitos dessas disciplinas. Nesta segunda entrega, apresentamos o andamento do segundo mês de desenvolvimento realizado de um sistema computacional com arquitetura RISC-V de 32 bits que têm suporte ao conjunto RV32I.

## 2. Planejamento

No primeiro mês, as reuniões do grupo ocorreram semanalmente durante o horário da aula, o que muitas vezes resultava em dúvidas e discussões acumuladas até o momento do encontro. As comunicações eram principalmente feitas por meio de mensagens em grupo no WhatsApp, o que frequentemente levava à perda de documentação importante e dificultava o acompanhamento de atividades. Além disso, as discussões sobre tópicos diferentes se misturavam no mesmo canal, causando confusão e perda de informação.

A partir da primeira entrega, o sistema de comunicação e colaboração foi aprimorado. São realizadas reuniões duas vezes por semana, às terças e sextas-feiras, permitindo abordar questões de forma mais ágil. O grupo de mensagens é usado para assuntos pontuais e rápidos, mas também foi criado um servidor no Discord para aprimorar a colaboração. No Discord, foi criado um canal de voz para facilitar as discussões em tempo real e as mensagens de texto são organizadas em tópicos separados, incluindo dúvidas, entregas, relatórios e pull requests. Além disso, o Github está sendo utilizado para gerenciar pull requests, comentários e discussões, bem como para rastrear issues relacionadas a bugs.

Dado o resultado da primeira entrega, o grupo decidiu focar em finalizar a implementação e execução da instrução ADDI, montar uma planilha de sinais e instruções e iniciar a pesquisa e integração de um periférico.

Para o segundo mês do projeto, o grupo se dividiu novamente em duplas, com as seguintes responsabilidades:

- Guilherme e Mariana, foram responsáveis por realizar os testes para integração com um periférico, sendo que, o periférico escolhido foi o Serial Console. Além disso, com o andamento do projeto, iniciaram o desenvolvimento da instrução de BEQ (Branch if Equal) na CPU;
- Lucca e Gabriel, foram responsáveis por criar e atualizar a planilha com sinais e instruções. A partir dela foi possível realizar a decodificação das instruções do tipo R, S, I e B;
- Paulo e Daniel, foram responsáveis pela implementação da MMU (assim como seus testes automatizados) e refatoração da CPU em um único arquivo, e integraram os dois módulos. Implementaram os registradores de pipeline e sinais documentados na planilha feita por Lucca e Gabriel e foram responsáveis por ajustar a implementação e arrumar *bugs* de sincronia para que um primeiro programa em assembly (ADDI) funcionasse corretamente no sistema computacional do grupo.

### 3. Resultados

Sobre a integração com o Serial Console, o grupo criou uma pasta separada no projeto, com o exemplo de integração, permitindo a comunicação entre a placa e o console no computador, recebendo e enviando dados através do protocolo UART. Sendo que, por ser o código baseado em tutoriais da LushayLabs, esse código está separado e não é integrado no projeto atual da CPU.

A CPU foi refatorada de forma que os registradores de pipelines sejam bastante visíveis. A escolha de deixar os estágios em único arquivo evitou a criação de sinais duplicados e fios desnecessários, e facilitou o entendimento do *datapath* do sistema. Um ponto negativo é que a resolução de conflitos se tornou bem mais complicada em revisões de código, mas o grupo acredita que as vantagens superam as desvantagens.

A equipe terminou de implementar a primeira versão da MMU, que gerencia o acesso a ROM e RAM. Para isso, o grupo implementou uma máquina de estados que controla a leitura e escrita em seu espaço, assim como o mapeamento de endereços virtuais para físicos. Para garantir que tal módulo estava funcionando corretamente, foram escritos testes automatizados de leitura e escrita, simples, mas que assegurem que o módulo está funcionando corretamente.

A equipe concluiu na última semana um teste completo do sistema com um programa em assembly com a instrução ADDI. Foi possível verificar que o programa escreveu o resultado da operação no registrador com sucesso. A partir desse teste foi possível descobrir falhas de sincronia e atualização de registradores de *pipeline* que estavam incorretos, a partir do *gtkwave*. Tal teste e análise do *timing* do sistema computacional, a equipe conseguiu criar uma lista de priorização de tarefas para otimizar o sistema.

### 4. Aprendizados

A dupla, Guilherme e Mariana, ficou responsável por pesquisar opções de periféricos para o projeto e estudar sobre o FPGA kit e como desenvolver o projeto com ela, buscando entender quais problemas poderiam surgir no futuro. Para alcançar esse objetivo, a equipe acessou diversos recursos, incluindo tutoriais fornecidos por LushayLabs, que ofereceram insights valiosos sobre o funcionamento do kit FPGA. Os tutoriais abordaram tópicos como o uso do WaveTrace para análise de resultados, a manipulação do console serial e o reconhecimento do dispositivo pelo computador. Inicialmente, a equipe tinha a concepção de que o periférico necessário para o projeto deveria ser algo físico, como um monitor ou visor. No entanto, após estudar e dialogar com o professor orientador, foi esclarecido que seria possível utilizar o console serial (Serial Console) como um dos periféricos para o projeto. Essa abordagem baseia-se na transmissão e recebimento de dados de forma assíncrona, por meio do protocolo UART (transmissor/receptor assíncrono universal). Com essa configuração, tornou-se possível reconhecer as entradas do teclado dos notebooks como entradas para o console e observar as saídas sendo exibidas na tela.

Sobre a implementação da instrução de BEQ, algo que foi necessário ser atualizado no projeto foi a obtenção de imediato para essas instruções, pois o projeto estava estruturado para obter somente os imediatos dos outros tipos de instrução. Logo, a dupla responsável por essa funcionalidade (Guilherme e Mariana) teve que pesquisar sobre os imediatos nas instruções de branch para implementar a solução na etapa de ID (Instruction Decode). E por

ser um trecho de código específico a dupla decidiu criar um test bench, focando somente nessa parte, o que ajudou a garantir o funcionamento de um pequeno trecho do código que é essencial para toda execução da CPU como um todo.

Durante a implementação de testes automatizados, notamos a importância de garantir que o sistema seja *clock-cycle-accurate*. Muitos dos testes que estávamos implementando não tinham atenção a esse detalhe, e colocamos um número absurdamente alto de *clocks* só para que o teste passe com sucesso. Assim, passamos a utilizar o *gtkwave* para melhor *debugar* o estado do sistema a cada *clock* e ajustar sinais que estavam defasados ou adiantados. Tal aprendizado tem ajudado muito não só a solucionar problemas de sincronia, mas também a entender o funcionamento do sistema com um todo.

A equipe aprendeu que o correto uso de *wires* e *regs*, e como sua atribuição funciona em blocos combinacionais e concorrentes. Muitos dos problemas de sincronização com o *clock* foram frutos do mau entendimento desses conceitos. A refatoração do código para consertar esses *bugs* ajudou também a melhor documentar o código do sistema.

Um aprendizado que foi essencial para o andamento do projeto foi a documentação de melhorias através da ferramenta de Issues do Github. Um bom exemplo dessa prática foi o caso de atribuições dentro do bloco *always* de maneira errada, em que o grupo por não ter familiaridade com a tecnologia, estava atribuindo de forma sequencial os valores das variáveis, fazendo com que a CPU não funcionasse corretamente. Com a ajuda do professor, alguns integrantes do grupo notaram tal comportamento e criaram a Issue no Github, detalhando o motivo desse problema estar ocorrendo e como resolvê-lo. Logo, nos momentos de avaliação de código, se tornou natural referenciar essa Issue para que os outros integrantes pudessem entender o problema e corrigi-lo no código. O que tornou o processo de compartilhar conhecimento e de documentação do projeto muito melhor.

## 5. Futuro

Na atual implementação, nossa CPU acaba tendo 50% de utilização da pipeline, pois cada operação de leitura na MMU tem pelo menos 2 ciclos de clock de latência, o que faz necessário inserir muitas “bolhas” (instruções NOP) durante o estágio de fetch. Para aumentar o aproveitamento, será necessário a implementação de uma cache L1 com leitura em 1 ciclo, para que o estágio IF não atrase os outros estágios a cada leitura de nova instrução.

Além disso, para que seja possível implementar instruções como LOAD e STORE, será necessário a implementação de um módulo de controle de acesso da CPU à MMU. Se dois estágios tentarem acessar a memória simultaneamente, para que a pipeline possa prosseguir, será necessário priorizar o acesso do estágio mais adiantado e fazer com que o outro estágio aguarde sua vez de acessar a memória.

Por termos apenas uma instrução funcionando, ainda não conseguimos fazer um programa em assembly que rode em loop para implantação na placa FPGA. A priorização de instruções como *branches* e *jumps* para que seja possível criar laços de execução no programa deve ser feita. Da mesma forma, como ainda não tínhamos as instruções funcionando, fica para próxima entrega o módulo de comunicação com os periféricos, integrando a CPU e a MMU.

Com esses diversos desafios a se atacar, um plano de ação possível seria o seguinte: uma dupla responsável por implementação e testes das instruções do RISC-V de forma que estejam *clock-cycle-accurate*; outra dupla responsável pela implementação de módulo adicionais do *datapath* como controle de memória e cache LI; e, a última dupla, responsável

por criação de programas simples, implementação de periféricos, e testes nas placa FPGA. Tal priorização de tarefas será necessária para que a equipe consiga executar programas em *hardware* (na placa FPGA) e que otimize o sistema operacional implementado.

## Referências

1. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.
2. Repositório com código do sistema computacional:  
<https://github.com/izumizawa/mc851>