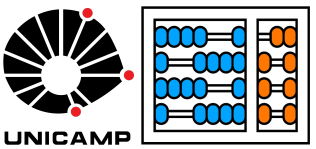


MC851 Projeto em Computação

# Relatório Entrega 3

***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 terceira entrega, apresentamos o andamento do terceiro mês de desenvolvimento realizado de um sistema computacional com arquitetura RISC-V de 32 bits que têm suporte ao conjunto RV32I.

## 2. Planejamento

Dado o resultado da segunda entrega, o grupo decidiu focar na integração de periféricos, desenvolver instruções dos tipos I e R, desenvolver a cache L1, implementar o controle de acesso à memória e implantar o sistema na placa FPGA.

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

1. Guilherme e Mariana, foram responsáveis por realizar a integração de periféricos e implantar o sistema na placa FPGA;
2. Lucca e Gabriel, foram responsáveis pelos testes das instruções dos tipos I e R;
3. Paulo e Daniel, foram responsáveis pela implementação das caches e o controle de acesso à memória principal.

Ao longo do mês, o grupo dividiu as funções novamente de acordo com as necessidades geradas.

## 3. Resultados

Lucca e Gabriel foram responsáveis por continuar a implementação *clock-accurate* das instruções do RV32I, entregando as instruções do tipo I e do R. Para cada instrução, foram implementados testes automatizados de forma a conferir que o comando toma os ciclos esperados teoricamente. Para tal inspeção e *debug* foi utilizado o programa **gtkwave**. Também consertaram bugs na ALU relacionados à representação de sinal de *bitstrings*.

A dupla Paulo e Daniel fez ajustes na pipeline de forma que a MMU ao invés de tomar 2 ciclos para leitura como fio na última entrega, agora toma só 1. Atrasos desnecessários haviam sido introduzidos em várias partes do código, causando atrasos na pipeline. Tal ajuste permitiu que o sistema computacional executasse um loop infinito com uma instrução `addi`, sem empecilhos. A dupla desenvolveu a *forwarding unit* do sistema, dessa forma, instruções consecutivas que utilizam os mesmos registradores funcionam corretamente, adiantando valores aos registradores que são dependentes das próximas

instruções. Um código em *assembly* como o abaixo não funcionaria corretamente sem essa estrutura:

```
addi t0, zero, 2
addi t0, t0, 2
```

Daniel também implementou as estruturas das cache L1 de instruções de dados, mas principalmente a lógica de controle de acesso de memória de diferentes estágios da pipeline, com o intuito de priorizar e dar *stall* no sistema em caso de leituras e escritas na memória.

Mariana e Guilherme ficaram responsáveis pela implantação da implementação em Verilog do nosso sistema computacional na placa FPGA. Foram encontradas diversas dificuldades: código não sintetizável, comportamento não esperado, documentação não clara… Para melhor atuar nesse problema, outros membros da equipe também participaram na experimentação com placa FPGA a fim de tentar implantar uma primeira demonstração. Depois de encontrar pedaços do código-fonte que sintetizam latches D, não sintetizados pela placa, e de ler o manual do *chip* FPGA da Tang Nano 9k, o GW1NR-9, o grupo conseguiu sintetizar uma primeira demonstração do sistema computacional em FPGA. Utilizando a UART como periférico, a demonstração consiste em rodar um programa em *assembly* que altera o valor de um registrador. Através da porta UART, é possível conferir o valor do registrador no qual a operação foi executada.

## 4. Aprendizados

Durante a implementação das caches foi encontrado um problema de possibilidade de dois estágios de pipeline tentarem acessar a memória principal no mesmo ciclo, porém a memória só faz, no máximo, uma operação por ciclo, dando origem a um hazard estrutural.

Na entrega anterior, o foco da equipe estava na conclusão de outras etapas da CPU e na execução de pelo menos um programa simples. Dado que a integração da cache à CPU dependia de mecanismos de controle de acesso à memória e do controle da pipeline, foi optado por adiar a implementação das instruções de carga/armazenamento (load/store), mantendo apenas o fetch de instruções.

No estágio atual do projeto, foram implementadas duas caches distintas, uma destinada a armazenar instruções e outra para dados. A expectativa é que o impacto do hazard estrutural diminua à medida que o tamanho das caches aumenta, uma vez que isso aumenta a probabilidade de os valores necessários já estarem disponíveis nas caches. No entanto, por enquanto, as caches possuem apenas alguns bytes de armazenamento, destinados principalmente a fins demonstrativos. Isso ocorre porque a prioridade do grupo foi o desenvolvimento dos protocolos de comunicação entre a cache e a memória principal.

Apesar da implementação de caches separadas para instruções e dados, foi observado que essa abordagem não é suficiente para resolver de forma definitiva o problema do hazard estrutural. Ainda há situações em que ambas as caches podem apresentar misses simultaneamente. Para contornar essa questão, foi introduzido um mecanismo que concede

prioridade de acesso ao estágio mais avançado da pipeline (MEM), evitando congestionamentos desnecessários nos estágios anteriores.

A implementação desse mecanismo revelou-se mais simples do que inicialmente previsto. Foi utilizado um multiplexador de "duas vias" (entrada/saída) no controlador de memória para as caches. Esse multiplexador escolhe qual cache conectar à memória com base em um sinal de miss gerado por elas, respeitando a prioridade de acesso. Assim, as próprias caches assumem a responsabilidade de conceder a vez, desativando o sinal de miss após completar o acesso à memória.

Outro aspecto do funcionamento da CPU é que, quando ocorre um miss em qualquer uma das caches, todos os estágios anteriores entram em estado de "stall". Embora os estágios seguintes possam avançar, é essencial evitar a duplicação de instruções. Para isso, é necessário inserir uma instrução nula no lugar da instrução ausente, realizando assim um "flush" no registrador de pipeline. Esse procedimento garante a consistência e integridade da execução das instruções subsequentes.

Durante o processo de sintetizar o código e executar na placa FPGA, foram enfrentados diversos desafios. Ao utilizar *includes* de outros componentes no código, percebeu-se que, embora essa abordagem funcionasse nos testes, a síntese no Yosys primeiro executava os arquivos unitários e, em seguida, consolidava todos em uma hierarquia. Isso resultava em redeclarações de componentes devido aos *includes*, o que precisou ser corrigido. Além disso, identificou-se que blocos always sensíveis a "posedge clk" e "negedge reset_n" geraram problemas durante a síntese e precisavam de tratamentos específicos para que não gerassem erros. Além disso, na síntese de cada arquivo, verificou-se que parâmetros vazios em ROMFILE geraram erros na função de leitura na ROM.

Com o código sintetizado e implementado com êxito na placa, o foco voltou-se para a utilização do módulo de demonstração da UART. A intenção era executar a instrução ADDI na CPU, enviar o valor do registrador destino via UART e imprimir o resultado no Terminal Serial. O terminal serial estava configurado para receber apenas 1 byte por vez, enquanto o registrador possuía 32 bits. Foi necessário criar um script em Node.js para manipular e imprimir os 32 bits com seus valores em hexadecimal.

Além do mais, foram enfrentadas dificuldades na execução, apontando erros de Latches D, resultando em falhas na comunicação UART para receber o valor do registrador. A identificação das causas desse problema exigiu uma análise mais aprofundada. Para solucionar alguns problemas específicos, foram realizados testes executando comandos do Yosys fora do ambiente do VSCode, o que contribuiu para a identificação e resolução de questões técnicas.

Embora os testes indicassem que a atribuição de valor nos registradores do register_file estava funcionando corretamente, observou-se falhas quando implementado na placa. Esse desafio exigiu revisões adicionais. Após uma série de testes e modificações, foi possível obter o valor do registrador modificado pela instrução ADDI e imprimi-lo com sucesso no terminal.

## 5. Futuro

Sendo o próximo mês a última entrega, a equipe quer entregar a maior quantidade de funcionalidades para o sistema computacional para que, mesmo que simples, seja um SoC RV32 funcional.

Com o controle de memória pronto, o grupo poderá terminar a implementação das caches L1 e implementar as funções de LOAD/STORE. Tal mudança também permitirá uma melhor interface com os periféricos como é feito na maioria dos computadores, onde a interface entre a CPU e outros *devices* é feito através de leitura e escrita em endereços de memória, por meio da MMU.

A equipe continuará a implementar as instruções da especificação do RV32I, terminando a RV32I ao completar a implementação das instruções de pulos e pulos condicionais.

A ideia é que, para a última apresentação, o grupo demonstre uma aplicação mais complexa utilizando uma gama variada de instruções de RISC-V com o uso dos periféricos.

## 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 sitema computacional: https://github.com/izumizawa/mc851