

# MC851 - Projeto em Computação I

## arRISCado - Entrega 3

Ângelo Renato Pazin Malaguti - 165429

Claudio dos Santos Júnior - 195727

Elton Cardoso do Nascimento - 233840

Gabriel Costa Kinder - 234720

Iago Caran Aquino - 198921

João Pedro de Moraes Bonucci - 218733

### 1. Introdução

Neste relatório, continuamos com o trabalho iniciado no documento anterior, concentrando nossos esforços no desenvolvimento das entregas pendentes. Isso inclui a criação de um ambiente de execução funcional tanto para o processador quanto para o processador em FPGA, e o desenvolvimento da cache L1. Além disso, nosso objetivo é integrar os dois periféricos ao processador em FPGA, e expandir o conjunto de instruções de RV32IM para RV32IMAC, incorporando instruções atômicas, bem como instruções compactas.

### 2. Planejamento

Partindo do princípio que já tínhamos os dois periféricos parcialmente prontos, decidimos focar no pipeline nas primeiras semanas. Posteriormente, dividimos em 4 times, um para a integração de cada um dos periféricos, outro no desenvolvimento da terceira entrega, enquanto o último concentrou-se na realização de testes e correções de erros na simulação. As atividades realizadas serão listadas e detalhadas nas seções a seguir.

### 3. Desenvolvimento

#### a. Finalização do Pipeline

O Iago e o Ângelo fizeram as últimas ligações na pipeline para adicionar os sinais necessários às instruções de branch.

Além disso, visando aumentar a quantidade de instruções que podemos colocar na placa, o Ângelo implementou o *Forwarding* de dados para tratar Hazards do tipo RAW, com exceção do caso de Load seguido de alguma operação que utiliza o registrador que foi atualizado. Isso nos permitiu remover a maioria dos *nop* que adicionamos nos códigos de teste. Todavia, resta adicionar os *Stalls* relativos a acesso a memória para eliminar definitivamente a necessidade de *nop*. Esses stall foram desenvolvidos pelo João e Gabriel, tendo um sinal de stall adicionado em cada etapa do pipeline sendo acionado quando necessário pela etapa de *execute* e memória. No entanto, não conseguimos adicionar essa etapa na branch principal do projeto devido aos problemas causados por alterações realizadas em paralelo a esta implementação. Assim a versão com stall está separada da branch principal até o momento da entrega, sendo testada assim que a branch principal estiver estabilizada.

## **b. Implementação do Pipeline na FPGA**

O Iago modificou o código para garantir que algumas instruções importantes funcionassem na FPGA, sendo elas o *add*, *addi*, *lw* e *sw*.

A maioria dos testes foi feita removendo a UART para garantir a quantidade necessária de *SLICES* na FPGA, o que tem se mostrado um limitador do que podemos desenvolver. Enfrentando este problema nas últimas semanas, o Iago mudou o foco da cache para estudar a toolchain da Gowin e criar um script para substituir as ferramentas do Yosys.

## **c. Melhorias para a UART**

O Gabriel otimizou o módulo UART, diminuindo *SLICES* usadas na implementação e alocando menos memória para instruções repetidas. Também assegurou o correto funcionamento do controle da CPU e da aquisição das instruções.

Além disso, foi garantido que sua interface de comunicação mantenha-se idêntica ao do módulo da ROM, facilitando que estes sejam trocados, permitindo a realização de testes mais rápidos sem a preocupação da utilização da comunicação serial.

## **d. Testbenches**

O Cláudio ficou responsável pela reformulação dos testbenches, assim como documentação e correção dos problemas no pipeline. De início, existiam muitos testes e arquivos que não se sabiam a utilidade, segundo, que quase todos estavam já desatualizados e não funcionavam.

A primeira parte da tarefa consistiu em analisar a utilidade e funcionalidade de cada teste, seguido pela remoção dos considerados não mais úteis. A parte 2 foi documentar e corrigir os testes, na maior parte deles foi necessário fazer correções nos módulos, pois foram encontrados resultados incorretos. Por conseguinte, os testes foram melhorados de forma a conseguirem identificar erros mínimos no pipeline, como utilizar operações específicas para cada instrução.

Por fim, os testes foram melhorados para verificar se o pipeline tinha suporte para todas as instruções do tipo RV32IM. Nesse processo foram encontrados alguns problemas, um deles é que as operações de multiplicação não estão conseguindo trabalhar com as instruções de sinal, os resultados estão saindo incorretos. Ademais, não foi possível testar e verificar as instruções atômicas e compactas, pois o pipeline ainda não tem suporte para elas.

Em paralelo, o Elton ficou responsável de realizar testes para checar passo-a-passo o avanço de cada instrução, executando os testes de código gerados anteriormente (Figura 1). O código de teste criado para essa funcionalidade foi atualizado constantemente para obter os sinais necessários para avaliar cada instrução. Testes também foram realizados regularmente com as alterações realizadas na branch principal, avaliando se ocorreram problemas de regressão.

Step 0	Step 1	Step 2	Step 3
IF 0 instr: 20000537 pc: 4	IF 1 instr: 00000013 pc: 8	IF 2 instr: 00000013 pc: 12	IF 3 instr: 00000013 pc: 16
Decode -1 DECODE.IN----- DECODE.INTERNAL----- instruction: 00100013 DECODE.OUT----- imm: 1 rs1: 0 rs2: 1 shamt: 1 func3: 0 func7: 0 opcode: 19 MemWrite: 0 MemRead: 0 RegWrite: 1 RegDest: 0 AluSrc: 1 AluOp: 2 AluControl: 2 Branch: 0 MemToReg: 0 RegDataSrc: 0	Decode 0 DECODE.IN----- DECODE.INTERNAL----- instruction: 20000537 DECODE.OUT----- imm: 536870912 rs1: 0 rs2: 0 shamt: 0 func3: 0 func7: 16 opcode: 55 MemWrite: 0 MemRead: 0 RegWrite: 1 RegDest: 10 AluSrc: 1 AluOp: 4 AluControl: 2 Branch: 0 MemToReg: 0 RegDataSrc: 0	Decode 1 DECODE.IN----- DECODE.INTERNAL----- instruction: 00000013 DECODE.OUT----- imm: 0 rs1: 0 rs2: 0 shamt: 0 func3: 0 func7: 0 opcode: 19 MemWrite: 0 MemRead: 0 RegWrite: 1 RegDest: 0 AluSrc: 1 AluOp: 2 AluControl: 2 Branch: 0 MemToReg: 0 RegDataSrc: 0	Decode 2 DECODE.IN----- DECODE.INTERNAL----- instruction: 00000013 DECODE.OUT----- imm: 0 rs1: 0 rs2: 0 shamt: 0 func3: 0 func7: 0 opcode: 19 MemWrite: 0 MemRead: 0 RegWrite: 1 RegDest: 0 AluSrc: 1 AluOp: 2 AluControl: 2 Branch: 0 MemToReg: 0 RegDataSrc: 0

Figura 1: exemplo de visualização da execução passo-a-passo do processador.

### e. Ampliação do conjunto de instruções

O Ângelo assumiu a responsabilidade de incorporar as instruções ausentes do conjunto RV32IMA nos módulos de decodificação, execução e na ALU. Até o momento, as instruções do conjunto RV32AC não foram implementadas, priorizando-se as do conjunto RV32IM.

No módulo de decode, o opcode das instruções foi declarado como "localparam" no início do módulo, visando melhorar a legibilidade do código. Nesta etapa, também foram instanciados os sinais de controle associados a cada instrução da extensão M. Os sinais das instruções de Branch foram melhor definidos e mais claramente explicitados, melhorando fazendo com que fosse mais fácil debugar e entender os erros que aconteciam.

No módulo execute, os sinais de controle AluOP e AluSrc são utilizados para determinar os operandos a serem enviados para a ALU, enquanto na ALU, o sinal AluControl é empregado para determinar a operação aritmética ou lógica a ser executada entre os dois operandos. Os possíveis valores para o AluControl também foram definidos como "localparam" no início do módulo da ALU.

### f. Integração do Periférico

O Elton e o João ficaram responsáveis pela validação dos periféricos com a pipeline. A validação foi realizada primeiro em simulação, checando se a escrita na memória da porta PWM funcionava corretamente e se ela gerava os valores desejados. Ainda não foi possível validar se a implementação funciona na FPGA.

## **g. Implementação da cache L1**

O Iago ficou responsável pelo desenvolvimento da cache L1, a qual está na branch *feat/add-cache* do Pull Request #64. Atualmente, o módulo está pronto para acessos de memória alinhados, porém ainda não foi integrado na branch principal.

O objetivo atual é trazer o módulo para a versão mais recente da *main* afim de utilizar a mesma interface de stall da pipeline que é utilizada pelos periféricos, além de já integrá-lo ao controlador que diferencia a memória principal dos periféricos.

## **4. Pendências**

Abaixo segue uma lista de pendências para a próxima entrega:

- Garantir funcionamento do conjunto RV32M
- Adicionar as instruções do conjunto RV32AC
- Finalização da Cache L1
- Finalizar script para utilizar a toolchain da Gowin
- Utilizar os testes para validar alterações no pipeline

## **5. Aprendizados**

Abaixo segue uma lista de aprendizados que tiramos desse mês de trabalho:

- WaveTrace é mais fácil para debugar o projeto;
- Usar um único modelo de simulação
- Yosys ainda não está pronto, ou seja, não fornece todo suporte e otimização para a 9k
- Planejar todo o comportamento dos sinais antes de desenhar os módulos
- Verilog complica projetos com muitos arquivos