CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES II - Turma 01

PRÁTICA 2 - PROCESSADOR MULTICICLO

Natan Rodrigues Rocha / 20203013242
Gabriel Andrade Quezada / 20183021240

Belo Horizonte
22 de Julho de 2024

COMPONENTES DO PROJETO

O projeto da 1ª parte da prática 2 do Laboratório de Arquitetura e Organização de Computadores II inclui os seguintes arquivos principais:

- **Principal.v**: Contém a entidade de nível superior e instancia as conexões com a FPGA.
- **Decodificador.v**: Liga os fios do display de sete segmentos.
- **Register.v**: Registrador de uso geral do processador.
- **ProgramCounter.v**: Registrador que marca a passagem das instruções.
- **Processor.v**: Módulo principal que gerencia todo o funcionamento do circuito.
- **FlipFlopW.v**: Módulo que simula um flip-flop especializado que direciona o sinal de escrita da memória RAM.
- **Dec3To8.v**: Recebe um sinal de 3 bits e retorna um sinal de 8 bits sinalizando qual registrador deve ser selecionado.
- **Barrel.v**: Módulo que implementa as operações de shift e rotate.
- **ALU.v**: Módulo que gerencia as operações aritméticas, de shift, e lógicas.
- **RAMLPM.v**: Memória RAM da biblioteca LPM de porta única. Usada para guardar dados.
- **ROMLPM.v**: Módulo ROM da biblioteca LPM de porta única. Usada para armazenar instruções.
- **ClockDivider.v**: Recebe o sinal de clock, que é extremamente rápido, e retorna um sinal que oscila de um em um segundo, para que seja possível mostrar o passo a passo das instruções na placa.
- **Testbench.v**: Módulo de testes que cria uma instância do processador e simula uma bateria de testes de 1500ps.
- **dados.mif**: Armazena os dados salvos previamente na memória RAM.
- **instrucoes.mif**: Armazena as instruções. Na pasta do projeto, existe um arquivo de instruções, representando um teste.

IMPLEMENTAÇÃO DO PROCESSADOR

O objetivo do trabalho foi implementar o processador que atenda à seguinte especificação de projeto:

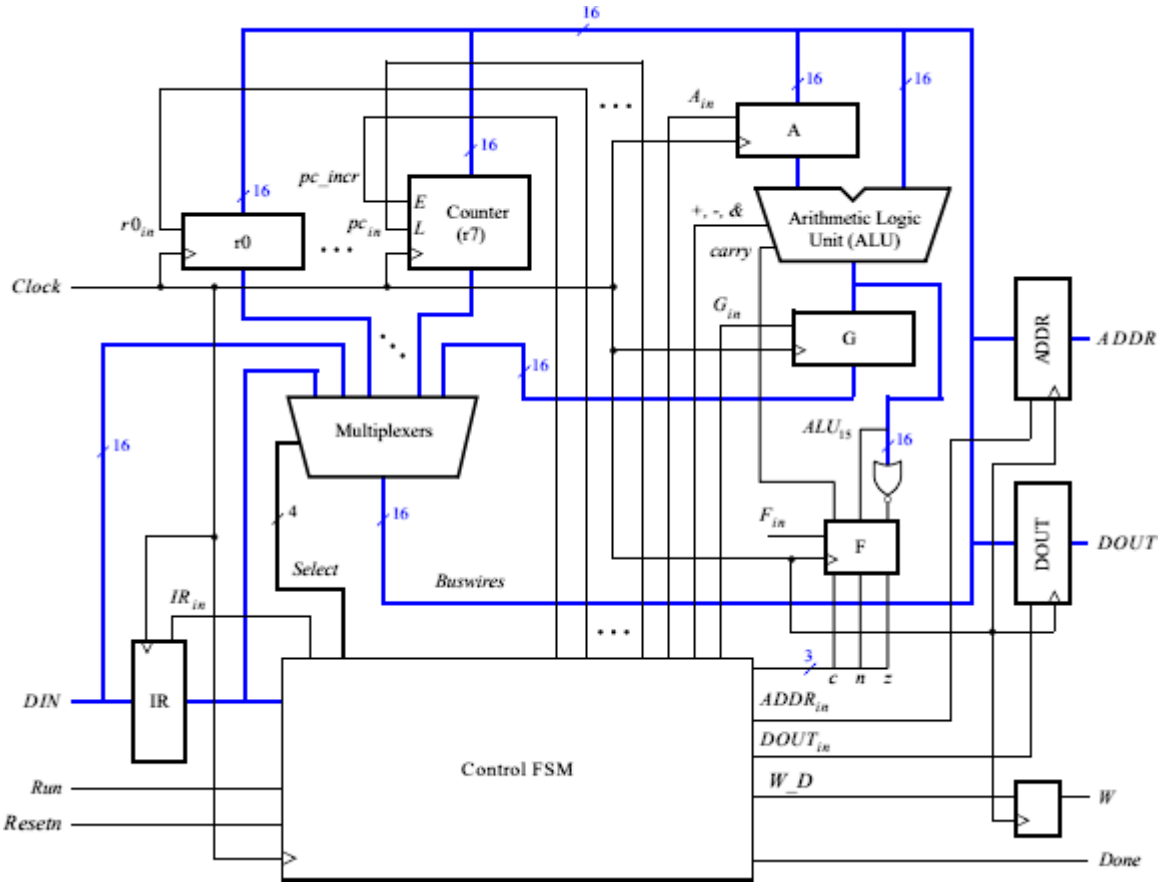


Figura 1 — Versão melhorada do processador

Para iniciar o desenvolvimento do projeto, utilizamos o código esqueleto fornecido na parte A da prática. Inicialmente, o código apresentava algumas lacunas, que precisaram ser preenchidas para o funcionamento correto do sistema.

Primeiramente, foi implementado o módulo Register, que representa os registradores R0 até R6, além dos registradores auxiliares A e G utilizados na ALU, os registradores ADDR e DOUT utilizados pelas memórias, e o IR utilizado para ler as instruções. Esse módulo recebe uma entrada de 16 bits (BusWires) compartilhada por todos os registradores. Cada registrador possui um sinal de escrita individual (RIn), uma entrada de clock, e um fio de saída que representa seu conteúdo. É interessante notar que apesar da nossa ULA ser simples, optamos por usar um barrel shifter, que facilita os cálculos e deve ser útil na 2ª parte da implementação. Isso foi uma dica que pegamos de

AOC1, pois usamos no desenvolvimento da arquitetura do projeto do processador uniciclo e ainda que diferente, a ULA funciona de maneira similar.

Em seguida, desenvolvemos o módulo ProgramCounter, que é um registrador especializado responsável por armazenar a sequência das instruções (uma espécie de índice). Este módulo funciona como um registrador tradicional, mas inclui um sinal adicional, PCIncr, que indica quando o seu conteúdo deve ser incrementado em um.

Os últimos componentes a serem implementados foram as memórias LPM, que foram conectadas aos registradores ADDR e DOUT.

Finalmente, criamos os wires e regs necessários para interligar todos esses componentes e seguimos a tabela de tempos que indica quais sinais devem ser ativados em cada ciclo para todas as instruções.

A tabela a seguir mostra os sinais que precisam ser ativados em cada tempo para todas as instruções:

	T_0	T_1	T_2	T_3	T_4	T_5
mv	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY* or *IR*, rX_{in}, *Done*		
mvt	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select IR*, rX_{in}, *Done*		
add	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, G_{in}	*Select G*, rX_{in}, *Done*
sub	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, *AddSub*, G_{in}	*Select G*, rX_{in}, *Done*
and	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, *ALU_and*, G_{in}	*Select G*, rX_{in}, *Done*
ld	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY*, $ADDR_{in}$		*Select DIN*, rX_{in}, *Done*
st	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY*, $ADDR_{in}$	*Select rX*, $DOUT_{in}$, W_D, *Done*	

Tabela 1 — Mapa da FSM

Com isso, conseguimos implementar um sistema funcional que realiza as operações conforme especificado, garantindo a correta execução de cada instrução.

MODELSIN ALTERA:

Nesta seção, vamos explorar uma série de testes de instruções básicas utilizando um modelo de simulação em Altera. Cada teste executa uma instrução específica, e vamos analisar o funcionamento e o resultado esperado após a execução de cada instrução. As instruções cobertas incluem movimentação de valores imediatos para registradores, operações aritméticas como soma e subtração, além de operações de carga e armazenamento em memória. Vamos detalhar a inicialização dos registradores, a execução de cada instrução, e o estado final dos registradores após a execução.

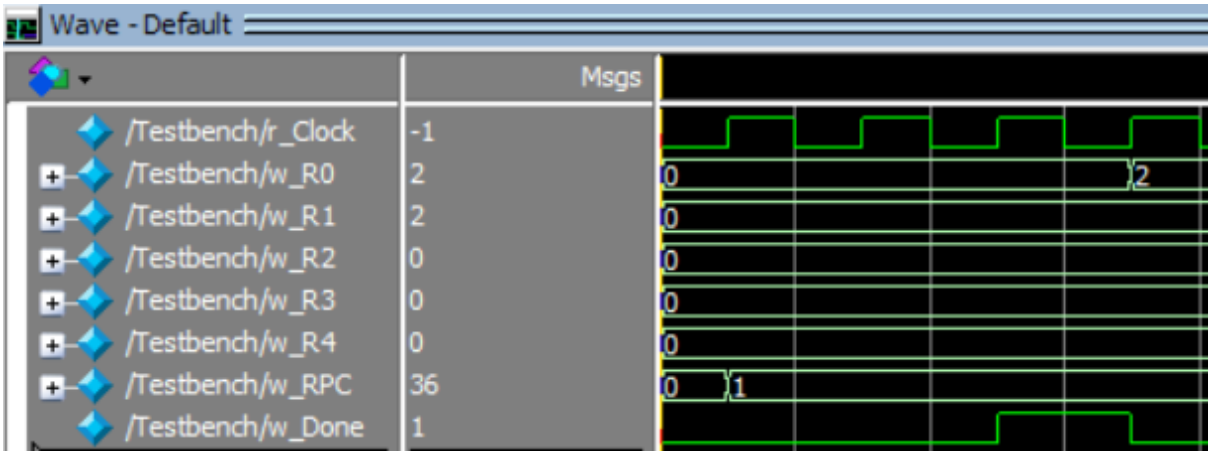
- MV

  Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry
MV R0, #2	000	1	000000	000010

  Execução: O teste em questão executa uma instrução de MV envolvendo o registrador R0 e o imediato #2, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor imediato #2 para o registrador R0, que inicialmente contém o valor 0. Portanto, ao fim da execução R0 receberá o valor 2. (Inicialização dos Registradores: R0 = 0.)

  Simulação:



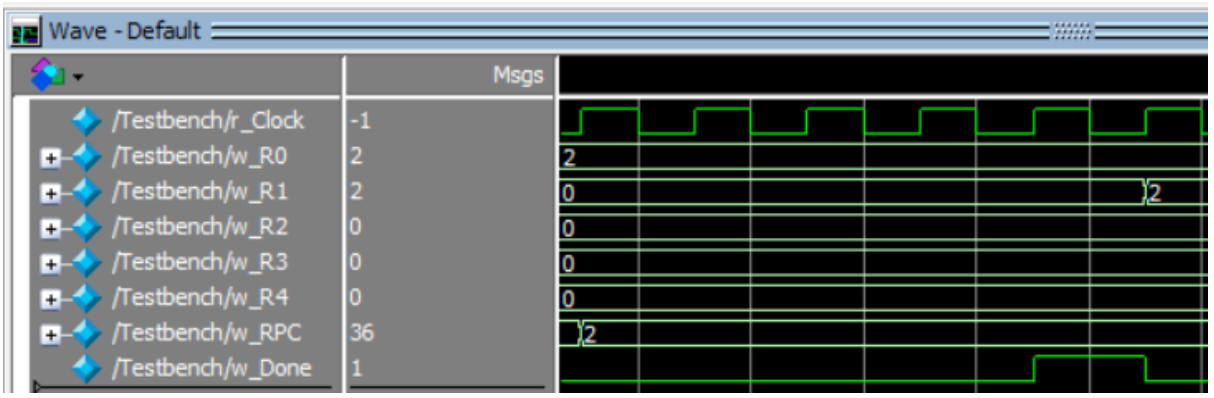
- ADD

  Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry
ADD R1, R0	010	0	000001	000000

Execução: O teste em questão executa uma instrução de ADD envolvendo os registradores R1 e R0, ou seja, o correto funcionamento desse teste acarretará na soma dos valores presentes em R1 e R0, que são respectivamente 0 e 2, e acarretará também no salvamento da soma desses valores no registrador R1. Portanto, ao fim da execução R1 receberá o valor 2. (Inicialização dos Registradores: R0 = 2, R1 = 0)

Simulação:



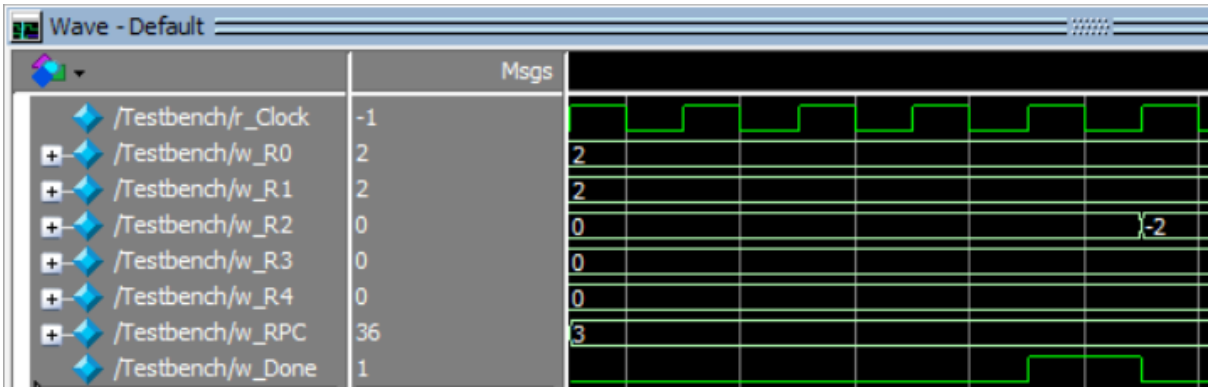
- SUB

Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry
SUB R2, R1	011	0	000010	000001

Execução: : O teste em questão executa uma instrução de SUB envolvendo os registradores R2 e R1, ou seja, o correto funcionamento desse teste acarretará na subtração dos valores presentes em R2 e R1, que são respectivamente 0 e 2, e acarretará também no salvamento dessa subtração no registrador R0. Portanto, ao fim da execução R0 receberá o valor -2. (Inicialização dos Registradores: R2 = 0, R1 = 2)

Simulação:



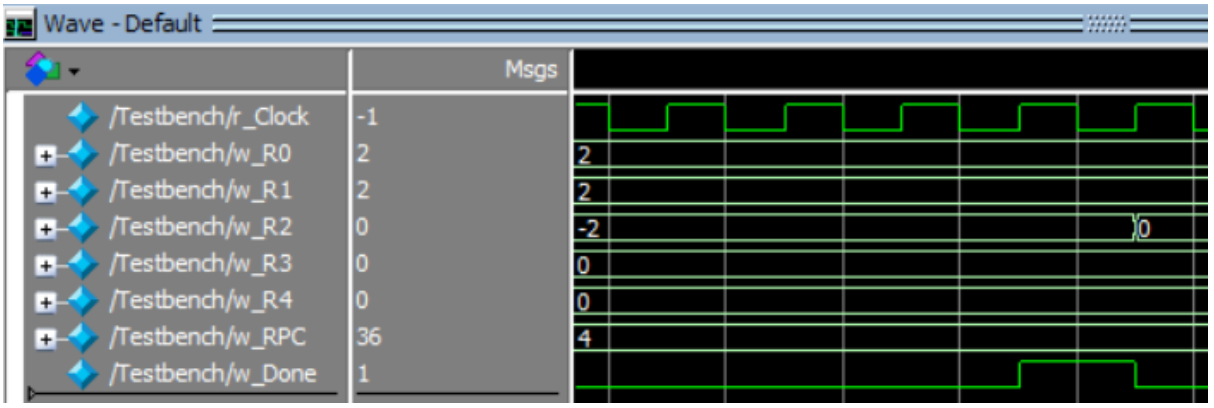
- LD

Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry
LD R2, R0	100	0	000010	000000

Execução: O teste executa uma instrução de LD que acessa a posição de memória referente ao valor salvo no registrador R0, e salva o valor presente nessa posição de memória no registrador R2, ou seja, uma correta execução acarretaria no salvamento do valor 0 no registrador R2. (Inicialização dos Registradores: R0 = 2, R2 = -2)

Simulação:



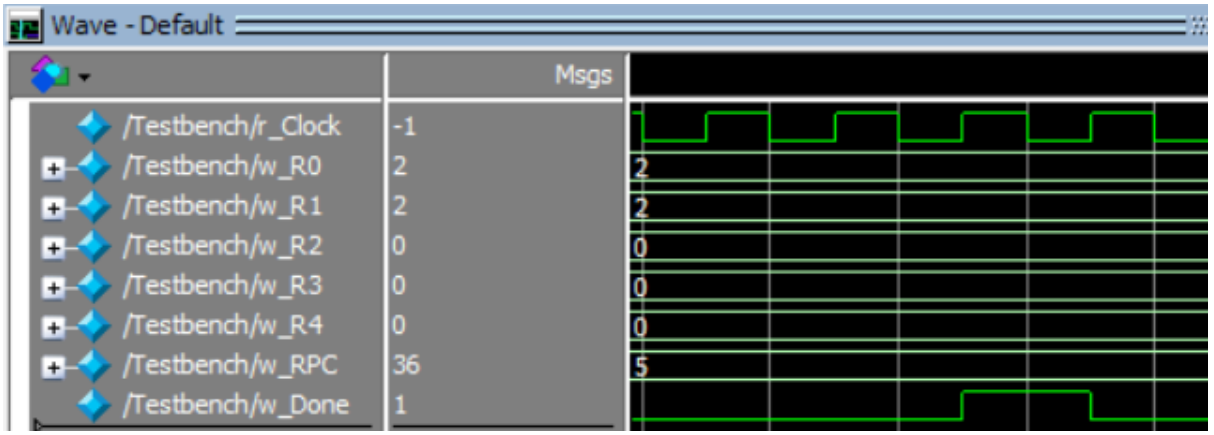
- SD

Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry
SD R2, R0	101	0	000010	000000

Execução: O teste em questão deve executar uma instrução de SD, que irá salvar o valor do registrador R2, o valor -2, na posição de memória referente ao valor presente no registrador R0, que é o valor 2, ou seja, uma correta execução acarretaria no salvamento do valor -2 na posição 2 da memória.

Simulação:



- MVNZ

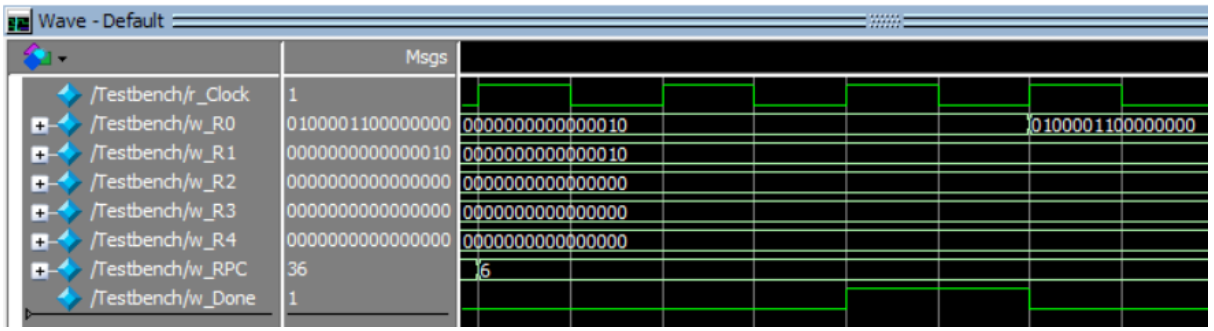
Instrução executada:

Instrução	Opcode	Imediato	Rx	Ry

MVNZ R1, #3	001	1	000001	000011

Execução: O teste em questão executa uma instrução de MV envolvendo o registrador R0 e o imediato #2, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor imediato #2 para o registrador R0, que inicialmente contém o valor 0. Portanto, ao fim da execução R0 receberá o valor 2. (Inicialização dos Registradores: R0 = 0.)

Simulação:



CONCLUSÃO:

Ao final da aula de hoje, devido a uma persistência em leitura dos slides A e B além de debug no código, fixamos diversos problemas no código que nos permitiu finalmente executar a instrução de MV(a primeira), durante a aula. Ainda que não tenhamos apresentado na FPGA, isso nos deu uma nova motivação para que após a aula, seguindo cuidadosamente as orientações fornecidas pela professora, conseguimos identificar e corrigir erros na inicialização do arquivo MIF, além de testes unitários por módulos. Alguns dos erros variavam uso impróprio da sintaxe do verilog até inicialização e importação inadequadas de módulos em outros(algo realmente perceptível quando testamos o MV sozinho). Este processo de depuração foi fundamental para compreendermos melhor a estrutura e o funcionamento do código.

É importante notar que houve uma dificuldade, em um primeiro momento, na análise do processador e como ele seria implementado, a aula do dia 15/07 nos ajudou a repensar no funcionamento do processador. Além disso, a sintaxe extremamente tipada da linguagem dificultou a junção de módulos/arquivos separados para o funcionamento mútuo.

Sendo assim, com as correções realizadas, fomos capazes de executar as instruções restantes, alcançando os resultados esperados. E, portanto, concluir com êxito a primeira parte do projeto.