#### UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE CENTRO DE INFORMÁTICA - CIn

#### RELATÓRIO DE PROJETO

#### Julio Cesar de Araujo Sobral Filho - jcasf

#### Alvaro Nascimento - av

#### RECIFE, 2023

**ÍNDICE**

1. [**Introdução 3**](#_gjdgxs)
2. [**Materiais e Métodos 3**](#_30j0zll)
3. [**Implementações 3**](#_1fob9te)
   1. [Instruções Registrador - Registrador 3](#_3znysh7)
      1. [SUB 4](#_2et92p0)
      2. [SLT 4](#_tyjcwt)
      3. [XOR 4](#_3dy6vkm)
      4. [OR 4](#_1t3h5sf)
   2. [Instruções Registrador - Imediato 4](#_4d34og8)
      1. [ADDI 4](#_2s8eyo1)
      2. [SLTI 4](#_17dp8vu)
      3. [SLLI 4](#_3rdcrjn)
      4. [SRLI 5](#_26in1rg)
      5. [SRAI 5](#_lnxbz9)
      6. [LUI 5](#_35nkun2)
   3. [Instruções de Transferência de Controle 5](#_1ksv4uv)
      1. [JAL 5](#_44sinio)
      2. [JALR 5](#_2jxsxqh)
      3. [BNE 5](#_z337ya)
      4. [BLT 6](#_3j2qqm3)
      5. [BGE 6](#_1y810tw)
   4. [Instruções de Load e Store 6](#_4i7ojhp)
      1. [LB 6](#_2xcytpi)
      2. [LH 6](#_1ci93xb)
      3. [LBU 6](#_3whwml4)
      4. [SB 6](#_2bn6wsx)
      5. [SH 6](#_qsh70q) [7](#_1pxezwc)
4. **Simulações de Referência 7**
   1. [Simulações ALU 7](#_49x2ik5) [8](#_147n2zr)
   2. [Simulações LOAD 9](#_3o7alnk) [10](#_ihv636)
   3. [Simulações de Desvio 11](#_32hioqz) [16](#_1v1yuxt)
   4. [Simulações de STORE 18](#_4f1mdlm) [19](#_3tbugp1)
5. [**Conclusão 20**](#_28h4qwu)

# Introdução

O projeto visou a implementação de um processador de arquitetura RISC-V utilizando o modelo de pipeline, como parte da disciplina de Infraestrutura de Hardware. O objetivo era aplicar conhecimentos adquiridos ao longo do curso, enfrentando desafios práticos na implementação de instruções em linguagem de descrição de hardware, especificamente SystemVerilog. O projeto teve como orientadora a professora Edna Natividade.

# Materiais e Métodos

A base do projeto foi um modelo disponibilizado pelos monitores da disciplina. As implementações das instruções foram feitas a partir deste modelo, usando o software de simulação ModelSim, um ambiente que permite simulações para linguagens de descrição de hardware como SystemVerilog. O projeto foi versionado e integrado pela equipe através da plataforma GitHub, facilitando a colaboração entre os membros.

# Implementações

## Instruções Registrador - Registrador

Para implementar as funcionalidades dessa categoria, foi preciso definir no controlador da ALU um caso que levasse em conta as partes do OPcode que são específicas de cada instrução (funct3 e funct7) para permitir o tratamento da mesma na ALU. Nesse componente, por sua vez, realiza-se a execução da operação específica da instrução, com base no resultado provindo do controlador.

### SUB

Utilizamos o caso da operação ser (0110) e definimos a saída ALUResult = SrcA - SrcB.

### SLT

Utilizamos o caso da operação ser (1100) e separamos a saída em três casos:

* + - * Quando o primeiro registrador tem um valor negativo e o segundo um valor positivo, o primeiro sempre será menor, logo ALUResult = 1;
      * Quando o primeiro registrador tem um valor positivo e o segundo tem valor negativo, o primeiro nunca será menor que o segundo, logo ALUResult = 0;
      * Para outra configuração diferente dos dois casos citados acima, usamos um operador ternário que verifica se (SrcA < SrcB) e atribui o resultado a ALUResult.

### XOR

Utilizamos o caso da operação ser (0101) e definimos a saída ALUResult = SrcA ^ SrcB;

### OR

Utilizamos o caso da operação ser (0001) e definimos a saída ALUResult = SrcA | SrcB;

## Instruções Registrador - Imediato

Para implementar as instruções que requerem um imediato, nós modificamos o Gerador de Imediato para reconhecer as instruções I-Type e o controlador para para permitir o uso da ALU e da escrita do Registrador. Adicionamos na Alu novos casos para as instruções. No ALUController, usamos o código da operação, a Funct3 e Funct7 de cada instrução para encaminhar para o caso da Alu correspondente a instrução. O registrador da instrução fica no fio “SrcA” e o imediato fica no fio “SrcB”.

### ADDI

Utilizamos o caso da operação ser (0010) e definimos a saída ALUResult = SrcA + SrcB;

### SLTI

Utilizamos o caso da operação ser (1100) e separamos a saída em três casos:

* + - * Quando o registrador tem um valor negativo e o Imediato um valor positivo, o registrador sempre será menor que o Imediato, logo ALUResult = 1;
      * Quando o registrador tem um valor positivo e o imediato tem valor negativo, o

registrador nunca será menor que o imediato, logo ALUResult = 0;

* + - * Para outra configuração diferente dos dois casos citados acima, usamos um operador ternário que verifica se (SrcA < SrcB) e atribui o resultado a ALUResult.

### SLLI

Utilizamos o caso da operação ser (0111) e o operador shift lógico à esquerda do SystemVerilog e atribuímos o resultado a ALUResult na ALU.

### SRLI

Utilizamos o caso da operação ser (1111) e o operador shift lógico à direita do SystemVerilog e atribuímos o resultado a ALUResult na ALU.

### SRAI

Utilizamos o caso da operação ser (1110) e o operador de shift lógico aritmético à direita do SystemVerilog e atribuímos o resultado a ALUResult na ALU.

### LUI

Utilizamos o caso da operação ser (1011), retorna o valor de ScrB e põe 0 nos últimos 12 bits e atribuímos o resultado a ALUResult na ALU.

## Instruções de Transferência de Controle

**Desvio indondicional:** Para implementar as funções de desvio incondicional, o controller foi modificado para reconhecer as intruções e enviar o sinal (um para indicar que é um desvio incondicionl e outro para indicar que o resultado da busca nos registradores deve ser usado)para o data path o sinal “é computado” na segunda parte do pipeline e é guardado no banco B de registradores, na terceira parte do pipeline foi adicionado um mux o qual seleciona o valor do sourceA de acordo com o sinal, esse valor é o que será guardado no registrador, e será o valor vindo do registrador ou o próprio Pc+4. O Branch unit também foi adaptado, ele agora recebe o valor obtido do registrador e os sinais do tipo-J, caso o jalr seja identificado o valor do próximo pc é atualizado para o valor guardado no registrador + Imm.

### JAL

O sinal jal é ativo, o pc+4 é escrito nos registradores e o próximo pc atualizado para Pc+Imm.

### JALR

Os dois sinais jal e jalr são ativos, o pc+4 é escrito nos registradores e o próximo pc é atualizado para FAmux\_Result+Imm.

**Desvio Condicional:** Os desvios condicionais foram implementados modificando o “Controller” para reconhecer as instruções do tipo Branch, a “ALU” e “ALUController” para enviar o resultado das operações que decidem se ocorrerá desvio e a “Branch Unit” para mandar o sinal vindo da ALU e, se ocorrer desvio, atribuir PC + Imediato ao endereço de desvio.

### BNE

Utilizamos o caso da operação ser (1001) e o operador lógico “ != ” do SystemVerilog para verificar se os valores nos registradores são diferentes.

### BLT

Utilizamos o mesmo caso de operação do SLT e SLTI já que suas operações na “ALU” são equivalentes.

### BGE

Utilizamos o caso da operação ser (1010) e separamos a saída em três casos, similar ao que foi feito para verificar se o valor de SrcA é menor que SrcB:

* + - * Quando o primeiro registrador tem um valor negativo e o segundo registrador tem um valor positivo, o primeiro registrador nunca será maior que o segundo, logo ALUResult = 0;
      * Quando o primeiro registrador tem um valor positivo e o segundo registrador tem

valor negativo, o primeiro registrador sempre será maior que o segundo, logo ALUResult = 1;

* + - * Para outra configuração diferente dos dois casos citados acima, usamos um operador ternário que verifica se (SrcA >= SrcB) e atribui o resultado a ALUResult.

## Instruções de Load e Store

**Load:** As funções de load envolvem capturar diferentes números de bits, a escolha de implementação foi capturar a palavra completa (32 bits) da memória e resetar os bits que não eram necessários estendendo o bit de sinal (fora a lbu que despreza o sinal), assim só é escrito o que é necessário.

### LB

Considera os byte menos significativos e o sinal.

### LH

Considera os 2 bytes menos significativos e o sinal.

### LBU

Considera o byte menos significativo e desconsidera o sinal.

**Store:** Para implementar os stores nós modificamos o sinal “Wr” que é enviado para saber qual Byte vai ser escrito na memória, no caso de uma half, desativamos o sinal de escrita para os dois Bytes mais significativos, assim só os 2 bytes menos significativos serão escritos.

### SB

Escreve o byte menos significativo da palavra passada na memória.

### SH

Escreve os 2 bytes menos significativos da palavra passada na memória.

1. **Simulações de Referência**

Nesta seção, apresentaremos as simulações realizadas em cada uma das instruções implementadas neste projeto, as simulações foram previamente informadas no repositório de referência.

## Simulações ALU

Estas simulações foram fornecidas no repositório de referência para o teste das instruções que utilizam a ALU para realizar as suas operaçõe

### 

## Simulações LOAD

Estas simulações foram fornecidas no repositório de referência para o teste das instruções que utilizam o acesso à memória e carregam valores para realizar operações.

## Simulações JAL, BEQ

Estas simulações foram fornecidas no repositório de referência para o teste das instruções que realizam desvios no código para as suas operações.

## Simulações STORE

Estas simulações foram fornecidas no repositório de referência para o teste das instruções que realizam armazenamentos na memória para as suas operações.

# Conclusão

O projeto revelou-se um desafio significativo, proporcionando valiosos aprendizados sobre a complexidade da implementação de processadores. A implementação de um pipeline, ainda que em versão simplificada, foi um exercício que aprofundou a compreensão da equipe sobre a intrincada natureza dos sistemas de hardware.