PUC
CAMPINAS
PONTIFÍCIA UNIVERSIDADE CATÓLICA

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS
ENGENHARIA DE COMPUTAÇÃO

Gustavo Yudi Ito RA: 23004097
Matheus Pithon RA:23005808

Campinas, 2025

1. Introdução

Este projeto apresenta o desenvolvimento de uma CPU simples com arquitetura pipeline de cinco estágios. O objetivo é compreender o fluxo de execução dentro do pipeline e os impactos da segmentação no desempenho. A CPU implementa operações aritméticas, acesso à memória e desvios básicos. Não foram incluídos mecanismos de detecção ou resolução de hazards, de modo que dependências entre instruções devem ser tratadas manualmente com NOPs. O relatório descreve a topologia da CPU, sua especificação e os testes realizados.

1.1 Descrição textual do projeto – Topologia da CPU

O projeto consiste na implementação de uma CPU simples, utilizando arquitetura pipeline com os estágios clássicos:

1. **IF – Instruction Fetch**
2. **ID – Instruction Decode**
3. **EX – Execute / ALU**
4. **MEM – Access Memory**
5. **WB – Write Back**

A CPU foi projetada para executar o conjunto reduzido de instruções especificado pelo professor (NOP, LW, SW, ADD, SUB), porém **sem implementação de tratamento de hazards**.
Ou seja:

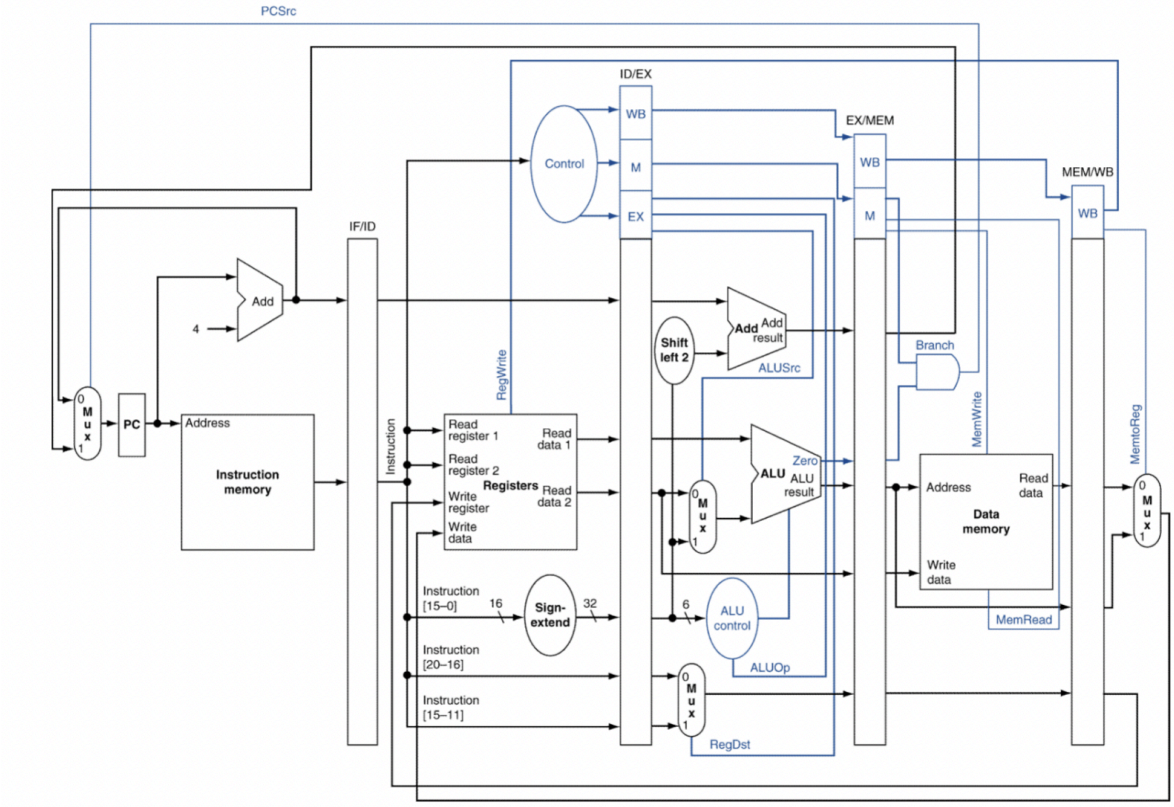
- **Não há forwarding/bypassing**
- **Não há stall automático**
- **Não há detecção de hazard de dados**
- **Não há tratamento de control hazard**

Na prática, isso significa que a CPU apenas executa sequencialmente o pipeline, podendo produzir resultados incorretos caso existam dependências entre instruções consecutivas. Portanto, cabe ao usuário inserir NOPs manualmente entre instruções dependentes.

A topologia geral inclui:

- **Banco de registradores** com 16 registradores de 16 bits
- **Memória de dados** e **memória de instruções**

- **Unidade de Controle** responsável por gerar sinais de controle conforme o opcode
- **ALU** responsável por operações de soma e subtração
- **PC (Program Counter)**
- **Registradores de pipeline** (IF/ID, ID/EX, EX/MEM, MEM/WB)



2.1 Registradores (quantidade, endereço e tamanho)

O banco de registradores é composto por 16 registradores de largura 16 bits, endereçados por 4 bits (0 a 15). Possui duas portas de leitura assíncronas e uma porta de escrita síncrona habilitada pelo sinal RegWrite. O registrador R0 é protegido contra escrita (WRITE_REGISTER ≠ "0000" é requisito), atuando como registrador zero. As leituras retornam READ_DATA_1 e READ_DATA_2 a partir dos seletores READ_REGISTER_1 e READ_REGISTER_2, enquanto a escrita ocorre na borda de subida do clock em WRITE_REGISTER, com o dado WRITE_DATA quando RegWrite = 1.

2.2 Formato das instruções (OPCODE)

Em relação às instruções, temos:

Tipo R:

3 bits	4 bits	4 bits	4 bits	1 bits
OP	**RS**	**RT**	**RD**	**ADD/SUB**

Tipo I:

3 bits	4 bits	4 bits	5 bits
OP	**RS**	**RT**	**OFFSET END**

INSTRUÇÃO	OPCODE	SIGNIFICADO
NOP	000	No operation
LW	001	Rt ← MEM[rs+offset end]
SW	010	MEM[rs+offset end] ← Rt
ADD	011	Ri ← Rj + Rk
SUB	011	Ri ← Rj - Rk
ADDI	100	Ri ← Rj + imediato
SUBI	101	Ri ← Rj - imediato

Segue o formato de como funciona o nossa CPU.

Tipo R:

XXX	XXXX	XXXX	XXXX	X
OPCODE	RS	RT	RD	ADD/SUB
011(SOMA SUB)	0001	0010	0011	0(soma)

Obs: se o último bit for 1 vira subtração.

Tipo I:

XXX	XXXX	XXXX	XXXXX
OPCODE	RS	RT	IMEDIATO
100(ADDI)	0000	0001	00011

Obs: se o opcode for 101 é SUBI.

A tabela abaixo resume como a ControlUnit mapeia o Opcode para os vetores de controle, conforme definido no processo principal

Instrução	Opcode (bits 0-2)	WB (2bits)	MEM(3bits)	EX (5 bits)	Descrição
NOP	000	00	000	00000	Nenhuma operação.
LW	001	10	010	00001	Leitura de memória (Load).
SW	010	0X	001	00001	Escrita em memória (Store).
R-TYPE	011	11	0X0	10100	Operações aritméticas tipo R (ADD/SUB).
ADDI	100	11	0X0	00111	Soma imediata.
SUBI	101	11	0X0	01001	Subtração imediata.
Outros	*Default*	00	0X0	XXXXX	Estado padrão para opcodes inválidos..

Mapeamento da Placa

Usamos:

7-Segment Displays

HEX 0 - Vai mostrar o Rd

HEX 1 - conteúdo reg2

HEX 2 - conteúdo reg1

HEX 3 - Resultado da Ula

HEX 4 - B

HEX 5 - A

HEX 6 - imediato

HEX 7- PC

LEDR(0) <='1' quando escreve em registrador

3. Resultados

O programa de teste foi definido na memória de instruções. Ele foi projetado para exercitar as instruções suportadas pela CPU (NOP, LW, SW, ADD, SUB, ADDI, SUBI). Os testes foram executados na placa EP4CE115F29C7, no programa Quartus.

3.1 Descrição dos testes realizados

Segue a memória de instrução, mostrando as seguintes ordens de instruções:

```
-- 0: ADDI $1, $0, 3   ; R1 = 3
-- 1: ADDI $2, $0, 2   ; R2 = 2
-- 2: NOP
-- 3: NOP
-- 4: SUBI $3, $1, 2   ; R3 = R1 - 2 = 1
-- 5: NOP
-- 6: NOP
-- 7: ADD  $3, $1, $2  ; R3 = R1 + R2 = 5
```

https://youtu.be/W0M5HZU_gFU

3.2 Resultados e Discussão

O pipeline IF–ID–EX–MEM–WB funcionou conforme projetado, com todos os registradores de estágio propagando corretamente os valores.

Entretanto, devido à ausência de mecanismos de detecção e resolução de hazards, foi possível observar situações em que o pipeline produz resultados incorretos quando instruções dependentes são colocadas consecutivamente. Por exemplo, uma instrução do tipo R que escreve em um registrador seguida imediatamente por uma instrução que lê o mesmo registrador exibe comportamento indeterminado. Nesses casos, a inclusão manual de NOPs entre as instruções eliminou o problema, confirmando o comportamento esperado para um pipeline sem forwarding e sem stall automático.

- O PC avança de 2 em 2
- OPCODE tem 3 bits

Outro ponto observado foi que os sinais exibidos nos displays HEX0 a HEX7 facilitaram a identificação do fluxo de dados na CPU, permitindo verificar corretamente o valor do PC, o imediato, os operandos A e B da ULA, o resultado da operação e os registradores acessados.

De forma geral, os resultados confirmaram que a CPU implementada está funcional, executa corretamente seu conjunto de instruções e apresenta o comportamento típico de uma arquitetura pipeline sem mecanismos de hazards — funcionando corretamente desde que o código seja escrito com o espaçamento adequado entre instruções dependentes.

4. Bibliografia

1. PATTERSON, David A. e HENNESSY, John L. *Computer Organization and Design – The Hardware and Software Interface*. Morgan Kaufmann, 5ª ed., 2013.