THE AMERICAN UNIVERSITY IN CAIRO
الجـامـعة الأمـريكيـة بالقـاهـرة

School of Sciences and Engineering
Department of Computer Science and Engineering
Fall 2024

CSCE 3301: Computer Architecture

Project 01: Single Cycle CPU

Dr. Cherif Salama

Submitted By:

Farida Bey - 900212071
Arwa Abdelkarim - 900221451
Date: Oct 28, 2024

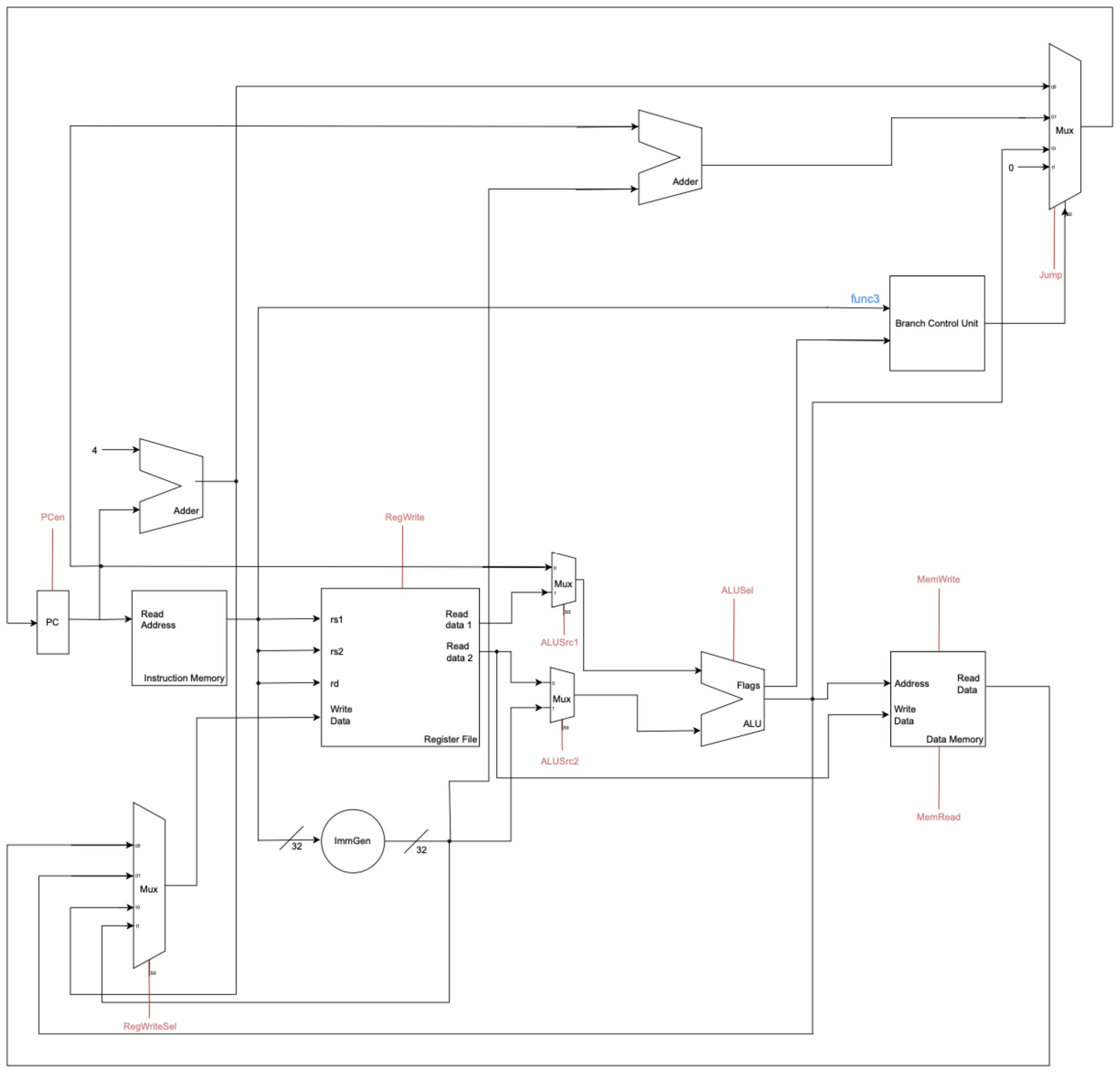
TABLE OF CONTENTS

GitHub repo link: https://github.com/arwaabdelkarim/RISC-V-Processor.git

1. Project Objectives:

The main aim of this milestone is to create a fully functional RV32I single cycle processor which supports the 43 instructions including FENCE, ECALL and EBREAK where FENCE and ECALL act as nops and EBREAK acts as a halting instruction.

2. Design of the Datapath:



3.Implementation Overview:

This is a brief overview of of our project modules.

PC: The Program Counter (PC) is implemented as a register that increments by 4 when the PC enable (PCen) signal is active. It also checks the BranchAnd and Jump signals to determine whether to add the immediate value from ALU_out instead of incrementing by 4, facilitating branching and jumping operations.

Instruction Memory: This component is designed as a simple 256-entry register file, with each entry being 32 bits wide. It is dedicated to instruction storage for our single-cycle processor, receiving instruction addresses from the PC to fetch the corresponding instruction.

Register File: The register file manages read and write operations based on the instruction inputs from the Instruction Memory. Upon activation of the reset signal (rst), all registers are cleared to zero. It continually reads two source registers and only writes to the destination register (RD) if the WriteReg signal is enabled.

ALU: The Arithmetic Logic Unit (ALU) takes inputs ALU_A and ALU_B, with the ALUSel signal dictating the operation type (arithmetic or logical). It also generates flags used by the Branch Control Unit, primarily by computing the difference between inputs A and B.

Data Memory: Similar in structure to the Instruction Memory, the Data Memory accepts ALU_out as the address for read/write operations. It reads data from the specified address when the readMem signal is active and writes data when the WriteMem signal is enabled. The funct3 signal is also included to determine the number of bytes to read or write for load/store instructions.

Control Units: These units are not depicted in the datapath for simplicity:

- **Control Unit**: This unit assigns control signals for each instruction based on the opcode.
- **ALU Control Unit**: It takes the func3 of the instruction and the ALUOp to generate the ALUSel signal, determining the specific operation for the ALU.
- **Branch Control Unit**: It evaluates func3 and the relevant flags for branch instructions to assign the BranchAnd signal based on specific flag comparisons.

We also included A description for each module in our .v files.

4.Test:

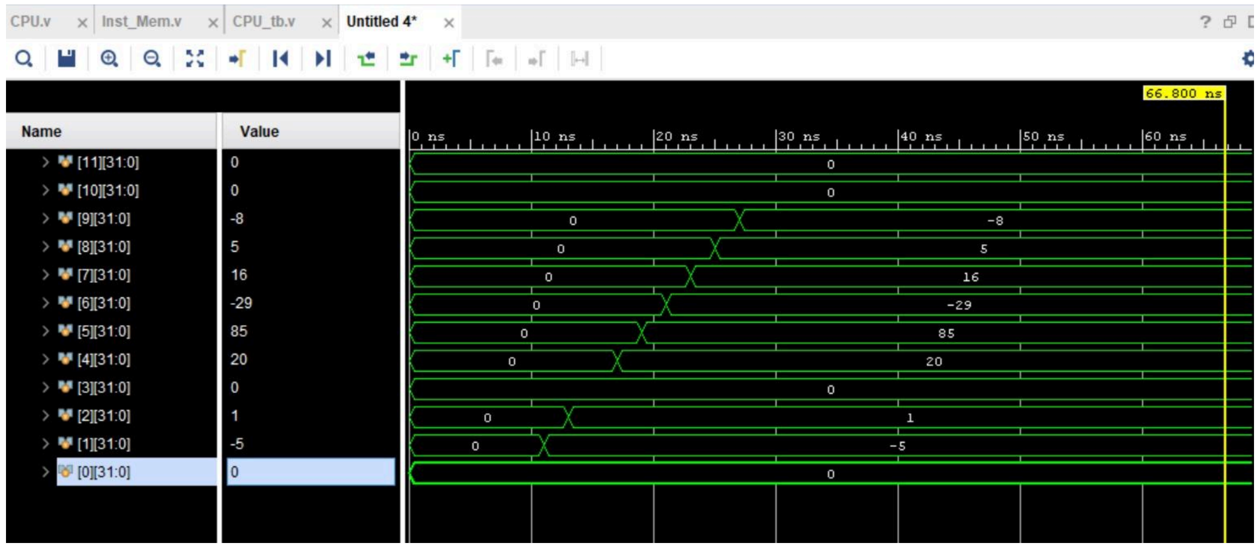
We divided the testing of the 40 instructions across several test programs, creating a total of 4 test programs. 3 of the programs were designed specifically to verify the functionality of various instructions, rather than performing specific operations. The fourth program implemented a function that writes an array to memory and swaps its elements.

Test program 1:
Assembly:

```
addi x1, x0, -5
slti x2, x1, 3
sltiu x3,x1, 3
xori x4, x3, 20
 ori x5, x4, 85
andi x6, x1, -25
slli x7, x2, 4
 srli x8, x4, 2
srai x9, x6, 2
```

Machine Code:

```
11111111101100000000000010010011
00000000001100001010000100010011
00000000001100001011000110010011
00000001010000011100001000010011
00000101010100100110001010010011
11111110011100001111001100010011
00000000010000010001001110010011
00000000001000100101010000010011
01000000001000110101010010010011
```



Name	Value
[11][31:0]	0
[10][31:0]	0
[9][31:0]	-8
[8][31:0]	5
[7][31:0]	16
[6][31:0]	-29
[5][31:0]	85
[4][31:0]	20
[3][31:0]	0
[2][31:0]	1
[1][31:0]	-5
[0][31:0]	0

Test program 2:

Assembly:

```
lbu x1, 0(x0)
lh x2, 4(x0)
lb x3, 8(x0)
lhu x4, 12(x0)
sll x5, x1, x2
slt x6, x3, x1
sltu x7, x3, x1
xor x8, x6, x5
srl x9, x3, x2
sra x10,x3, x2
```

Machine Code:

```
inst_mem
00000000000000000100000010000011
00000000010000000001000100000011
00000000100000000000000110000011
00000000110000000101001000000011
00000000001000001001001010110011
00000000000100011010001100110011
00000000000100011011001110110011
00000000010100110100010000110011
00000000001000011101010010110011
01000000001000011101010100110011
data_mem
00000000000000000000000000011000
00000000000000000000000000000010
00000000000000000000000010000010
00000000000000000000000000000011
```