P2 Design Documentation

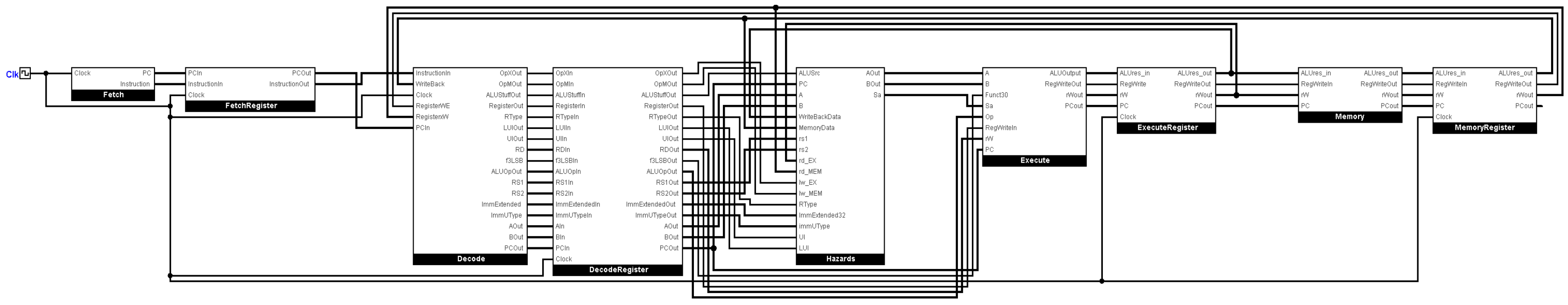
Oscar So (ons4) Michael Rigney (mjr372)

1 Introduction

The purpose of this document is to understand our design of pipelining a RISC-V CPU on Logisim. The document will cover an overview of the document, as well as descriptions for each stage in the CPU and also include sub-circuits and testing. This document should provide people, with knowledge of circuits, who view our design a thorough understanding of each stage component.

2 Overview

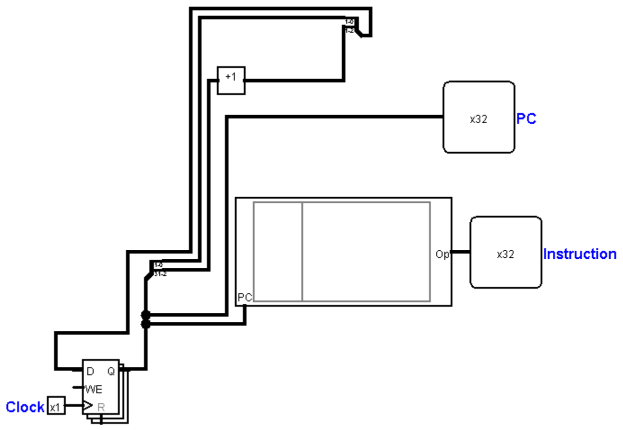
The RISC-V CPU that we will be working on is a 5-staged pipelined CPU that takes in operations and executes it to be written in the memory. In simple words, this is the backbone behind simple codes like logical and arithmetic statements.



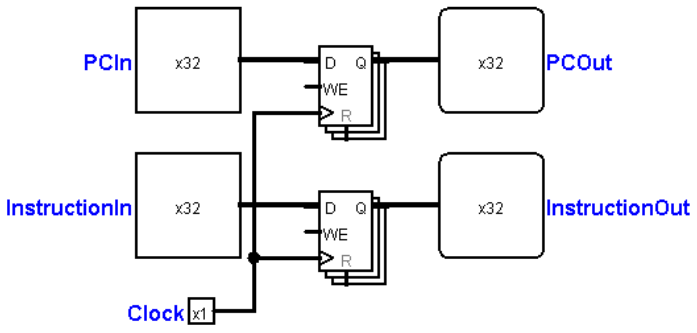
3 The Fetch Stage

3.1 Circuit Diagram

Fetch



IF/ID Pipeline



3.2 Program Counter (PC)

Register holding the address of the current instruction in instruction memory. Each instruction is 32 bits.

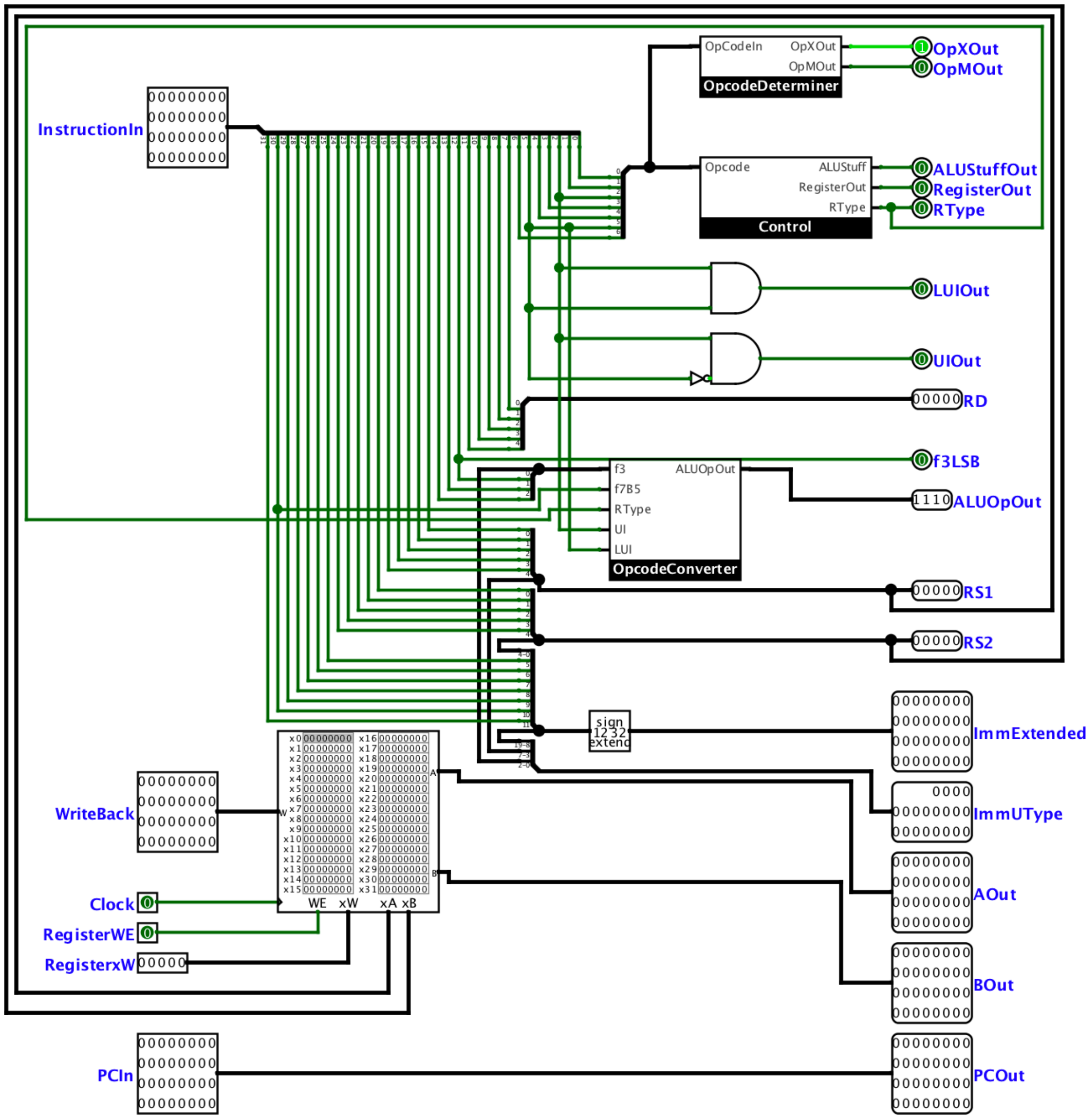
3.3 IF/ID Pipeline Register

Takes in the 32 bit instruction to be sent to the decoder, as well as the PC + 4 for the address of the next instruction to be fed back into instruction memory.

Note: To increment the PC by 4 (decimal) we add 1 to the 3rd bit of the PC.

4 The Decode Stage

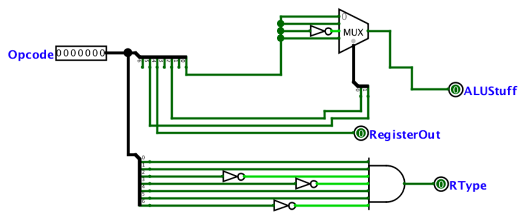
4.1 Circuit Diagram



Takes in instructions to determine everything that needs to be passed through to the next stage like A, B, immediates, different instruction types, ALUoutput, RS1, RS2, RD, control system, etc… Also has a register file that takes in write back data, clock, register write (WE) and the register location storage (xW)

4.2 Control Unit

Take in the instruction bits from IF/ID pipeline register and route the read registers and write register to the Register file. Route the operation to the ALU.

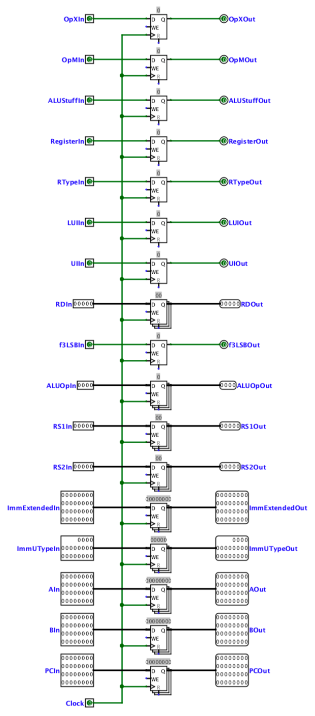


The control unit, as stated above, takes in the Opcode and determines whether 1) the ALU should be used for the particular instruction, 2) the register should be writing this particular instruction into the register file, and 3) if the instruction is RType, which would be helpful when converting the instruction into ALUop.

4.3 Register File

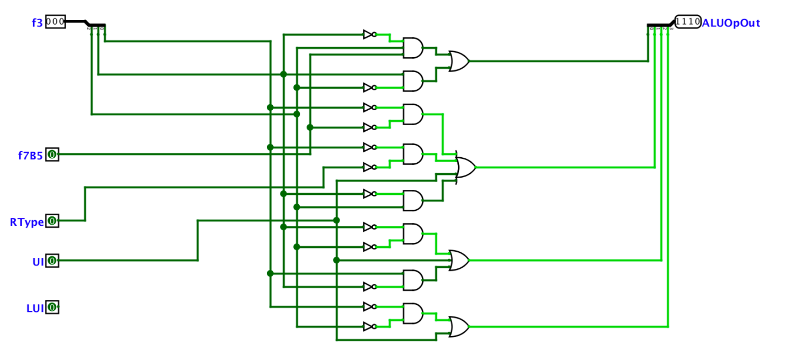
Contains registers with values to be accessed and operated on per the instructions. Stores write back results of operations in registers as well. Outputs A and B 32-bit values to the ID/EX pipeline to be used in the ALU.

4.4 ID/EX Pipeline Register



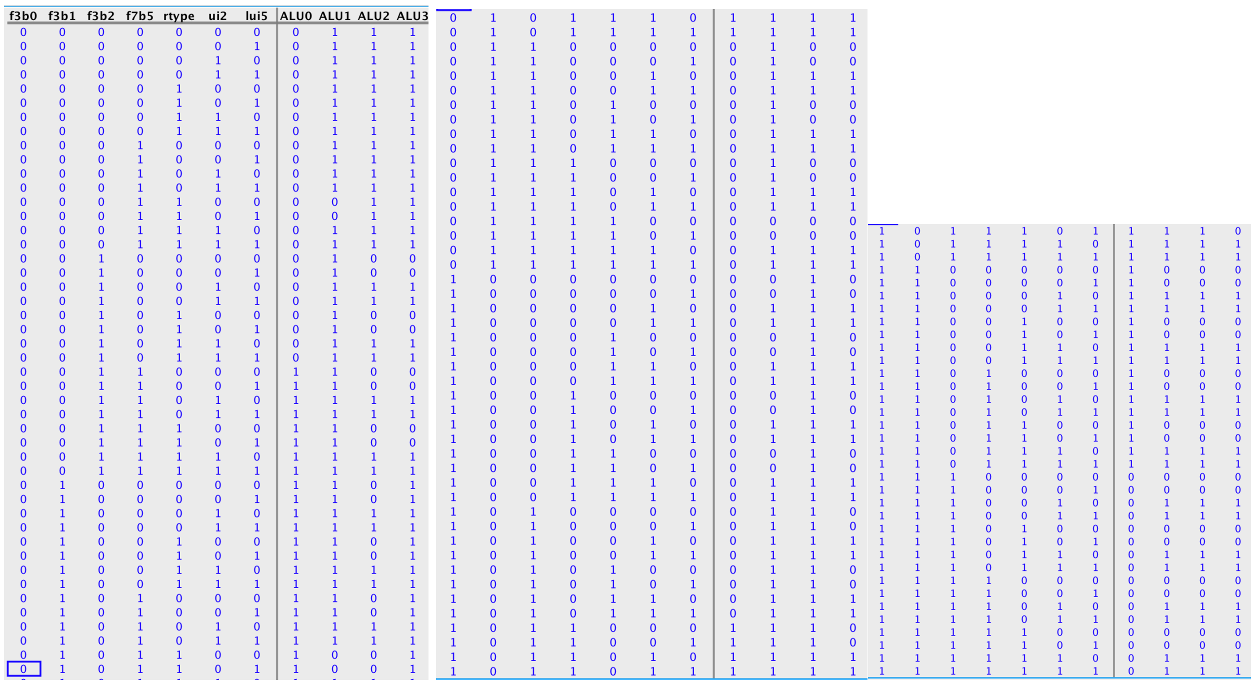
Takes in control information for the execution stage (A,B, ctrl) as well as immediates and offsets. Also takes in PC + 4 for computing branch targets later on

4.5 Instruction to ALUOp Converter



After determining which bits are most important for determining the ALU operator, the truth table is then run through combinations logical circuit analysis on logism to generate the optimal circuit given inputs of f3, bit 5 of f7, RType signal, and UI (2nd bit of instruction).

4.5.1 Truth Table



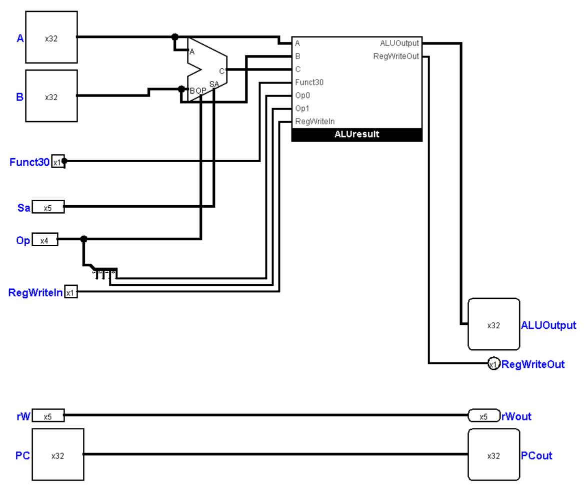
f3b0	f3b1	f3b2	f7b5	rtype	ui2	lui5	ALU0	ALU1	ALU2	ALU3
0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	1	0	0	1	1	1
0	0	0	0	0	1	1	0	1	1	1
0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	1	0	1	0	1	1	1
0	0	0	0	1	1	0	0	1	1	1
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	0	0	1	1	1
0	0	0	1	0	0	1	0	1	1	1
0	0	0	1	0	1	0	0	1	1	1
0	0	0	1	0	1	1	0	1	1	1
0	0	0	1	1	0	0	0	0	1	1
0	0	0	1	1	0	1	0	0	1	1
0	0	0	1	1	1	0	0	1	1	1
0	0	0	1	1	1	1	0	1	1	1
0	0	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0	1	1	1
0	0	1	0	0	1	1	0	1	1	1
0	0	1	0	1	0	0	0	1	0	0
0	0	1	0	1	0	1	0	1	0	0
0	0	1	0	1	1	0	0	1	1	1
0	0	1	0	1	1	1	0	1	1	1
0	0	1	1	0	0	0	1	1	0	0
0	0	1	1	0	0	1	1	1	0	0
0	0	1	1	0	1	0	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	0	1	1	1	0	0	1	1	0	0
0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	1	1	0	1
0	1	0	0	0	0	1	1	1	0	1
0	1	0	0	0	1	0	1	1	1	1
0	1	0	0	0	1	1	1	1	1	1
0	1	0	0	1	0	0	1	1	0	1
0	1	0	0	1	0	1	1	1	0	1
0	1	0	0	1	1	0	1	1	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	0	0	0	1	1	0	1
0	1	0	1	0	0	1	1	1	0	1
0	1	0	1	0	1	0	1	1	1	1
0	1	0	1	0	1	1	1	1	1	1
0	1	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	1	1	0	0	1
0	1	0	1	1	1	0	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1
0	1	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	1	0	1	0	0
0	1	1	0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	1	1	1
0	1	1	0	1	0	0	0	1	0	0
0	1	1	0	1	0	1	0	1	0	0
0	1	1	0	1	1	0	0	1	1	1
0	1	1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	0	0	1	0	0
0	1	1	1	0	0	1	0	1	0	0
0	1	1	1	0	1	0	0	1	1	1
0	1	1	1	0	1	1	0	1	1	1
0	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	0	1	0	0	0	0
0	1	1	1	1	1	0	0	1	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	0	1	0
1	0	0	0	0	1	0	0	1	1	1
1	0	0	0	0	1	1	0	1	1	1
1	0	0	0	1	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0	1	0
1	0	0	0	1	1	0	0	1	1	1
1	0	0	0	1	1	1	0	1	1	1
1	0	0	1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	1	1
1	0	0	1	0	1	1	0	1	1	1
1	0	0	1	1	0	0	0	0	1	0
1	0	0	1	1	0	1	0	0	1	0
1	0	0	1	1	1	0	0	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	0	0	0	0	1	1	0
1	0	1	0	0	0	1	0	1	1	0
1	0	1	0	0	1	0	0	1	1	1
1	0	1	0	0	1	1	0	1	1	1
1	0	1	0	1	0	0	0	1	1	0
1	0	1	0	1	0	1	0	1	1	0
1	0	1	0	1	1	0	0	1	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	0	1	1	1	0
1	0	1	1	0	0	1	1	1	1	0
1	0	1	1	0	1	0	1	1	1	1
1	0	1	1	0	1	1	1	1	1	1
1	0	1	1	1	0	1	1	1	1	0
1	0	1	1	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	1	1	0	0	0
1	1	0	0	0	1	0	1	1	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	0	1	0	0	1	0	0	0
1	1	0	0	1	0	1	1	0	0	0
1	1	0	0	1	1	0	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1	0	0	0
1	1	0	1	0	0	1	1	0	0	0
1	1	0	1	0	1	0	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	0	1	1	0	0	1	0	0	0
1	1	0	1	1	0	1	1	0	0	0
1	1	0	1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	0	0	1	1	1
1	1	1	0	0	1	1	0	1	1	1
1	1	1	0	1	0	0	0	0	0	0
1	1	1	0	1	0	1	0	0	0	0
1	1	1	0	1	1	0	0	1	1	1
1	1	1	0	1	1	1	0	1	1	1
1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0
1	1	1	1	0	1	0	0	1	1	1
1	1	1	1	0	1	1	0	1	1	1
1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	0	1	0	0	0	0
1	1	1	1	1	1	0	0	1	1	1
1	1	1	1	1	1	1	0	1	1	1

5 The Execute Stage

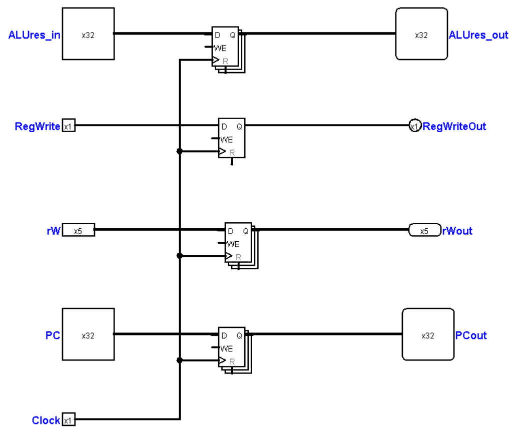
This circuit takes in A, B and ALU Op and ALU Sa. It feeds them into the ALU and then uses the ALUresult circuit to determine whether the ALU output or SLT/SLTU output should be passed into ALUOutput

5.1 Circuit Diagram

Execute



EX/MEM Pipeline



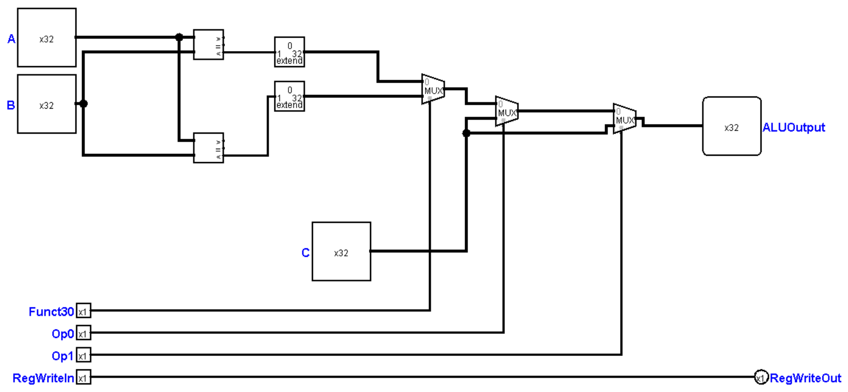
5.2 Arithmetic Logic Unit (ALU)

Takes A, B (32 bits each), and an operation code and outputs the computed result into the EX/MEM pipeline

5.3 EX/MEM Pipeline

Takes in the output of the ALU, PC + 4 and the past instruction index, and the write register and sends them to the MEM stage

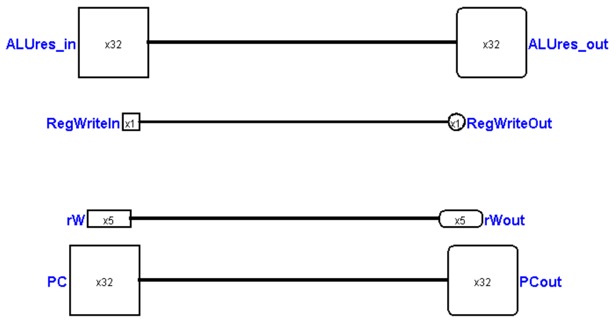
5.4 ALUResult



This circuit is **ALUresult**. This circuit chooses the output of the ALU to be either C, the computed value, or SLT if the 0th bit of funct3 = 0 or SLTU if the 0th bit of funct3 = 1. The 0 and 1 bits of Op are also used in subsequent mux's to determine whether the C value (ALU output) or SLT/SLTU is used. This also includes the comparisons used for SLT and other similar instructions.

6 The Memory Stage

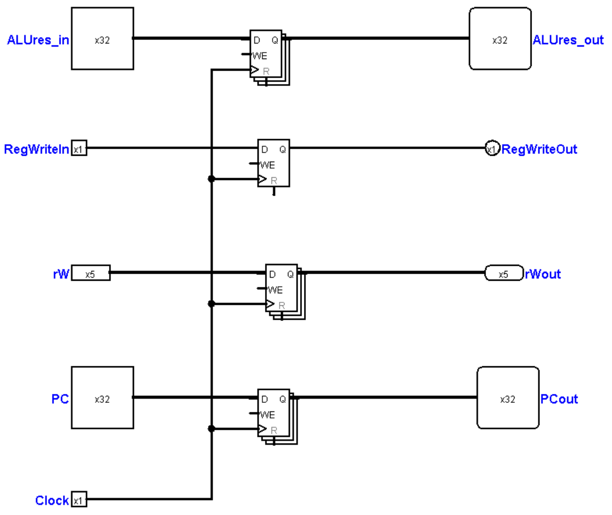
6.1 Circuit Diagram



This stage is simply a passthrough of the above inputs in this project. This step is to be used in load and store instructions after the instruction finishes executing in full RISCV CPUs.

7 The Writeback Stage

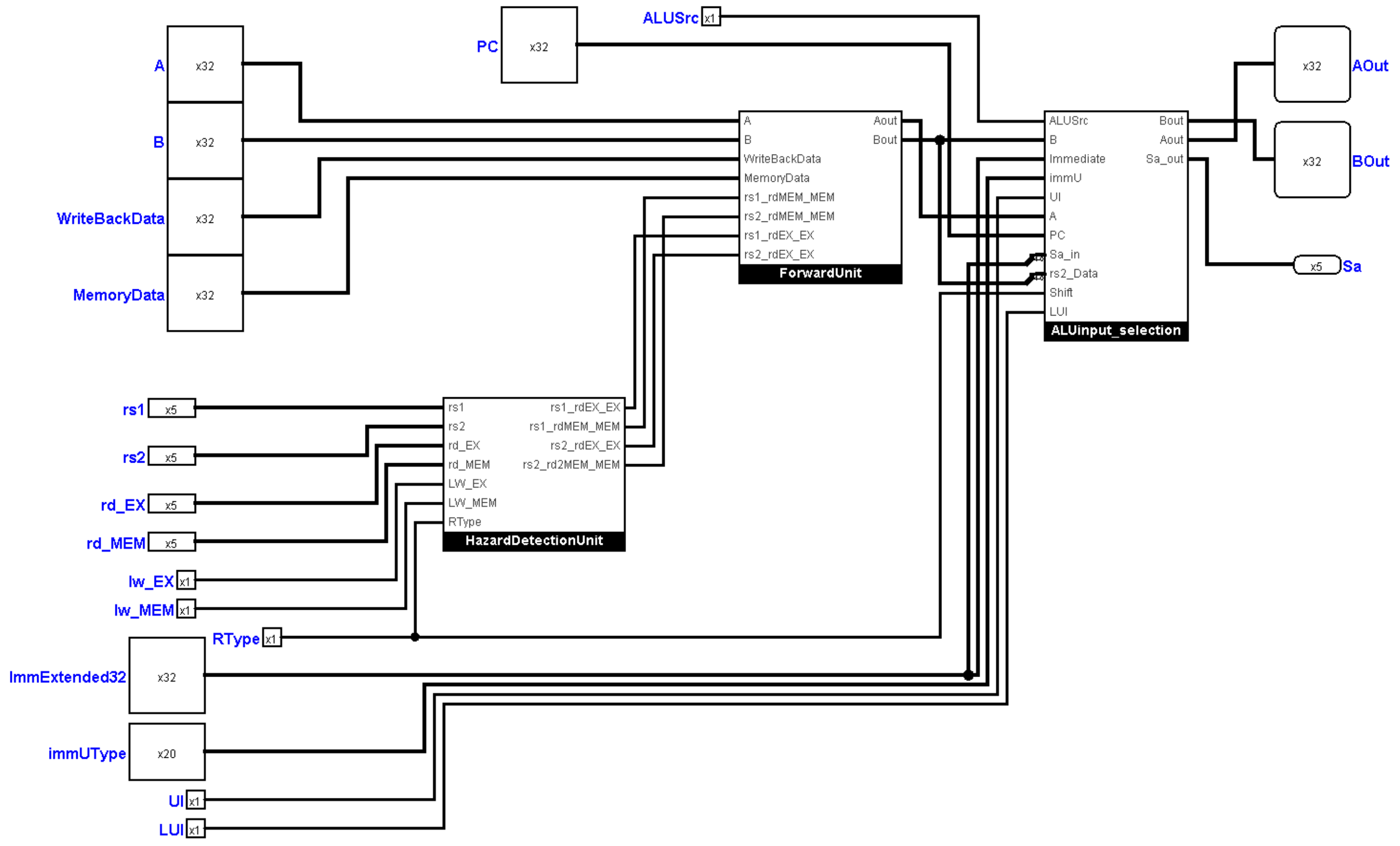
7.1 Circuit Diagram



7.2 MEM/WB Pipeline Register

Takes in the ALU operation result, result of memory operation, and control information for next instruction register index. Sends ALU result to write back to register file and updates PC in the event of a branch or jump.

8 Hazards and Forwarding



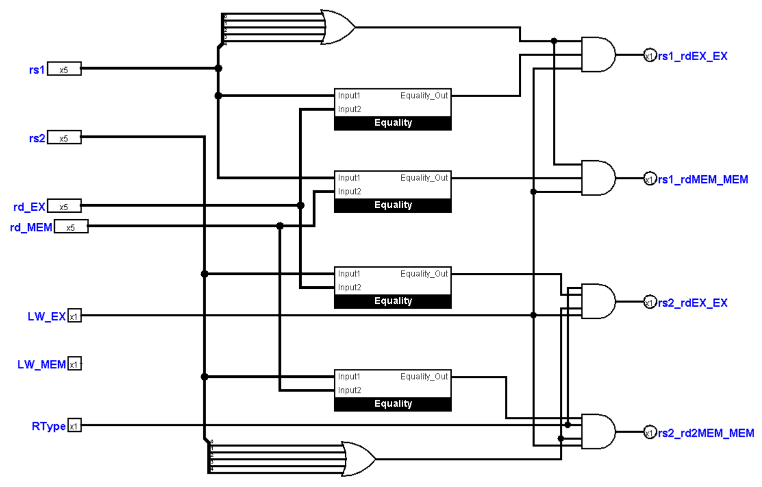
The following logical statements are given in the project guidelines:

EX Hazard

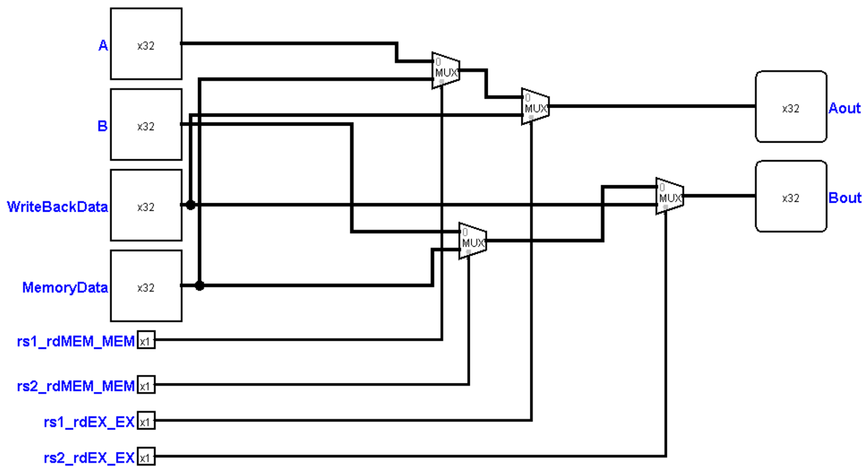
```
if (rs1_D == rd_X) and (rd_X != 0)
if (rs2_D == rd_X) and (rd_X != 0)
```

MEM Hazard

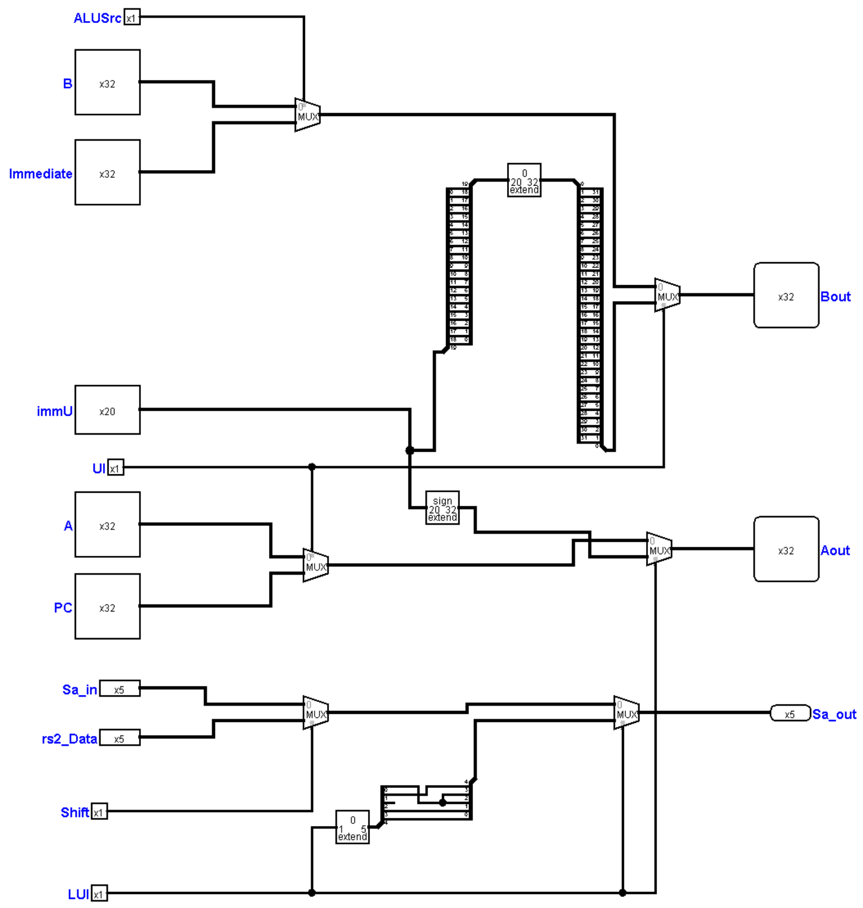
```
if (rs1_D == rd_M) and (rd_M != 0)
if (rs2_D == rd_M) and (rd_M != 0)
```



To resolve such hazards, we can forward values from the EX/MEM or MEM/WB registers. The circuit to the left is the **HazardDetetctionUnit** which outputs 0/1 values for rs1 and rs2 hazards.



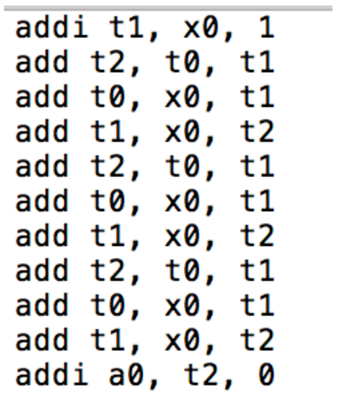
This circuit is the **ForwardUnit**. It takes in the hazard signals for memory and writeback hazards and forwards if needed. If rs1 MEM and rs1 EX are 0 then the original A value is passed into Aout. If rs1 MEM = 1 and rs1 EX = 0 then the MemoryData is used for Aout. If rs1 EX = 1 and rs1 MEM = 0 or 1 then the WriteBack Data is used for Aout. The same logic holds for rs2 and B/Bout.



This circuit is **ALUinput_selection**. It selects B either from the register or from an extended 32-bit immediate value depending on the ALUSrc value given from the Control unit. It selects A either from the register or the PC in the event of an AUIPC instruction. For shifting, the shift amount is selected from the Sa_in or the rs2Data depending on the value of the Shift 1-bit signal being fed in. In the event of a LUI instruction, the Sa_out is automatically set to 01100 as the smallest 12 bits are set to 0 in an LUI.

9 Testing

9.1 Fibonacci 4

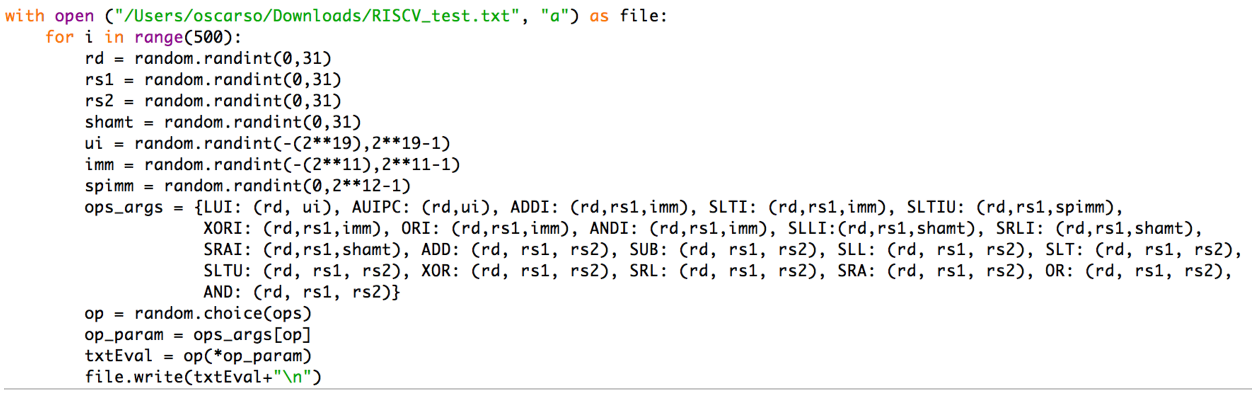


```
addi t1, x0, 1
add t2, t0, t1
add t0, x0, t1
add t1, x0, t2
add t2, t0, t1
add t0, x0, t1
add t1, x0, t2
add t2, t0, t1
add t0, x0, t1
add t1, x0, t2
addi a0, t2, 0
```

This fib4 assembly instruction is based off the C code that was given to us in the project description.

9.2 Random Testing

Code for random tests were also written in Python script by using Python's random package and generating different numbers for all types, following the guidelines from the simplified documentation link provided by the project description.



```python
with open ("/Users/oscarso/Downloads/RISCV_test.txt", "a") as file:
    for i in range(500):
        rd = random.randint(0,31)
        rs1 = random.randint(0,31)
        rs2 = random.randint(0,31)
        shamt = random.randint(0,31)
        ui = random.randint(-(2**19),2**19-1)
        imm = random.randint(-(2**11),2**11-1)
        spimm = random.randint(0,2**12-1)
        ops_args = {LUI: (rd, ui), AUIPC: (rd,ui), ADDI: (rd,rs1,imm), SLTI: (rd,rs1,imm), SLTIU: (rd,rs1,spimm),
                    XORI: (rd,rs1,imm), ORI: (rd,rs1,imm), ANDI: (rd,rs1,imm), SLLI:(rd,rs1,shamt), SRLI: (rd,rs1,shamt),
                    SRAI: (rd,rs1,shamt), ADD: (rd, rs1, rs2), SUB: (rd, rs1, rs2), SLL: (rd, rs1, rs2), SLT: (rd, rs1, rs2),
                    SLTU: (rd, rs1, rs2), XOR: (rd, rs1, rs2), SRL: (rd, rs1, rs2), SRA: (rd, rs1, rs2), OR: (rd, rs1, rs2),
                    AND: (rd, rs1, rs2)}
        op = random.choice(ops)
        op_param = ops_args[op]
        txtEval = op(*op_param)
        file.write(txtEval+"\n")
```

9.3 Edge Case Testing

Multiple tests were written for each "instruction" in Table A and Table B to see if the RISC-V CPU would behave as it should.