University of California Irvine

EECS 31L: Introduction to Digital Logic Laboratory (Spring 2025)

Lab 5: Single-Cycle RISC-V Processor

Anoop Koganti(koganti1@uci.edu)

Single-Cycle RISC-V Processor
(All bolded words are submodules of the Processor)

The Single-Cycle Processor is a CPU that provides an instruction within 1 clock cycle. The design is based off 2 inputs and an output, where the inputs are the clock and the reset button, whilst the output is the 32-bit Result variable, which provides the arithmetic results that it obtains. Inside the processor are three major components, and wires that connect these components inputs and outputs together. These three components are the Controller, ALUController, and the Datapath. The Datapath executes instructions by carrying out the actual computation, data movement, and register updates. The Controller generates the control signals needed to steer the datapath correctly depending on the instruction. Finally, the ALUController generates the exact ALU control signal needed for the specific operation. Block diagrams for these components are shown below, as well as the components for the submodules of Datapath:

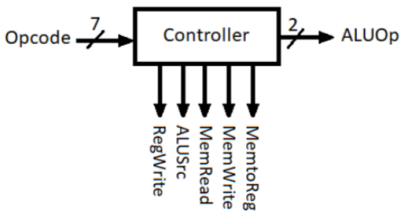


Figure 2 : Controller.

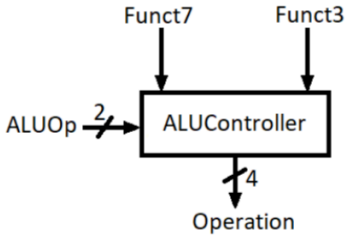
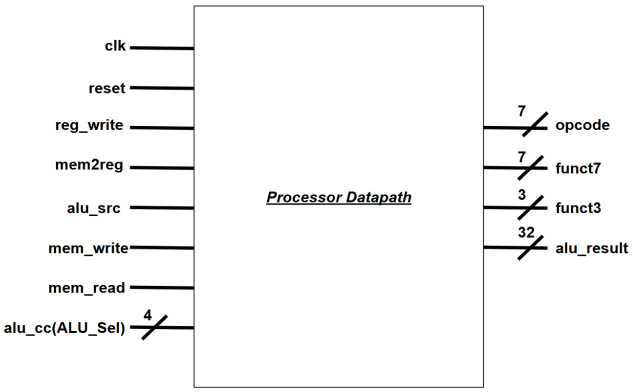
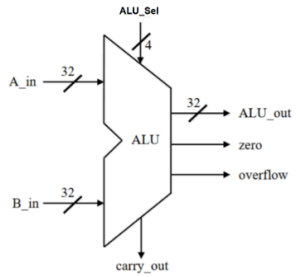


Figure 3 : ALUController.

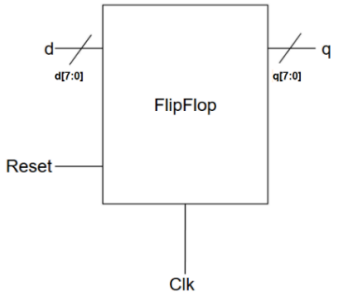
Datapath:



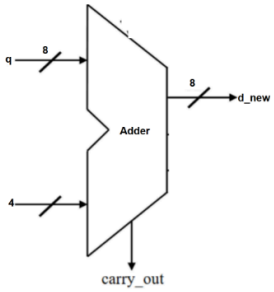
Arithmetic Logic Unit:



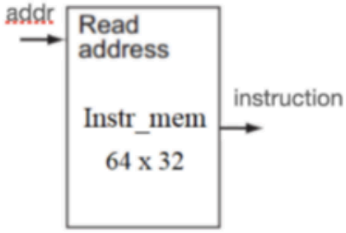
D-Flip Flop:



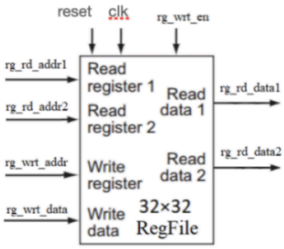
Adder:



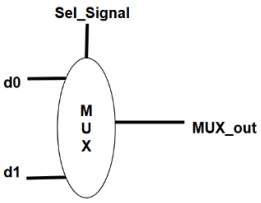
Instruction Memory:



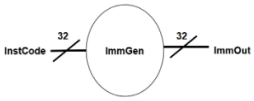
Register File:



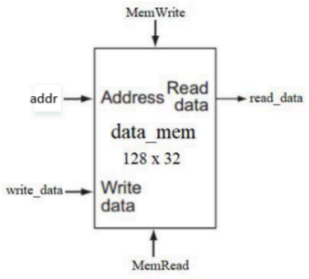
2-1 Multiplexer:



Immediate Generator:



Data Memory:



The Datapath consists of 9 components, which are the D Flip-Flop, Adder (These two combine to design the program counter), ALU, InstMem, RegFile, ImmGen, two 2-1 multiplexers, and a DataMem. Below are the descriptions of these components and their value, from Lab 4.

The **Program Counter** is implemented using an 8-bit D Flip-Flop that updates every rising clock edge. This PC holds the address of the current instruction. It is updated by a simple addition of 4 to the current PC value (pc_next = pc_out + 4) since all instructions are 4 bytes long. This adder is implemented directly using the + operator in Verilog and is not designed as a separate module.

The **Instruction Memory** uses the current PC value to obtain a 32-bit instruction. From this, the extractions for the opcode, rd, rs1, rs2, funct3, and funct7 are done. These fields are used to determine how the rest of the datapath behaves and which operands are used for computation.

The **Register File** module uses the rs1 and rs2 fields to read the values of the source registers. The rd

field is used as the write address if reg_write is enabled. To avoid waveform errors on Windows, as stated in the lab manuel, wires are used to assign the rs1, rs2, and rd fields separately before being passed to the register file.

The **Immediate Generator** interprets the instruction type (I-type, S-type, etc.).

The first 2-1 **multiplexer** in the datapath is placed before the ALU and chooses between rs2_data and the immediate output, based on the alu_src control signal input for the datapath.

With a similar design to Instruction Memory, the **Data Memory** module is used to load and store values. It receives the address from the ALU output, and either reads from memory (lw) or writes to memory (sw) depending on the control signals for MemRead and MemWrite. The address indexing uses bits [8:2] of the address value provided from the ALU.

The final 2-to-1 **multiplexer** decides whether the data to be written back to the destination register rd comes from the ALU result (e.g., arithmetic/logical ops) or from the data memory (e.g., lw). This selection is controlled by the mem2reg input signal.

These components are all connected in such order to design the processor datapath, shown through the diagram below, with the intended pathways of where the inputs and outputs of each submodule relay to, along with the inputs and outputs of the datapath in and of itself:

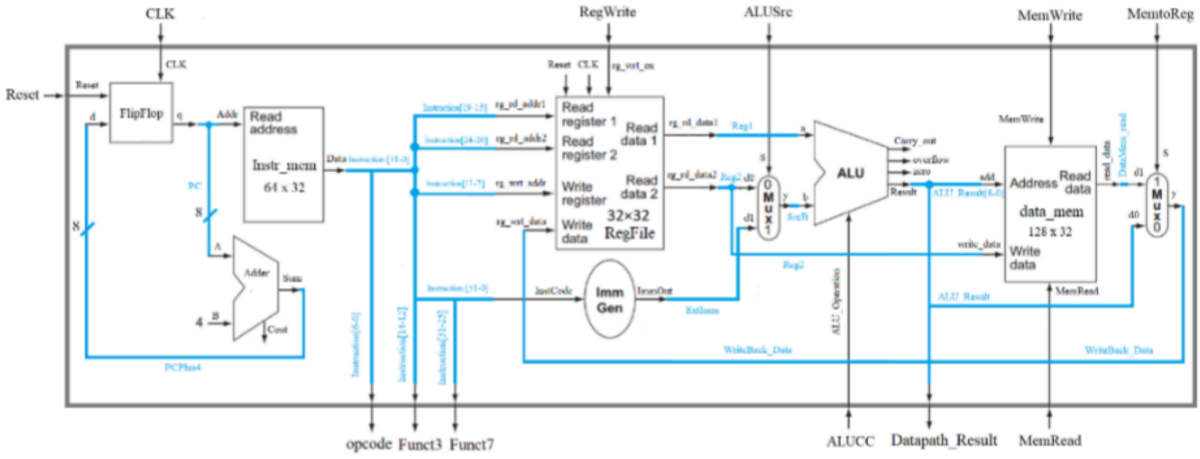
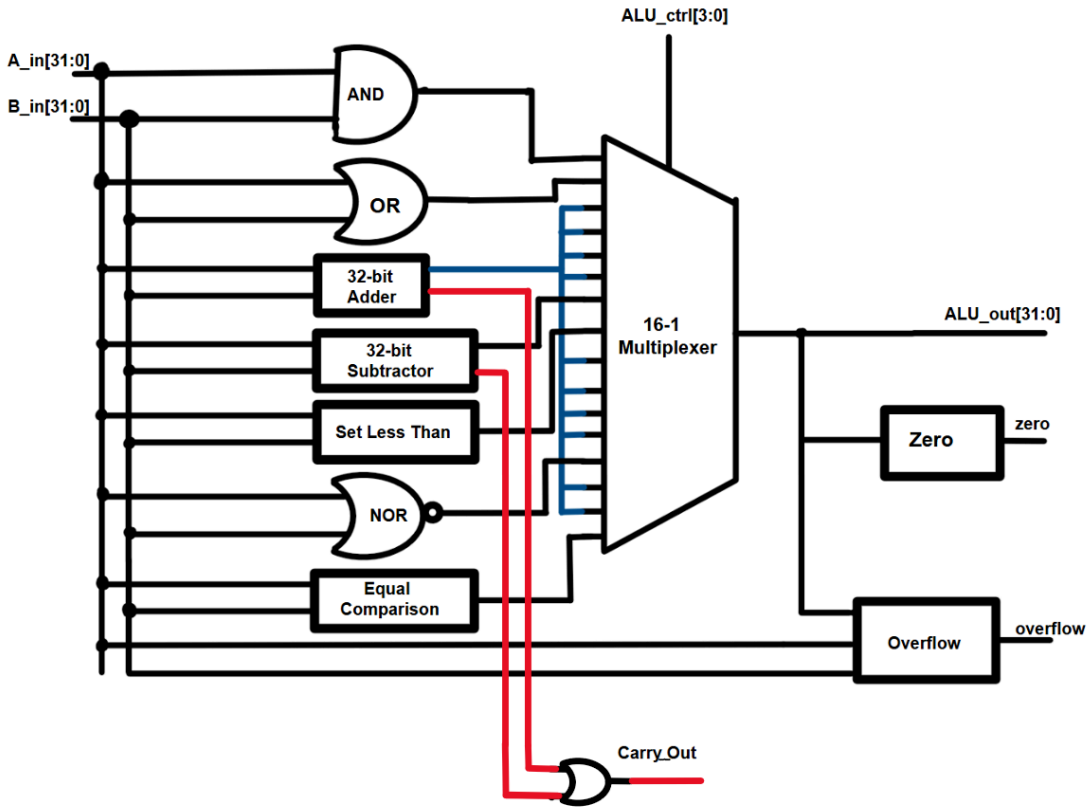


Figure 1: RISC-V Datapath

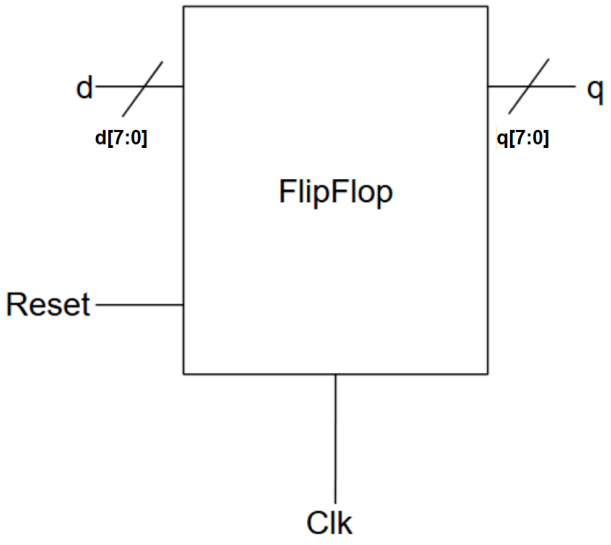
Here are the gate diagrams for most of the components inside of the processor datapath from prior labs (not accounting for differing variable names and sizes, variable name and size differences will be listed below each diagram):

ALU:



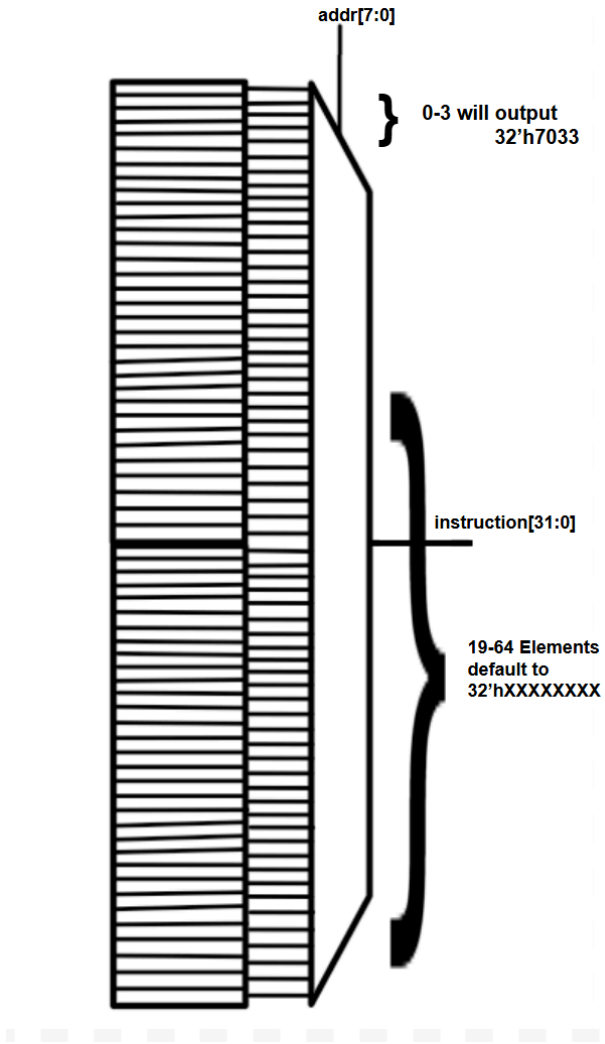
- ALU_ctrl should be named alu_cc

D-FlipFlop:

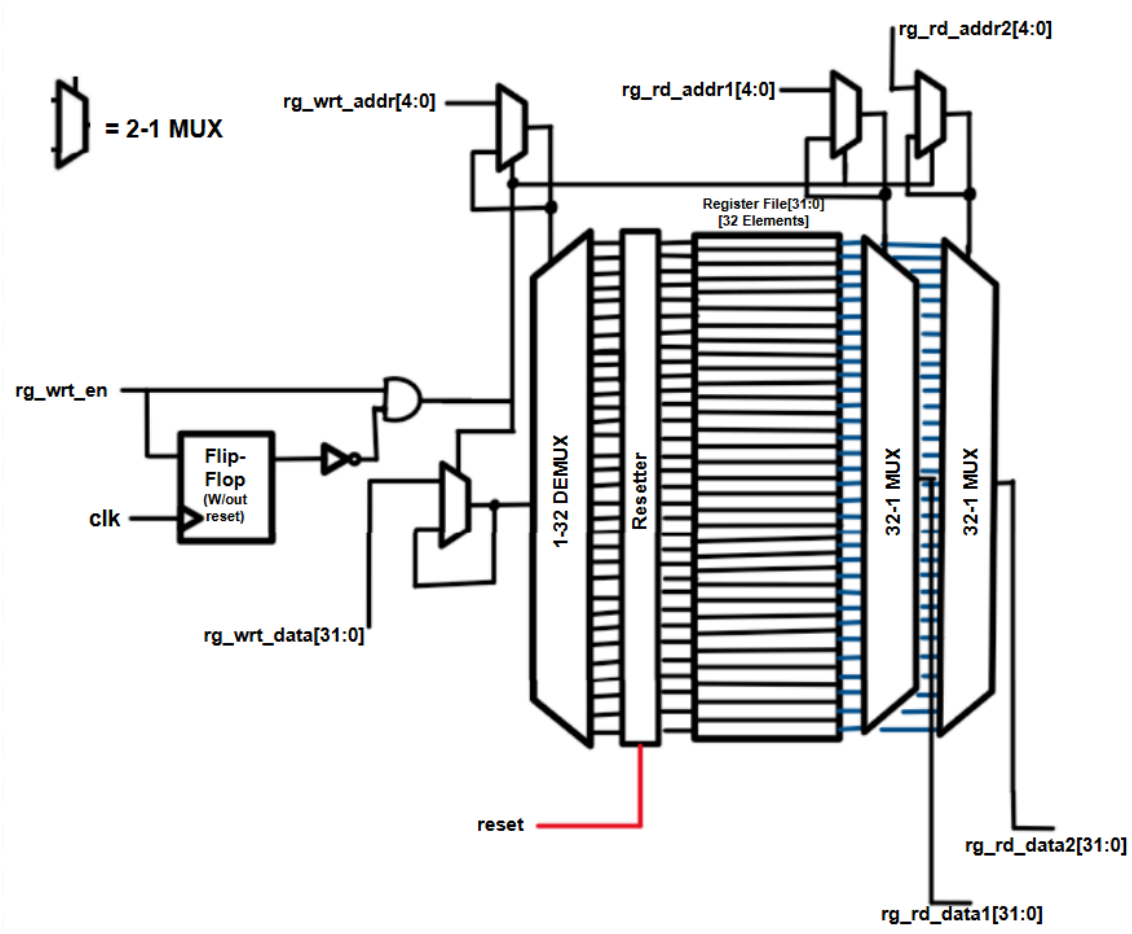


- Clk = clk and Reset = reset.

Instruction Memory (Similar logic for Data Memory):

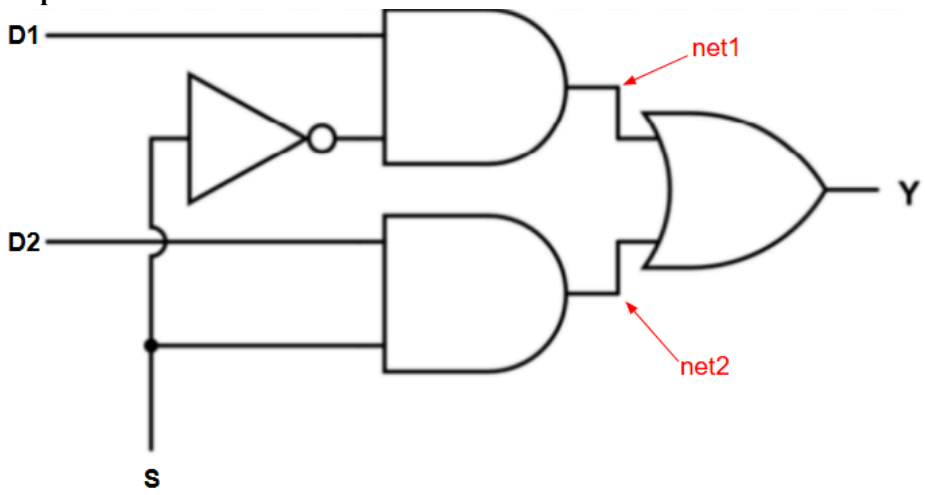


Register File:



- reg_wrt_en = reg_write

2-1 Multiplexer:



1. D1, D2, and Y are all 32 bits for the design of the processor Datapath multiplexers.

Controller:

The design of the controller was implemented to account for specific cases from the input value of Opcode. Depending on the 8-bit numbers value, specific test cases will be accounted for and provide specific outputs, which will relay towards the ALUController and the Datapath. Shown below are the testcases written in the code, including a default case, which just provides outputs of 0 for distinct input cases, that have outputs ALUSrc, MemtoReg, RegWrite, MemRead, and MemWrite transmit signals to the Datapath, whilst ALUOp transmits a signal to the ALUController:

```
7'b0110011: begin // add, or, and, sub, slt, nor
    ALUSrc   = 0;
    MemtoReg = 0;
    RegWrite = 1;
    MemRead  = 0;
    MemWrite = 0;
    ALUOp    = 2'b10;
end

7'b0010011: begin // addi, andi, ori, slti, nori
    ALUSrc   = 1;
    MemtoReg = 0;
    RegWrite = 1;
    MemRead  = 0;
    MemWrite = 0;
    ALUOp    = 2'b00;
end

7'b0000011: begin // lw
    ALUSrc   = 1;
    MemtoReg = 1;
    RegWrite = 1;
    MemRead  = 1;
    MemWrite = 0;
    ALUOp    = 2'b01;
end
```

```
7'b0100011: begin // sw
    ALUSrc   = 1;
    MemtoReg = 0;
    RegWrite = 0;
    MemRead  = 0;
    MemWrite = 1;
    ALUOp    = 2'b01;
end

default: begin
    ALUSrc   = 0;
    MemtoReg = 0;
    RegWrite = 0;
    MemRead  = 0;
    MemWrite = 0;
    ALUOp    = 2'b00;
end
```

ALUController:

The ALUController was designed to generate a 4-bit control signal (Operation) based on the values of ALUOp, Funct3, and Funct7 from the instruction. These inputs are used to determine the exact operation that the ALU should perform, such as addition, subtraction, AND, OR, SLT, and more. The 2-bit ALUOp signal, which is provided by the Controller, categorizes the instruction type (e.g., R-type, I-type, or memory-based), while Funct3 and Funct7 further specify the exact type of operation that is being outputted to the Datapath's ALU. The ALUController uses combinational logic to check for bit patterns and generate a corresponding 4-bit output, which are sent to the ALU within the Datapath to execute the correct operation. The code and chart below will show these specific bit patterns, and how they affect the output of the code:

operation	Operation code
AND, ANDI	0000
OR, ORI	0001
ADD, ADDI, SW, LW	0010
SUB	0110
SLT, SLTI	0111
NOR, NORI	1100

```
// Define the ALUController module's behavior
assign Operation[0] = ((Funct3 == 3'b110) || ((Funct3 == 3'b010) && (ALUOp[0] == 1'b0))) ? 1'b1 : 1'b0;
// Bit 1 assignment ^
assign Operation[1] = ((Funct3 == 3'b010) || (Funct3 == 3'b000)) ? 1'b1 : 1'b0;
// Bit 2 assignment ^
assign Operation[2] = ((Funct3 == 3'b100) ||((Funct3 == 3'b010) && (ALUOp[0] == 1'b0)) || ((Funct7[5] == 1'b1) && (Funct3 == 3'b000) && (ALUOp[1] == 1'b1))) ? 1'b1 : 1'b0;
// Bit 3 assignment ^
assign Operation[3] = (Funct3 == 3'b100) ? 1'b1 : 1'b0;
```

These Components ultimately all provide inputs and outputs that satisfy the other components. Through the instantiation of these components with verilog, and the connection of the components inputs to outputs to other inputs via wire variables, the gate schematic design of the Single-Cycle RISC-V Processor shown below is formed:

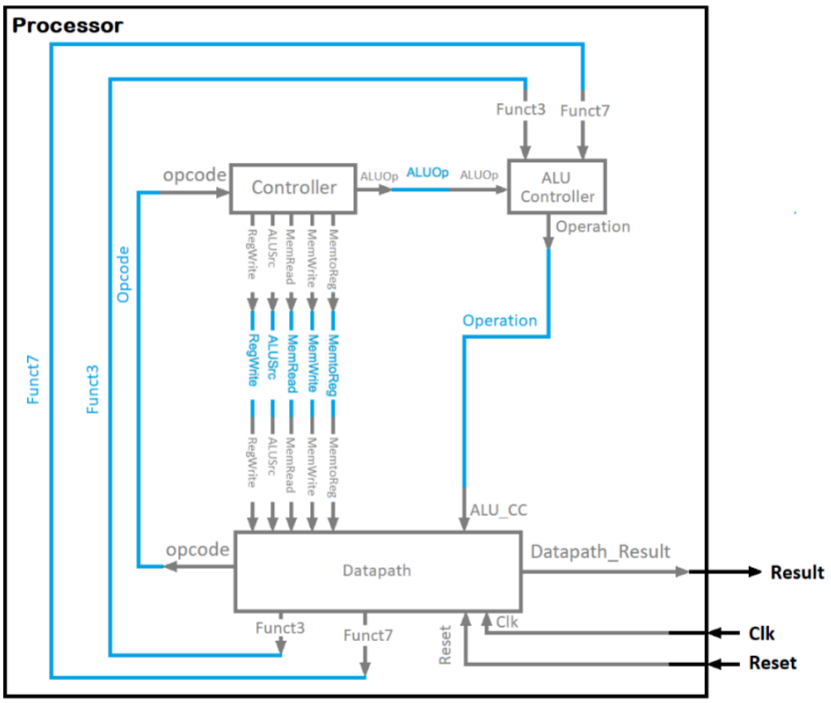
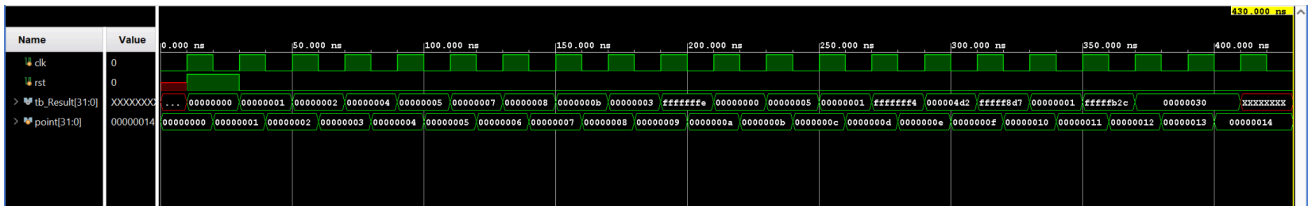


Figure 1: Submodules in a RISC-V processor

Simulation Results

Through the provided testbench, the following waveform diagram was generated:



Twenty operations were formed from the testbench to test 20 different scenarios for the processor. These testcases happened every 20 nanoseconds starting from the 30 nanosecond where the first input change occurs, where the reset signal is held and released, changing the value of tb_Result. After that, countless RISC-V operations were tested, which included addi, sub, and, or, slt, nor, andi, ori, nori, sw, and lw, in order to account for all possible testcases.