Lab 5 Report

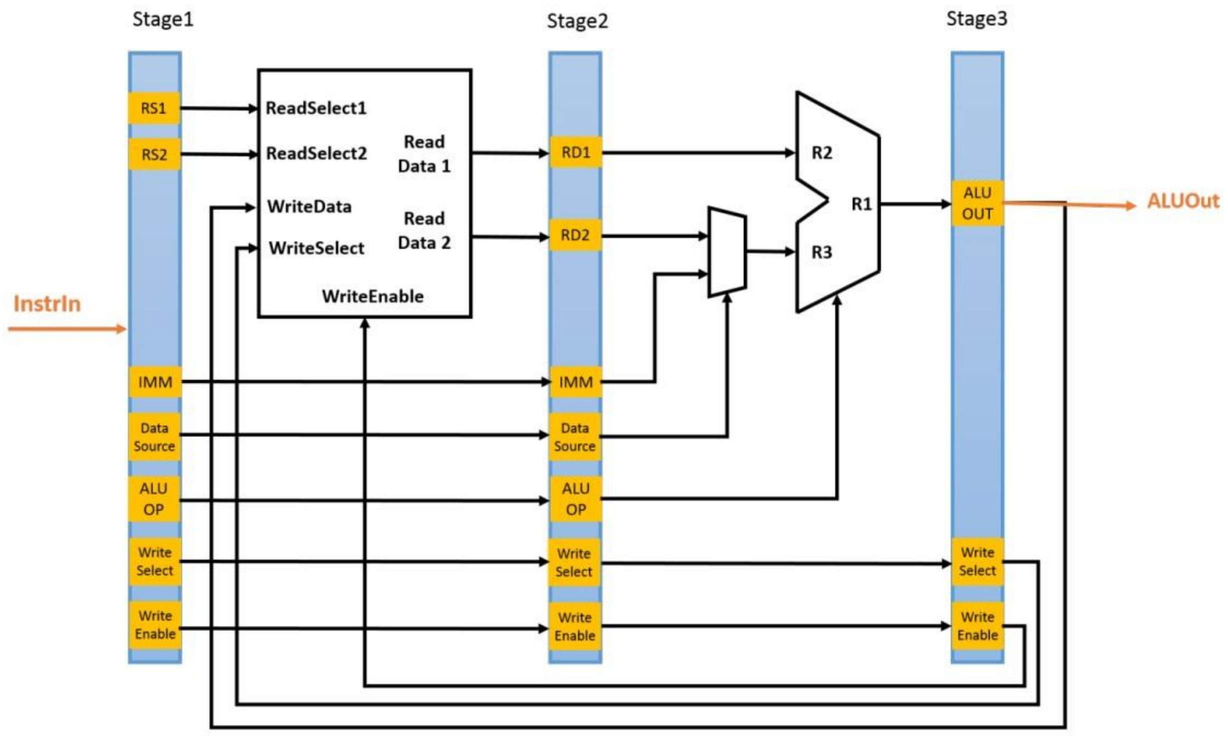
Leanorine Lorenzana-Garcia

EC 413, Professor Herbordt

Introduction:

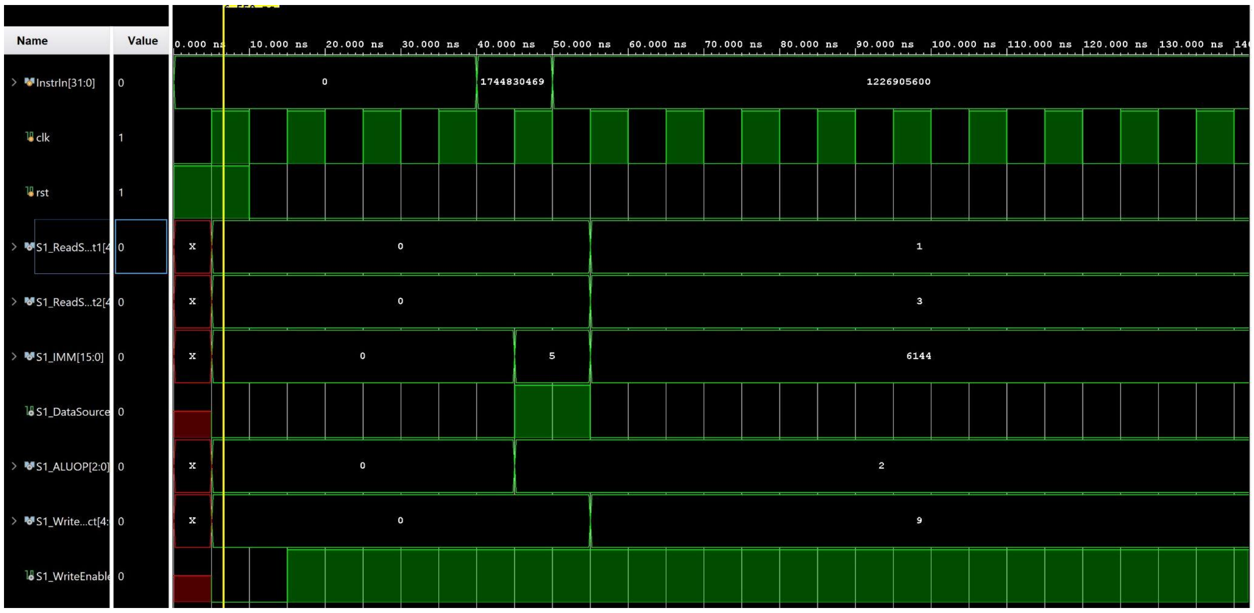
The purpose of Lab 5 is to build, debug, and test a 4-stage 32-bit pipelined datapath in Verilog. We were provided files for the n-bit register, an n-bit demux, an n-bit register file, a D Flip-Flop. Using these files and our files from Lab 5, where we built an ALU, we were tasked with creating Verilog modules for each stage of the datapath and a top module that instantiates all of these components to create the total pipeline. With each module, we were to also create testbenches and test cases to ensure that each stage and component worked properly. Lastly, in order to ensure our datapath was working, we were to create paper test cases and test by passing 32-bit instruction sequences into our datapath and ensuring that the register values for each individual stage matched the register values in our CPU.

Design (`myPipeline.v`):



We were given the design diagram below in which we were supposed to use to create our pipeline. For each stage of the pipeline we created our own modules that we eventually used in our top module. Our top module, `myPipeline.v` instantiates `S1_Reg.v`, `nbit_register_file.v`, `S2_Reg.v`, `myALU.v`, `and S3_Reg.v`.

<u>Stage 1 (`S1_reg.v`):</u>



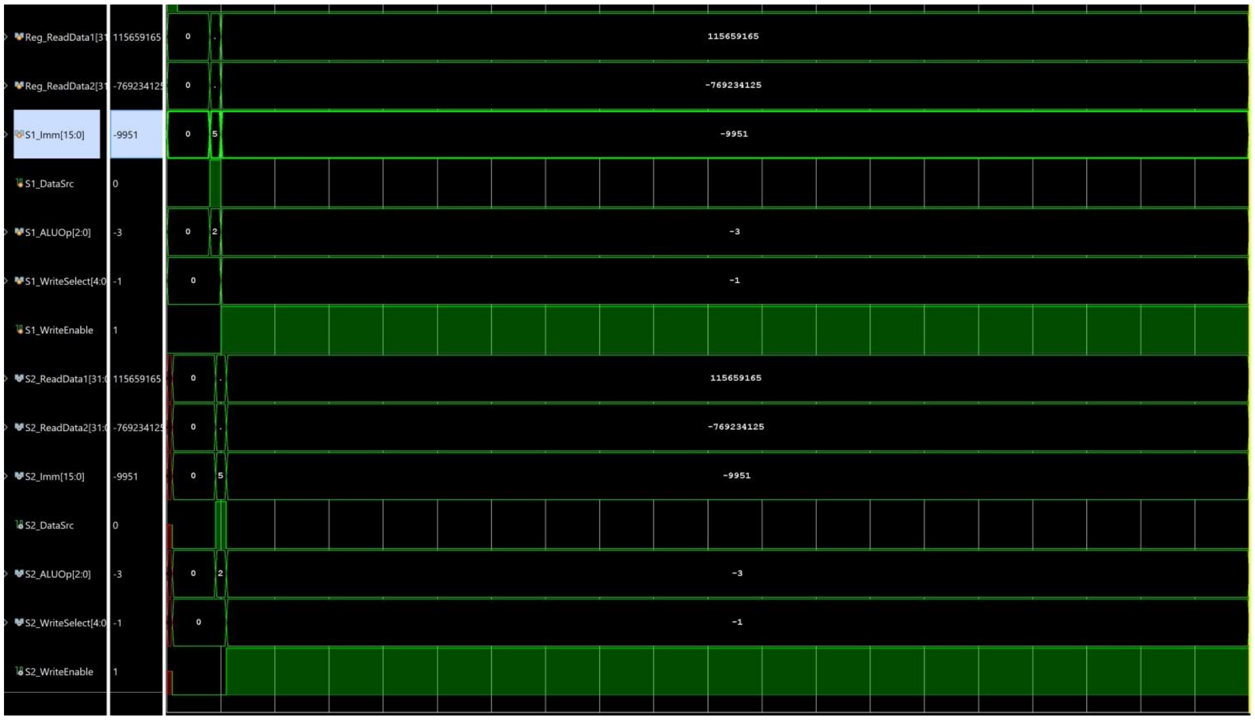
For Stage 1 we created a `S1_reg.v` module that takes in clk, rst, and a 32-bit input instruction, `InstrIn`, as its inputs. We have the following output registers, which are set on the positive edge of the clock, two 5-bit read select registers (`S1_ReadSelect1` and `S1_ReadSelect2`), a 16-bit immediate register (S1_IMM), a 1-bit data source register (`S1_DataSource`), a 3-bit ALU Operation code register (`S1_ALUOP`), a 5-bit write select register (`S1_WriteSelect`), and a 1-bit write enable register (`S1_WriteEnable`). The `S1_ReadSelect1` is defined as bits 20 to 16 of the `InstrIn` and `S1_ReadSelect2` is defined as bits 15 to 11 of the input `InstrIn`. Our immediate value, `S1_IMM`, in the case of an

I-type instruction is equivalent to bits 15 to 0 of the instruction input. The `S1_DataSource` tells us where the instruction type is an R-type (0) or an I-type(1) and is found in bit 29 of the instruction Input. The ALU Operation code `S1_ALUOP` is found in bits 28 to 26 of the instruction input and tells the ALU which operation to perform on the data contained in the two read select registers. `S1_WriteSelect` is defined by bits 25 to 21 of the instruction input and specifies what register the output of the ALU operation goes to. The `S1_WriteEnable` is set to 1 and this allows the contents of the write select register to be updated.

N-Bit Register File (`nbit_register_file.v`):

This module is a register file that takes the a 32-bit write data (`write_data`) that contains the value of the output from the ALU operation. It has two 5-bit read select outputs from the `S1_Reg` module (`read_sel_1` and `read_sel_2`). In addition, a 5-bit write select (`write_address`) which contains the value from stage 3 `S3_Reg_WriteSelect`. An input of a 1-bit write enable (`RegWrite`) from the `S3_Reg` module, `S3_Reg_WriteEnable`. The outputs are two 5-bit read data outputs from the `S1_Reg` module (`read_data_1` and `read_datal_2`). The `write_data` input contains the 32 bit value that is going to be written to the `write_address` register if the `RegWrite` input is 1. When we instantiate the register file, all 32-bit registers are initialized to 10 times the register number. For example, `registerfile[2] = 2* 10`.

Stage 2 (S2_Reg.v):

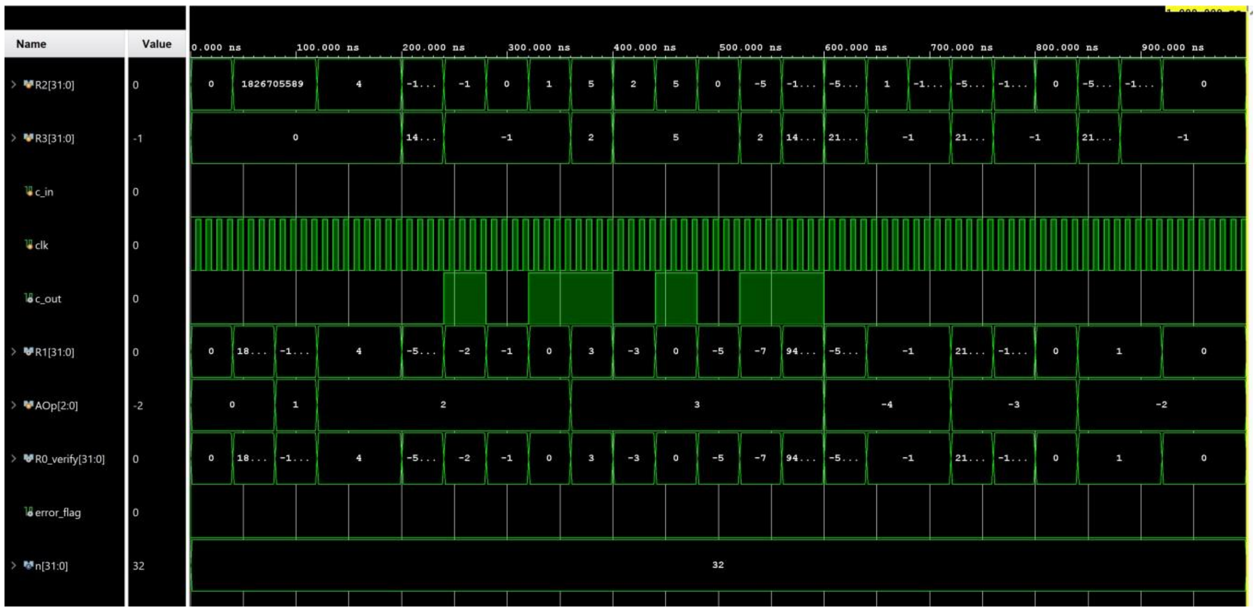


For Stage 2 we created a `S2_reg.v` module that takes in `clk`, `rst`, two 32-bit read data registers (Reg_ReadData1 and Reg_ReadData2), the immediate value from stage 1 (S1_Imm), the data source from stage 1 (S1_DataSrc), a 3-bit ALU operation from stage 1 (S1_ALUOp), a 1-bit write enable from stage 1 (S1_WriteEnable). We have the following output registers, which are set on the positive edge of the clock, two 5-bit read data registers (`S1_ReadData1` and `S1_ReadData2`), a 16-bit immediate register (S2_Imm), a 1-bit data source register (`S2_DataSrc`), a 3-bit ALU Operation code register (`S2_ALUOp`), a 5-bit write select register (`S2_WriteSelect`), and a 1-bit write enable register (`S2_WriteEnable`). The `S2_ReadData1` is defined as the input `Reg_ReadData1` and likewise for `S2_ReadData2` with `Reg_ReadData2`. Our immediate value, `S2_Imm` is set to `S1_Imm`. The data source for stage 2, `S2_DataSrc,` is set to S1_DataSrc. Our ALU operation from stage 1 is stored in S2_ALUOp. Our write select from stage 1 is passed into

`S2_WriteSelect`. Lastly, the write enable for stage 2, `S2_WriteEnable`, is set to `S1_WriteEnable`.

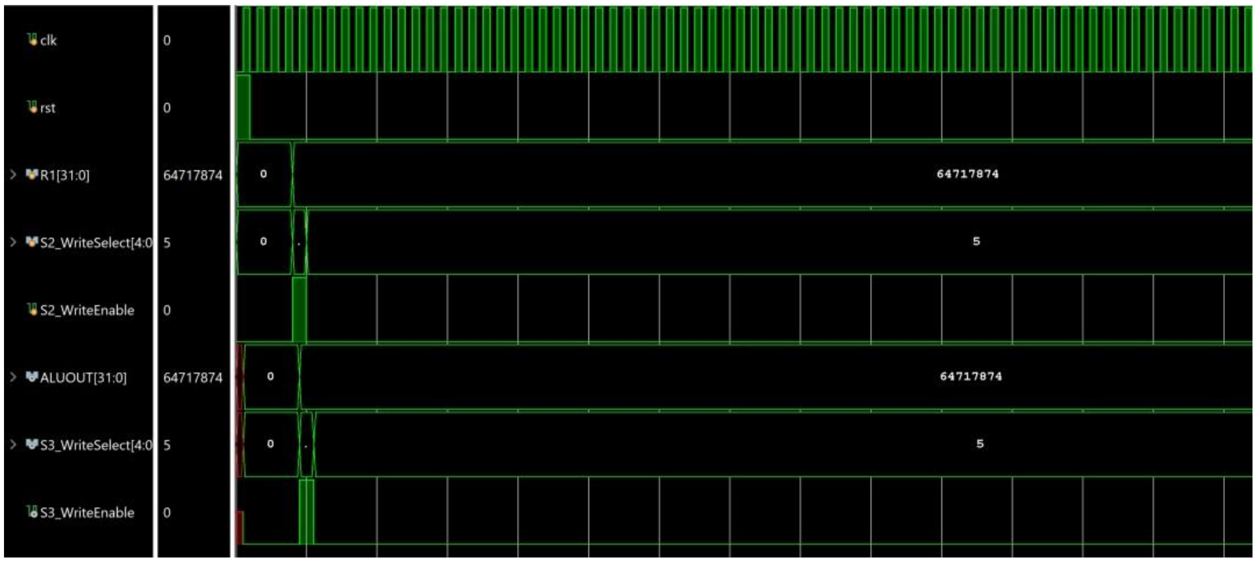
<u>MUX and ALU (`myALU.v`):</u>

The multiplexer takes the 32-bit value of Read Data 2 (`S2_Mux_ReadData2`), a 16-bit immediate (`S2_Mux_Imm`), and the 1-bit data source output from the `S2_Reg` module (`S2_Mux_DataSrc`) as its inputs and stores the selected value into a 32-bit register `R3`. Using behavioral verilog, the mux will output `S2_Mux_ReadData2` if `S2_Mux_DataSrc` is 0 meaning that the instruction is an R-type. If `S2_Mux_DataSrc` is 1 it will output the immediate value `S2_Mux_Imm` because the instruction is an I-type.



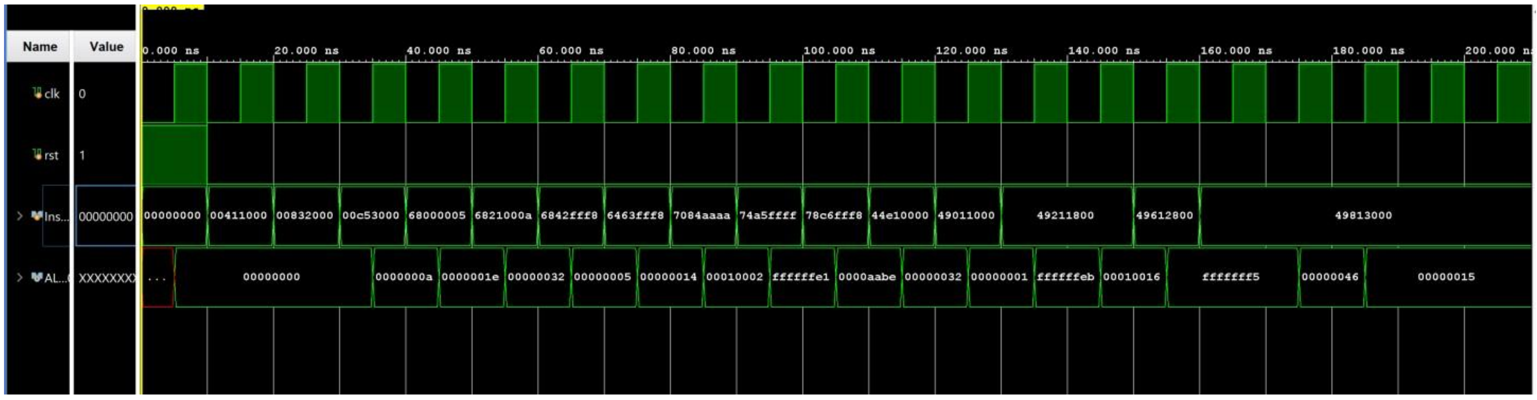
The ALU component comes from lab 5 takes the following inputs: 32-bit register `R2` that contains `ReadData1` from stage 2, a 32-bit register `R3` that contains the output of the multiplexer, and the 3-bit ALU Operation code from stage 2. The ALU then uses a mux to select the specified operation (MOV, NOT, ADD, SUB, OR, AND, or SLT) based on the imputed value from the 3-bit ALU Operation code. The ALU will then run the specified function and output the 32-bit result into `R1` which will eventually be output in Stage 3 of the pipeline.

Stage 3 (`S3_Reg.v`):



Stage 3 of the pipeline is written in the `S3_Reg.v` module. It takes in the following inputs: `clk`, `rst`, the 32 bit result of the ALU operation (`R1`), a 5-bit write select from stage 2 (`S2_WriteSelect`), a 1 bit write enable from stage 2 (`S2_WriteEnable`). It has a 32-bit output register `ALUOUT` which is set to the value of R1, the result of the ALU Operation. A 5-bit write select, `S3_WriteSelect`, which is set to the `S2_WriteSelect`. Lastly `S3_WriteEnable` is set to `S2_WriteEnable`.

<u>Test Bench of Pipeline (`Lab5_Pipeline_Test.v`):</u>


Name
Value
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
200.000 ns
clk 0
rst 1
Ins... 00000000
00000000 00411000 00832000 00c53000 68000005 6821000a 6842fff8 6463fff8 7084aaaa 74a5ffff 78c6fff8 44e10000 49011000 49211800 49612800 49813000
AL... XXXXXXXX
... 00000000 0000000a 0000001e 00000032 00000005 00000014 00010002 ffffffe1 0000aabe 00000032 00000001 ffffffeb 00010016 fffffff5 00000046 00000015

Our `Lab5_Pipeline_Test.v` module is the testbench we used to verify that our pipeline worked properly. In this module, we instantiated our top-level module myPipeline so we could test.. After performing a global reset we began to pass a series of 32-bit instruction sequences provided to us in Pipeline_test.v to the datapath, with 10 time units between inputting each instruction.

<u>Design Hierarchy:</u>

```
Lab5_Pipeline_Test.v
    myPipeline.v
        S1_Reg.v
        nbit_register_file.v
        S2_Reg.v
        myALU.v
        S3_Reg.v
```