Module Descriptions:

S1_Register:
Stage 1 register responsible for taking an instruction as input and parsing it for ReadSelect1, ReadSelect2, Immediate, Data Source, ALU OP, and WriteSelect values. This module was repurposed from the pre-lab.

nbit_register_file:
This parameterized register file was unmodified aside from adding a reset switch, and is responsible for indexing the necessary registers for each instruction, and writing to them when necessary as well.

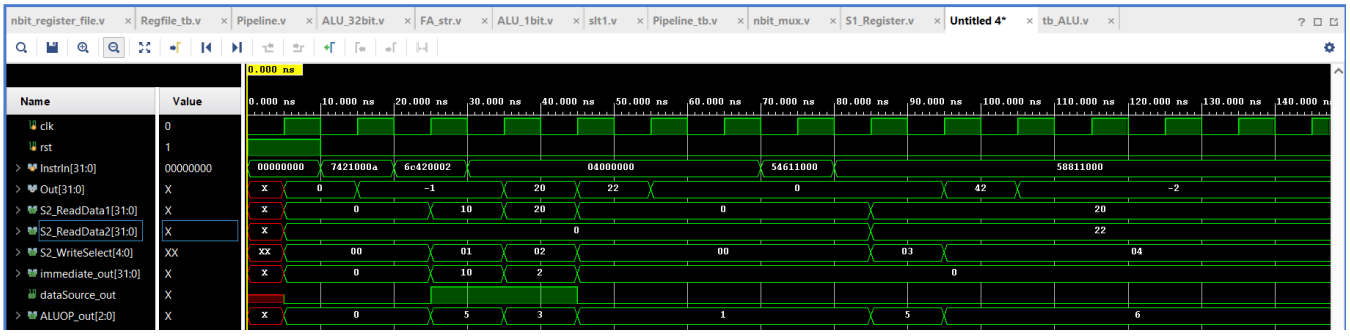
S2_Register:
Stage 2 register responsible for passing ReadData1 and ReadData2 from the register file. ReadData1 gets passed to the ALU while ReadData2 gets passed to the MUX. It also takes the immediate, data source, ALU OP, Write Select, and Write Enable from the Stage 1 register. It passes the immediate and data source on to the MUX, the ALU OP on to the ALU, and the Write Select and Write Enable on to the Stage 3 register.

S3_Register:
Stage 3 register responsible for passing Write Select, ALU OUT, and Write Enable on to the register file (for updating the contents of registers).

nbit_mux:
Parameterized mux that uses conditionals to compare Data Source and determine if an instruction is R-Type or I-Type, and subsequently whether it should pass the immediate from the Stage 2 register on to the ALU, or the ReadData2 value from the Stage 2 register to the ALU.

ALU_32bit:
This is actually a parameterized ALU responsible for actually performing the logical/arithmetic instructions, and outputs the result to the Stage 3 register for further processing.

Waveforms:
Pipeline Testbench:



I tested the pipeline/datapath by feeding it the provided example instructions and tracing through the ReadData1, ReadData2, WriteSelect, immediate, dataSource, and ALU OP wires and checking them against the final result “Out”. With each register being initialized to 10 * (Register Number), I saw the correct output for each instruction.

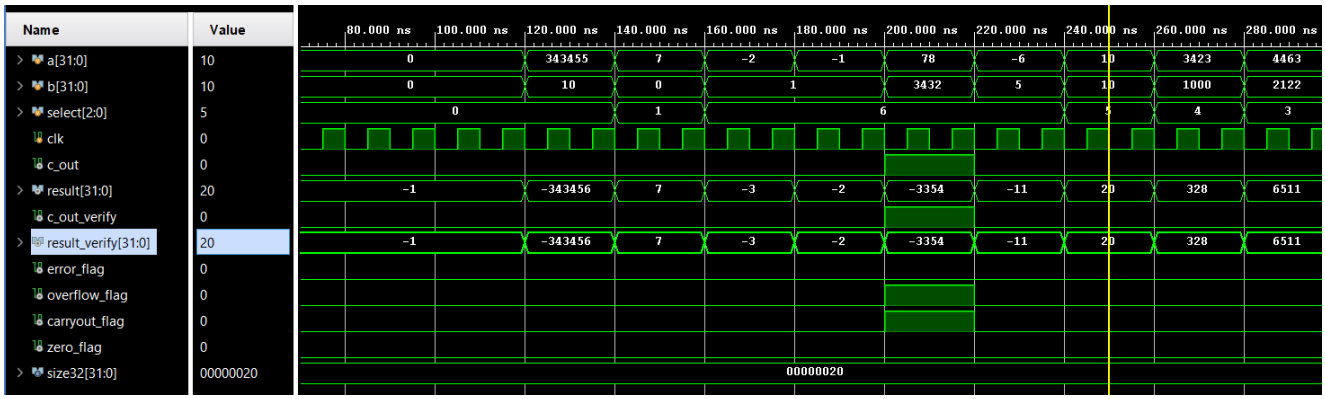
The first instruction is input at 10ns, and the output corresponding to this ADD operation propagates at 35ns.

The second instruction is input at 20ns, and the output for this OR operation propagates at 45ns.

After a null operation delay to allow the register file to update the contents of registers, the next instruction is input at 70ns, and the output for this ADD operation propagates at 95ns.

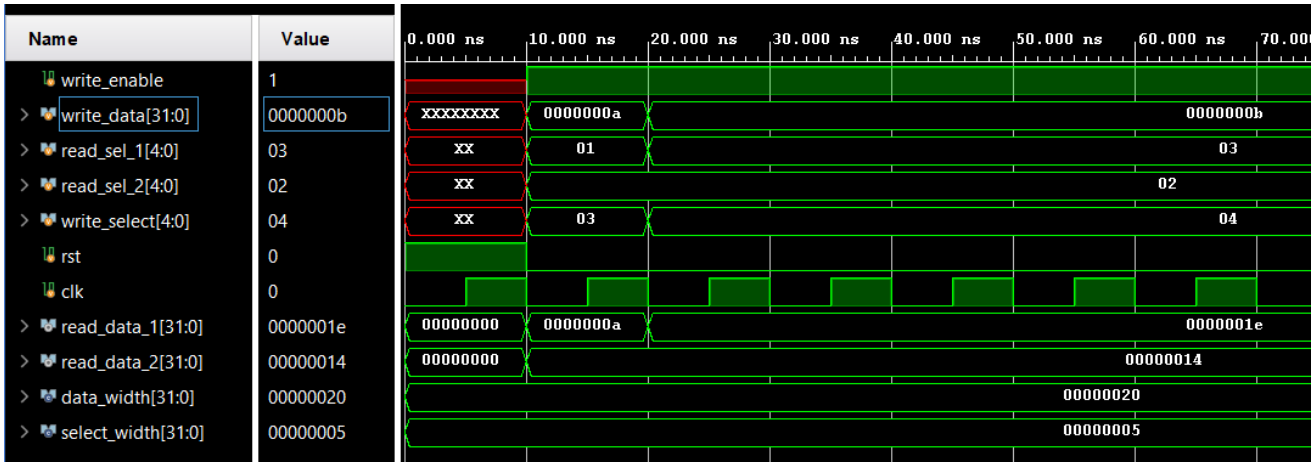
The next and final test instruction is input at 80ns, and the output for this SUB operation propagates at 105ns.

ALU Testbench:



I tested the ALU using a separate testbench and verification module to test several of the operations from the given operation table. As you can see, there are no error flags.

Register File Testbench:



For this testbench, I feed two example write_data inputs to the register file, and see it update the contents of the specified registers accordingly. This way I knew the register file was successfully able to take an input and update the appropriate registers.