EECE 3324: Computer Architecture and Organization

Project Part 1

Alexander Wise

002907473

June 7, 2024

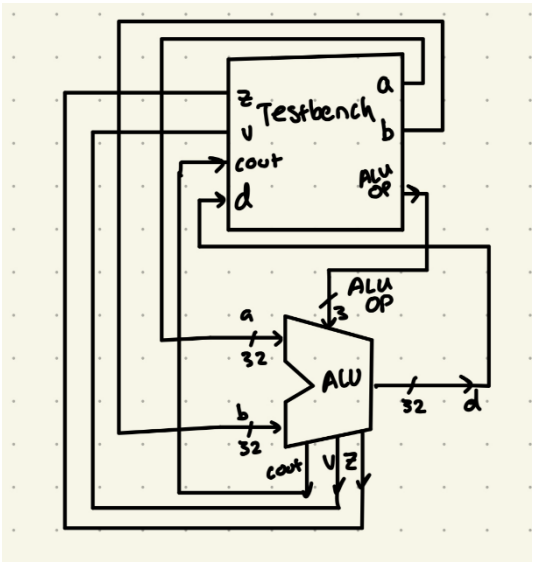
Summary and Approach:

The purpose of this project was to build a basic pipelined processor to complete simple operations. This was done in stages starting with the implementation of the ALU with three operations: addition, and operations, and left shifting. After this, the Pipelined ALU was designed next with the introduction of registers for the 2 inputs, op code, and the output from the ALU. Finally, everything was put together with a central register file to control the inputs and store the outputs from the ALU.

My approach to this project was to consider the larger idea behind the entire project and focus on the individual components and think about how they would be implemented into a system. In doing so, I was able to go assignment by assignment with ease and ensure that each component worked with each step.

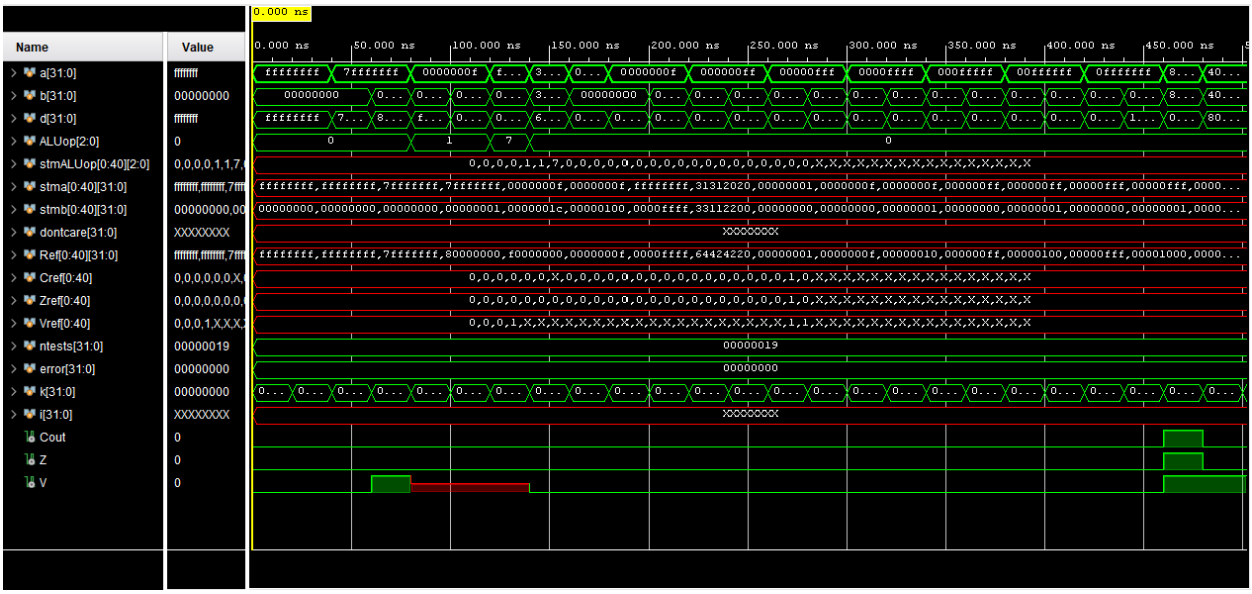
Assignment 1

The purpose of this assignment is to create the central ALU that will be used as the main component of the processor designed in this project. With the three operations, I designed the ALU to do each operation based on the opcode with a switch statement allowing for easy control over each operation. The block diagram for this assignment is as follows:



As seen above, the testbench directly interfaces with the ALU, providing the ALU with test values. When run, the following console output and waveform was produced:

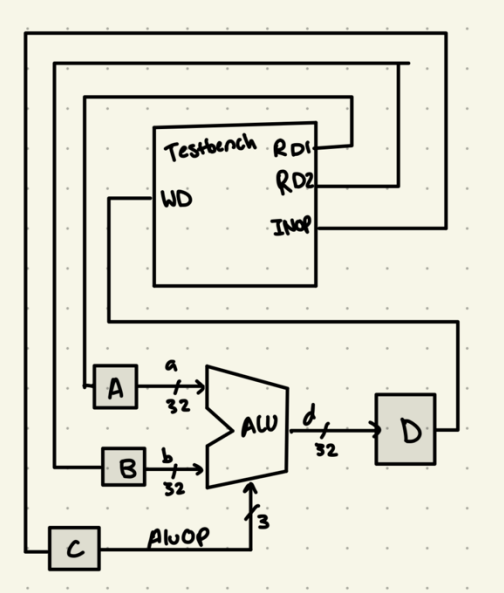
```
 Zref=1
 Cout=1
 Cref=1

-----  TEST FOR A + B  -----
Test#          24
 Time=     500.0ns
 ALUop=000
 a=      01000000000000000000000000000000
 b=      01000000000000000000000000000000
 d=      10000000000000000000000000000000
 Ref=    10000000000000000000000000000000
 V=1
 Vref=1
 Z=0
 Zref=0
 Cout=0
 Cref=0

---------YOU DID IT!! SIMULATION SUCCESFULLY FINISHED----------
 Number Of Errors =           0
 Total Test numbers =          25
 Total number of correct operations =          25
```



As seen above, the ALU module passed all tests and worked as expected. The test bench for this module worked by creating an array of tests. The testbench then went through each index of the array and compared the output values to the reference values. If the values were correct, then a point was awarded. This was the same process for each of the test benches within this project.

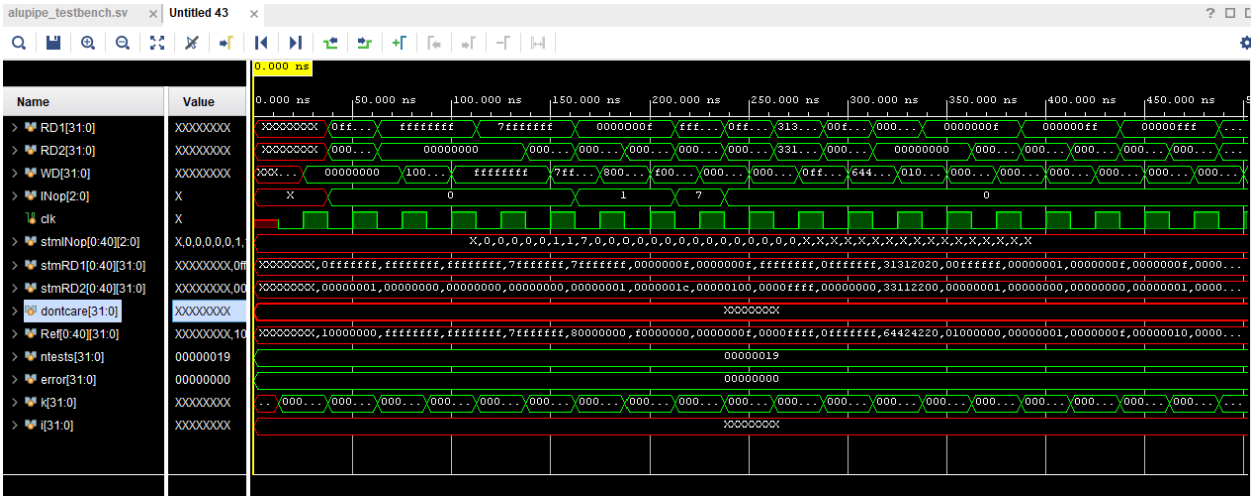
Assignment 2

Assignment 2 involved adding registers to the ALU to turn it into a pipelined ALU. This allowed for the initial implementation of pipelining to the pipelined processor. To do so, a series of registers were created to take in the next input and store it until the next clock cycle. These registers were placed before all ALU inputs and the ALU output to store the respective values. The implementation of this looks like the following with the testbench wiring:



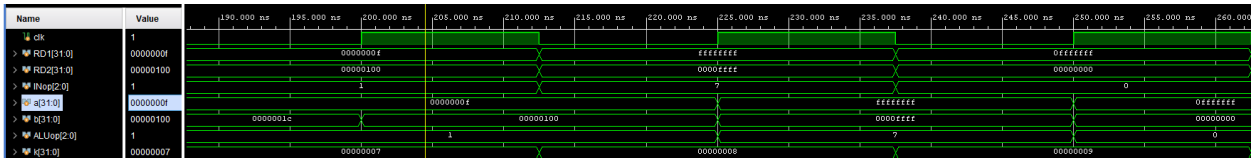
As seen above, just like in Assignment 1, the test bench interfaces directly with the ALU and the inputs go directly into the registers prior to entering the ALU.

The results of the test are as follows:



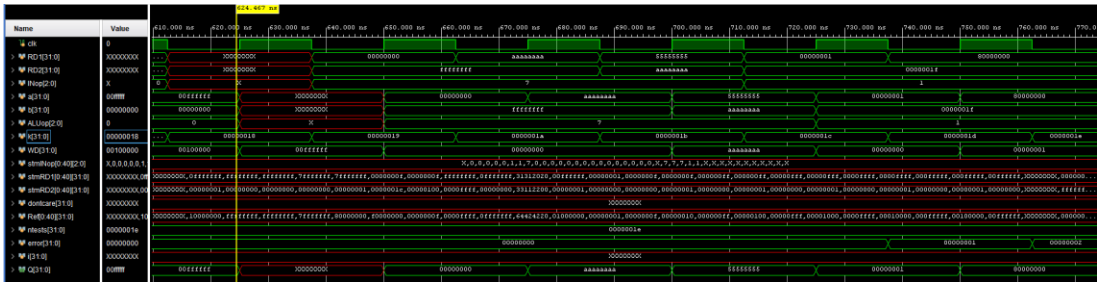
```
Current operation #          23
 S=000
 a  =   00000000111111111111111111111111
 b  =   00000000000000000000000000000000
 d  =   00000000111111111111111111111111
 ref=   00000000111111111111111111111111

>>>>>>>>>> YOU DID IT!! SIMULATION SUCCESFULLY FINISHED<<<<<<<<<<<
```

As seen in the screenshots, the pipelined ALU successfully worked and passed all tests.



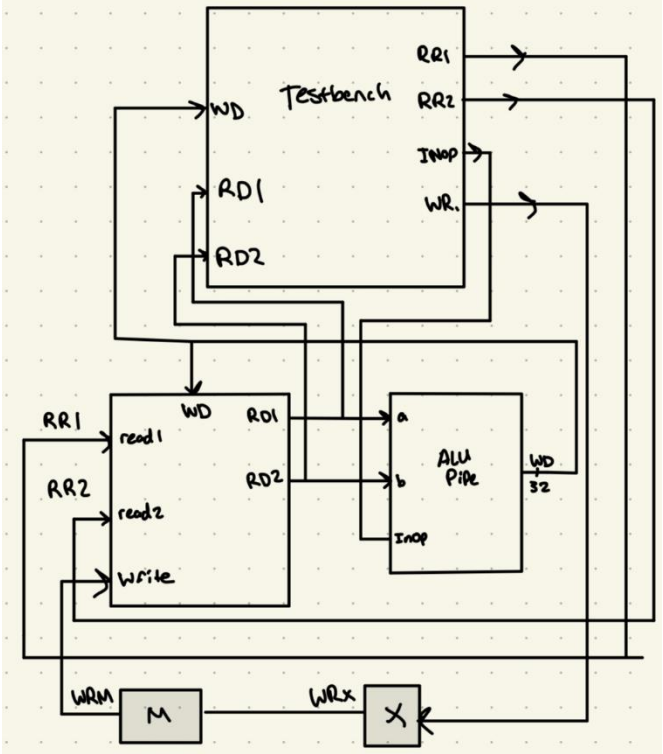
Here, for tests 7, 8, and 9, We can see that inputs RD1, RD2, and INop all complete at the same time while inputs a and b finish half a clock cycle later. This is because a and b are values from the registers themselves and the next values is triggered on the posedge of the clock. This results in them being half a clock cycle later than the values above a and b in the waveform. Similarly, that gives the processor time to modify the values if they are changed during the various operations that those values may undergo. These changes won't be reflected yet because we don't have a register file implemented.

Next, 5 more test cases were generated and added to the test bench. The output of these cases are as follows:



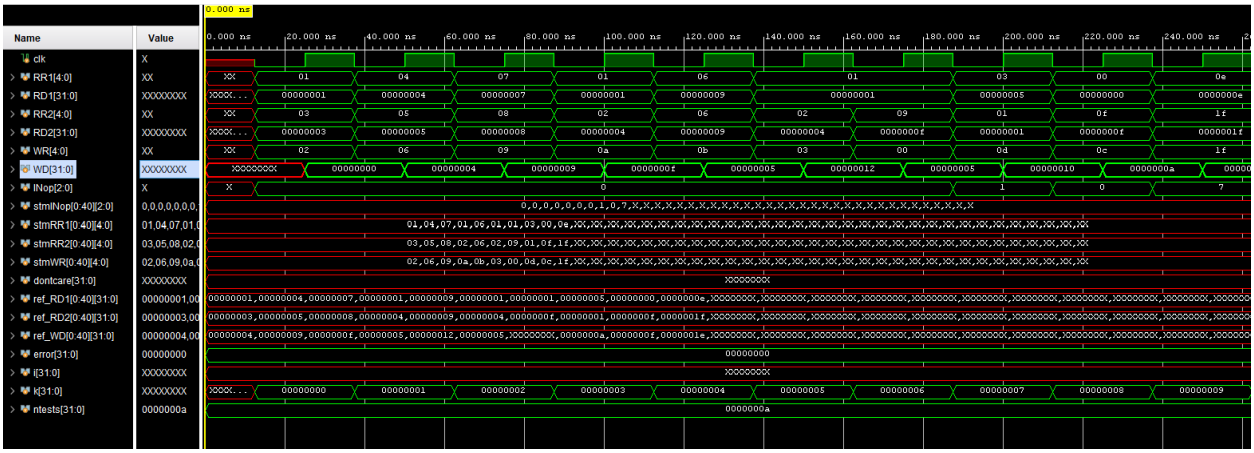
Assignment 3

Assignment 3 involved the implementation of a register file and top-level module to control the pipelined ALU with register file. This step involved the creation of two new files: the top-level module, and the register file. The register file consisted of an array of 32 32-bit registers to store data. This data is the values from the register file operations. Next, the top-level module was developed. The top-level module instantiated the register file and implemented two registers for the write input. These registers essentially applied a delay so that the write input would lineup with the data that we wanted to store from the ALU. The block diagram for the implementation of this as well as the testbench is as follows:



As seen from the block diagram above, the testbench controls the register file values and ALUOp. Everything here has built off of the previous assignments within this project.

The module was then run with the test bench and the following waveform and console output was produced:




```
RD2=00000000000000000000000000011111
ref_RD2=00000000000000000000000000011111

>>>>>>>>>> YOU DID IT!! SIMULATION SUCCESFULLY FINISHED<<<<<<<<<<<<
```

As seen above, the module worked as expected and passed all testbench cases.

If we were to replace the blocking assignments with non-blocking assignments, then we get 2 errors. These two errors are both from case 7 and return the following values:

```
For the next operation #           7
 RR1=00011
 RR2=00001
 WR=01101
 RD1=00000000000000000000000000000011
 ref_RD1=00000000000000000000000000000101
 RD2=00000000000000000000000000000001
 ref_RD2=00000000000000000000000000000001

++++++++++ ERROR. A Mismatch Has Occured ++++++++++
Time=     212.5ns

-----  TEST FOR A XNOR B  -----
Current operation #           7
 For the current operation ALUop=001
 WD=00000000000000000000000000000110
 ref_WD=00000000000000000000000000001010
```

The reason for this error is that because we are using non-blocking assignments which means that everything is completed sequentially. Register 3 is assigned a new value in the previous step but this happens in parallel with the system calling for the value of register 3. This means that we get the previous value of register 3 instead of the updated one which is why there is a mismatch in the reading step and ALU operation step.

Conclusion

All components of this pipelined ALU with register file built off of one another to ensure a proper understanding of the pipelined processors construction process as well as the idea behind the importance of each individual module within the processor.

Some improvements I would make is to make the testbench operations more coherent. It can be difficult to trace values across the test bench and the waveform graph to find error and fix them.