- Ben Gothard CSSE232 Project Journal

-- Entry #1: 10.1.2018

After talking with my group about which processor design would be best to implement, we unanimously decided to go with a hybrid design that incorporated Matthews idea of explicitly stating in the assembly code which processor to store information into and my idea of implicitly deciding a "default" accumulator register in which to store the value. We then discussed whether we wanted different opcodes for commands that did operations on the 2 accumulators versus using a "funct" flag. I felt that using a funct code was a better idea since it prevented duplication of similar operations and kept the instruction set from getting large. I wanted to use the funct code since we also came up with the idea of having the funct code be the first bit in the machine code so we could easily tell whether it was doing an accumulator operation or an immediate operation. I then worked on a rough assembler to convert our instruction set assembly line-by-line (entered manually by us) to convert the instruction to machine code.

-- Entry #3: 10.3.2018

We met to discuss who is doing what for the next milestone. We broke up all of our instructions into 4 even 6 instruction sections so each of us could do some of the RTL. By some luck, each of us seemed to get a different "group" of instructions, like how I got all of the jump instructions. Since almost all jumps behave like jimm when the flag bit is set, they were all pretty similar in layout.

-- Entry #3: 10.5.2018

We met for about 45 minutes and talked over our RTL and started working on grouping our RTL steps into parallel sections and figuring out which instructions have similar blocks to combine.

-- Entry #4: 10.6.2018

We planned out a state diagram for each "group" of instructions with each person who wrote the RTL doing most of the planning with the rest giving input and insight. I had the jump RTL and diagram. I started working on making Euclid's algorithm into a table and started working on an assembler/compiler for our instruction set to make the process go faster.

-- Entry #5: 10.9.2018

We met the day of Milestone 2's due date to make sure everything was together. While converting our Euclid's algorithm into machine code, I noticed a few bugs in our assembly code, but since I had written a compiler, I checked with the group on bugfixes to the assembly and then re-ran the program to output the correct machine code. I added and made our assembly fragments a little more organized as well.

-- Entry #5: 10.15.2018

We started working on our Xilinx components for our registers, ALU, memory, zero/sign extenders and shifters. We divided up the components between each of our members and I ended up with the register component. For the rest of the period I worked on learning the basics of Xilinx in order to properly create our register.

-- Entry #6: 10.16.2018

During class we worked together on our components and started talking through our data path. I finished the register and started working on the tests for it. We also divided up the rest of the tasks for the milestone such as converting our RTL to the data path and double checking our RTL to make sure it works correctly.

-- Entry #7: 10.17.2018

We finished up all of our components and tests and did a final revision on our RTL and data path. We found an old RTL where we were using a B register and corrected it to match our new data path. We also figured out our remaining control signals and added them. We worked together figuring out the rest of our troubles with Verilog and I went through all of our committed components and ran each test to make sure they worked on other machines. We discovered a feedback loop with our swap command and our registers, so we worked as a group to figure out how to solve that. As of this writing, we are storing one value of the registers in ALUOut before swapping it to the other register.

-- Entry #8: 10.18.2018 - 10.24.2018

This entry is a group of dates since multiple meetings were just continuations of what we were working on previously. I worked on making sure our RTL was correct and uniform across our document, I then worked on making our memory block, but due to our datapath being out-of-date for a bit and working with/helping others with their parts, I only started the base-work for the component. I met with Joy a few times to work on the PC block component. I also helped write up some of the sections of the design document

-- Entry #9: 10.26.2018 - 10.31.2018

We started working on integrating all of our separate components together into larger blocks before merging those blocks together. I was tasked with writing our memory unit along with the logic to determine which data goes where. While writing the tests for the component, I came across a tough bug where 2 values would be written almost simultaneously to two different addresses. After looking through the affected test and experimenting with it a bit (in this case, a bit = 4-5 hours), the cause of the issue was found due to invalid/old address and value data still present in the wire and being written to

the memory before the logic that controls what values are sent to which wires was executed. After switching up the order of execution, all tests passed and worked correctly. After finishing the memory block, I worked with Joy on integrating her PC/SP component block with my memory block component and writing the test for that step of the integration.

-- Entry #10: 11.1.2018 – 11.7.2018

I was tasked with working on IO for Frankie, and after trying to implement it in 3 different modules at various levels and working through the data path with Sid I finally got input IO working. After that, I spent lot of time looking over my output IO and attempting different methods of displaying output from Mary. Despite these attempts, I couldn't get the X's on the simulation from going away. After almost reviewing the entirety of Frankie and finding some other issues, it was discovered that the output wire for our test was actually a "reg", thus preventing the correct value from being displayed on the waveform. With that now working, I simplified the IO controller logic and verified that IO stor/load instructions would function properly.