CSSE 232 COMP ARCH 1 JOURNAL

TEAM ORANGE:

Drew Kilner

M6:

More debugging and implementing the entire CPU

11/6/23: (10 hours) began by fixing some stage files and then brought them all together in the CPU file, connected every component together according to the datapath. Hazards are the only thing left not implemented. Then later started testing the file and found lots of errors everywhere spent about 3-4 hours just working out errors between code to get the program to simulate in modelsim. Finally got a simulation to work, but after about 3 clock ticks everything stops, need to go back into the stage tests to see what is messed up. Ended up trying to compile in quartus to see what errors were brought up here. Found quite a few and fixed them to get the project to completely compile. Went back into modelsim and started to retest all the stages, finished the fetch it works great, then found that the debug test is lacking some and need some work. Will meet with the group tomorrow to split up testing and get it to a finished state.

11/7/23: (2 hours) fixed a handful of errors in the decode stage that were causing problems.

11/8/23: (3 hours) Fixed a couple things in the execute stage and successfully ran an instruction through the CPU, added 1 to a register multiple times. All the I type and R type instructions work, store and load is broken and the rest is untested.

Verliog debugging and Euclid no hazard code

11/5/23: (4 hours) Began by running through our Euclid's algorithm code and added adding 0 to 0 to have a safe stall before/after any instructions that needed it so that we can run Euclid in a hazard free case. This was fairly simple just took some time to work through it. Went into the forwarding unit code and added the new tests that I found last week. It can now forward to decode for branches, and I fixed the weird load data 2 instructions after it is changed error. This should hopefully be the finalized forwarding unit. Lastly I looked into Ziyu's immediate generator code and found that the only problem was a few things in the test that were false failures, after about an hour all kinks were ironed out.

M5:

Other

11/1/2023: (3 hours) found some missing forward problems in the forwarding unit and implemented them (debugged and worked through more flow diagrams for instruction combinations). Created a table with all possible hazards and forwards for each type of instruction on each type of data update. I now hate the forwarding unit 😊

Adder and Comparator

10/30/23: (2 hours) implemented the simple ALU adder and the comparator for branching and jumps. Wrote the tests and tested the components. ALU adder worked great on the first try, but the comparator messed up for branch if less than and I had to do some debugging to fix it. Overall I ended up putting the opcode from the IR into the comparator to be able to determine what kind of branch or jump is used to output the correct signals.

Forwarding unit

10/26/23: (2 hours) implemented the Forwarding unit in verilog and wrote the test bench files. Have not yet tested I still think I have something missing that is needed as an input into the forwarding unit for proper operation. Also left some don't care situations such as when x0 is used.

10/25/23: (3 hours) Went through all possible hazards, and forwards that our instructions could encounter. Planned out what instructions will need to be forwarded or flushed according. Drew forwarding tables for each hazard and forward. Jump instruction is still a mystery and needs to be handled if we add a delay slot after all jumps.

M4:

Hazards and forwarding

10/25/23: (2 hours) Worked through all possible hazards including where necessary flushing, forwarding, and inserting bubbles. Will only be flushing on an incorrect branch prediction. Loading a word from memory and using it right after in any instruction is bad always causes stalls. Loading a word and using in a branch causes 2 stalls.

M3(datapath design):

Registers

10/12/23: (1hour) Began datapath drawing in online software. Decided to output 3 arguments from the register file due to needing rs1 and rs2 regularly, but also rd is needed for the input data in our store word instruction.

Pipeline implementation

10/13/23: (2 hours) Continued datapath implementation. Started the pipleline type implementation of the datapath. Split memory into instruction and data. Started the pipeline register groups. Put our PC incrementor in the decode stage so that the PC relative branching is not effected by it.

Control

10/16/23: (1 hour) Datapath implementation of control and added registers in the pipeline blocks for each signal needed per cycle. Currently computing 7 control signals, write reg, alusrc, aluop, branch, memwrite, memread, and a regstore(decides what to write to the register).

Forwarding unit

10/17/23: (1 hour) Datapath implementation of the forwarding unit. Added new registers in the pipeline block to send the rs1, rs2, and rd registers for forwarding conflicts. This will fix most problems with registers being needed right after one is operated on. Still have issues when loading a word and then using that register. Must put a bubble and hold the registers when doing this. To prevent have better coding educate.

Branching changes and branch prediction

10/18/23: (2 hours) Changed the datapath for branching. Moved the branching to happen inside the decode stage and not the memory stage. Added muxes for ALU sources 1 and 2 from the forwarding unit, and left the second mux for the imm vs register ALUsrc control. Forwarding unit sends a 2 for normal operation, 1 for write back data, and 0 for memory data. Decided that we are not going to implement a BDS, but we are going to create some simple branch prediction logic stemming from if branched before, branch now, then correcting this with a flush if incorrect.

M2/M1:

9/27/23: (1 hour) Brainstorm what type of architecture we want. Then we landed on an 8 register design. Possible future problems could be not enough registers. Created all the instructions needed to run the algorithm. Key takeaways: focusing on speed, running Euclid algorithm as fast as possible, ended on Load/Store.

10/4/23: (after meeting): (30 minutes) Made ways to change our project from being a 16 bit RISC-V. Key takeaways: changed all symbols for instructions, rearranged it to rd operation rs1, rs2 (a = b + c type of structure).

10/4/23: (15 minutes) Updated symbols in existing document and found some new ideas and changes nothing major.

10/6/23: (2 hours) Created basic examples of functions. Translated from java type code to assembly to machine code. Key takeaways: still felt like RISC-V maybe find some things to change about it later.

10/8/23: (45 minutes) Changed symbols and instructions in my parts of the document to match new instructions. Key takeaways: now feels like our own project no longer RISC-V in 16 bits.

10/8/23 (3 hours) - Created the RTL path for M-Type (loading and storing memory) instructions and the 2 first base instructions for all types. Then started the table of components. Key takeaways: 2nd step creates 3 arguments from rs1 rs2 and rd for later use due to needing those based on our use of instruction types.

10/11/23 (1 and a half hours) – Edited RTL for implementation of pipeline. Added registers we need for pipelining and split memory. Planned for Milestone 3. I will be taking control of the datapath and planning out the pipelining. Key takeaways: RTL is now mostly set for pipelining, and I will be the main outlook for the pipelining.