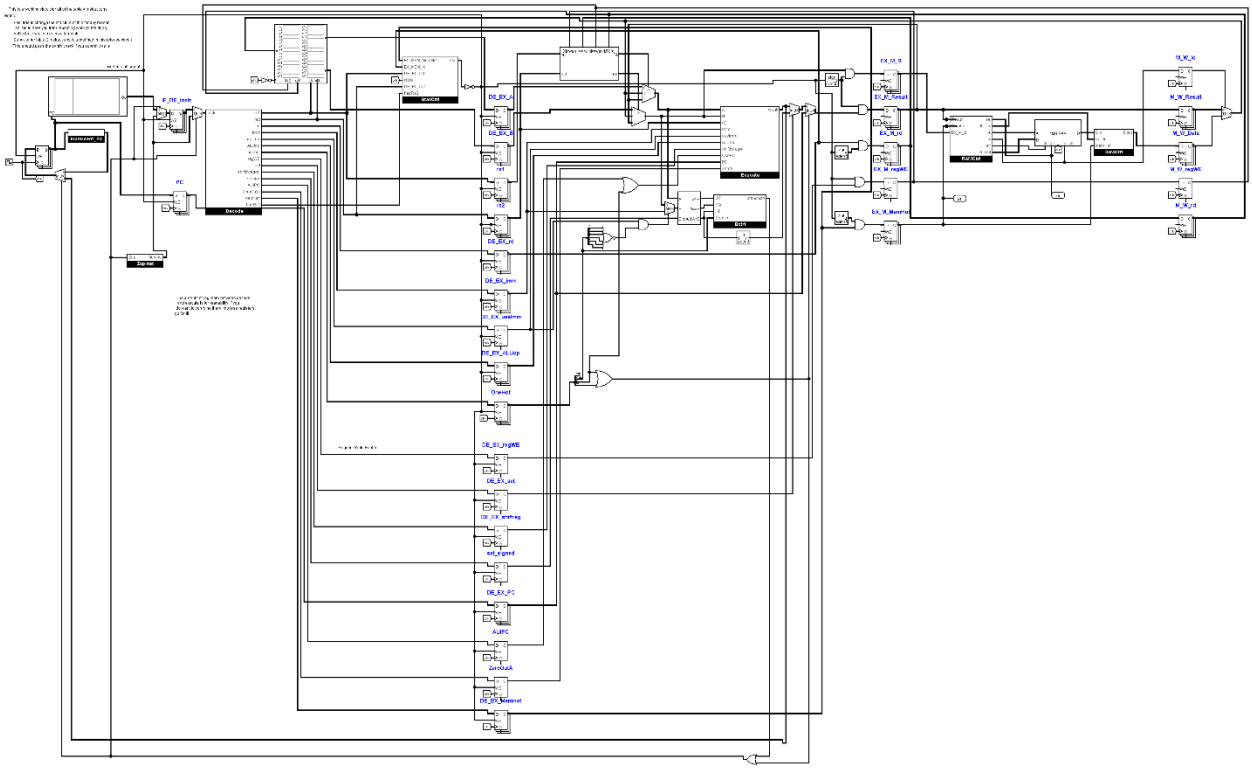
CS3410:P3 Design Documentation

Kevin Klaben (kek228) Hudson Fernandes (haf48) February 20, 2019

1 Introduction

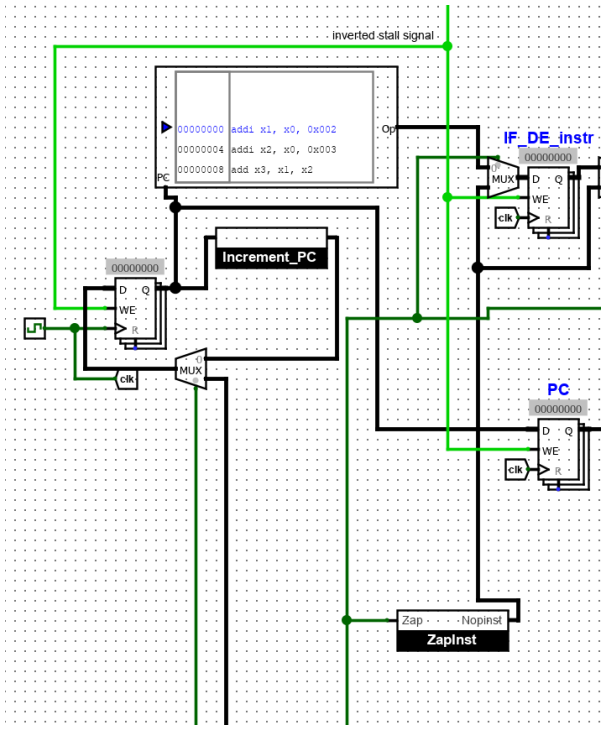
- **Purpose:** The purpose of this document is to act as a reference to the processor and explain our design choices
- **Scope:** The scope of this document is everything in a RISC-V pipelined processor that we've covered in class
- **Intended audience:** The intended audience is anyone with at least some basic knowledge of the concepts behind building a processor
- **Summary:** This will provide an overview of each of the components in our processor changes from the template processor given.

2 Overview



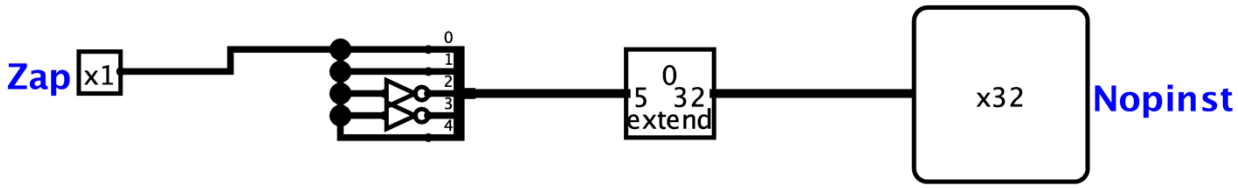
The overall pipelined RISC-V processor design can be seen above. Each stage of the processor will be outlined below as to what changes have been made to the release circuit. The Pipeline registers at the end of each stage are included as part of that diagram, thus fetch/decode registers are included as part of the Fetch stage and so on.

3 The Fetch Stage 3.1 Circuit Diagram



There are three main changes to the fetch stage to accommodate Table B instructions. The first is that a mux was added to the PC to decide between PC+4 to be used as the next PC or if another value sent from the execute stage should be used as the next PC. The control bit for this mux is 1 if the instruction in execute was a jump or if it was a branch and the branch was taken. Otherwise the control bit is and simply PC+4 is used. Next, the stage was changed to accommodate if the branch was taken. A mux is added after the ROM to select between the output of the rom and a signal from the zapinst subcircuit. In the case that a jump or branch was taken in execute, the signal form zapinst is used, otherwise the output of the ROM is used. The zapinst subcircuit is used to create the instruction of a nop to get rid of instructions that would not have been taken given a branch/jump was taken earlier. Finally, wiring was added to prevent the PC from updating during stalls. If a stall signal is sent from decode, write enable is turned off for all fetch registers

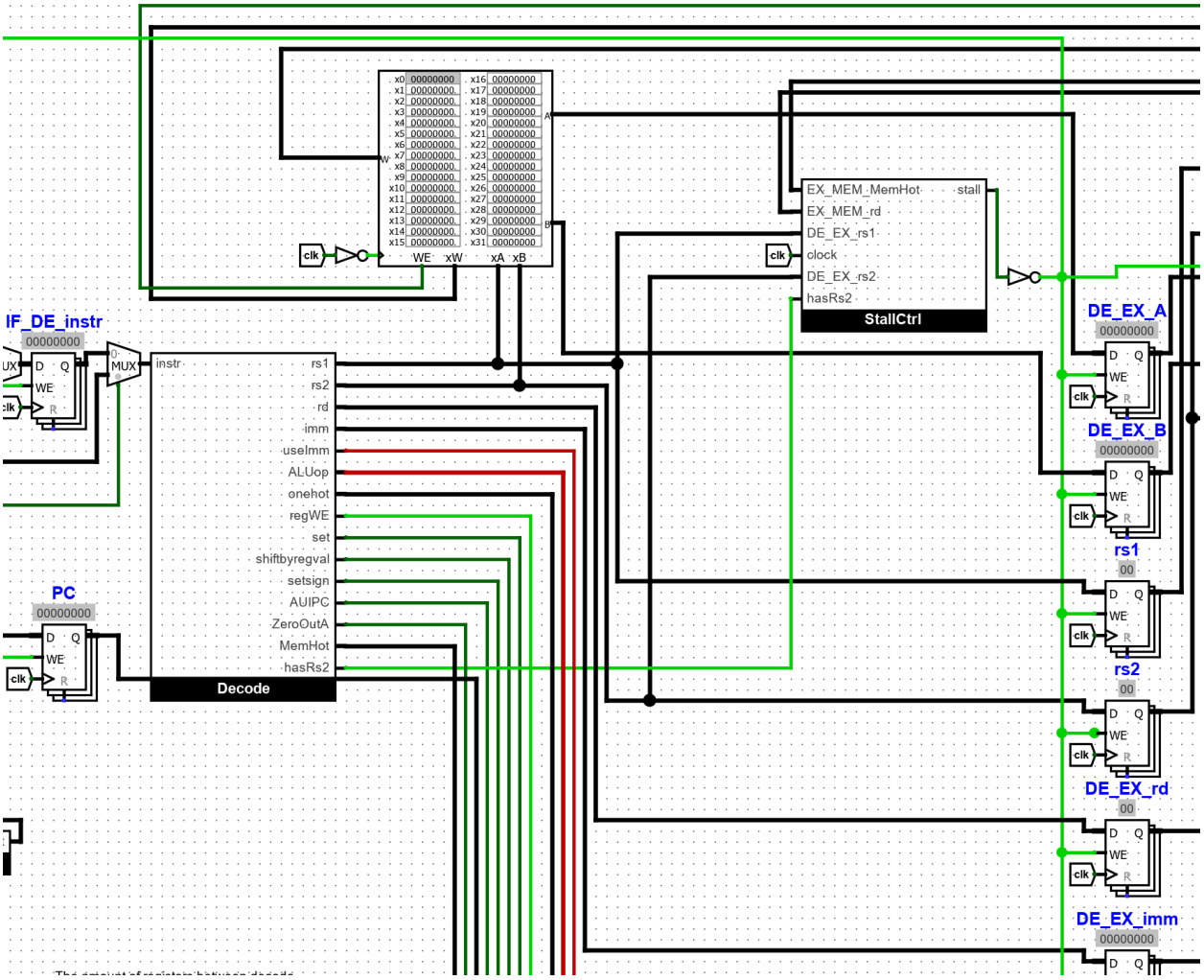
3.1.1- ZapInst Subcircuit



The ZapInst subcircuit will take in 1 if a branch or jump was taken and 0 otherwise. In the case that it takes in a 1, it will produce the instruction code

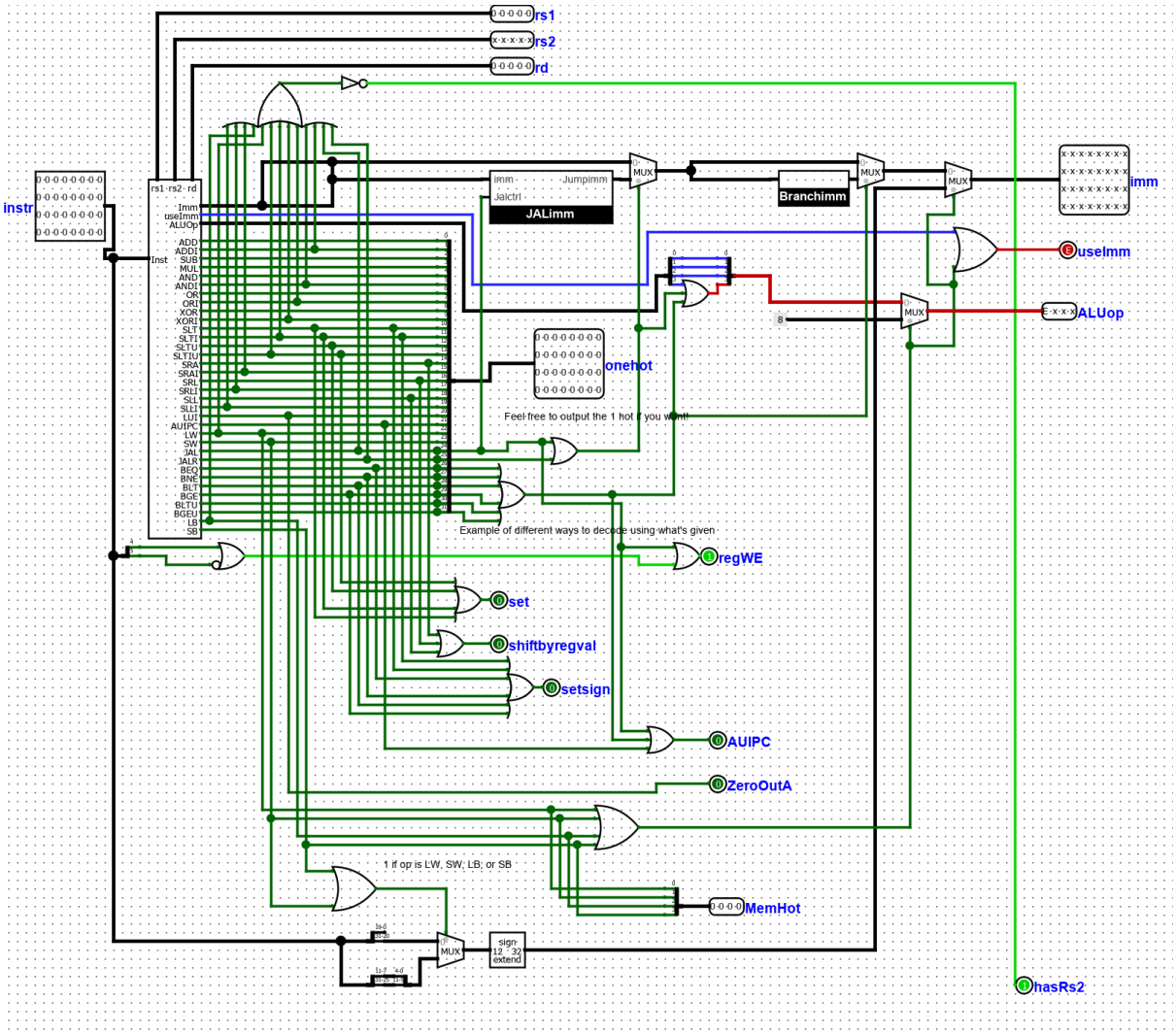
for addi x0, x0, 0 which is the instruction for a Nop. In the case that it is zero, it will produce nonsense, this is ok as this value is only used when Zap is 1.

4 Decode Stage



The decode stage very closely resembles what was initially given to us, with the addition of a StallCtrl subcircuit that determines whether the processor should stall. Beyond this, a mux has been added before the Decode subcircuit, which is used to insert a Nop from zapinst when a branch or jump has been taken in the execute stage.

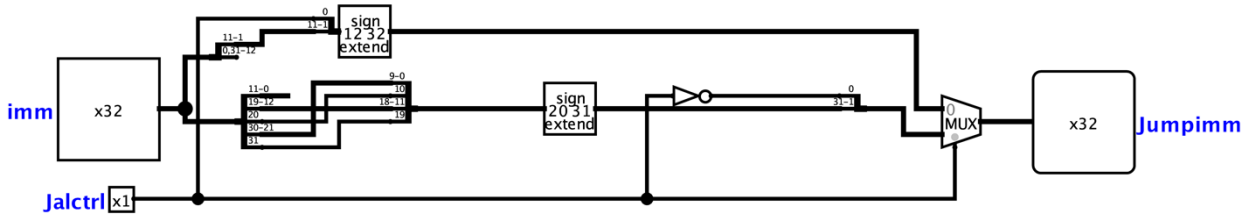
4.1 Decode



Several changes within the Decode subcircuit were made. Two changes were made along the imm line. The first is there is a mux between the immediate output by the Decode Black Box and the output of the JALimm subcircuit. The JALimm subcircuit's immediate value will be used if the instruction being called for is a jump and the original will be used otherwise. Following this mux is one more mux which will similarly mux between the output of the previous mux and the immediate output by the Branchimm subcircuit. The Branchimm subciruit is used when the instruction being called for is branch instruction. The ALUop output was changed in that in the case that a jump or branch instruction is called for, the ALUop is modified to be that of and ADDI instruction such that the addiction operations needed by branches and jumps can be executed. In addition, minor changes are made to out outputs to account for the values those should output for the jumps and branches.

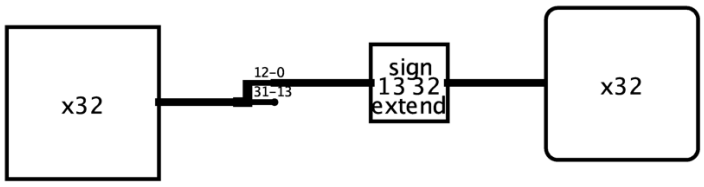
In order to accommodate stalling, two more outputs have been added to the Decode subcircuit. First, hasRs2 is 1 for all instructions with 2 source registers, and 0 otherwise. Next, MemHot is a one-hot encoding of all 4 memory operations. Also, a memory operation requires the addition of B and an immediate to determine the memory address. So, if a memory operation is detected then the ALUop will be set to 1000, which corresponds to addition.

4.1.1 JALimm



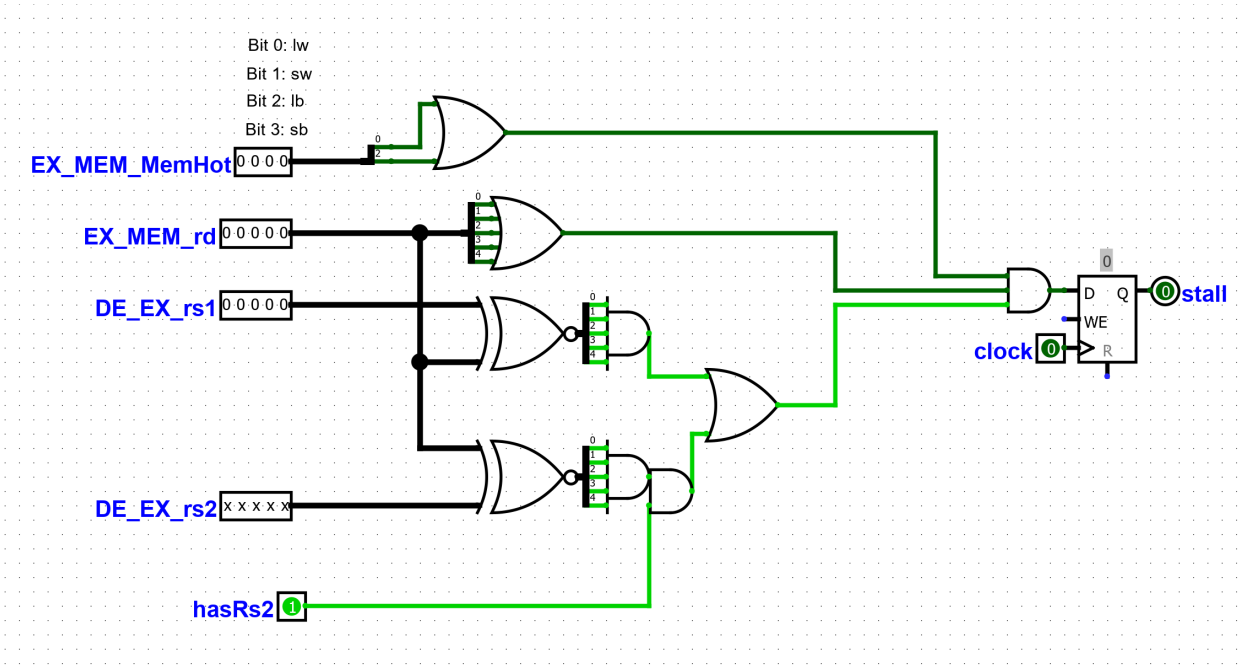
The JALimm subcircuit will take the immediate produced by the Decode black box and Jalctrl which is 1 if a Jal instruction was called for and 0 otherwise. The immediate will be rearranged and extended according to the JAL and JALR instructions or how the immediate should be made. At the end a mux is used to select the JAL immediate if Jalctrl is 1 and the JALR if Jalctrl is 0.

4.1.2 Branchimm



The Branchimm subcircuit is very simple and will create the immediate for a branch instruction.

4.2 StallCtrl



The stallCtrl subcircuit is responsible for determining whether the processor should load use stall. It has 5 inputs: EX_MEM_MemHot is the MemHot encoding of the instruction about to be written to the EX_MEM pipeline registers. EX_MEM_rd is the destination register of that same instruction. DE_EX_rs1, DE_EX_rs2, and hasRs2 all take input from the instruction currently in decode. A stall should occur if:

EX_MEM_MemHot represents a load

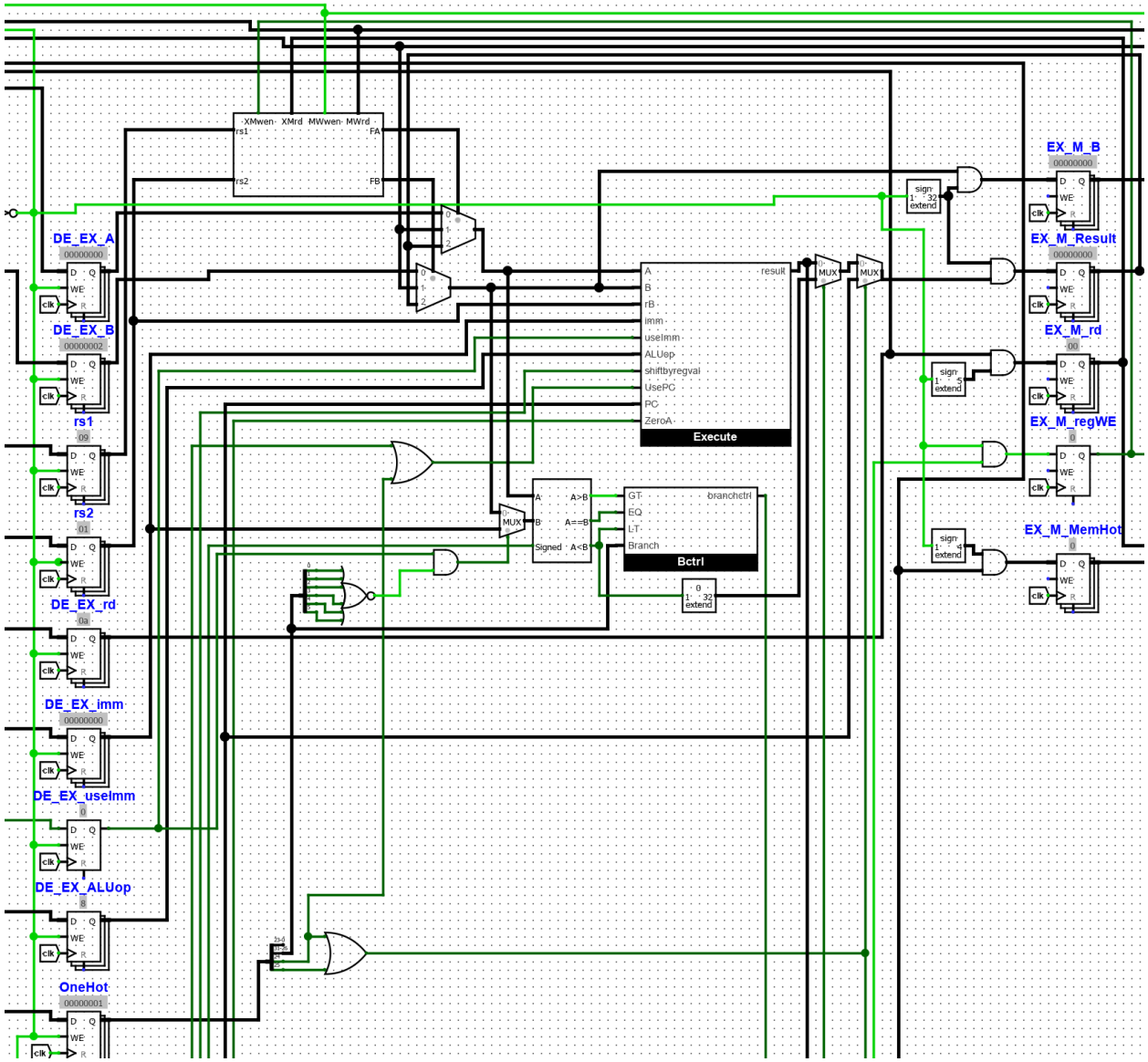
AND
EX_MEM_rd isn't 0

AND

(DE_EX_rs1 = EX_MEM_rd) OR (DE_EX_rs2 = EX_MEM_rd AND hasRs2)

If there is a stall, it is written to a register and carried out the next clock cycle.

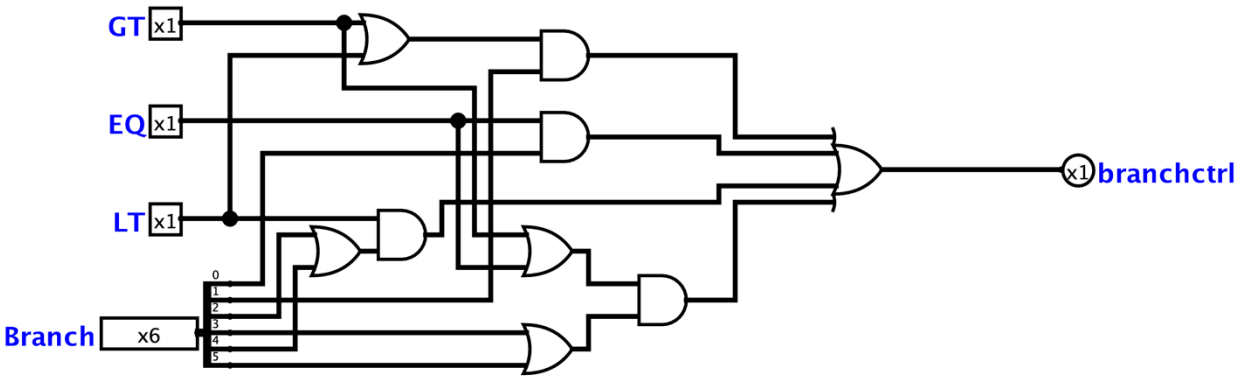
5. Execute Stage



There are several changes to the execute stage at the level of the RISCV circuit. These include the splitting of the Onehot signal into several pieces which are used as control bits to determine what is to be done for branch instructions and jump instructions. A comparator is used with these bits and the outputs are input to the Bctrl subcircuit which will determine if the conditions were met for the branch to be taken. The output of this subcircuit is sent back to be used to determine if instructions need to be zapped. In additions, several smaller controlling operations have been added to adapt for branching and jumping.

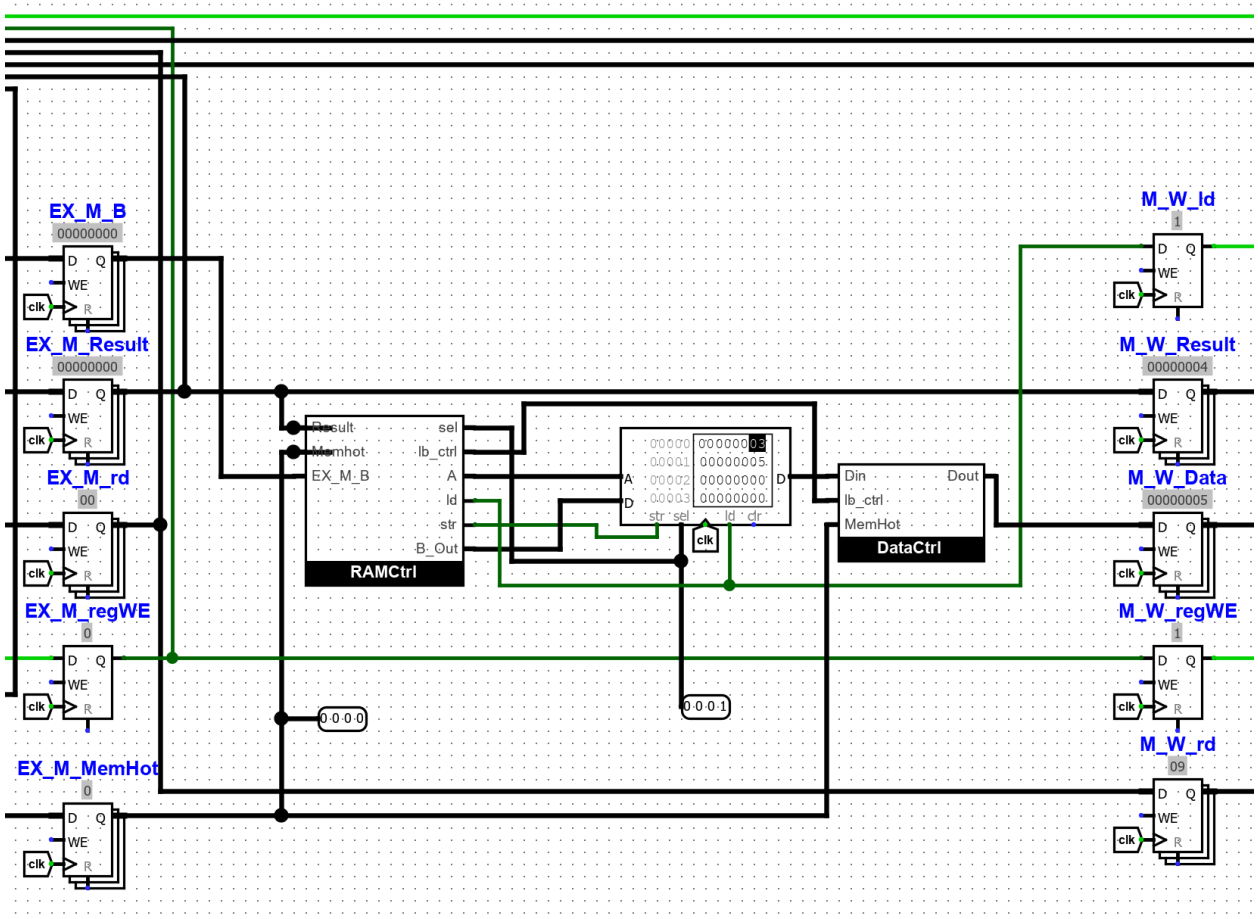
In order to perform a stall, a bubble must be inserted into the pipeline. This is done at the execute stage- if a stall signal is sent, the EX_M registers will be overwritten with zeroes.

5.1 Bctrl



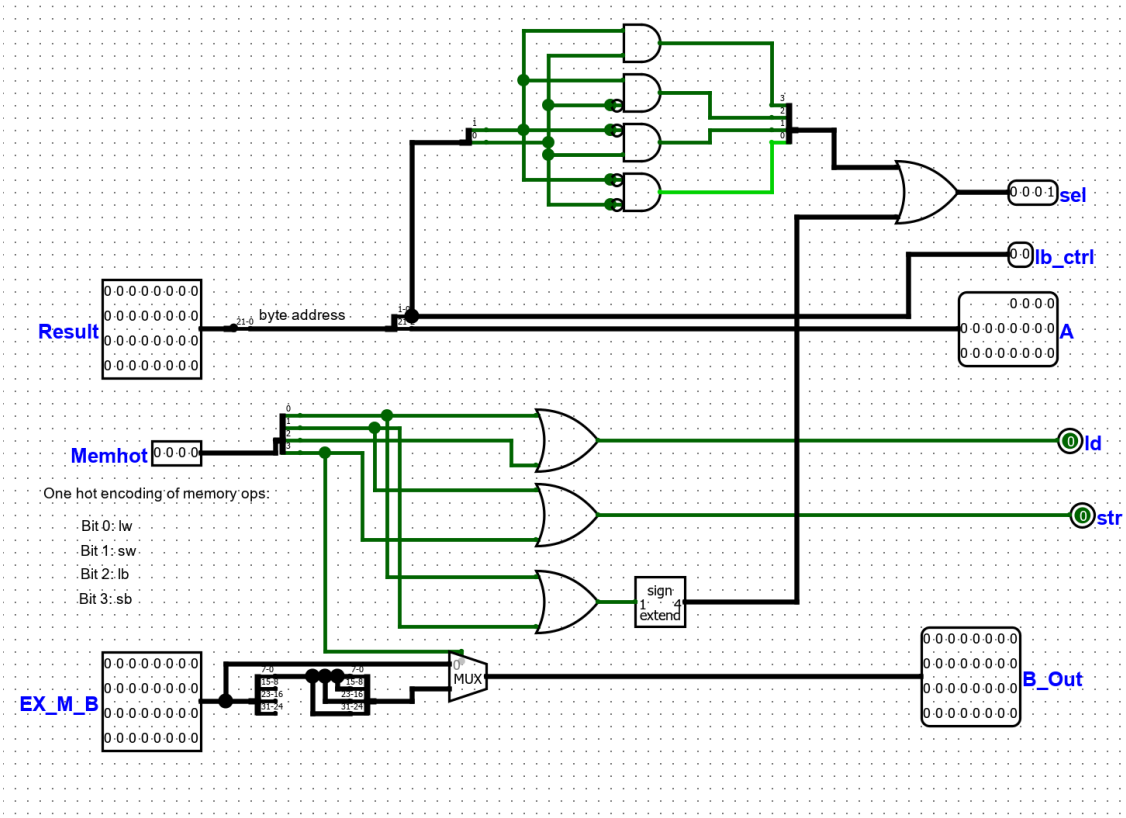
The circuit will take in Onehot encodings for if A was greater than, equal to, or less than B. In, addition, Onehot encodings of which branch instruction if any was called for. Using simple gates, this subcircuit will break apart these inputs to determine if the condition called for by the branch instructions were met and will output 1 to branchctrl if they were and 0 otherwise.

6. Memory Stage



The memory stage is designed to support lw, sw, lb, and sb operations. The RAMCtrl subcircuit forms proper inputs to the given RAM component, while the DataCtrl subcircuit transforms the RAM output to meet RISCV specifications.

6.1 RAMCtrl

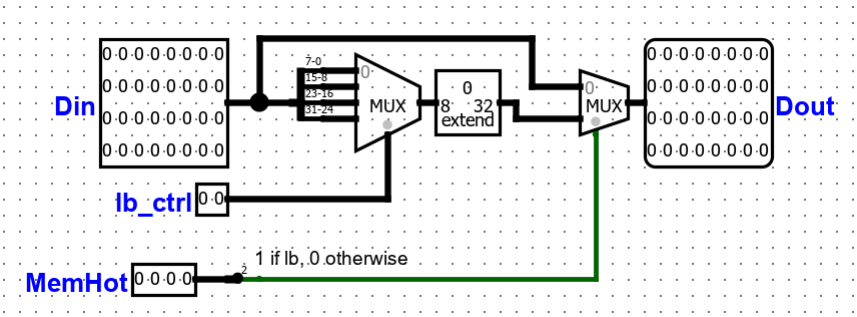


The RAMCtrl subcircuit has 3 inputs: Result is the desired byte address in RAM, Memhot is a one-hot encoding of the memory operation, and EX_M_B is the data being written to RAM (for stores). Sel is input to the sel pin of RAM, it is 1111 for word operations. For byte operations, the last 2 digits of Result determine sel as follows:

Result[1]	Result[0]	sel
0	0	0001
0	1	0010
1	0	0100
1	1	1000

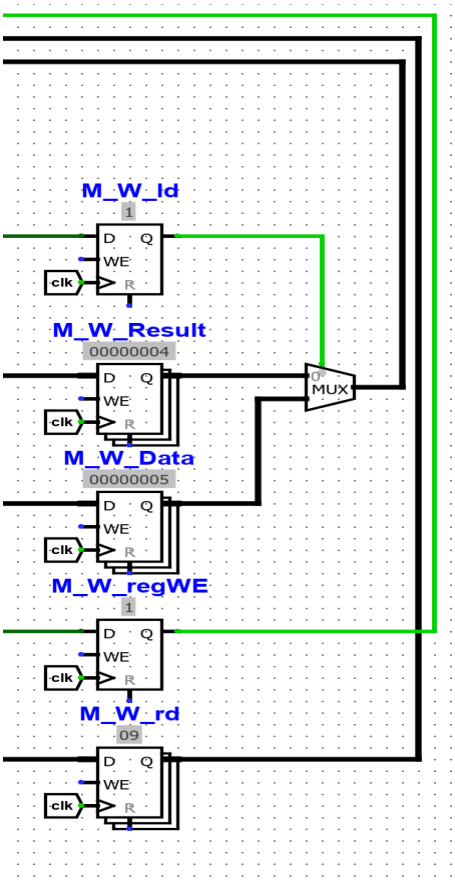
These 2 bits of result are output for later use as lb_ctrl. The remaining 30 bits represent the word address desired in RAM, and are output as A. ld and str are Booleans calculated from Memhot that are needed for RAM. Lastly, B_Out is the data to be input into RAM (for stores). This is just EX_M_B for most operations, with the exception of sb. For sb instructions, the least significant 8 bits are copied 4 times and output as B_Out. This is to guarantee that whichever byte of the data is accessed, it will be the byte sb is meant to store in memory.

6.2 DataCtrl



In the case of a lb operation, the byte output from RAM must be moved to the least significant 8 bits of the 32-bit output (with the other 24 being zeroes). DataCtrl makes this change for lb operations, and does nothing otherwise.

7. Writeback Stage



The writeback stage passes data back to be written to the register file or to be forwarded to the execute stage. The mux determines whether the data

written back should be the data from RAM in the case of a load, or just the result from execute otherwise.

8 Testing-

Each instruction from tables A and B are tested out thoroughly in out testing file on the over all circuit as well as hazard detection