REPORT OF THE IMPLEMENATIO OF THE CPU THROUGH LOGISIM

**Strategy**:

Overall, followed schematic in the notes. Some of the controls were different,

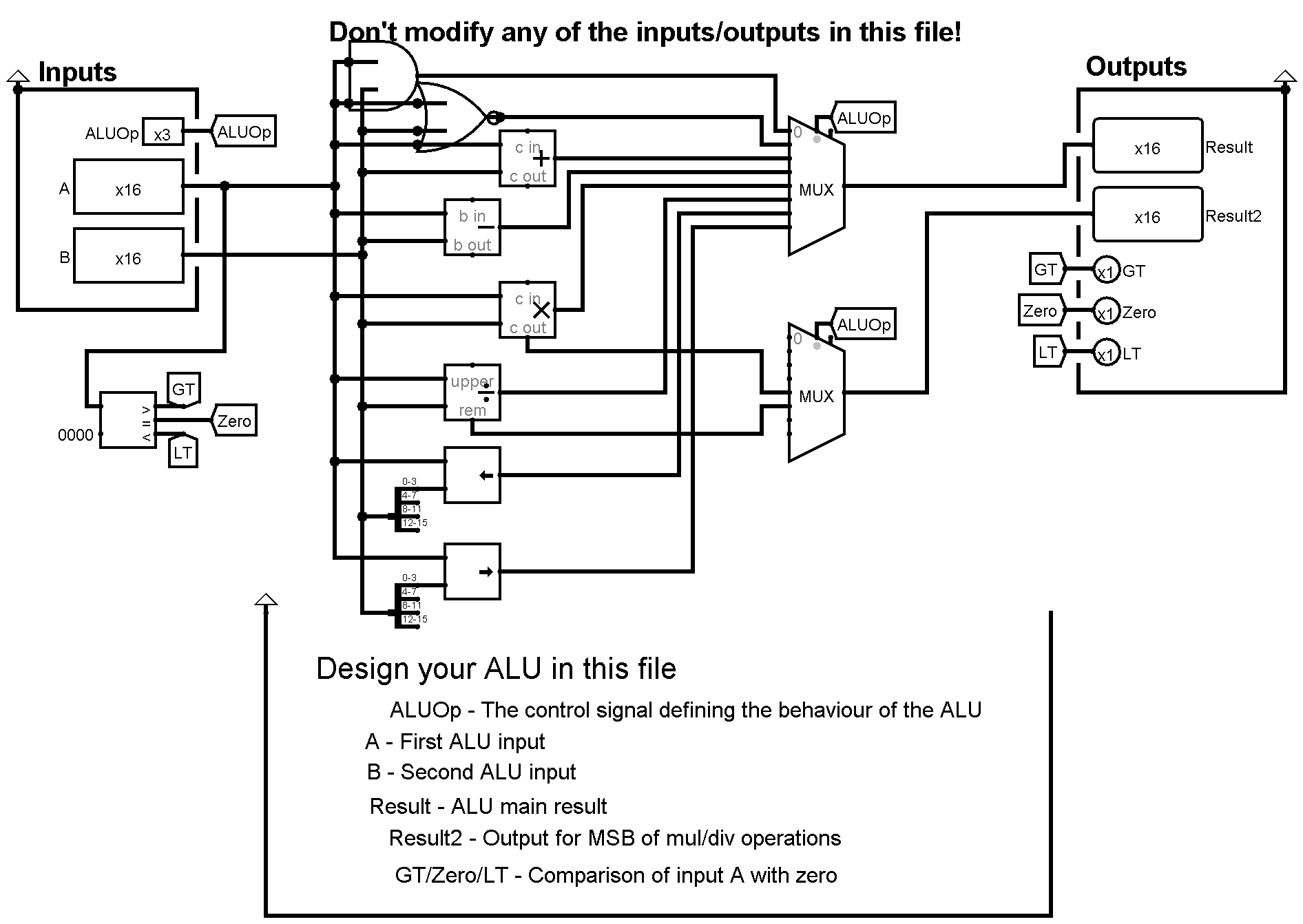
particularly the jumps and branches.

Control Signals:

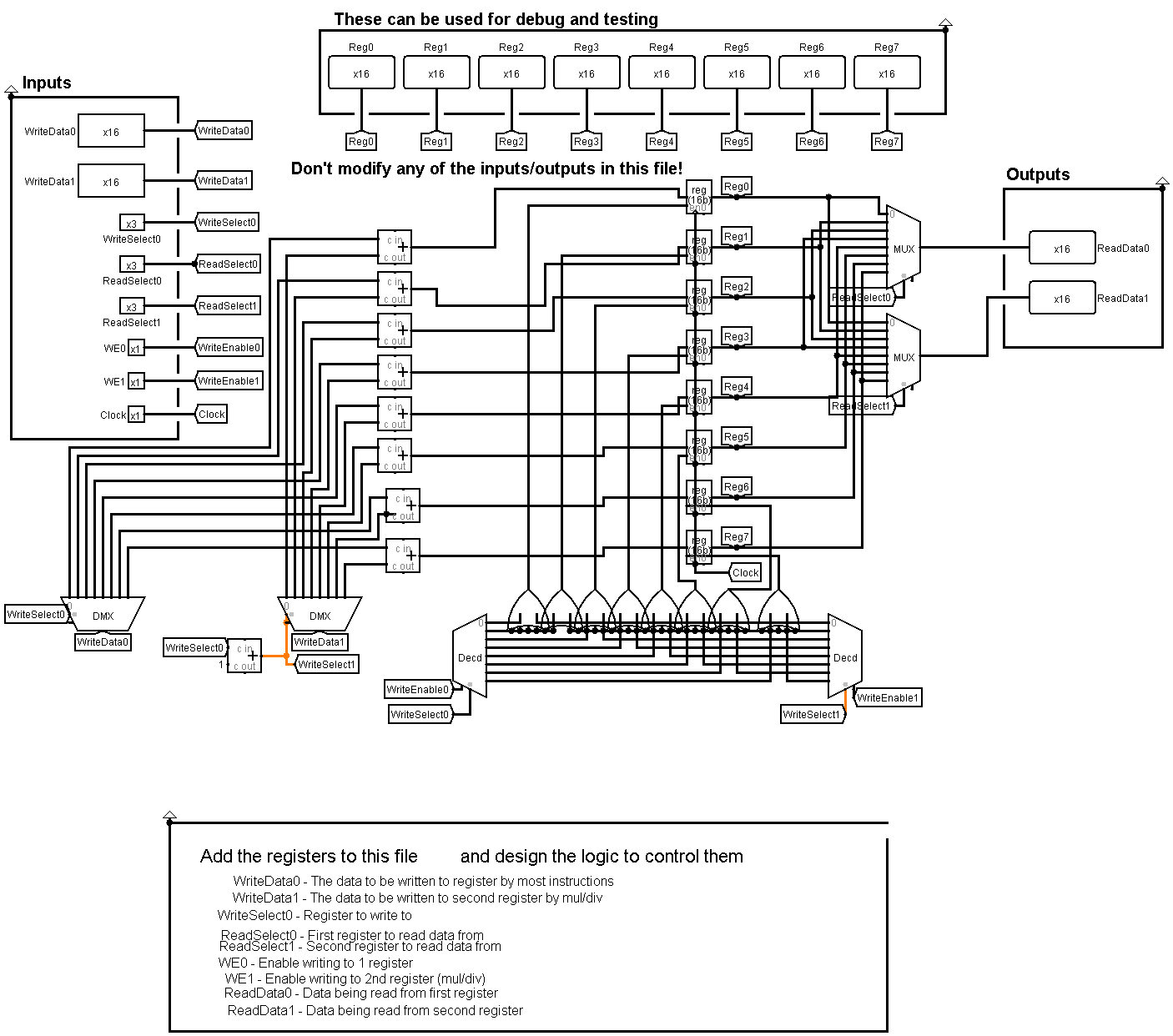
**INPUTS**:

Opcode & Subop from read instruction. Split w/ splitter.

Schematic Design for The ALU



**Schematic Design For The Register Unit**



**OUTPUTS**:

JAL: True on jal instruction. Selects PC+1 as WriteData0, sets PC to the immediate.

Jump: True on j instruction. Sets PC to immediate

JR: True on jr instruction. Sets PC to $rs.

MemToReg: True on LW. Selects memory output as WriteData0.

MemWrite: True on SW. Enables memory store.

ALUSrc: Controls source for ALU input B. True for immediate arithmetic.

RegWrite: Enables Write0. True for anything that sets $rs.

RegWriteB: Enables Write1. True only for div and mul.

ALUOp: Selects ALU operation. 3 bits.

AddZero?: Used in combination with ALU add, sets input A to Zero to allow immediates

to pass thru ALU unchanged. (Perhaps could've been eliminated with another immediate

tunnel and MUX port.)

put: True on put instruction. Updates display register.

halt: True on halt instruction. Disables PC and instruction memory. Lights LED.

BranchCase: Controls branching. 1 on bp, 2 on bn, 3 on bx, 4 on bz.

Goes into BranchControl subcircuit along with ALU comparators.

BranchTrue: Output from BranchControl. True if the branch condition is true. Sets PC

to immediate.

**Notes**:

The second MUX on ALU input B ensures that rt is not read in as an immmediate

on stores.

I didn't have the control distinguish between r and i.

Branch controlling is done in a subcircuit that takes a BranchCase signal from the

main control and the comparisons from the ALU, and outputs a binary BranchTrue signal.

Jumps have a separate control signal for each kind. In retrospect, I could've consolidated

them into a subcircuit, but separate controls work nicely because each jump

does different things with the registers

The MUX on ALU input A is my way of allowing data to pass thru the ALU unchanged.

If I want the passthru, I singal AddZero.

I ended up adding a DataA tunnel right out of the registers for the JR instruction,

and I probably could've consolidated the two passthru methods, but hey, it works.

The main control should be self-explanatory. Just note that Loads and Stores trigger the AddZero passthru, and BranchCase is a 3-bit output.