CS 3432   Fall 2021   Lab 3

For this lab, you will implement part of processor to execute a subset of the RISC-V RV32I arithmetic-logical instructions. The instructions that you will implement for lab 3 are shown in Table 1 below. The tasks you are to carry out are as follows. You should thoroughly test your circuits at each step to make sure each one works correctly before going on to the next step.

Milestone 1 due Sunday, October 24

1. Fill in the truth table shown in Table 2 for the ALU_Control circuit and use the truth table to generate the circuit. You may use a tool such as Logisim's Analyze Circuit to generate the circuit if you wish, or you can construct it by hand. Although we are implementing only the arithmetic-logical instructions for this lab, we will go ahead and include lw and sw inputs to the ALU_Control, since those instructions will use the ALU.
2. Build a 32-bit ALU that can perform and, or, xor, add, and sub. Your ALU should have inputs A (first operand - 32 bits), B (second operand - 32 bits), and ALU_Ctrl (4 bits), and should have output Result (result of A op B). To build your ALU, carry out the following tasks:
   a. Build a 1-bit full adder circuit. Your 1-bit full adder should have inputs A, B, C_in and outputs S and C_out.
   b. Use your 1-bit full adder, together with logic gates and multiplexors available in Logisim to build a 1-bit ALU that can perform and, or, xor, add, and sub. Your 1-bit ALU should have inputs A (1 bit), B (1 bit), C_in (1 bit), ALU_Ctrl (4 bits), and outputs R (1 bit) and C_out (1 bit).
   c. Use your 1-bit ALU circuit to build a 32-bit ALU by chaining the 1-bit ALUs together. You may do this in stages – for example, you could first build a 4-bit ALU by chaining togther four 1-bit ALUs and then build the 32-bit ALU by chaining together eight 4-bit ALUs.

Table 1. Instructions and ALU-Control outputs

| Instruction | Format | Opcode | Funct3 | Funct7 | ALU_Ctrl | ALU_Src |
|---|---|---|---|---|---|---|
| AND | R | 0110011 | 111 | 0000000 | 0000 | 0 |
| OR | R | 0110011 | 110 | 0000000 | 0001 | 0 |
| XOR | R | 0110011 | 100 | 0000000 | 0010 | 0 |
| ADD | R | 0110011 | 000 | 0000000 | 0011 | 0 |
| SUB | R | 0110011 | 000 | 0100000 | 0111 | 0 |
| ANDI | I | 0010011 | 111 |  | 0000 | 1 |
| ORI | I | 0010011 | 110 |  | 0001 | 1 |
| XORI | I | 0010011 | 100 |  | 0010 | 1 |
| ADDI | I | 0010011 | 000 |  | 0011 | 1 |
| LW | I | 0000011 | 010 |  | 0011 | 1 |
| SW | S | 0100011 | 010 |  | 0011 | 1 |

You need only turn in your circuits for Milestone 1. You do not need to schedule a demo until you have completed the entire lab.

Table 2. Truth table for ALU_Control circuit

| Inst'n | $I_{30}$ | $I_{14}$ | $I_{13}$ | $I_{12}$ | $I_6$ | $I_5$ | $I_4$ | $ALU\_Ctrl_3$ | $ALU\_Ctrl_2$ | $ALU\_Ctrl_1$ | $ALU\_Ctrl_0$ | ALU_Src |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AND** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **OR** | | | | | | | | | | | | |
| **XOR** | | | | | | | | | | | | |
| **ADD** | | | | | | | | | | | | |
| **SUB** | | | | | | | | | | | | |
| **ANDI** | X | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **ORI** | | | | | | | | | | | | |
| **XORI** | | | | | | | | | | | | |
| **ADDI** | | | | | | | | | | | | |
| **LW** | | | | | | | | | | | | |
| **SW** | | | | | | | | | | | | |

Milestone 2 due Sunday, October 31

3. Build a register file with 32 32-bit registers. Use the Register component provided in the Logisim Memory library for each individual register. Your register file should be constructed as shown in lecture. It should have inputs RReg1 (5 bits), RReg2 (5 bits), WReg (5 bits), WData (32 bits), Write (1 bit), and Clock_in (1 bit). It should have outputs RData1 and RData2.
4. Connect your ALU_Control, 32-bit ALU, and register file circuits with the necessary wiring and additional logic to construct a datapath that executes the arithmetic-logical instructions in Table 1. Your completed circuit should look similar to the one shown in Figure 1. Test your completed circuit by entering machine language instructions in the instruction input, simulating them, and checking that the execution is correct.
5. **Extra Credit** (optional, 10 extra lab points): In addition to the instructions from Table 1, implement the shift instructions shown in Table 3. You will need to extend the ALU_Control circuit to generate the appropriate outputs for the shift instructions and extend your 32-bit ALU to handle the shift instructions. You may use the Shifter component available in the Logisim Arithmetic library.

**Table 3. Shift instructions and their ALU_Control outputs**

| Instruction | Format | Opcode | Funct3 | Funct7 | ALU_Ctrl | ALU_Src |
|---|---|---|---|---|---|---|
| SLL | R | 0110011 | 001 | 0000000 | 1000 | 0 |
| SRL | R | 0110011 | 101 | 0000000 | 1001 | 0 |
| SRA | R | 0110011 | 101 | 0100000 | 1010 | 0 |
| SLLI | I | 0010011 | 001 | 0000000 | 1000 | 1 |
| SRLI | I | 0010011 | 101 | 0000000 | 1001 | 1 |
| SRAI | I | 0010011 | 101 | 0100000 | 1010 | 1 |

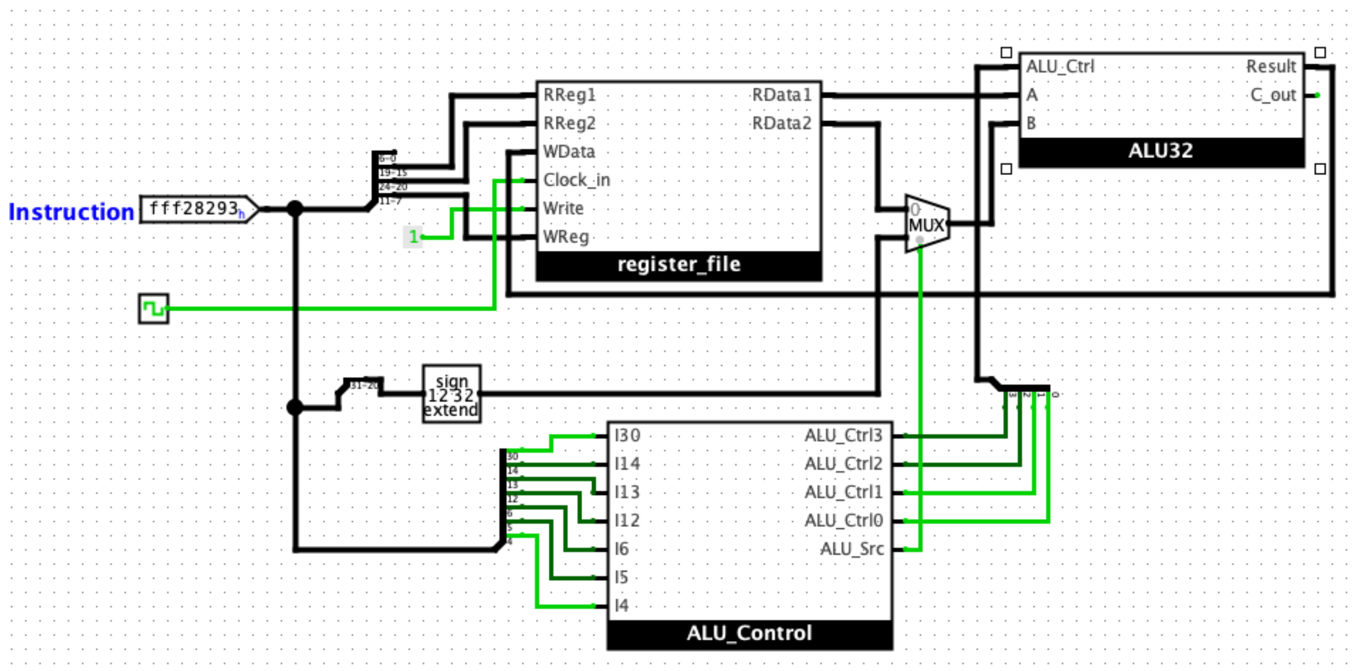

Figure 1. Datapath and control for RISC-V RV32I arithmetic-logical instructions