CSC258H Lab 7: A Simple Processor

Introduction

This week is the final Logisim lab, in which you will finally create a simple 8-bit processor, capable of executing real machine-code.

You'll be building the datapath as well as the control unit for this basic processor. A processor's "data-path includes all of the circuits that store and manipulate the data a program uses, and the control unit stores and manipulates the instructions in the program. Those instructions are used by the control unit to tell the datapath what to do.

To complete the components of the datapath, you will construct the arithmetic logic unit (ALU). This ALU will support two operations of your choosing, that can be used to manipulate data in programs. Then, you will construct the instruction decoder, part of the control unit. Finally, you will hook up the ALU and instruction decoder blocks you just built to a register file, and test the resulting processor by manually feeding it instructions in our custom machine code.

You are provided a starter file `lab07_base.circ` which can be downloaded from Quercus.

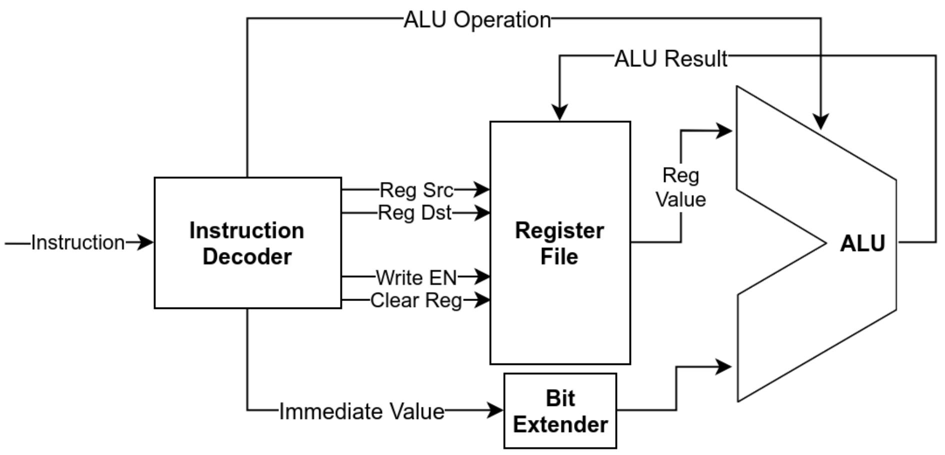
[TASK] Download the starter file and use it as the base for your solution.

This starter circuit contains an implementation of a register file from the previous week with an additional clear input. It also has "skeletons" (the input and output pins) of the sub-circuits you need to build.

Notes: For this lab, you may any Logisim component from **Wiring**, **Plexers**, **Gates**, **Arithmetic**, and **Input/Output** blocks. You may also use **Registers**, from the **Memory** block. You can also use the circuit analyzer (although it's not hugely helpful here).

Processor Structure

Below is a schematic of the processor you will implement. Note this processor is simply enough that all instructions can be done in exactly one clock cycle, and there is no need for an entire control unit FSM. Instead, we will use a combinatorial instruction decoder to determine the control signals.



- The instruction decoder, one of the blocks you will be constructing, is a combinational circuit that decodes an 8-bit instructions fed as input, and outputs the correct values on the control signals (ALU Operation, Reg Src, Reg Clear, etc.) to make the operation happen.
- The register file in the diagram is similar to the one you saw in the lectures, with one read port and one write port. It has four 8-bit registers. The register file accepts an 8-bit data input, two 2-bit select inputs to select which register to read from ("read_reg") and write to ("write_reg"), the clock, a write-enable input, and a reset input. Use Logisim registers to implement it.

  The one difference from what we saw in class: this register file features a Clear Register input ("clear"): if it is high on the positive edge of the clock and if write_enable is high, the register file will ignore "data_write" and will instead write the value 0 to the register.

  [TASK] Implement the register file inside the "register_file" subcircuit using the input and output pins already there.
- The ALU receives two 8-bit inputs, and a 1-bit input to control what operation it runs on those inputs. The output is sent back into the register file. There are no output flags.
- Since the immediate value is 2-bit, and the ALU accepts 8-bit inputs, it needs to be **zero-extended** to 8-bit. You can use the Logisim Bit Extender for this (found **Wiring** in the Design panel), but make sure to configure the extension type to be zero extension, not sign extension.

ALU Sub-Circuit

Given two 8-bit inputs A and B, your ALU sub-circuit will perform two operations of your choosing, selected using a 1-bit control input "op". You do not need to implement these operations yourself; instead, you will select from the built-in **Arithmetic** blocks in Logisim-Evolution. Make sure to set the width of the operations to 8-bits to match the inputs and the output "data_out". Once again, you do not need to output any flags.

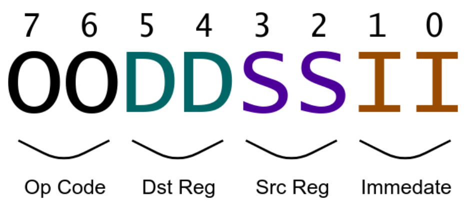
[TASK] Implement the ALU in the "alu" subcircuit using the input and output pins already there.

Historical Context

Some early processors contained a huge number of special purpose circuits. Since then, we have scaled back to a more limited set of functionality, but even today, most ALUs can compute dozens of different functions. It turned out that CPUs that support a more limited set of instructions can be made to run faster and more efficiently. Complex functionality can then be implemented in software. In fact, modern CPUs support more assembly instructions then actually implemented by the ALU. Instead, some operations are performed by running using multiple ALU steps over multiple cycles.

Instruction Decoder Sub-Circuit

The decoder receives an 8-bit input with the instruction. It outputs the 2-bit destination register index, 2-bit source register index, 2-bit immediate value, ALU operation (1-bit), write enable (1-bit), and register clear (1-bit).



For this processor, there is only one type of instruction, in the format shown above. Each instruction contains a 2-bit operation code (op-code), a destination register (indexed from 0 to 3), a source register, and a 2-bit immediate value. This instruction set has been chosen to require minimal logic to decode.

The two ALU operations are operations of **your** choosing. Remember, the ALU's input consists of the register file's output and the immediate value from the instruction, and the output is fed into the register file. It is up to you to determine what the control lines should be to execute these four instructions. The table below shows the specific meaning of each op-code. An example instruction is `01110111` which performs ALU Operation A on the value in register 1 and the immediate value 3, and stores the result in register 3

Opcode	Operation	Description
00	No-Op	do nothing
01	ALU Operation A	Dst reg $\leftarrow$ src reg (OP-A) immediate
10	ALU Operation B	Dst reg $\leftarrow$ src reg (OP-B) immediate
11	Clear Reg	Dst reg $\leftarrow 0$

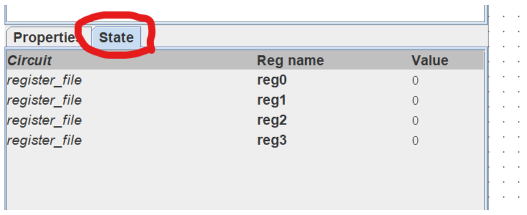
[TASK] Build the decoder unit in the "instr_decoder" subcircuit using the input and output pins already there.

When implementing, you have a choice of either using a truth-table, using the circuit analyzer, or even just looking at the table and figuring things out on your own. It is entirely your choice!

Wiring it together

Now that you have an ALU, a register file, and an instruction decoder, creating the processor in the schematic on the first page should be straightforward. Create a "main" circuit, and wire all of the sub-circuits together according to the schematic. Use Logisim buttons for the CLK and Reset. You may find the **Bit Extender** component (in Wiring) useful in wiring up the immediate to the ALU. You should choose zero extension.

Once you've done this, test a few instructions manually by editing the instruction input and running a clock cycle. To quickly inspect the state of the registers, beside the `Properties` tab, there is a `Registers` tab, which lists the four registers and their values at all times. See figure to the right.



Circuit	Reg name	Value
register_file	reg0	0
register_file	reg1	0
register_file	reg2	0
register_file	reg3	0

[TASK] Create a "main" circuit, wire up the processor, test it, and upload to Quercus.

Summary of tasks

1. Implement the register file, decoder, and ALU in their respective subcircuits.
2. Create the main circuit and wire everything up.
3. Test and submit to Quercus.
4. Demonstrate your processor to your TA.

Evaluation

As always, marks are based not only on submitted work but also on oral examination by TA. You need to be able to make reasonable explanations about any details to the TA.

Solution	Submitting everything on time	1 mark
	Decoder	1 mark
	Register file	1 mark
	ALU	1 mark
	main circuit and wireup	1 mark
Understanding	TA oral score	1 to 3

Final lab marks (up to 5) are determined by multiplying your solution subtotal by the oral score (1–3) and dividing by 3:

$$\text{total} = \text{solution marks} \times \frac{\text{oral score}}{3}$$