CSCB58 Lab 7: Basic Processor

1 Introduction

This week is the final Logisim lab and we will finally create a simple 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_starter.circ` which can be downloaded at:

`https://cscb58f20.ml/labs/lab07_starter.circ`

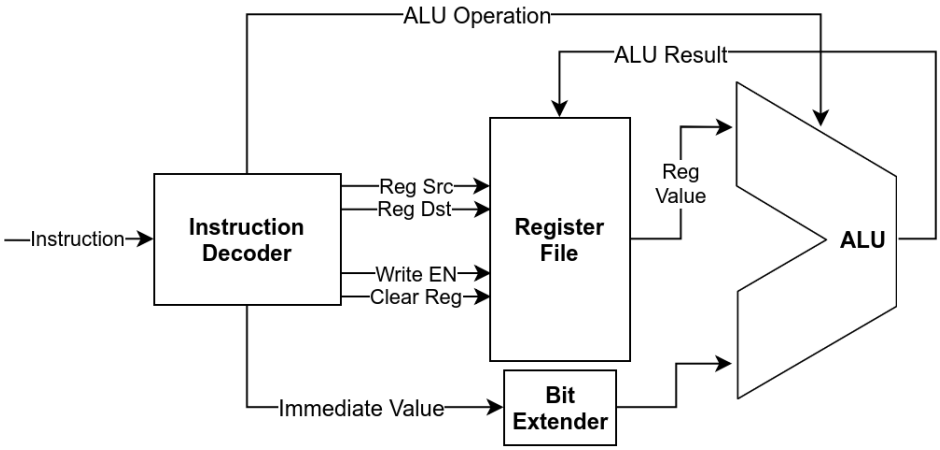
This starter circuit contains an implementation of the register file from the previous week with an additional clear input. It also has skeletons of the sub-circuits you need to build.

PRELAB REPORT: This lab requires the submission of a prelab report to Quercus to **by the start of your lab section**. The deadline shown on Quercus may not apply to you. Your TA will access your prelab submission during the lab and ask you about it.

2 Processor Structure

Below is a schematic of the processor you will implement. Note that the register file is a sequential component, which outputs into combinational

logic in the ALU, then back into itself, making our processor similar to the finite state machines you've seen throughout the course.



- The instruction decoder, one of the blocks you will be constructing, is a combinational circuit that determines what the control lines (ALU Operation, Reg Src, Reg Clear, etc.) should be to perform each operation, based on the 8-bit instruction fed in as input.
- The register file in the diagram is the one you created in the previous lab, with an additional input. It accepts a 4-bit data input as well as clock, write enable, two 2-bit select inputs to select which register to read from and write to, and a register clear input. The Reg Clear input, if high on the positive edge of the clock, will set the selected write register to 0. The register file produces a 4-bit data output that is sent to the ALU.
- The ALU receives two 4-bit inputs, and a 1-bit input to control what operation it runs on those inputs. It's output is sent back into the register file.
- Since the immediate value is 2-bit, and the ALU accepts 4-bit inputs, it needs to be extended to 4-bit.

3 ALU Sub-Circuit

An arithmetic logic unit (ALU) is a computational circuit. Theoretically, all we need is a single NAND gate to compute any function, but computing

with just NAND is very slow. Instead, ALUs usually contain a selection of commonly used circuits. This lets us compute with higher-level circuits (like `add`) which are useful but not universal. In the 1980's, processors contained a huge number of special purpose circuits (including things like "Fast Fourier Transform"). Since then, we have scaled back to a more limited set of functionality, but even today, most ALUs can compute a dozen different functions.

Your ALU sub-circuit will perform two operations of your choosing, which you do not need to implement yourself, instead, you will select from the built-in `Arithmetic` blocks in Logisim-Evolution. These computational circuits will execute in parallel (at the same time), and a 2-to-1 4-bit multiplexer selects the correct output, similar to the register file's implementation. Your ALU should take two 4-bit data inputs and a 1-bit control input. It should produce a single 4-bit output. The control input (ALU Operation) determines which operation should be performed on the inputs.

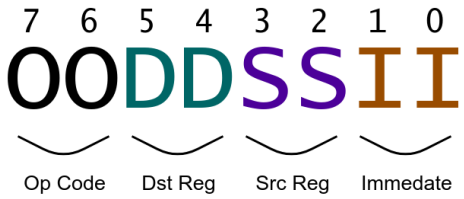
PRELAB REPORT: In your prelab report, include the circuit diagram of the ALU using two high-level blocks of your choosing and a 4-bit 2-to-1 mux. This should be a very simple circuit. "High level blocks" means you can use an "adder" block. Make it clear which wires are buses and their bit-widths.

Inputs: Two 4-bit inputs for operands, a 1 bit control signal to control the selected operation.

Outputs: 4-bit ALU result.

4 Instruction Decoder Sub-Circuit

For this processor, instructions are 8-bit wide, containing the operation code (op-code), the source and destination registers (indexed from 0-3), and an immediate value. This instruction set has been chosen to require minimal logic to decode. For example, to decode the source register, one only needs to split off bits 3 and 2 from the instruction into a separate bus. The only logic required should be to decode the Op-Code into the various control lines, for each instruction. An example instruction is `01110111` which performs ALU Operation A on register 1 and immediate value 3, and stores in register 3.



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.

Op Code Bits:

Bit 1	Bit 0	Operation	Description
0	0	No-Op	No-Operation (do nothing)
0	1	ALU Operation A	Dst Reg = Src Reg (operation A) Immediate
1	0	ALU Operation B	Dst Reg = Src Reg (operation B) Immediate
1	1	Clear Reg	Clear register Dst Reg

PRELAB REPORT: In your prelab report, include a simple circuit to decode these instructions into the various control lines. Some of the control lines may end up being "don't-cares", which means that they don't affect the result. An instruction that "does not affect the result" simply does not change the state of the registers. Make it clear which wires are buses and their bit-widths.

Inputs: 8-bit instruction

Outputs: 2-bit destination register index, 2-bit source register index, 2-bit immediate value, ALU operation, write enable, register clear.

5 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. 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. Here are some additional tips:

- You may find the `Bit Extender` block useful in wiring up the immediate to the ALU.
- You should choose zero extension for the bit-extender.
- For help with buses, refer to the Logisim-Evolution Reference Section 3.

6 Summary of TODOs

1. Before the lab, read through the lab handout. Complete and submit the **PRELAB REPORT** with the following information:
   - Your name and student number.
   - The circuit diagram of the ALU
   - The circuit diagram of the instruction decoder
2. Before the lab, implement the ALU and the instruction decoder sub-circuits.
3. Wire up up the ALU, instruction decoder, and register file in a main circuit.
4. Demonstrate your individual ALU / decoder circuits and the main CPU circuit to your TA in your practical session.

Evaluation (5 marks in total):

- 3 mark for the prelab report / design.
- 2 mark for explaining design and demonstrating running an instruction on the processor.

Congratulations! We are now done with all the Logisim labs. Next week we will start working with a new tool – assembly!