ELEC3441: Computer Architecture
Second Semester, 2023–24

Lab 3

`Objective: Learn about Logisim-Evolution`

In this lab, you will practice using the software Logisim-Evolution for building hardware. Logisim-Evolution is a modern enhancement to the classic Logisim software, which is an excellent software to learn about digital hardware design. In this course, we will use the Logisim-Evolution software to studey how the hardware of a microprocessor works.

1 Getting the Code

You may find the source files needed for this lab from:

`https://www.eee.hku.hk/~elec3441/sp24/handout/elec3441lab3.zip`

2 Getting the Software

Logisim-Evolution is a free open source program written in Java. You can find the source code from:

`https://github.com/logisim-evolution/logisim-evolution`

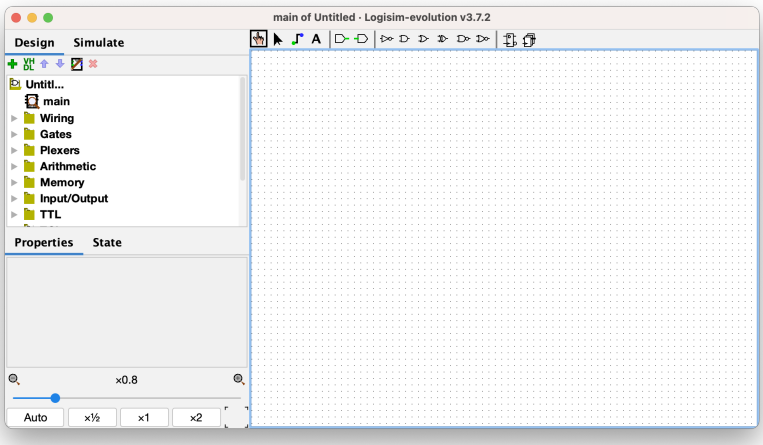
The easiest way to start using the software is to download one of the pre-built installation files for your platform (Windows, Mac, Linux) from the Download page:

`https://github.com/logisim-evolution/logisim-evolution#download`

If you downloaded the `jar` file, you will need the Java run time to execute the `jar` file. Download it here `https://www.java.com/en/download/`.

3 Orienting Yourself

3.1 Starting Logisim-Evolution Run the Logisim-evolution program. You should see a screen similar to the one below:



3.2 Learn about the Basic You can learn about the basic operations of Logisim-Evolution from the built-in tutorial. Access the tutorial by selecting:

Help → Tutorial

For our purposes, make sure you go through the following sections:

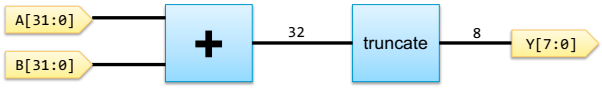
- Beginner's Tutorial
- Additional features → Creating Bundles
- Additional features → Splitters
- Additional features → Wire colors

3.3 Check Yourself

Make sure you know the answers to the following:

- How do you add an input pin to your circuit?
- What is the function of the Poke tool ()?
- Given a 32-bit signal, how do you split the signal and extract only the 8 least significant bits to form a new 8-bit signal?
- What is a "Tick"? How do you use that to test your circuit?

3.4 A Simple Adder Using Logisim-Evolution, build and test the following circuit. This circuit takes two 32-bit values, `A[31:0]` and `B[31:0]` and add the two values using a built-in 32-bit adder. The result is truncated to the lower 8 bit only and output as an 8-bit signal `Y[7:0]`.



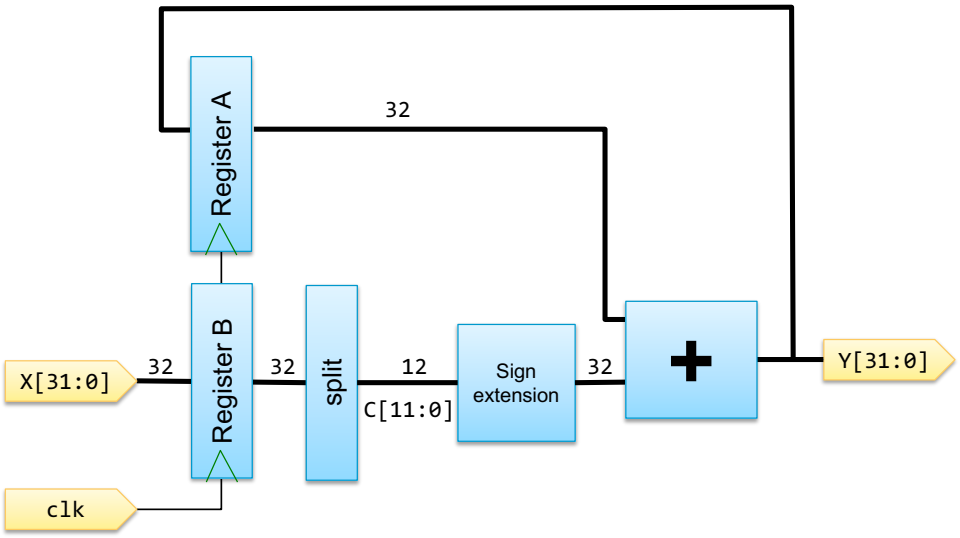
3.5 Checkoff 1

(i) Submit your adder circuit above.
(ii) Show screenshots showing the adder working with various different input?

4 A Small Accumulator

Implement the following mysterious `small-accumulator` in Logisim-Evolution. The circuit takes a single 32-bit signal `X[31:0]`. The signal `X` is first stored in a 32-bit register (Register B). On the next cycle, 12 bit of values from X[19:8] are extracted through the `split` module to form the value `C[11:0]`, which is then expanded back to 32 bit through a *sign extension* block.

Finally, the sign extended value of `C` is added with the current value of register A to produce `Y[31:0]` with a 32-bit adder. The value of `Y[31:0]` is written back to Register A for next cycle.



4.1 Sign Extension In class, we have discussed the operating principle of the sign extension block. Here, you will implement one such block in Logisim-Evolution

- Open the file `smallaccu.circ` from the downloaded source files.
- **Double Click** the circuit icon `signext` on the left (right below `main`).

You should now see the sign extension circuit that is inside the main small accumulator circuit.

The `signext` circuit has one input `C[11:0]` and one 32-bit output `CX[32:0]`. The entire circuit has been done completed for you except that there is problem with the two **Constant** blocks.

Your task is to complete the circuit by setting the correct values for the two constant blocks. To do so,

- Click the constant block at the input the multiplexer.
- On the bottom left, in the "Attribute Table" pane, you will see the attribute **value**. Change from the default value of `0x0` to the correct value. Remember, the constant is meant to be 12-bit wide.

4.2 Complete Circuit Now, complete the rest of the circuit by:

- Open the file `smallacc.circ` if you have not done so.
- **Double-click** the `main` circuit. You should see a template circuit with all the necessary sub-blocks.

Your task is to connect the blocks and implement the necessary logic that extracts the 12 bits of signals from Register B to form C and connect to `signext`.

4.3 Checkoff 2

Implement and test your small accumulator circuit. Submit:

- Your completed circuit file (`smallacc.circ`).
- Screenshot showing your test cases and how the values are accumulated on each cycle.

4.4 Submission Submit your answers to Checkoff 1, 2 above.

4.5 Going Beyond If you compared the design of `smallacc` and the single cycle processor, you should find a lot of similarity. Think about these similar design:

- Register A $\leftrightarrow$ Register File
- Register B $\leftrightarrow$ PC
- Adder $\leftrightarrow$ ALU
- Split $\leftrightarrow$ Instruction Decode