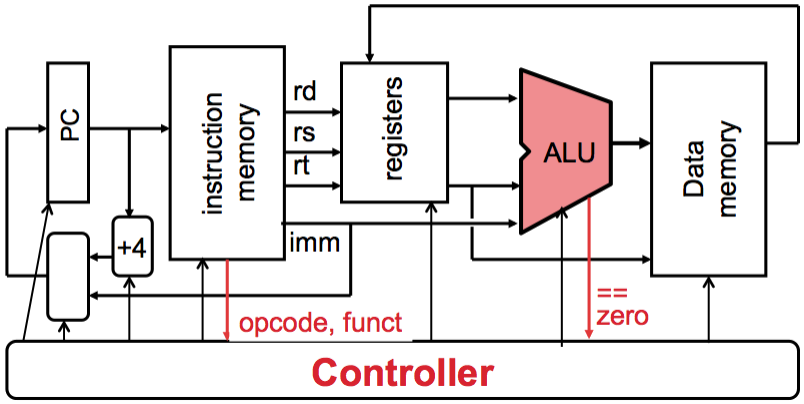
CPU Project Prep in Logisim

Exercise 1: Logisim ALU

In this exercise, you will first implement a 32 bit ALU in Logisim. Remember: we have provided a starter file here!

As a reminder, recall that ALU stands for *Arithmetic Logic Unit*. An ALU is a fundamental building block of a CPU (central processing unit) and it performs integer arithmetic and logical (bitwise) operations. The function that the ALU performs (e.g. add, xor) is determined by the control of our datapath, which is determined by the instruction the processor is executing. The ALU is highlighted in a simplified datapath diagram below:



This lab assignment is similar to the CPU design project (it is essentially a slightly simpler version of Project 2's ALU). Hopefully by getting a headstart here in lab, Project 2 will go a bit more smoothly!

The 8 functions that you will implement are: shift left logical, shift right logical, shift right arithmetic, rotate left, rotate right, and, or, and xor. The ALU will perform a desired function on 2 32-bit inputs and output the result. The selected function will be determined by the value of the control signal, as listed below. Here's what occurs for each operation:

Control	Operation	Description
000	Shift Left Logical	Shift source1 left by source2
001	Shift Right Logical	Shift source1 right logical by source2
010	Shift Right Arithmetic	Shift source1 right arithmetic by source2
011	Rotate Left	Rotate the bits of source1 left by source2
100	Rotate Right	Rotate the bits of source1 right by source2
101	And	Compute `source1 & source2`
110	Inclusive Or	Compute `source1 | source2`
111	Exclusive Or	Compute `source1 ^ source2`

How the heck do I start doing this? Good question! If you've read through the descriptions of the ALU above, you should know that it's just a circuit that can take two inputs and spit out a binary operation on them based on a control signal. What piece of hardware lets you choose between some inputs? (Everybody all together: A _ _ _!!)

So, how do we get the things to wire into the _ _ _? Well, we've laid out the 8 operations we want to be able to do, and if you look at the "Arithmetic" library in the left-hand side dropdown menu, you should find pretty much everything you need to implement these operations. By the way, you ARE allowed to use any and all of these now.

You'll notice, if you try to use the SHIFTER, that you can't just stick **source1** and **source2** into the box. This is because if you set the "data bits" field of a shifter to 32, it automatically knows that it only makes sense to shift a 32-bit number by a 5-bit number, so it will start to only accept a 5-bit value for its second input. Smart, huh? But inconvenient for you. Make use of those bit extender things to fix this problem.

YOU SHOULD USE SOME TUNNELS. The circuit harness (AKA skeleton) is designed to fit an implementation of the ALU that makes good use of tunnels. You might have to reuse the "source1" and "source2" tunnels a few times.

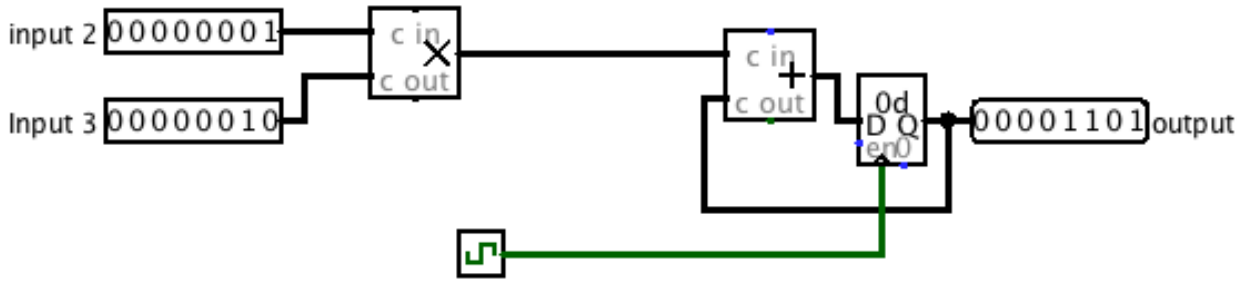
Checkoff

- *Set two inputs to your ALU. Toggle the control bits to change the function being performed on the output and verify with your TA that the output is correct.*

Exercise 2: Pipelining

This next exercise will get you some hands-on practice with pipelining. Assume that on power-on, registers initially contain zeros.

Consider the following 2-input circuit. Its output is computed by multiplying the inputs and adding it to the current value.



Say the propagation delay of a adder block is 50ns, the propagation delay of a multiplication block is 55 ns, and the clk-to-q delay of a register is 5ns. Calculate the maximum clock rate at which this circuit can operate. Assume that the register setup time is negligible, and that both inputs come from clocked registers that receive their data from an outside source.

Checkoff: *Show your TA the calculations you performed to find the maximum clock rate (non-pipelined).*

We want to improve the throughput of this circuit, and let it operate at a higher clock rate. To do so, we will divide up the multiplication and addition into two different pipeline stages; in the first pipeline stage, we will perform the multiplication of the two inputs. In the second pipeline stage, we will add the product to the state.

Our definition of "correctness" will be simple: we will consider the sequence of outputs from this circuit "correct" iff it corresponds to the sequence of outputs the non-pipelined version would emit, potentially with some leading zeros. For example, if for some sequence of inputs the non-pipelined version emits [3,5,1,2,4, ...], a correct circuit might emit the sequence of outputs [0,3,5,1,2,4, ...] for that same sequence of inputs.

For your convenience and to help standardize check-offs, we are providing a starting point in the files pipeline.circ and ROMdata (use the links to download). In pipeline.circ, the sub-circuit Non-pipelined is set up exactly as the figure above. The main circuit is set up to produce the output sequence [3,5,1,2,4,-1,0,0,...] on the non-pipelined version of this circuit. It is also a handy example of how to use memory from a file. The ROM block should be initialized to the proper data, but if it is zero-ed out, right-click it and choose "Load image..." and select ROMdata.

Note that we need a register to hold the intermediate value of the computation between pipeline stages. This is a general theme with pipelines.

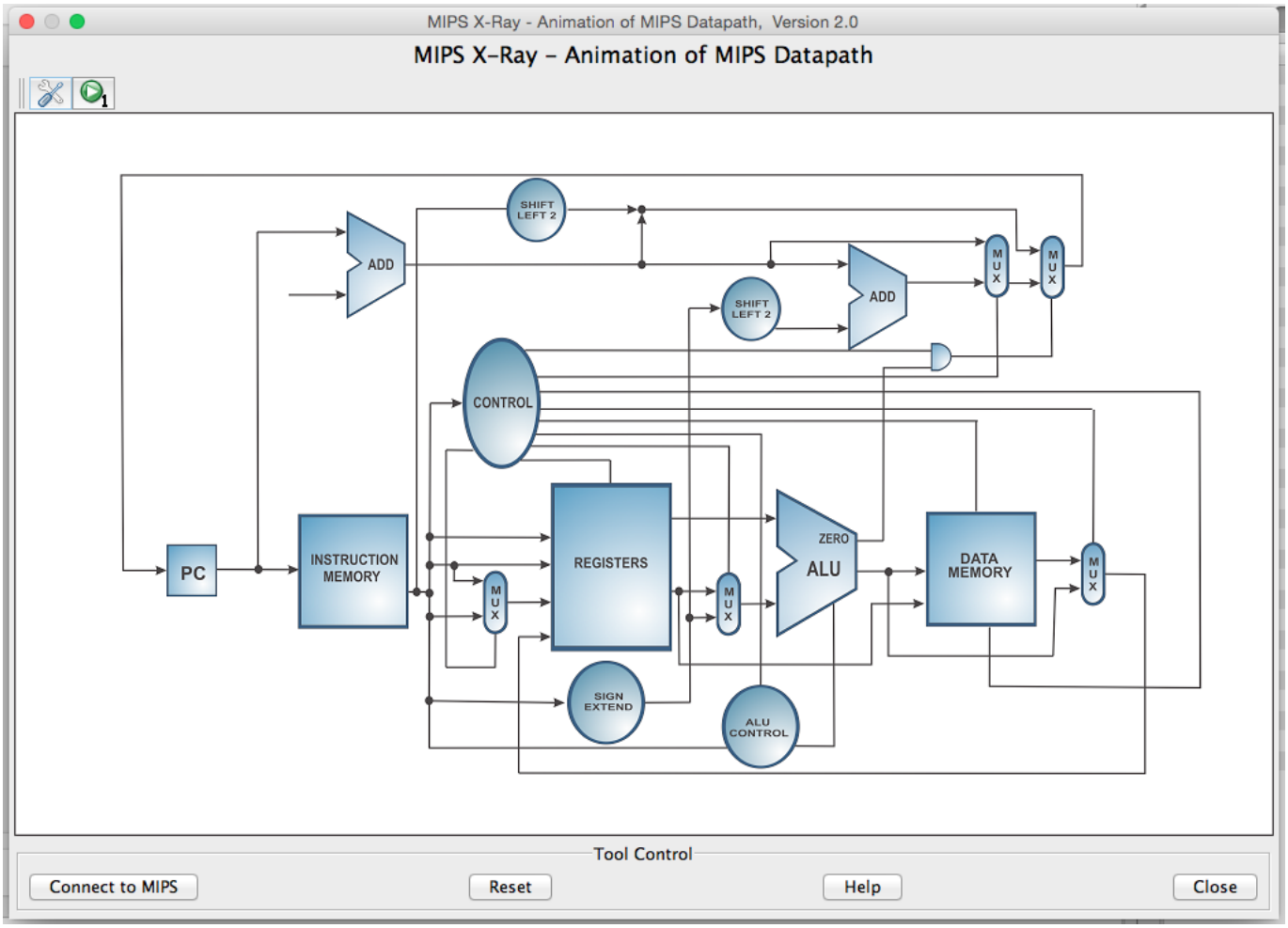
1. Complete the sub-circuit Pipelined. You will need to add a register to separate the multiplication and addition into 2 pipeline stages.
2. Calculate the maximum clock rate for the pipelined version of the circuit.
3. When we talked about pipelining in lecture, we discussed that if a computation depends on the output of a previous computation, it's difficult to pipeline them and we often need to insert a pipeline "bubble" (or several) to ensure that the output of the first computation is ready to be an input to the second. Explain why inserting such "bubbles" is unnecessary for this particular circuit.

Checkoff

- *Show your TA the completed, pipelined circuit.*
- *Show your TA the calculations you performed to find the maximum clock rate (pipelined).*
- *Explain to your TA why bubbles are unnecessary in this circuit.*

Optional exercise: MIPS X-RAY

It turns out that MARS 4.5 has a new feature called MIPS X-RAY that will visualize the datapath while you are running your instruction. This might be useful for you for part 2 of the project! To run MARS 4.5 run the command mars-4.5. To open up this feature go under "Tools" and select "MIPS X RAY". It should open a window that looks like this:



We will use the file fib.s to see how each instruction goes through the datapath. Open fib.s and the MIPS X-RAY feature. You can connect the X-RAY to the instructions by pressing the "Connect to MIPS" button on the bottom left of the screen. Assemble the file by pressing the "Assemble" button on the upper left of the screen. You can then step through the code using the green "Step" button next to the "Assemble" button. The wires that are red signal that the control signal is on. You can also click on any control block to see what is going on inside of it. Play around with the feature, and the following questions will be based off what is going on in the control block.

Note: The X-RAY feature generalizes instructions to have the same opcode. You can see this in control with the instructions add and addi. For the questions below, base your answers on the logic of the control block.

Questions:

- What do the bits BIT 0 - BIT 5 represent?
- For what instruction or type of instructions is opALU0 active? What operation in the ALU does opALU0 represent?
- For what instruction or type of instructions is opALU1 active?
- For what instruction or type of instructions is ALUSrc active? What does this signal represent with respect to the input to the ALU?