Lab 12: MiniMIPS Example of Datapath and Control
CS/CoE 0447: Fall 2017

Introduction

To help get started on the CS 447 JrMIPS project, we have developed an even simpler variant of MIPS, called MiniMIPS. This simple version is intended to guide you through how to build a basic datapath and control unit that can do some simple operations.

MiniMIPS has only a handful of instructions, two registers (A and B), no memory instructions and no control transfer instructions. By following this guide, you'll build MiniMIPS. You can take the same concepts that you learn and apply them to JrMIPS.

OK, now, let's get started!

Instructions

The instruction set is very simple { there are only ve opcodes. There are only two registers, A and B. The registers are 8 bits. Every instruction has a destination register (either A or B). The source registers are always A and B. The instructions are:

Instructions		
Opcode	Assembly	Definition
0000	set $r,imm8	$r imm8
0100	add $r	$r A + B
0101	sub $r	$r A - B
1000	nor $r	$r A !| B
1001	nand $r	$r A !& B
$r is register A or B. Register A is $r= 0 and register B is $r= 1.		

The instruction format specifies the width of an instruction is 16 bits with a 4-bit opcode, a 1-bit destination register, and an 8-bit immediate. The format is:

Instruction Format			
Opcode	Register	Unused	Immediate
bits 15 to 12	bit 11	bits 10 to 8	bits 7 to 0

Instruction addresses are 8 bits. Thus, a program can have up to 256 16-bit instructions.

Components and Design

To implement MiniMIPS, requires only a few components, many of which are available in Logisim's library. The components include the following:

- Registers { hold A and B, 8 bits
- ROM { 8-bit address, 16-bit data width (instruction) Program Counter { 8-bit register

- Fetch adder - 8-bit adder incremented by 1 on each cycle
- ALU - four functions: add, sub, nand, nor
- Control unit - decodes opcode for Muxes and ALU
- Mux - select between imm8 for set and output of ALU
- Fields - extract opcode, register and immediate from instruction
- Decoder - generates control signals from opcode

In the following steps, we will add or create these components.

Step 1: Now, it's time to start. Open Logisim to a blank circuit page. When Logisim starts, this is the “main" circuit. We will build the datapath here and connect everything together. We will also create some subcircuits.

Step 2: The left window pane in Logisim contains the “Component Library". We can use prebuilt components from this library. Let's add a memory to hold instructions. Click on “Memory". This opens the selection of memory components. Click on the ROM. Then move the pointer to the circuit grid. This will add the ROM to the circuit.

Step 3: By default, Logisim creates a ROM with an 8-bit address and an 8-bit data width. The ROM will hold the program, composed of 16-bit MiniMIPS instructions. Thus, we need to change the data width to 16 bits. Click on the ROM in the main circuit grid. This will open a window pane on the left with the ROM's attributes. One of the attributes is “Data Bit Width". Con gure this to 16 bits by clicking on it, and scrolling down to 16.

Step 4: A memory typically has a select signal that controls whether the memory will do any operation. The ROM has a select signal labeled “Sel". We need to set this to logic-1. Select \Wiring" in the component library, and then add a “1 constant" to the circuit. Draw a line, using the pointer, between the 1 and the Sel of the ROM.

Step 5: We need a program counter. The PC is just a register. Select the Memory library components, and then grab a Register. Add it to the circuit. By default, the Register is 8 bits, which matches the address size for MiniMIPS (8 bit address for instructions). Connect the Q output of the register to the A input of the ROM. This connects the PC to the address input of the ROM.

Step 6: Each clock cycle, we need to increment the PC by 1 instruction. Thus, we need to add an adder to the circuit and wire it to the PC. Select the Arithmetic components and add an Adder to the circuit. By default, the adder is 8 bits. Connect the output of the Adder to the D input of the PC. We need to add a Constant 1 for addition. The simplest solution is to add a 1 from the wiring, and then configure it as an 8-bit value. Set the attribute to 0x01. Connect the constant 1 to the second input of the Adder. If you didn't change the width of the constant, you'll get an error (marked in orange). Once the constant is done, then connect the Q output of the PC to the first input of the Adder. Lastly, the PC register has a “En" signal. This is an “enable" signal. When set to 1, the register can be written on a clock cycle. Since we update the PC every cycle, connect the enable to the 1-bit constant 1 you connected to the ROM's Sel. The Fetch portion of the datapath should now be completed.

Step 7: We need two registers, A and B. Add two registers to the circuit diagram. We will connect these registers to the output of the ROM, so you will probably want to put the registers to the right of the ROM in the diagram.

Step 8: Now, it's time to build the ALU. We will do this as a “Subcircuit". Click on the drop-down menu Project -> Add Circuit. Let's give subcircuit the name “ALU". You should now have a blank circuit grid, labeled “ALU" at the top of the window. We build the ALU by adding 8-bit NAND, NOR, Adder and Subtractor components from the Component Library. Go ahead and try this! (Hint: NAND and NOR gates can be configured to have two 8-bit inputs.)

We need to add two 8-bit input pins: Click on the little box on the toolbar (the box with the green circle). This is a Pin. By default, the Pin is 1 bit. Configure it to be 8 bits. In the Label field, put “A". Add a second input pin, again configuring as 8 bits and labeling “B". Connect the pins for A and B to the AND, OR, Adder and Subtractor.

Now, we need a 4:1 mux to select one of the four ALU functions. Under the Plexers components, select a Mux. Because we need a 4:1 mux, configure it to have 2 select signals (2^2 = 4 outputs) and an 8-bit data width. By default, a Mux has an Enable signal. You can configure the Mux to not use the Enable -- do this now by clicking on “Include Enable?" in the attributes to change to No. Next, connect the Mux: connect input 0 to the output of the Adder, input 1 to the Subtractor, input 2 to the AND, and input 3 to the OR. Add a 2-bit input Pin, labeled “Operation".

Finally, add an output pin (the circle on the toolbar), configure it to 8 bits, and labeled “Result". Connect the output pin to the output of the mux. Whew! You have built an ALU.

Your ALU should look something like the one in Figure 1.

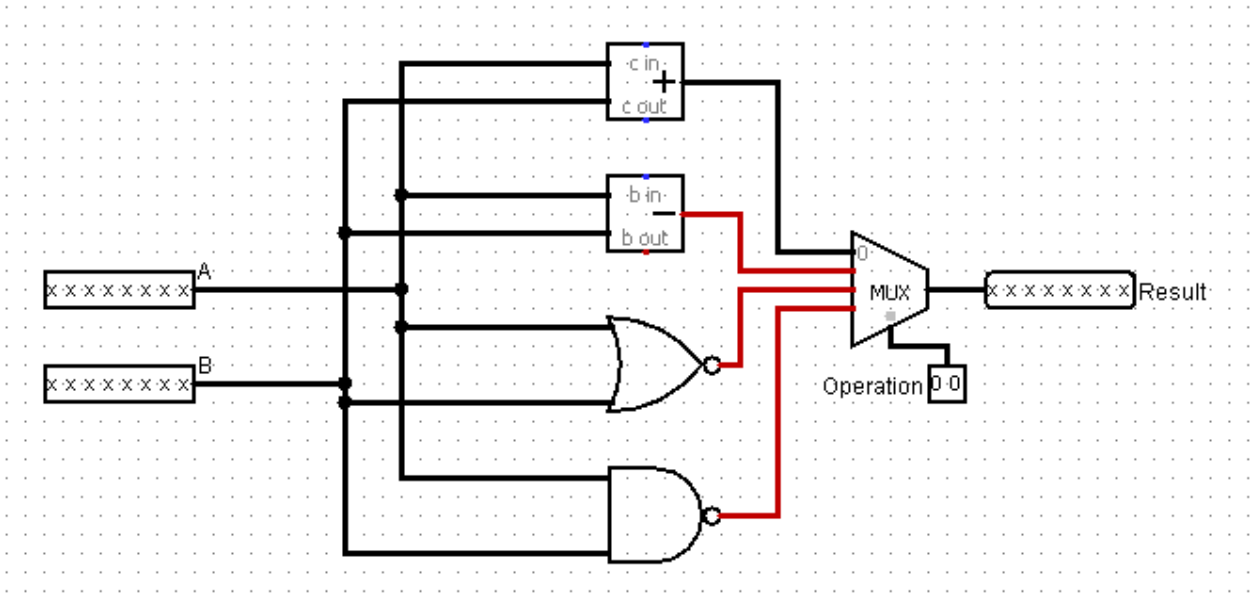

Figure 1: Internal view of ALU.

Step 9: Go back to the main circuit diagram by double clicking on “main" in the project pane (top left pane). Add the ALU subcircuit to the main diagram. For our simple instruction set, the ALU's inputs come only from the A and B registers. So, connect the Q signal of A and B to the A and B inputs of the ALU. We also need to connect the ALU's output to the registers. Notice that the registers can be written from the immediate of the set instruction, or the ALU instructions. So, we need a mux between the output of the ALU and the inputs of A and B. Add a 2:1 8-bit mux. Let's call this mux, SetMux. Connect input 0 of SetMux to the output of the ALU. For now, leave input 1 unconnected. Next, connect the mux's output to the D input of A and B. (Hint: You can change the orientation of the Mux to make your drawing easier to understand.)

Step 10: We are nearly finished with the datapath. Our next task is to add the control. For this purpose, we need to get access to individual elds in an instruction. Add a subcircuit called “Fields". The Fields subcircuit will contain a “splitter" that separates signals. The input to Fields is 16 bits, and there are three outputs: 4-bit opcode, 1-bit register, and 8-bit immediate. You can do this by configuring the splitter with Bit Width In of 16, and Fan Out of 3. Configure the splitter bits to select the bits to output of the three fields of the instruction (opcode, register and immediate). Add a 16-bit input pin and connect it to the 16-bit input of the splitter. Add three output pins (4 bits, 1 bit and 8 bits) and connect to the splitter. Label the input pin “Instruction" and the output pins “Opcode", “Register" and “Immediate". (Hint: You can change the orientation of the splitter to make your drawing easier to understand.)

Your Fields subcircuit should be similar to the drawing in Figure 2.

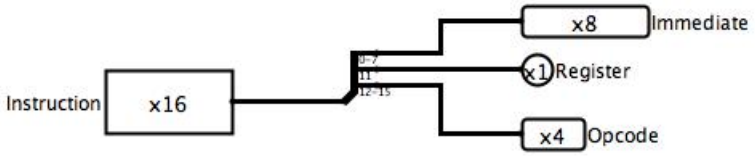

Figure 2: Internal view of Fields.

Step 11: Now, we need to build the control unit, a.k.a, Decoder. The Decoder is nothing more than a combinational circuit. It takes as an input the opcode from the instruction to output the control signals. The control signals are the operation signals for the ALU and the select signals for SetMux. You can draw a truth table and then minimize the table with pencil and paper. Better yet, Logisim includes a tool “Combinational Analysis" to help. Look under the drop-down menu Window -> Combinational Analysis and select the analysis tool. With this tool, you can specify the truth table and then press a button “Build Circuit" to get the optimized circuit. So, go ahead and try doing that. You'll need to specify the 4 bits of opcode input as individual inputs (e.g., Opcode3, Opcode2 and so forth) and the individual outputs (e.g., Operation1 and Operation0 for the ALU, and Select for the SetMux).

Step 12: Once you have the truth table ready, press “Build Circuit". Give the subcircuit a name, “Decoder", when prompted. Go to the Decoder subcircuit -- we need to make a couple changes. Let's group the pins into multi-bit pins. For instance, you'll see four pins for Opcode3 to Opcode0. Create a single 4-bit input pin, Opcode, and use a splitter to connect it to the individual signals. Remove the single bit pins when you are done. Do this for the ALU operation output as well.

When you are done with the Decoder, it should appear similar to Figure 3.

Step 13: Whew. You've come along way! We're now ready to finish wiring everything on the main circuit. Make sure you have the following components on the main circuit: ROM, Fields, Decoder, ALU, A and B registers, PC register, an Adder for PC, and a mux (SetMux).

Step 14: Connect the Immediate output of Fields to the second input of SetMux. This is the second possible location for a value to write to the registers.

Step 15: Connect the Opcode output of Fields to the decoder. Connect the Operation output of the Decoder to the ALU. Connect the Select output of the Decoder to the SetMux. At this point, everything should be connected, except the clock and the Enable signals for registers A and B.

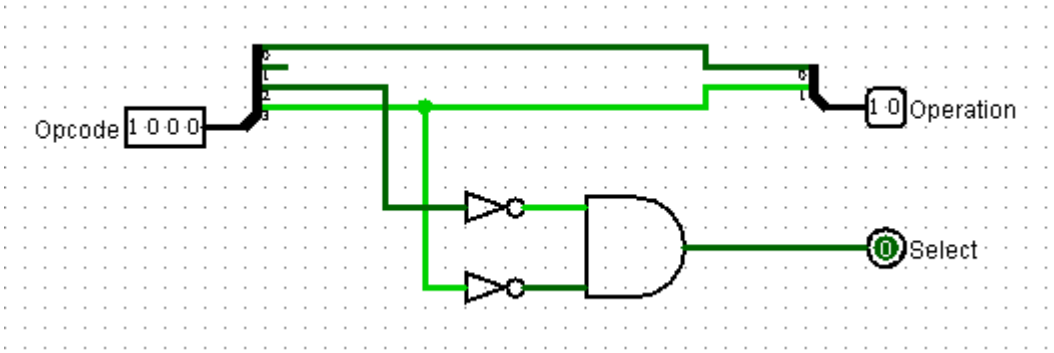

Figure 3: Internal view of Decoder. Currently showing NOR being decoded.

Step 16: Notice that every instruction writes a register. So, we need to set Enable for A or B on every clock cycle. When the register field is 0, we should enable the A register and disable the B register. When the register field is 1, we should enable the B register and disable the A register. Thus, we have two boolean functions -- one for A and one for B. The functions are: A <- $r and B <- $r, assuming $r is the register from the instruction (i.e., the Register output of Fields). Add the necessary NOT gate to your circuit and wire the enable inputs of the registers.

Step 17: OK, one more thing to do! Under Wiring, select a Clock. Connect the clock to all registers|PC, A and B. Assuming you've done everything correctly, you should be ready to test the processor!

When you are done, the main circuit should appear similar to Figure 17.

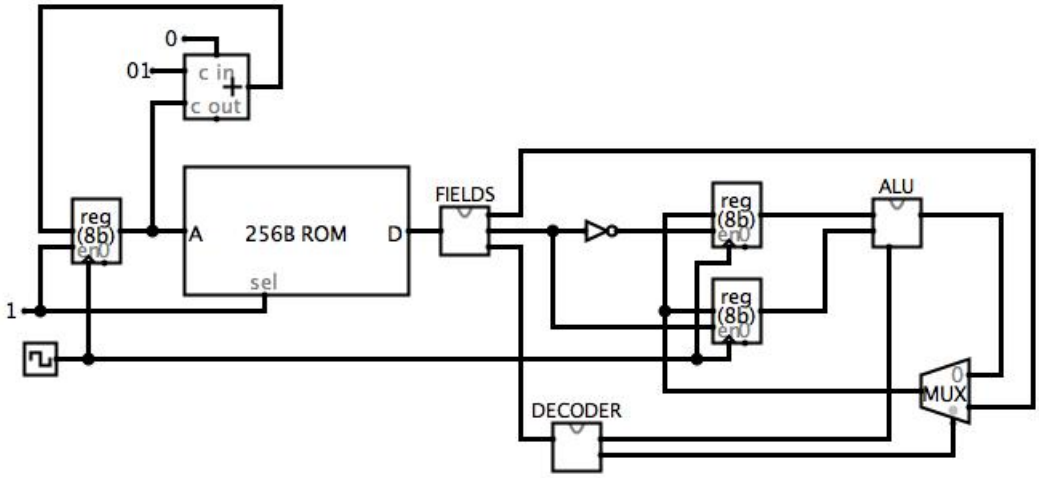

Figure 4: Main diagram of overall design.

Step 18: Use the “hand" to poke at the ROM. Look at the attributes and select “(click to edit)" for the Data Contents. Let's try a set A,1 instruction. This instruction has the value 0x0001. Put the value 0x0001 in the rst location of the ROM (address 00, upper left corner). Try putting the value 0x0802 in the second location of the ROM. Close the data window. Reset the simulation with Simulate!Reset. Now, simulate the circuit by clicking on the clock box with the Hand tool. You should see the values of the A and B registers change to 1 and 2. Congratulations! Write a few more instructions and add to the ROM to test add, subtract, nor and nand.

Some Notes

If you feel confident, you can build the circuit directly without following the steps above. Here are some notes that may help.

The enable control signals for A and B can be tied directly to the $r bit in the instruction. The enable control signals are set as A <- ~$r and B <- $r. Either A or B are enabled on every clock cycle since each instruction has a destination register.

Fields can be implemented simply as a wire splitter. I like to put it inside a subcircuit to hide the splitter, but that is not really necessary for this simple case.

The ROM is configured with 8-bit address and 16-bit data width for the 16-bit instructions. The Select signal should be set.

ALU is implemented as components from Logisim with a 4:1 mux. I like to hide the ALU in a subcircuit. For my example, I used 00 for add, 01 for sub, 10 for NOR and, 11 for NAND.

There is a Mux (SetMux) on the input to the registers. This Mux will need to select either the Immediate field (for set), or the ALU output for all other instructions.

Submission

Upload your Logisim circuit to CourseWeb. Name the le lab11-<Your Pitt Mail ID>.circ (for example, lab11-xyz123.circ).