CS61C Spring 2016 Project 3-1: ALU and Regfile

TAs: Rebecca Herman, William Huang, Howard Mao

Due Tuesday, March 15th, 2016 @ 11:59 PM

Updates and Clarifications

- Nothing yet!

IMPORTANT INFO - PLEASE READ

- You are allowed to use any of Logisim's built-in blocks for all parts of this project.
- **Save often.** Logism can be buggy and the last thing you want is to lose some of your hard work. There are students every semester who have had to start over large chunks of their projects due to this.
- Approach this project like you would any coding assignment: construct it piece by piece and test each component early and often!
- Sample tests for a completed ALU and Regfile have been included in proj3-starter - just run the Makefile `make p1`. We recommend running the sample tests locally, but they only work with **python 2.7**. As always, keep in mind that these tests are NOT comprehensive and you will need to do further testing on your own.
- **MAKE SURE TO CHECK YOUR CIRCUITS WITH THE GIVEN HARNESSES TO SEE IF THEY FIT! YOU WILL FAIL ALL OUR TESTS IF THEY DO NOT.**
  (This also means that you should not be moving around given **inputs** and **outputs** in the circuits).
- Because the files you are working on are not plain code and circuit schematics, they can't really be merged. **DO NOT WORK ON THE SAME FILE IN TWO PLACES AND TRY TO MERGE THEM. YOU WILL NOT BE ABLE TO MERGE THEM AND YOU WILL BE SAD.**

Overview

In this project you will be using Logisim to implement a simple 32-bit two-cycle processor. Throughout the implementation of this project, we'll be making design choices that make it compatible with machine code outputs from MARS and your Project 2! When you're done, you'll be able to run most, but not all, instances of MIPS code through your assembler and linker, and then on your very own CPU! We have left out some functionality to make the project easier.

In part I, you will make the Regfile and ALU.

0) Obtaining the Files

Similarly to project 2, we will be distributing the project files through Bitbucket. You can look back on the step 0 for project 2 for more specific steps. However, make sure that you are using your newly created project 3 repository! The repository that contains the starter code is named `proj3-starter`.

An abridged version of the commands is reproduced below:

```
cd ~                            # Make sure you are outside of any existing repositories (eg. ~/work)
git clone https://bitbucket.org/mybitbucketusername/proj3-xx.git
cd proj3-xx
git remote add proj3-starter https://github.com/cs61c-spring2016/proj3-starter.git
git fetch proj3-starter
git checkout master
```

1) Register File

Your task is to implement all 32 registers promised by the MIPS ISA. Your regfile should be able to write to or read from any register specified in a given MIPS instruction without affecting any other registers, with one notable exception: your regfile should NOT write to $0, even if an instruction should try. Remember that the Zero Register should ALWAYS have the value 0x0.

You are provided with the skeleton of a register file in `regfile.circ`. The register file circuit has six inputs:

INPUT NAME	BIT WIDTH	DESCRIPTION
Clock	1	Input providing the clock. This signal can be sent into subcircuits or attached directly to the clock inputs of memory units in Logisim, but should not otherwise be gated (i.e., do not invert it, do not "and" it with anything, etc.).
Write Enable	1	Determines whether data is written on the next rising edge of the clock.
Read Register 1	5	Determines which register's value is sent to the Read Data 1 output, see below.

INPUT NAME	BIT WIDTH	DESCRIPTION
Read Register 2	5	Determines which register's value is sent to the Read Data 2 output, see below.
Write Register	5	Determines which register to set to the value of Write Data on the next rising edge of the clock, assuming that Write Enable is a 1.
Write Data	32	Determines what data to write to the register identified by the Write Register input on the next rising edge of the clock, assuming that Write Enable is 1.

The register file also has the following six outputs:

OUTPUT NAME	BIT WIDTH	DESCRIPTION
Read Data 1	32	Driven with the value of the register identified by the Read Register 1 input.
Read Data 2	32	Driven with the value of the register identified by the Read Register 2 input.
$s0 Value	32	Always driven with the value of $s0 (This is a DEBUG/TEST output.)
$s1 Value	32	Always driven with the value of $s1 (This is a DEBUG/TEST output.)
$s2 Value	32	Always driven with the value of $s2 (This is a DEBUG/TEST output.)
$ra Value	32	Always driven with the value of $ra (This is a DEBUG/TEST output.)
$sp Value	32	Always driven with the value of $sp (This is a DEBUG/TEST output.)

The outputs $s0-$s2, $ra, and $sp are present because those registers are more special than the others - they are for testing and debugging purposes, and will be used in the autograder tests! If you were implementing a real regfile, you would omit those outputs. In our case, be sure they are included correctly! If they are not, we won't be able to grade your submission (and you'll get a zero. :( ).

You can make any modifications to `regfile.circ` you want, but the outputs must obey the behavior specified above. In addition, your `regfile.circ` that you submit *must* fit into the `regfile-harness.circ` file we have provided for you. This means that you should take care to not reorder inputs or outputs. If you need more space, however, you can move them around if you are careful and maintain their relative positioning to each other. To verify your changes didn't break anything, simply open `regfile-harness.circ` and ensure there are no errors and the circuit functions well. We will be using a similar file to test your register file for grading, so you should download a fresh copy of `regfile-harness.circ` and make sure your `regfile.circ` is cleanly loaded before submitting.

2) Arithmetic Logic Unit (ALU)

Your second task is to create an ALU that supports all the operations needed by the instructions in our ISA (which is described in further detail in the next section). The ISA mostly consists of instructions from the MIPS ISA, **with one extra instruction** Fortunately, you don't have to try to figure out all the required operations as we provide the list to you.

We have provided a skeleton of an ALU for you in `alu.circ`. It has three inputs:

INPUT NAME	BIT WIDTH	DESCRIPTION
X	32	Data to use for X in the ALU operation.
Y	32	Data to use for Y in the ALU operation.
Switch	4	Selects what operation the ALU should perform (see the list of operations with corresponding switch values below).

... and three outputs:

OUTPUT NAME	BIT WIDTH	DESCRIPTION
Signed Overflow	1	High iff the operation was an add or sub and there was signed overflow.
Equal	1	High iff the two inputs X and Y are equal.
Result	32	Result of the ALU Operation.

And as previously promised, here is the list of operations that you need to implement (along with their associated Switch values):

SWITCH VALUE	INSTRUCTION
0	sll: Result = Y << X
1	srl: Result = Y >>> X
2	sra: Result = Y >> X
5	add: Result = X + Y (Set Signed Overflow if overflow)
6	addu: Result = X + Y
7	sub: Result = X - Y (Set Signed Overflow if overflow)
8	subu: Result = X - Y
9	and: Result = X & Y

SWITCH VALUE	INSTRUCTION
10	or: Result = X | Y
11	slt: Result = (X < Y) ? 1 : 0 Signed
12	sltu: Result = (X < Y) ? 1 : 0 Unsigned

Some additional things to keep in mind:

- The output `Equal`, which is true iff X and Y are equal, must always output the correct comparison result regardless of the `Switch` value.
- The output `Signed Overflow` should only be true if the operation being performed is an add or a sub and the computation results in signed overflow. For a refresher on signed overflow follow the link here.

Note: Your ALU must be able to fit in the provided harness `alu_harness.circ` Follow the same instructions as the register file regarding rearranging inputs and outputs of the ALU. In particular, you should ensure that your ALU is correctly loaded by a fresh copy of `alu-harness.circ` before you submit.

Instruction Set Architecture (ISA)

Your CPU will support the instructions listed below. In all of the instructions you recognize from MIPS, the instruction format, opcode, funct, and register numbers should be taken directly from your greensheet. Notice that not all of the native MIPS functions are listed in the CORE INSRUCTION SET section of the greensheet - you may also need to look in the ARITHMETIC CORE INSTRUCTION SET section, and in the OPCODES, BASE CONVERSION, ASCII SYMBOLS table on the back! There is also one instruction that is foreign to MIPS: `swinc`. Specifications for this new instruction will follow in part II.

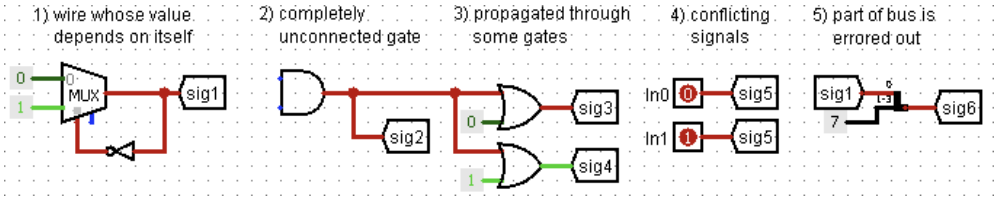
INSTRUCTION	FORMAT
Add	add $rd, $rs, $rt
Add Immediate	addi $rt, $rs, immediate
Add Immediate Unsigned	addiu $rt, $rs, immediate
Add Unsigned	addu $rd, $rs, $rt
And	and $rd, $rs, $rt
And Immediate	andi $rt, $rs, immediate
Branch on Equal	beq $rs, $rt, label
Branch on Not Equal	bne $rs, $rt, label
Jump	j label
Jump and Link	jal label
Jump Register	jr $rs
Load Upper Immediate	lui $rt, immediate
Load Word	lw $rt, offset($rs)
Or	or $rd, $rs, $rt
Or Immediate	ori $rt, $rs, immediate
Set Less Than	slt $rd, $rs, $rt
Set Less Than Immediate	slti $rt, $rs, immediate
Set Less Than Immediate Unsigned	sltiu $rt, $rs, immediate
Set Less Than Unsigned	sltu $rd, $rs, $rt
Shift Left Logical	sll $rd, $rt, shamt
Shift Right Arithmetic	sra $rd, $rt, shamt
Shift Right Logical	srl $rd, $rt, shamt
Sub	sub $rd, $rs, $rt
Sub Unsigned	subu $rd, $rs, $rt
Store Word	sw $rt, offset($rs)
Store Word and Increment **NEW!**	swinc $rt, offset($rs)

Logisim Notes

If you are having trouble with Logisim, ***RESTART IT and RELOAD your circuit!*** Don't waste your time chasing a bug that is not your fault. However, if restarting doesn't solve the problem, it is more likely that the bug is a flaw in your project. Please post to Piazza about any crazy bugs that you find and we will investigate.

Things to Look Out For

- Do **NOT** gate the clock! This is very bad design practice when making real circuits, so we will discourage you from doing this by heavily penalizing your project if you gate your clock.
- **BE CAREFUL with copying and pasting from different Logisim windows.** Logisim has been known to have trouble with this in the past.
- When you import another file (Project --> Load Library --> Logisim Library...), it will appear as a folder in the left-hand viewing pane. The skeleton files should have already imported necessary files.
- Changing attributes *before* placing a component changes the default settings for that component. So if you are about to place many 32-bit pins, this might be desirable. If you only want to change that particular component, place it first before changing the attributes.
- When you change the inputs & outputs of a sub-circuit that you have already placed in `main`, Logisim will automatically add/remove the ports when you return to `main` and this sometimes shifts the block itself. If there were wires attached, Logisim will do its automatic moving of these as well, which can be extremely dumb in some cases. Before you change the inputs and outputs of a block, it can sometimes be easier to first disconnect all wires from it.
- Error signals (red wires) are obviously bad, but they tend to appear in complicated wiring jobs such as the one you will be implementing here. It's good to be aware of the common causes while debugging:



Logisim's Combinational Analysis Feature

Logisim offers some functionality for automating circuit implementation given a truth table, or vice versa. Though not disallowed (enforcing such a requirement is impractical), use of this feature is discouraged. Remember that you will not be allowed to have a laptop running Logisim on the final.

Testing

For part 1, we have provided you with a Makefile for running tests in the project directory as well as a few test files in `tests`. Running `make p1` will copy your alu and regfile into the tests directory and run two ALU tests and two Regfile test. These tests drop in your work into a very slightly modified version of the harness and run it with a small set of inputs. The output is then compared against the provided reference which came from running the staff solution. Keep in mind, as always, that these tests are not comprehensive, so take a look at how `ALU-addu.circ` and `regfile-read_write.circ` are created to see how you can make your own tests. Basically, you'll want to come up with the values to put inside the different memory units to exercise different behaviors of the ALU and RegFile.

Note: the autograder only works with python 2.7, so it may be easier to run it remotely off of the `hive*` servers if you haven't set up your python environments.

Submission

There are **two** steps required to submit proj3-1. Failure to perform both steps will result in loss of credit:

1. First, you must submit using the standard unix submit program on the instructional servers. This assumes that you followed the earlier instructions and did all of your work inside of your `git` repository. To submit, follow these instructions after logging into your -XX class account:

   ```
   cd ~/proj3-XX                          # Or where your shared git repo is
   submit proj3-1
   ```

   Once you type `submit proj3-1`, follow the prompts generated by the submission system. It will tell you when your submission has been successful and you can confirm this by looking at the output of `glookup -t`.

2. Additionally, you must submit proj3-1 to your **shared** Bitbucket repository:

   ```
   cd ~/proj3-XX                          # Or where your shared git repo is
   git add -u
   git commit -m "project 3-1 submission"
   git tag "proj3-1-sub"                     # The tag MUST be "proj3-1-sub". Failure to do so will result in loss of credit.
   git push origin proj3-1-sub               # This tells git to push the commit tagged proj3-1-sub
   ```

Resubmitting

If you need to re-submit, you can follow the same set of steps that you would if you were submitting for the first time, but you will need to use the `-f` flag to tag and push to Bitbucket:

```
# Do everything as above until you get to tagging
git tag -f "proj3-1-sub"
git push -f origin proj3-1-sub
```

Note that in general, force pushes should be used with caution. They will overwrite your remote repository with information from your local copy. As long as you have not damaged your local copy in any way, this will be fine.

Deliverables

```
regfile.circ
alu.circ
```

We will be using our own versions of the `*-harness.circ` files, so you do not need to submit those. In addition, you should not depend on any changes you make to those files.

Grading

This project will be graded in large part by an autograder. If you would like a regrade, we will give you a chance to request one, but we will also automatically deduct 5% from your final proj3-1 grade, unless it is an error with our test cases.