CS/COE 0447 Project 2 - Spring 2015 - Project Information

Project is assigned March 23 and due April 17 by 11:59 PM. Because we must turn in grades shortly after finals week, the maximum late date is April 20 at 11:59 PM. After this date, projects will not be accepted.

For this project, you need Logisim. You can get the logic simulator from its home web site: http://www.cburch.com/logisim/index.html. We have tested my implementation of the project with Logisim version 2.7.1. To download:

1. Follow the Dowload link on the Logisim web site to SourceForge.net.
2. Next, select View all files.
3. Pick the folder for your platform.
4. Select version to download the simulator.

Be sure to start on the project early. It has a learning curve with Logisim and logic design. The project follows the lecture slides and book closely. You can use many of the diagrams in the slides and book to help with your implementation.

The project description, assembler and assembler manual are available on Courseweb. Below are some example programs.

- increment.asm Tests add
- load.asm Tests loading from memory
- addshift.asm More complete add, shift, branch test
- loadstore.asm Checks storing and loading
- function.asm Tests jal and jr with load and store

Hints and Updates

Do you have a question? Caught a typo or problem in the description? Please let your instructor know by e-mail, and we'll try to address the issue and put the reply below. We may not post all answers, but we will list the ones that are useful for everyone.

- **Halt missing in test cases:** Put a halt instruction at the end of all the example assembly files to make the processor halt at the last instruction, if you wish.

- **Decompose your circuits!** We recommend that you do not create a single "super ALU" that does every function in one subcircuit. Instead, we suggest that you use a few different subcircuits, including the four function ALU, a shifter subcircuit (selects among left and right shift), a branch comparison subcircuit, and a hex display subcircuit.

- **Use multiple control units:** There are likely too many control signals to create a single control unit. Instead, group the control signals into reasonable partitions, such as all control signals for branches and jumps, or all control for arithmetic operations. Turn these into separate subcircuit control units. My prototype implementation of the project has six subcircuits for control. The idea is like programming: break the task into smaller chunks that can be done by procedures (except, this time it's subcircuits).

- **K-maps? Who needs that?** Logisim has a nifty tool call Combinational Analysis (look under the drop-down menu "Window" for "Combinational Analysis"). In this tool, you create a truth table, and then you press a button, and by magic, Logisim creates a simplified combinational circuit for the truth table! Nifty! And a HUGE time saver. You'll want to use the Combinational Analysis to create your control units. It's a huge help. Another trick: After you've made the circuit, group individual pins into a multi-bit pin. For example, the Opcode bits can be grouped into a single 4-bit input to the control unit. This will make wiring simpler.

- **Use Logisim's library:** Logisim has a lot of components that you can use directly. Be sure to use them! You don't have to build the ALU from the ground up, i.e., you don't need to create a 16-bit ripple-carry adder, as Logisim already offers an adder that can be configured to add 16 bit quantities. Similarly, there's a shifter, sign and zero extenders, etc.

- **Implementing Halt instruction:** The project description is vague about how to halt the processor. The program counter has an enable signal (it is a register). This enable signal should be set to 1 for all instructions, except halt. When a halt instruction is encountered, set the enable for the program counter to 0. This can be done simply with control: Check for the halt opcode, and then generate enable=0 for this case. This has the effect that the program counter won't be updated any further, and hence, the halt instruction will continue to be executed (keeping enable=0), which stops the processor from fetching any new instructions! Neat!

- **Where to start?** The hardest part is to start the project. It seems daunting! If you do it a little bit at a time, it is much simpler. We recommend that you start with the ALU. Make a simple four function ALU to support sub, add, and, and nor. Next, make the register file. Connect the register file to the ALU (read and write). Add some pins for the control signals. Now, poke at the pins to make the ALU perform operations on the register values. You can also poke at the registers to set them to some values so you can operate on numbers other than 0 (trying only 0s doesn't really help test the circuit!). After you've got this part, then start to add the instruction memory and a simple decoder. Implement halt and the put instruction. These will be useful for debugging. Now, continue with more instructions, like the shifts, add immediate, etc. Then it's on to branches, loads and stores, and finally jumps. Whew! You're done!!

Test Cases

Here is a collection of simple tests that check the operation of a few critical instructions. You will probably want to write some more tests, particularly for any instructions not tested by these cases. We will use some of these in grading the project, so they are good tests to make sure work.

- addi.asm Tests addi
- put.asm Tests put
- put2.asm Another put test of individual hex digits
- addui.asm Tests addui
- sllv.asm Tests sllv by itself
- sll.asm Tests sll and sllv
- bz.asm Tests branch zero

- bp.asm Tests branch positive
- bn.asm Tests branch negative
- srl.asm Tests sll and srl
- jal.asm Tests jal by itself
- jr.asm Tests jr by itself
- lwsw.asm Tests lw and sw

Important: Put the instruction ROM, data RAM, and Hex Digits in the main circuit. We need to work with these components to grade the assignment and this will make it easier.