Carleton University

Department of Systems and Computer

Engineering SYSC 3006 Fall 2016

Computer Organization

Lab #4

Prelab Due: Your Prelab should be done *before* the start of your scheduled lab session. Bring your Prelab with you in hard or soft copy form (e.g. print it or have a word/pdf document ready) for TA inspection during the lab session (no online submission).

In this lab you will:

- Program a revised datapath and extend the design to include the microarchitecture feature of loading instructions from Main Memory (as discussed in lectures).
- Implement and test using Logisim.

Read this document carefully before deciding what to do.

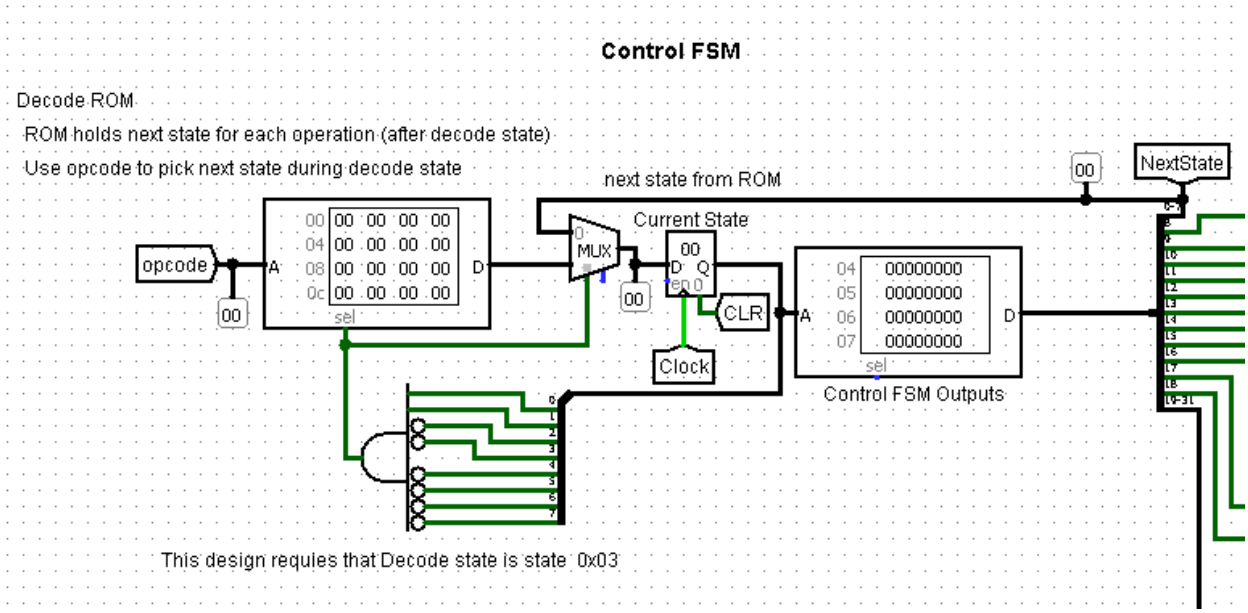
A folder containing 2 Logisim circuits has been included (not needed for the Prelab). **You MUST UNZIP the folder** (i.e. extract all files) before trying to load these circuit files into Logisim. If you do not unzip the files, they may load into Logisim, but will not simulate properly!!!

NOTE: In this lab description, all slide numbers refer to Lectures 6 and 7 (equivalent notes can be found in pages 28-40 of TowardsHardware)

Part 1

Load the supplied Lab4Part1 circuit into Logisim. It is very similar to your solution to Lab 3. Do not modify the components or wiring of this circuit in Part 1. A major difference from Lab 3 is that the FSM has been expanded to allow for fetching an instruction from Main Memory in Part 2. The FSM has been designed for 7 states: states 0 through 6. States 0, 1 and 2 will be used for fetching (in Part 2). The Decode state is state 3 (in both Parts). The three execution states are states 4, 5 and 6 (in both Parts).

The Decode ROM has been introduced into the FSM to deal with the Decode state. If you completed Lab 3 Part 2, you should find the Decode ROM reasonably easy to understand; however, the OneSourceOp circuit is no longer used. The Decode ROM contains the next state that follows the Decode state for each instruction (for example, for the ADD instruction, the next state that follows the Decode state is state 4). The revised circuit is shown below:



The FSM uses the Next State encoded in the FSM as the next state for all FSM transitions except for the transition out of the Decode state. When the state machine is in the Decode state (state 3), it does not use the Next State encoded in the FSM; instead, it uses a value encoded in the Decode ROM. The Decode ROM is addressed using the instruction opcode (from the IR). The Decode ROM must be programmed to contain the correct "Next State" that follows the Decode State for each instruction. For example, the opcode for the ADD instruction is 0x01, and therefore the contents of the Decode ROM word at address 0x01 should be 0x04 (the correct state following the Decode State for the ADD instruction). ☺

The Lab 4 Instruction Register is 32 bits wide (8 hex digits: H_7 H_6 H_5 H_4 H_3 H_2 H_1 H_0), as discussed in class. In the instruction encoding, each reference to a register requires 4 bits (since there are 16 registers) and the opcode is encoded in 8 bits. The 32-bit Instruction Register uses the bit layout from slide 123:

bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
use	Op								Rd				Rsx				Rsy				Not used (always 0) for now!											

Terminology note: *Instructions* are fetched from Main Memory and then decoded and executed over several states; while *operations* are performed by the ALU in one state.

Recall that slide 123 says that the 8-bit opcodes for the instructions that use the ALU are obtained by extending the 4-bit ALU operation with leading 0's. For example, the ADD instruction will have the 8-bit opcode = 0x01, which is obtained by extending the 4-bit ALU ADD operation (0x1) with leading 0's. The complete 32-bit ADD instruction R3 ← R2 + R1 would be encoded in hexadecimal as: 0x01321000.

The MOV (Move) instruction has been introduced as an instruction for copying a register to another register (i.e. using the RY ALU operation).

A NOP instruction is different from all of the other instructions that use the ALU since it does not generate any useful result. Think carefully about how to deal with a NOP instruction in the lab (hint: in this lab it can be resolved at the Decode state).

In the lab: Demonstrate your working solution for the following instructions with all registers initially containing 0x0:

NOT: R9 ← NOT RB

SUB: R8 ← RB – R9

NOP

MOV: RA ← R8

Some tables have been supplied to help with the PreLab (**Lab4PrelabTables.docx**). [PreLab: Fill in the Part 1 Instruction Encoding Table and complete the FSM Output ROM and FSM Decode ROM Tables for Part 1. NOTE: In the supplied circuit, the Control FSM outputs the RegSEL and RegLD signals with the behaviour expected by the Registers RAM. The FSM Output ROM does not use WordR and WordW signals as done in Lab 3 and in class; the RegSEL and RegLD signals (with Logisim's RAM signalling behaviour) are used instead. Do not change any of the existing values that have been pre-entered into the table, except for the "x" values in the hexadecimal encodings. The first 3 states are there only as placeholders in Part 1 and have been designed to have no undesirable effects on the circuit. This allows you to always start the FSM in state 0.]

The resulting FSM Output ROM and FSM Decode ROM values should work for any of the instructions discussed in class that use the ALU (i.e. NOP, ADD, SUB, MOV, AND, OR, XOR and NOT).

Save your circuit as Lab4Part1.circ.

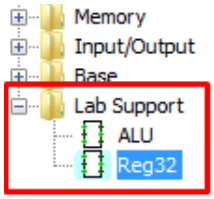
Part 2

Extend your solution to Part 1 to add the ability to load instructions from Main Memory (as discussed in class). You will not need to modify the circuit beyond the description of modifications given below.

The circuit in Part 1 has been set up to simplify the extension. The Decode ROM values used in Part 1 do not require any modification for Part 2. The Part 1 FSM ROM Output Table includes all of the new signals needed in this part (however, many of the values used in Part 1 for these signals won't work in Part 2). To get access to the new bits needed in Part 2, **replace the "NotUsedInPart1" tunnel with a splitter**. ☺ You might find it useful to break out the splitter outputs using tunnels with meaningful names (as done with the FSM outputs used in Part 1). The names of the new signals in the table are the same and have the same function as those used in the lecture slides.

You will need to add the "new" Processor hardware parts discussed in:

- **Slide 128**: connect the IR to the Internal Data Bus, add the IRCE and clock signals (make sure you get the right clock signal ☺), and add the PC address generator with the PCOE control signal.
- **Slide 132**: add the 32-bit constant 1 and the C1OE control signal.
- **Slide 137**: add the Interconnection Bus Interface as shown.
    - **Note**: use Reg32 registers to implement MDR and MAR.
- **Slide 149**: add the ADD operation and its AADD control signal.



You will need a Main Memory component, similar to that shown in slide 120. To simplify this part, a Main Memory component and surrounding circuit for use in Part 2 has already been added … do not modify the given Main Memory circuit . Build your solution to be compatible with the existing names used in the tunnels. For simplicity, a 64 x 32-bit Main Memory has been included (i.e. the lower 6 address bits on the Interconnection Address Bus are used to select a word, while the upper 26 bits must all be 0 to be a valid memory address). Also note that slide 120 shows the Bank of Memory Words that was developed in class, while Logisim's RAM component has a few differences (as you discovered in Lab 3!). The supplied circuit already accounts for the differences!

[**PreLab**: Complete the Part 2 FSM Output ROM Table and the Part 2 Main Memory Table.]

In the lab: Save your solution as Lab4Part2.circ.

Good Luck!