Carleton University

Department of Systems and Computer Engineering

SYSC 3006 (Computer Organization) summer 2020

Lab / Assignment 4

Prerequisites

Lab / Assignment 3, series 5, series 6 part 1 and all related materials posted on cuLearn.

Goal

- Program the microarchitecture (from lab 3) to add new execution states for different instruction.
- Implement the instructions shown in Table 1 below, Reprogram Decode ROM to have different execution path for each different form.
- Implement and test using Logisim.

Introduction

Until this lab, we have assumed that all instructions could be managed by (at most) 3 generic execution states. The Decode ROM introduced in Lab-3 provided additional flexibility by allowing the NOP instruction to be implemented with no execution states.

An instruction can be added to the microarchitecture quite easily using the following steps:

1. Design the execution state of the instruction. This requires designing how the instruction will be executed as a sequence of steps utilizing the components in the datapath (i.e. the registers and ALU). (Recall: we did this in lectures for the initial set of operations.) Your design must be concerned with issues such as: how the required behavior can be realized, what operations (if any) the ALU should preform, and how/when the temporary registers should be used to avoid conflicts in the use of the Internal Data Bus.
2. Realize the design as a sequence of Control FSM states using the provided Instruction Execution States Table. The FSM outputs in each state will realize a step in your designed execution of the instruction. In order to fill in the Next State fields of the Table, you must decide which Control FSM Output ROM words will be used to implement each state. Select some unused words in the Control FSM Output ROM to be used to hold the encoding of the execution states. “Unused words” are words that do not currently hold the encodings of any FSM states. The appropriate addresses of the selected words should be entered in the appropriate “Next State” fields of the Table. Program the resulting states in the table into the corresponding Control FSM Output ROM words.
3. All instructions use the same fetch and decode states, but the first execution state for each instruction is selected by the Decode ROM (Figure 1). Using the new instruction’s opcode as the

Decode ROM address, program the Decode ROM to point to the instruction's first execution state (i.e. the one that you programmed in Step 2).

The extended system is now ready for testing!

The main goal is to implement the instructions shown in Table 1: (Note: the instruction associated with the RY operation is named "MOV" (MOVE) and uses Rsy (not Rsx) as a source operand.)

Instruction	Effect	OpCode (hex)
NOP	Do nothing	00
ADD	Rd ← [Rsx] + [Rsy]	01
SUB	Rd ← [Rsx] – [Rsy]	02
MOV	Rd ← [Rsy]	03
AND	Rd ← [Rsx] AND [Rsy]	04
OR	Rd ← [Rsx] OR [Rsy]	05
XOR	Rd ← [Rsx] XOR [Rsy]	06
NOT	Rd ← invert([Rsy])	07
NEG	Rd ← – ([Rd])	17

Table 1 Lab-4 instructions

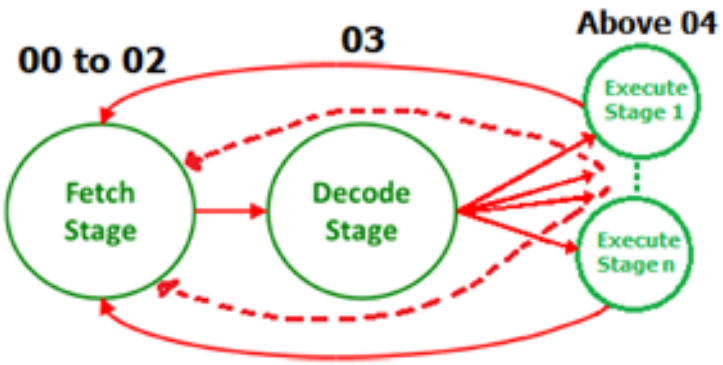

Figure 1 Same fetch and decode states for all instructions, but different execution path for different instruction form

Logisim files setup and components information

A folder containing 2 Logisim circuits has been included. **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!!!

The distributed Lab-4 circuit is a working implementation of the microarchitecture discussed in lectures, with the variations from Lab-3 Part 2. In this lab you must program the ROMs to perform some instructions and demonstrate their execution.

Load the supplied Lab-4.circ circuit into Logisim and use it for all part. Do not change anything in the circuit, you just need program the memories.

Your Assignment

Use the included editable .docx file (MS-Word) for your answers, then save it as PDF file before submitting it. Here we refer to tables to be filled in, they are not shown here but they do in the assignment .docx file.

Part 1 – [0.75-mark/5]

Program the microarchitecture for the ADD through NOT instructions in Table 1 above (opcodes 0x01 through 0x07). If you followed the directions in Lab-3, the solution to this part should be identical to your solution in Lab-3 Part 2 (i.e. the supporting design tables are the same). If you did not get Lab-3 Part 2 working, this is a good opportunity to complete that part of Lab-3. Note that the Lab-3 version of two-operand instructions share the execution states with three-operands instructions.

1-1 Control FSM Output Table

[0.75-mark] Complete the provided Control FSM Output Table for Part 1 for the Fetch, Decode, and Execution States for opcodes 0x01 (ADD) through 0x07 (NOT).

Part 2 – [0.75-mark/5]

Recall that the Lab-III version of the NOP instruction did not have any execution states. Extend your solution to Part 1 (following the 3 steps process outlined in the introduction above) to include a new implementation of the NOP instruction. This version of the NOP instruction should have **3** execution states, and these execution states must not change the internal state of any components on the Datapath. This new implementation effectively "burns" (i.e. wastes) **3** execution states without performing any useful work.

2.1 - Control FSM Output Table

[0.75-mark] Complete the provided Control FSM Output Table for Part 2 for NOP Instruction Execution 3 States. This table will extend the Control FSM Output Table for Part 1 (same FSM Output ROM).

Part 3 – [2.0-mark/5]

Extend your solution to Part 2 to add the "Negate" (NEG) instruction shown in Table 1. This instruction has only one operand (the destination register) and the instruction calculates the 2's complement of the initial value in the destination register and loads the result back into the destination register. The instruction might be written as:

```
NEG R6          ; R6 is the source and destination register
                ; set the contents of R6 = 2's complement of the initial value of R6
                ; NEG R6 would be encoded as: 0x 1760 0000.
```

(Hint: The opcode of the NEG instruction has been carefully chosen to support the instruction's implementation. Think about the ALU operation represented in the least significant nibble of the opcode …could this be used to help in achieving the "negate" functionality?)

3.1 - Control FSM Output Table

[0.75-mark] Complete the provided Control FSM Output Table for Part 3 for NEG Instruction Execution States. This table will extend the Control FSM Output Table for Part 1 and 2 (same FSM Output ROM).

3.2 -

[0.50-mark] Describe how NEG instruction is executed at each execution state.

3.3 – Decode ROM Table

[0.75-mark] Complete the provided FSM Decode ROM Table to show any entries that must be programmed (for all parts).

Part 4 – Execution test [1.5-mark/5]

Implement your design and execute the Test Program. Validate your instruction designs following these steps:

1. Program the words of the Control FSM Output ROM and Decode ROM with the values from your previous parts completed tables.
2. Make sure the System clock is high (in the 1 state).
3. Make sure all components in the Datapath hold the value 0 (use the "Clear Internal State" button in the upper left corner of the circuit).

4.1 – Instructions Table

[0.75-mark] Complete the provided Main Memory Table to contain the encodings of the Test Program instructions as indicated. Then program the words of this table into the Main Memory. Be sure to include the Main Memory contents exactly as given in the table.

4.2 – Test Results

[0.75-mark] Cycle the System Clock through the execution of your Test program and show your logs here.

Submission deadline

Must be submitted on cuLearn, locate (Assignment 4 submission) and follow instructions. Submission exact deadline (date and time) is displayed clearly within the Assignment 4 submission on cuLearn.

Note: If you have any question please contact your respective group TA (see TA / group information posted on cuLearn) or use Discord class server.

Good Luck