Carleton University

Department of Systems and Computer Engineering

SYSC 3006 (Computer Organization) summer 2020

Lab / Assignment 2

Prerequisites

Lab / Assignment 2 and all materials on cuLearn related to series 1 and 2; and especially materials related to series 3 and 4.

Goal

- Understand and use an ALU circuit similar to one that might be included in a processor;
- Design a primitive datapath (in part I) that requires a unique control FSM for each operation;
- Introduce an encoded operation latch to enable a single control FSM to execute every operation (Part II);
- Implement and test all designs using Logisim.

Introduction

Simple Processing System circuit (part I)

The Simple Processing Circuit (shown in Figure 21) performs ALU operations on memory words and stores the result in a memory word. This circuit requires a unique Control FSM for each operation.

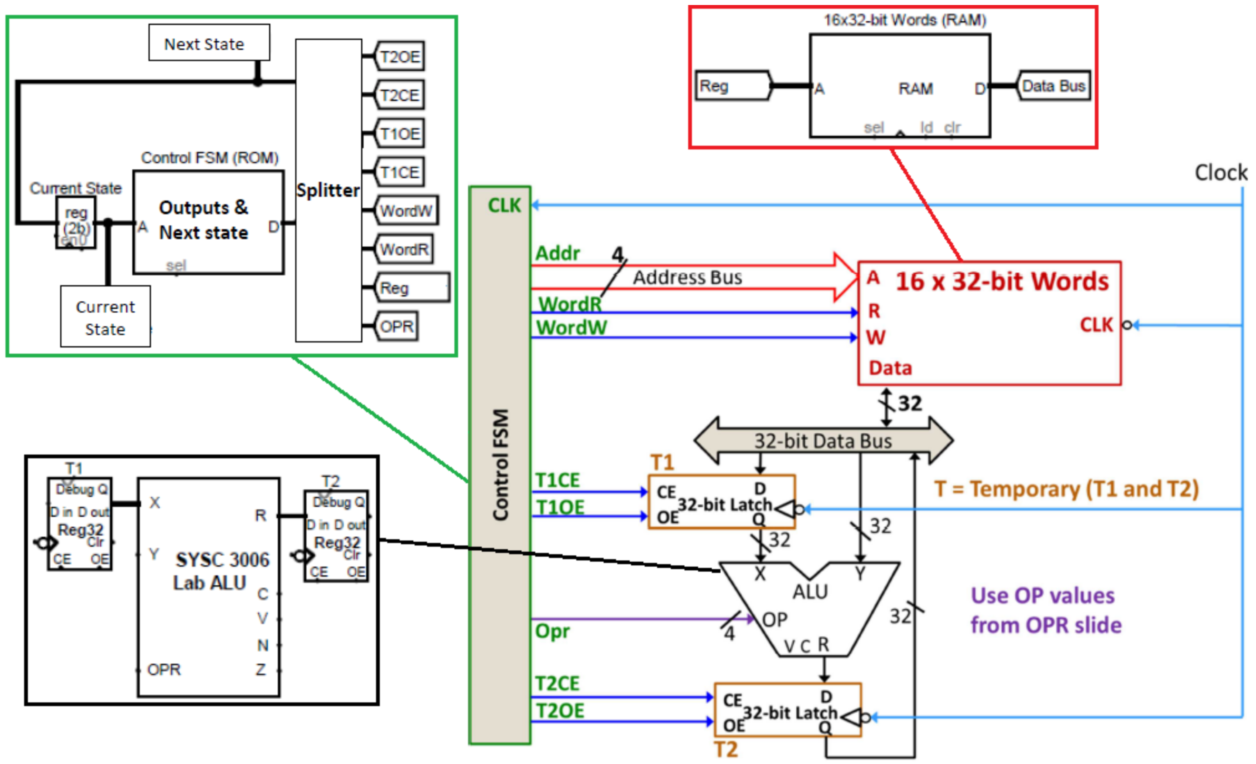


Figure 21: Simple Processing System Circuit

Generalized processing System circuit (Part II)

An encoded operation latch can be added to the circuit to generalize the (Mealy) Control FSM, as shown in Figure 34. The introduction of the latch enables a single Control FSM to execute every operation. When further developing the circuit into the microarchitecture, the memory words and encoded operation latch are renamed to registers and the Instruction Register (IR), respectively. This lab uses the microarchitecture terminology (i.e. registers and Instruction Register).

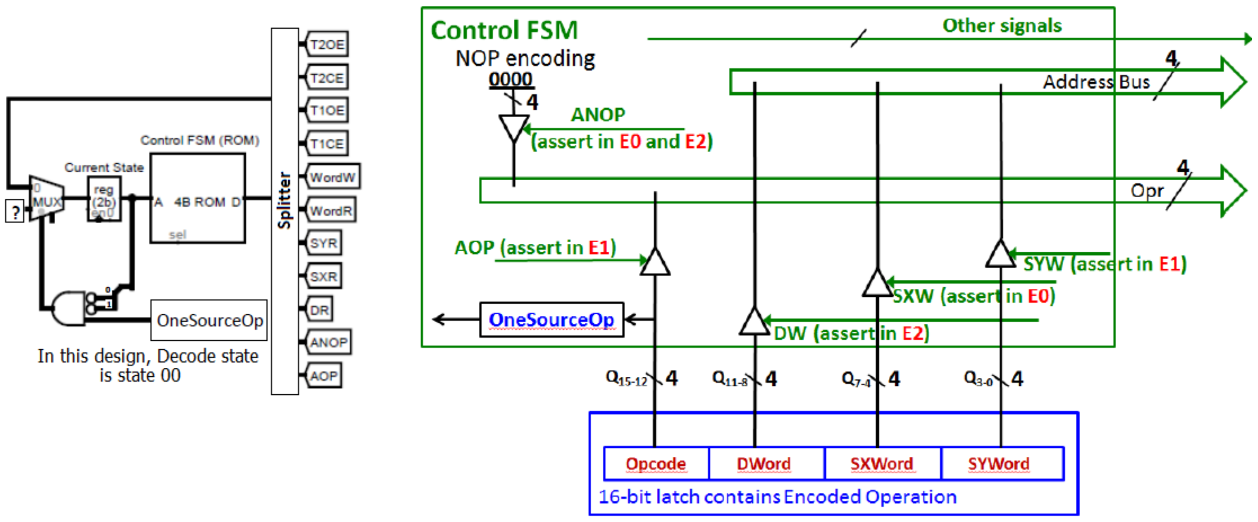
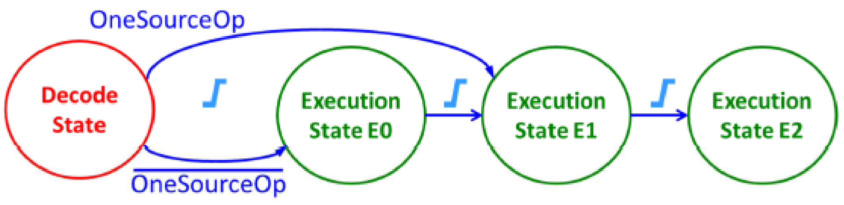


Figure 34: Encoded Operation in Control FSM

As discussed in class, your FSM in Part II will have an additional “Decode” state, thus there are total 4 states (D, E0, E1, E2, or 00, 01, 10, 11 as ROM address). You will have to redesign the FSM state transition logic to take appropriate action (Figure 35).



State	SXW	SYW	DW	AOP	ANOP	Other signals	Next State
Decode	0	0	0	0	1	no active outputs	$\overline{\text{OneSourceOp}} \rightarrow$ E0 OneSourceOp $\rightarrow$ E1
E0	1	0	0	0	1	as before	E1
E1	0	1	0	1	0	as before	E2
E2	0	0	1	0	1	as before	–

Figure 35: Revised Control FSM

The instruction will have the fields of figure 32.

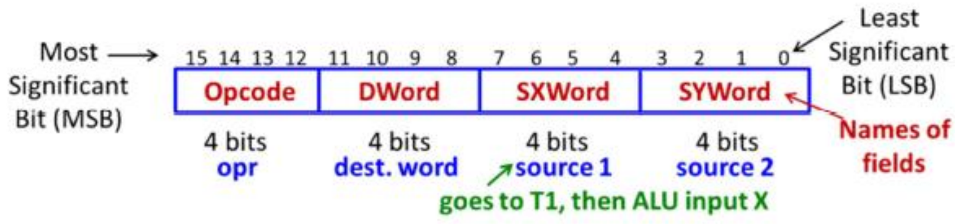


Figure 32: Fields in 12-bit Encoded Operation

To execute an operation, you will have to poke a value into IR, and then poke the Toggle Switch to advance the FSM through the operation.

Logisim files setup and components information

Two Logisim files are supplied (Lab-II_PartI.circ and Lab Support.circ), place both of them in same directory. Load Lab-II_PartI.circ circuit into Logisim. After opening the file, Logisim should looks like figure 1.

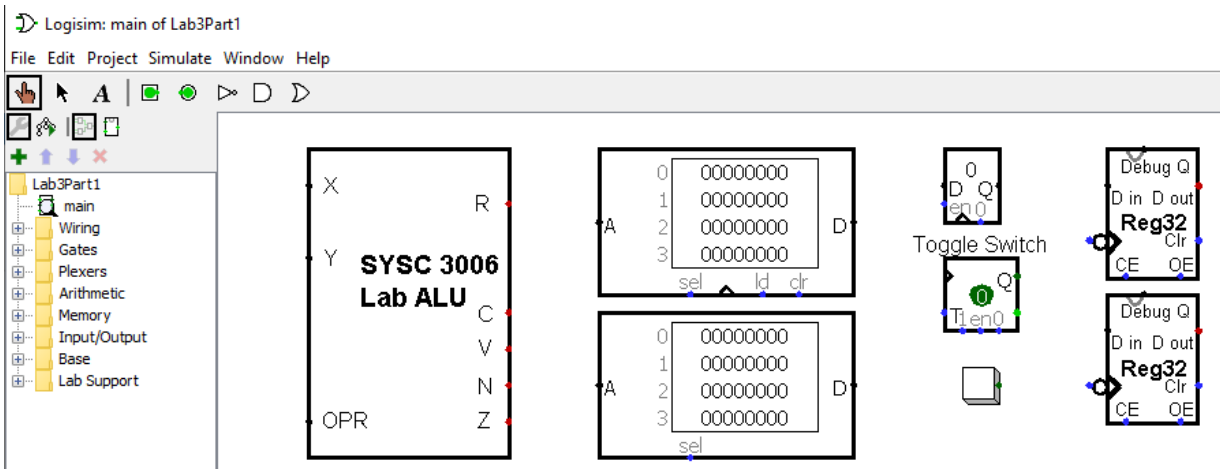
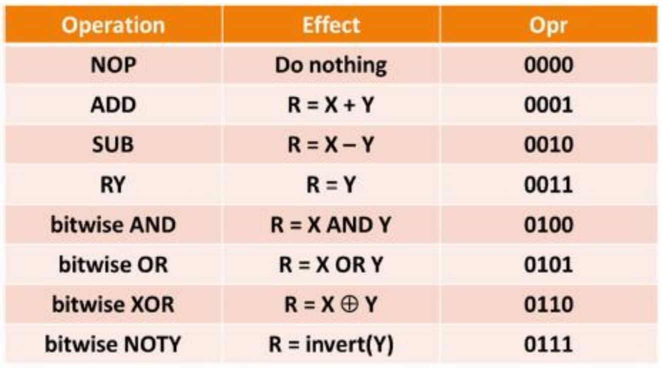


Figure 1: Lab-II_PartI.circ circuit

The file Lab Support.circ provides the internal circuit of the ALU box (can be instantiated in the main). By default Logisim look for this file in same directory as the main file (figure 1). If Logisim did not find the file, then you will need to provide the path to the file location. To check the content of the ALU (from within the main file), Right click your ALU block and select “View ALU”. Then you can return to main by clicking on “main”(on the left top side menu). In same menu, you can expand “Lab Support” to add an instance of the ALU to the main circuit (just in case you mistakenly deleted the ALU instance provided in Lab-II_PartI.circ). The ALU implements operations according to the OPR encoding presented in Figure 20.



Operation	Effect	Opr
NOP	Do nothing	0000
ADD	R = X + Y	0001
SUB	R = X – Y	0010
RY	R = Y	0011
bitwise AND	R = X AND Y	0100
bitwise OR	R = X OR Y	0101
bitwise XOR	R = X ⊕ Y	0110
bitwise NOTY	R = invert(Y)	0111

Figure 20: Subset of 2^m ALU Operations, m = 4

The RAM component

The RAM component in figure 1 has address and data connections (as expected) and is controlled using the SEL and LD inputs. The control provided by these inputs is a bit different than bank of words developed in class! Design and implement some simple logic using only a few logic Gates to interface the WordR and WordW signals from the Control FSM to the SEL and LD signals to implement Read and Write operations.

Reg32 component

The Reg32 component in figure 1 implements a 32-bit instance of the n-bit Data Latch abstraction shown in Figure 10. The names on the Reg32 pins are the same as those used in class, with the exception of "Debug Q". The Reg32 data outputs are controlled by the OE signal (as expected), and therefore the value stored in the component is only visible externally when OE is asserted. Debug Q outputs the currently latched value at all times, and is intended to allow you to monitor the contents with a probe while testing/debugging. Do not use the Debug Q output for any other purpose. It also has a Clear (CLR) input that should be 0 during normal operation. If CLR = 1, then the internal latches are reset to 0. You may find it useful to reset the latches while testing.

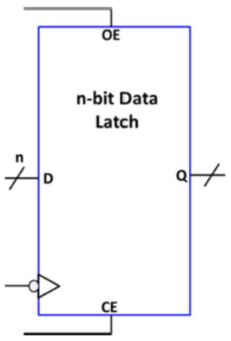


Figure 10: n-bit Data Latch Component

Other components

The Logisim Button (☐) included with the Lab-II_PartI.circ components has been provided to be used for reset (clear) function. Do not use the Button for any other purpose.

The Toggle Switch flip flop in figure 1 is to be used as the system clock.

Your Assignment

All questions below are included in an editable .dox file (MS-Word), use it for your answer, then save it as PDF file before submission.

Part I – Simple Processing System circuit [3-mark/5]

1-1

a) [0.30-mark] In your own words, describe how SEL and LD are used to control Read and Write operations on the RAM:

b) [0.30-mark] Then based on your observations from 1-1 a), design and implement some simple logic using only a few logic Gates to interface the WordR and WordW signals from the Control FSM to the SEL and LD signals to implement Read and Write operations. Show a screenshot of your design here:

1.2 - Circuit wiring [0.30-mark]

Implement the Simple Processing System circuit (Figure 21 in Towards Hardware) using the Logisim components included in the original Lab-II_PartI.circ circuit (i.e. those shown on the canvas of figure 1). You are asked to wire (interconnect) the circuit hardware properly. You must not add or exchange any components EXCEPT for a few simple gates to support interfacing to the SEL and LD pins. i.e., you may add Wires, Tunnels, Splitters, Probes, and Constants from the Logisim Wiring Library, and the permitted gates (for SEL and LD interfacing), but nothing else!
(your Lab2-Part-I.circ must be included with your submission. We need to be able to verify your circuit in order to give you the marks for this part)

Hint: Your implementation may result in many wires criss-crossing the circuit. Use Logisim Tunnels (in the Wiring Library) to simplify the circuit diagram. From the Logisim documentation: "A tunnel acts like a wire that binds points together, but unlike a wire the connection is not explicitly drawn. This is helpful when you need to connect points far apart in the circuit and a network of wires would make the circuit ugly." Always use appropriate names for the tunnels. A tunnelling example is intentionally included in Lab-II_PartI.circ, once familiar with the tunnelling technique; you should delete this example as it is not explicitly part of your circuit.

1.3 – ROM FSMs

Use the ROM-based implementation of the Control FSM (as you did in Lab 1). The state machine should cycle back to its first state after completing each operation. A word in ROM can be used to generate all of the outputs for a given state (i.e. specific bits in each ROM word will correspond to a specific output signals from the FSM). Since each operation requires a unique FSM, the ROM must be loaded with specific values for each operation.

Hint: 2 operand operations will use 3 ROM words, while 1 operand operations will use 2 … sound familiar? Refer to pages 18-25 in Towards the Hardware for various examples of the ROM design process.

1.3 a) [0.30-mark] Fill in a next state table (ROM table) for the following RTL:

R10 ← R1 OR R2

State	OPR (4 bits)	Addr (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State
E0									
E1									
E2									

1.3 b) [0.30-mark] Fill in a next state table (ROM table) for the following RTL:

R11 ← R2 – R10

State	OPR (4 bits)	Addr (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State
E0									
E1									
E2									

1.3 c) [0.30-mark] Fill in a next state table (ROM table) for the following RTL:

R12 ← NOT (R11)

State	OPR (4 bits)	Addr (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State
E0									
E1									
E2									

1.3 d) [0.30-mark] Fill in a next state table (ROM table) for the following RTL:

R13 ← R0 + R12

State	OPR (4 bits)	Addr (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State
E0									
E1									
E2									

1.4

Clear the RAM to Initiate all registers to zero, and initiate R0 to 0x00000001, R1 to 0x1000000F and R2 to 0xF0000000 (Just once before you execute any RTL). Then Implement and execute the 4 RTL from 1.3, one by one in the right sequence as below:

R10 ← R1 OR R2
R11 ← R2 – R10
R12 ← NOT (R11)
R13 ← R0 + R12

You need to implement the next state table for each of them into you Simple Processing System circuit ROM. Execute the first RTL starting at state E0 and finishing at state E3. Do the same for the rest and observe their execution to make sure that there is no errors. Do not forget to correct immediately any error into your tables before continuing, and then repeat from the beginning.

1.4 a) [0.30-mark] After finishing all executions, Log the execution of the sequence on your implementation and show the results here:

1.4 b) [0.30-mark] After the executions of R11 ← R2 – R10, what is the value of R11 (in decimal)?

1.4 c) [0.30-mark] What does the operations R12 ← NOT (R11) and R13 ← R0 + R12 accomplish (think about what does the value obtained in R13 represent in relation to R11 and R12, and why)?

Part II – Generalized processing System circuit [total of 2-mark/5]

2.1 - Circuit wiring [0.50-mark]

Save a copy of your part I (Save as) and name it "Lab2-Part-II". In this part you will introduce a 16-bit Instruction Register (IR) to generalize the FSM (see figure 34 above). Introducing the IR allows the same ROM contents to execute all operations. Use a Logisim Register to implement the IR, and poke values into the register as needed. You will need some tristate buffers in this part.
(your Lab2-Part-II.circ must be included with your submission. We need to be able to verify your circuit in order to give you the marks for this part)

2.2 [0.4-mark]

Complete the following next state table for the Control FSM that executes every instruction

State (ROM) address	AOP	ANOP	DR	SXR	SYR	Word R	Word W	T1CE	T1OE	T2CE	T2OE	Next State	Inst. (Hex)
(D) =													
(E0)=													
(E1)=													
(E2)=													

2.3 [0.4-mark]

Complete the Encoded Instruction Table for same 4 instructions of the Part I of this lab.

Operation	Encoded 16-bit Value (in binary)	Encoded 16-bit Value (in hex)
R10 ← R1 OR R2	0b:	0x
R11 ← R2 – R10	0b:	0x
R12 ← NOT (R11)	0b:	0x
R13 ← R0 + R12	0b:	0x

2.4

Clear the RAM to Initiate all registers to zero, and initiate R0 to 0x00000001, R1 to 0x1000000F and R2 to 0xF0000000 (Just once before you execute any RTL). Then Execute each of the instructions from the table in 2.3 in same sequence (same as you did in Part1). To execute the first operation, poke its 16-bit encoder value into IR, and then poke the Toggle Switch to advance the FSM through the operation. Do the same for the rest and observe their execution to make sure that there is no errors. Do not forget to correct immediately any error into your tables before continuing, and then repeat from the beginning.

2.4 a) [0.40-mark] After finishing all executions, Log the execution of the sequence on your implementation and show the results here:

2.4 c) [0.30-mark] Compare the results obtained here to Part I, they should be same. So, what is the advantage of the generalized machine in part II (here) over the simple machine of part I (above)?

Submission deadline

Must be submitter on cuLearn, locate (Assignment 2 submission) and follow instructions. Submission exact deadline (date and time) is displayed clearly within the Assignment 2 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