Carleton University

Department of Systems and Computer Engineering

SYSC 3006 (Computer Organization) summer 2020

Lab / Assignment 2 – Answers file

Student Name: ID#:

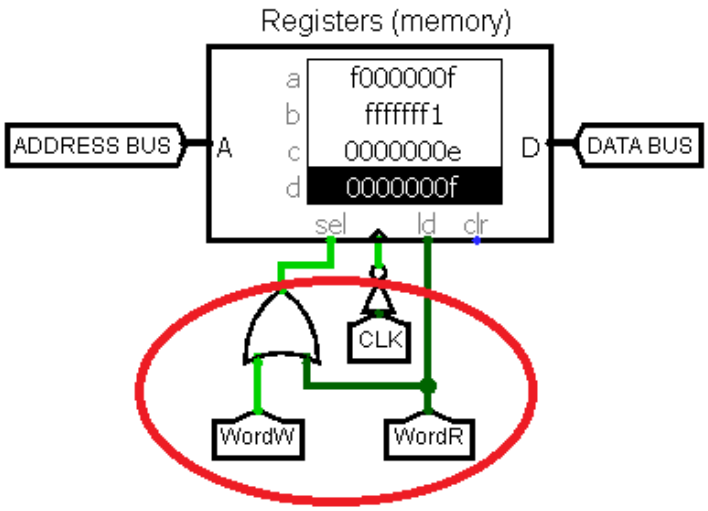
Part I – Simple Processing System circuit [3-mark/5]

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:

When SEL (or chip select) is asserted, it enable reading and writing to the device.
If not asserted, the device output in Z (high impedance state) and the input is inactive (we cannot white into the device). In other word, SEL is both CE (clock enable) and OE (output enable) of the device.
LD is equivalent to read/write' (read/write bar), when LD = 1, then we are reading the device into the data bus. When LD = 0, we are writing into the device from the data bus.

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)

Your (Lab2-Part-I.circ) circuit will be verified then you will get marked for this question

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 (2 bits)	OPR (4 bits)	Reg (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Hex
E0=00	0000	0001	1	0	1	0	0	0	01	01A1
E1=01	0101	0010	1	0	0	1	1	0	10	529A
E2=10	0000	1010	0	1	0	0	0	1	00	0A44

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

R11 ← R2 – R10										
State (2 bits)	OPR (4 bits)	Reg (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Hex
E0=00	0000	0010	1	0	1	0	0	0	01	02A1
E1=01	0010	1010	1	0	0	1	1	0	10	2A9A
E2=10	0000	1011	0	1	0	0	0	1	00	0B44

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

R12 ← NOT (R11)										
State (2 bits)	OPR (4 bits)	Reg (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Hex
E0 =	x	x	x	x	x	x	x	x	x	x

E1 = 00	0111	1011	1	0	0	0	1	0	01	7B89
E2 = 01	0000	1100	0	1	0	0	0	1	00	0C44

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

R13 ← R0 + R12										
State (2 bits)	OPR (4 bits)	Reg (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Hex
E0=00	0000	0000	1	0	1	0	0	0	01	00A1
E1=01	0001	1100	1	0	0	1	1	0	10	1C9A
E2=10	0000	1101	0	1	0	0	0	1	00	0D44

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 E2. 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:

Same log as in part 2

RAM(760,80)[0]	RAM(760,80)[1]	RAM(760,80)[2]	RAM(760,80)[10]	RAM(760,80)[11]	RAM(760,80)[12]	RAM(760,80)[13]
1	268435471	-268435456	0	0	0	0
1	268435471	-268435456	-268435441	0	0	0
1	268435471	-268435456	-268435441	-15	0	0
1	268435471	-268435456	-268435441	-15	14	0
1	268435471	-268435456	-268435441	-15	14	15
1	268435471	-268435456	-268435441	-15	14	15

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

-15

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)?

It performs the two's complement of the value in 11 (negate operation) -(n).

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) = 00	0	1	0	0	0	0	0	0	0	0	0	01	0801
(E0)= 01	0	1	0	1	0	1	0	1	0	0	0	10	0AA2
(E1)= 10	1	0	0	0	1	1	0	0	1	1	0	11	119B
(E2)= 11	0	1	1	0	0	0	1	0	0	0	1	00	0C44

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:0101 1010 0001 0010	0x5A12
R11 ← R2 – R10	0b: 0010 1011 0010 1010	0x2B2A
R12 ← NOT (R11)	0b:0111 1100 0000 1011	0x7C0B
R13 ← R0 + R12	0b:0001 1101 0000 1100	0x1D0C

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:

RAM(760,80)[0]	RAM(760,80)[1]	RAM(760,80)[2]	RAM(760,80)[10]	RAM(760,80)[11]	RAM(760,80)[12]	RAM(760,80)[13]
1	268435471	-268435456	0	0	0	0
1	268435471	-268435456	-268435441	0	0	0
1	268435471	-268435456	-268435441	-15	0	0
1	268435471	-268435456	-268435441	-15	14	0
1	268435471	-268435456	-268435441	-15	14	15
1	268435471	-268435456	-268435441	-15	14	15

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)?

The machine in part 1 requires a unique FSM (i.e. unique ROM contents) for each operation. So we need to change the content of the ROM for each operation. In part II, Introducing the IR allows the same FSM (same ROM contents) to execute all operations. We just need to encode each instruction in the IR for each instruction execution cycle.

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