Processing System Circuit

Ghassan Arnouk

SYSC 3006A
Summer 2020
Lab 2 Report
Group 1

Instructor: Michel Sayde
TA: Khalid Almahrog
Submitted: 2020/05/22

1 Simple Processing System Circuit

1.1 Discussion Questions

1.1.1 In your own words, describe how SEL and LD are used to control Read and Write operations on the RAM.

The **SEL** pin enables or disables the entire RAM module, based on whether the value is 1,floating, or 0. The input is meant primarily ofr situations where you have multiple RAM units, only one of which would be enabled at any time.

- If 0 is inputted, the RAM is disabled
- If 1 is inputted, the RAM is enabled
- If floating is inputted, the RAM is also enabled

The **LD** pin selectes whether the RAM should emit (on D) the value at the current address (A).

- If 0 is inputted, the RAM (writes) does not emit the value on the D
- If 1 is inputted, the RAM (reads) emits the value on the D
- If floating is inputted, the RAM also (reads) emits the value on the D

1.1.2 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.

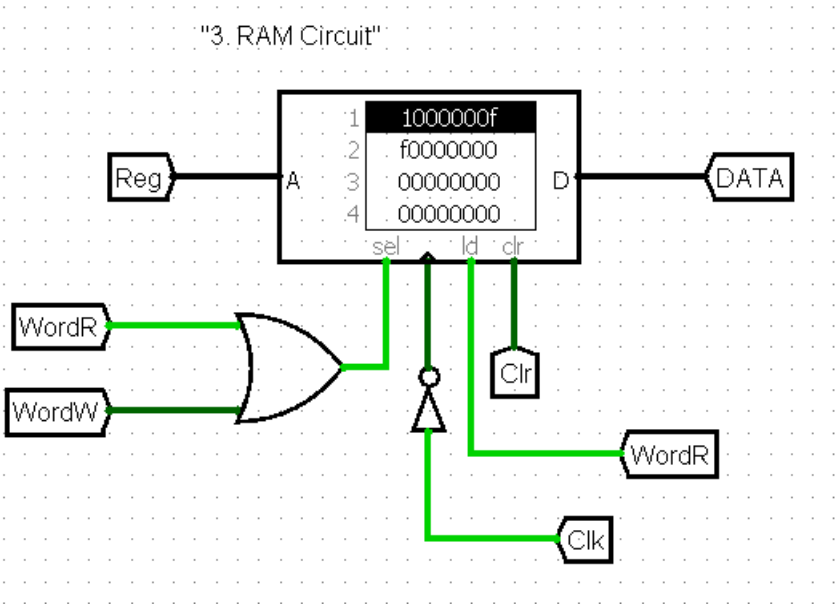

Figure 1: RAM Circuit Implementation

1.2 Circuit Wiring

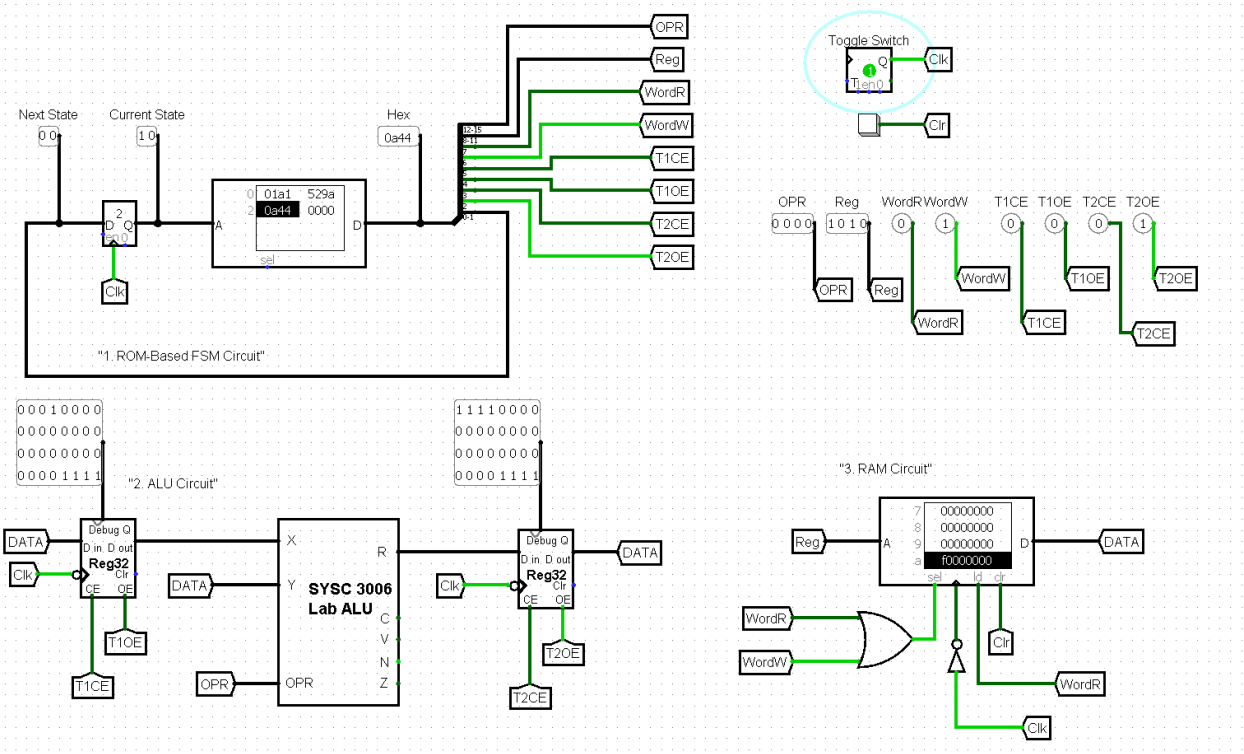

Figure 2: Simple Processing System Circuit

- Each FSM contains four states: E0 (00), E1 (01), E2 (10), and E3 (11). The output for each state contains a total of 16 bits

Table 1: Information contained within the 16 bits

OPR	4 bits	ALU operation code
Reg	4 bits	Register address
WordR	1 bit	Word read enable
WordW	1 bit	Word write enable
T1OE	1 bit	T1 output enable (read)
T1CE	1 bit	T1 clock enable (write)
T2OE	1 bit	T2 output enable (read)
T2CE	1 bit	T2 clock enable (write)
Next State	2 bits	Next State

- Note that the information is written in hexadecimal notation

1.3 ROM FSMs

1.3.1 Fill in a next state table (ROM table) for the following RTL.

Table 2: R10 ← R1 OR R2

State	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	0001	0010	1	0	0	1	1	0	10	529A
E2=10	0000	1010	0	1	0	0	0	1	00	0A44

1.3.2 Fill in a next state table (ROM table) for the following RTL.

Table 3: R11 ← R2 - R10

State	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

1.3.3 Fill in a next state table (ROM table) for the following RTL.

Table 4: R12 ← NOT(R11)

State	OPR (4 bits)	Reg (4 bits)	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Hex
E0=00	d	d	d	d	d	d	d	d	01	0001
E1=01	0111	1011	1	0	0	1	1	0	10	7B9A
E2=10	0000	1100	0	1	0	0	0	1	00	0C44

1.3.4 Fill in a next state table (ROM table) for the following RTL.

Table 5: R13 ← R0 + R12

State	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 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.

1.4.1 After finishing all executions, Log the execution of the sequence on your implementation and show the results here.

Table 6: Simulation Log of the Simple Processing Circuit

RAM(740,250)[10]	RAM(740,250)[11]	RAM(740,250)[12]	RAM(740,250)[13]
00000000	00000000	00000000	00000000
f0000000	00000000	00000000	00000000
f000000f	00000000	00000000	00000000
f000000f	fffffff1	00000000	00000000
f000000f	fffffff1	fffffff1	00000000
f000000f	fffffff1	0000000e	00000000
f000000f	fffffff1	0000000e	0000000f

1.4.2 After the executions of R11 ← R2 − R10, what is the value of R11 (in decimal)?

The value of R11 in decimal after the executions is 4294967281. The decimal from signed 2's complement is -15.

1.4.3 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)?

R12 is the 2's complement of R11. Furthermore, fffffff1 (-14) is the 2's complement of 0000000e (14). R13 is the sum of R0 and R12. Adding 00000001 (1) to 0000000e (14) will result in 0000000f (15).

2 Generalized Processing System Circuit

2.1 Circuit Wiring

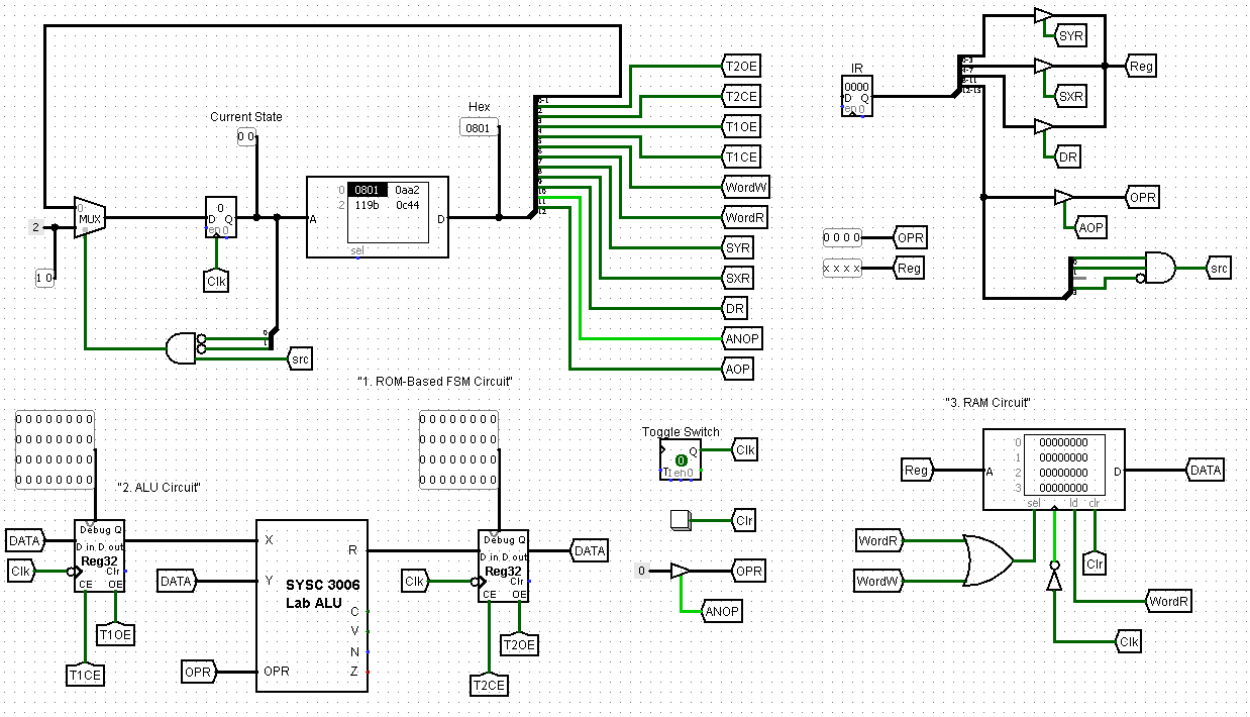

Figure 3: Generalized Processing System Circuit

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

Table 7: Control FSM

State (ROM) Adress	AOP	ANOP	DR	SXR	SYR	WordR	WordW	T1CE	T1OE	T2CE	T2OE	Next State	Inst.(Hex)
(D)=00	0	1	0	0	0	0	0	0	0	0	0	00	0801
(E0)=00	0	1	0	1	0	1	0	1	0	0	0	10	0AA2
(E1)=01	1	0	0	0	1	1	0	0	1	1	0	11	119B
(E2)=10	0	1	1	0	0	0	1	0	0	0	1	00	0C44

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

Table 8: Encoded Instruction

Operation	Encoded 16-bit Value (in Binary)	Encoded 16-bit Value (in Hex)
R10 ← R1 OR R2	0b: 0101 1010 0001 0010	5A12
R11 ← R2 - R10	0b: 0010 1011 0010 1010	2B2A
R12 ← NOT(R11)	0b: 0111 1100 0000 1011	7C0B
R13 ← R0 + R12	0b: 0001 1101 0000 1100	1D0C

2.4 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.

2.4.1 After finishing all executions, Log the execution of the sequence on your implementation and show the results here.

Table 9: Simulation Log of the Generalized Processing Circuit

RAM(740,250)[10]	RAM(740,250)[11]	RAM(740,250)[12]	RAM(740,250)[13]
00000000	00000000	00000000	00000000
f000000f	00000000	00000000	00000000
f000000f	ffffffff	00000000	00000000
f000000f	ffffffff	0000000e	00000000
f000000f	ffffffff	0000000e	0000000f

2.4.2 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 generalized processing system circuit is a lot more quicker as it allows the user to do multiple ALU operations without having to change the instructions in the ROM-based FSM. In the simple machine, each operation involving the ALU requires a unique FSM table. However, the generalized machine is implemented such that it encodes the differences among operations in a way that can be manipulated by the FSM.