Lab 2: Simple Data Path

Due Tuesday 28 April 2020, 11:59 PM

Minimum Submission Requirements

- Ensure that your Lab2 folder contains the following files (note the capitalization convention):
  - Lab2.lgi (you may need to rename your extension from .LGI to .lgi)
  - README.txt
- Commit and push your repository
- Completed Google Form with the correct commit ID of your final submission

Objective

The objective of this lab is to build a sequential logic circuit and introduce data paths.

Description

A data path is the path by which data flows in a system. In this lab, you will implement a simple data path with a register file, ALU, and user inputs.

Each processor contains a register file that holds the registers used in program execution. Registers are fast access local variables that can change after every instruction.

In this lab, you will be building a register file that contains four, 4-bit registers. Each of the four registers has an address (0b00 -> 0b11) and stores a 4-bit value.

The value saved to a destination register (write register) will be chosen from one of two sources, the keypad user input, or the output of the ALU. The ALU in this system is a 4-bit bitwise right rotation (right circular shift) circuit that takes two of the register values as inputs (read registers). **You may not use the MML library ALU and should instead build one out of muxes or logic gates.**

From the user interface, the user will select the data source (source select) and the addresses of the read and write registers.

A usage example is demonstrated in the Appendix.

Resources

Right Rotate Bitwise Operation

Multiplexors

Registers, Flip-Flops, and Modular Design

Specification

Template

Build your lab starting with the template file provided. The template file contains the user interface. **DO NOT MODIFY THE FIRST PAGE** except for your name and CruzID (NOT your student ID number).

Additional wires and logic circuits shall be drawn on subsequent pages. There are placeholder signal senders and receivers on the second page that you can use. You may remove these senders and receivers from the second page as you use them in your design. You are free to modify the template from the second page and add more pages. You are encouraged to add middle values to break a complicated circuit to several parts and add meaningful names to circuits to make them readable like the first page.

Remember to rename the template file to Lab2.lgi.

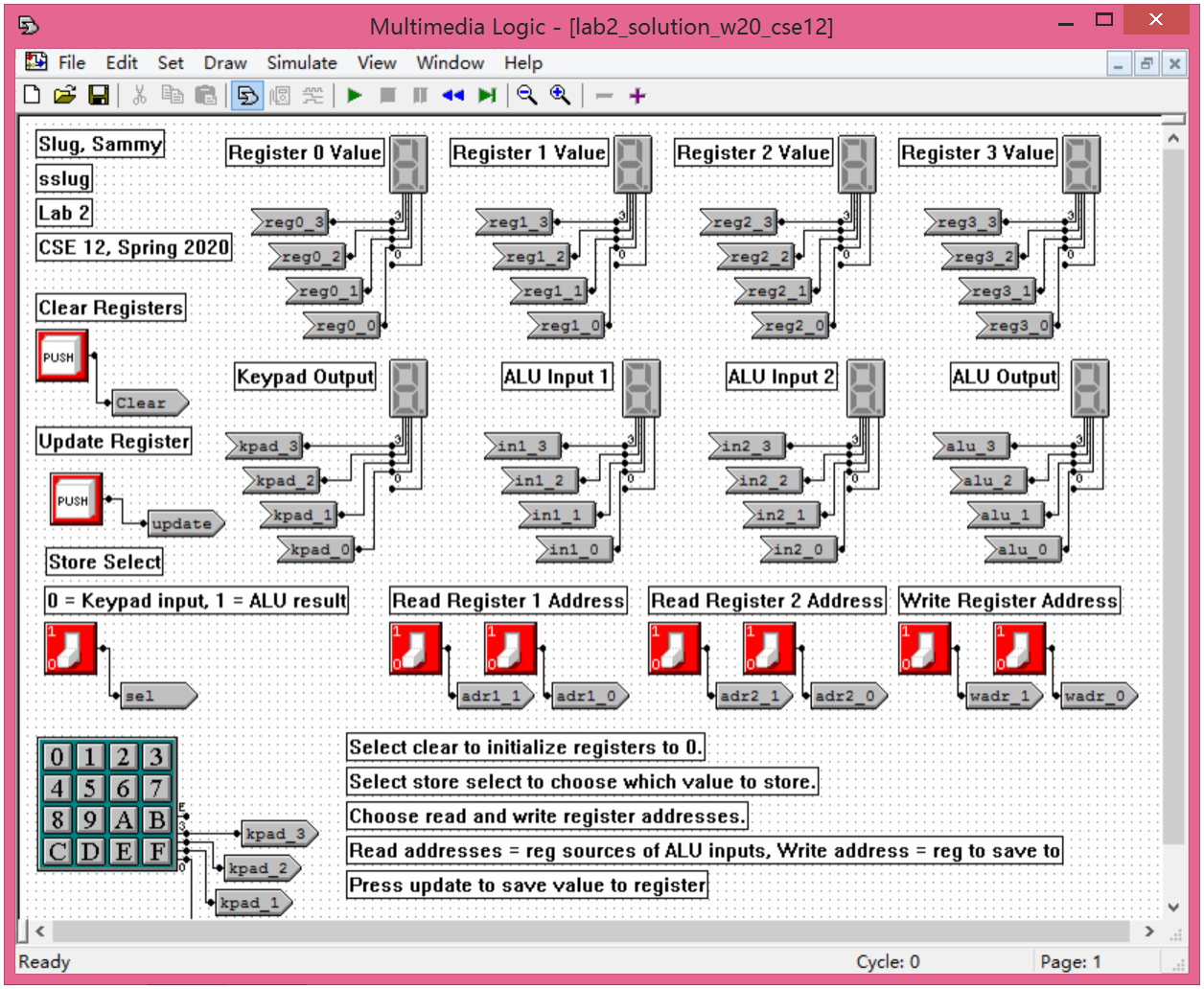


Figure: Lab 2 Template

User Interface

Inputs

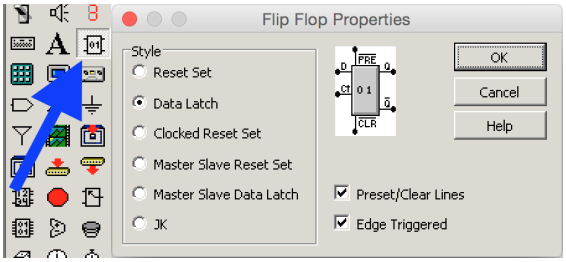
Clear	Resets all registers to 0, normally off
Update Register	Stores the appropriate value to the destination register indicated by the Write Register Address. This is like the "clock" for the register file.
Read Register 1 Address and Read Register 2 Address	Addresses of source registers to ALU
Write Address	Address of destination register
Keypad	User input value, this value will be stored directly to the register indicated by the write address register if Store Select is 0.

Outputs

Register # Value	4-bit raw value stored in register
Keypad Output	Value selected from keypad
ALU Input 1 and ALU Input 2	Value of the registers addressed by Read Register 1 Address and Read Register 2 Address, respectively
ALU Output	Result of the ALU computation; bitwise right rotation of ALU Input 2 by the amount in ALU input 1

Flip-Flops

For the register, use D flip-flops, and make sure they are edge triggered with a clear line.



For your convenience, here is a tabular description of how flip-flop with clear line works. You may also find practice_flipflop.lgi handy for understanding how flip-flops set and reset.

Examples of how to hook up the D flip-flop is included in practice_flipflop.lgi. This file can be found here.

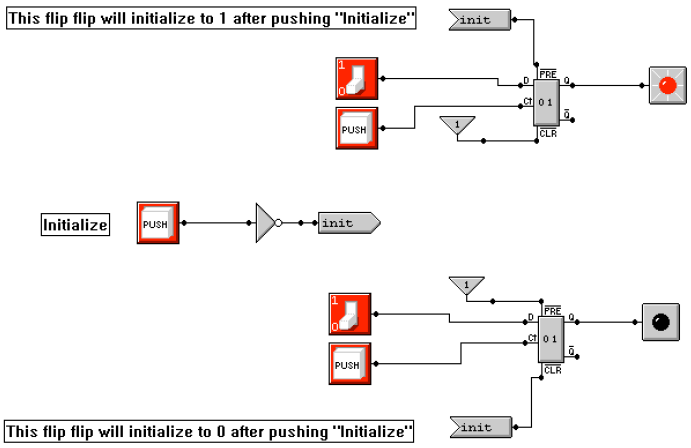


Figure: Flip-flop Usage Example

Top Level

Here is a top-level functional block diagram of the overall design. You do not need to follow this design exactly if you discover another way to meet the specification.

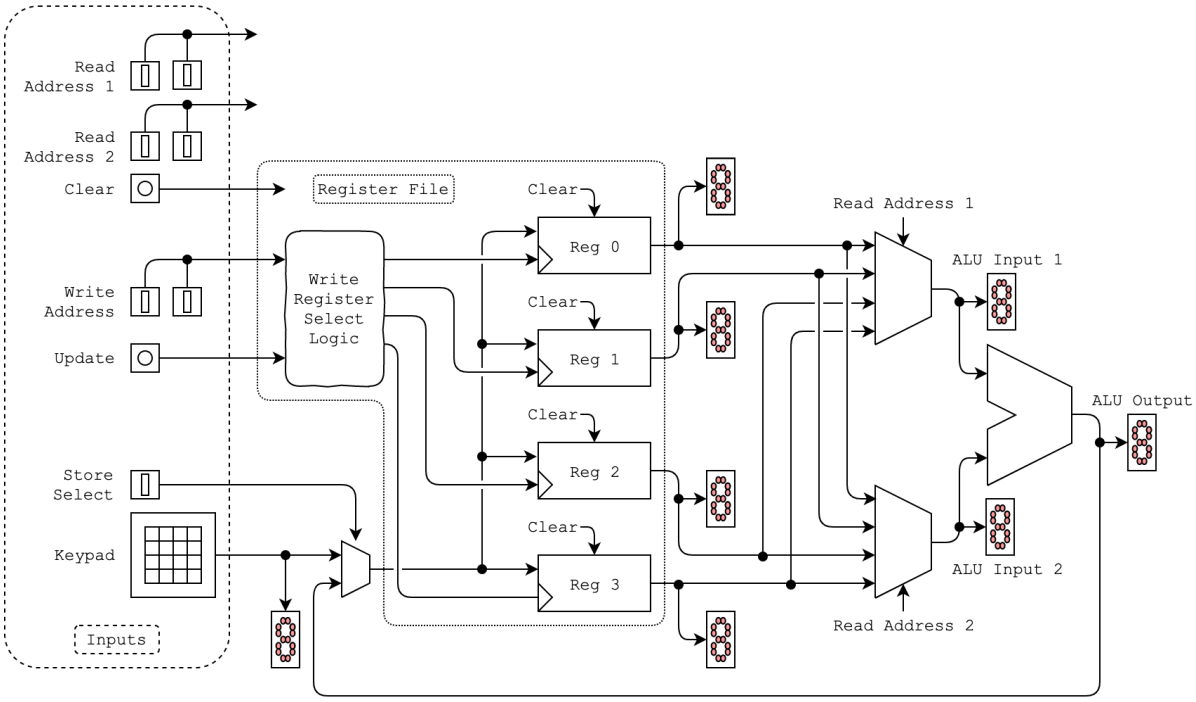


Figure: Top Level Diagram

Documentation Standards (README.txt)

Follow the documentation guidelines found here. Refer to the sections on the README and schematic visual structure. Diagram.pdf is not required for this lab.

Simulation

To ensure the circuit simulates without error, make sure there is **at least one receiver for every sender** and that **each receiver has exactly one sender**. In addition, **do not modify the canvas size**.

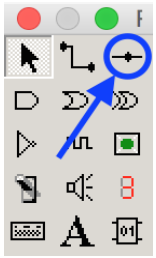
If you do not have at least one receiver for every sender or if you have more than one sender with the same name, your circuit will not simulate and you will lose points.

Google Form

You are required to answer questions about the lab in this Google Form:

Google Form

Missing Wire Best Practices



MML has a known bug which causes some wires to disappear during the save process. To reduce the likelihood of this occurring, **DO NOT use the "Node" tool** (it's a black dot located at the top-right of the tool palette). This tool is particularly vulnerable to the bug.

If this bug occurs, the grader will attempt to repair the missing wire in your file. **This is only possible if your circuit is very readable.** Make sure that wires **do not cross** whenever possible. Wire paths should be short and direct. **Use senders and receivers liberally**.

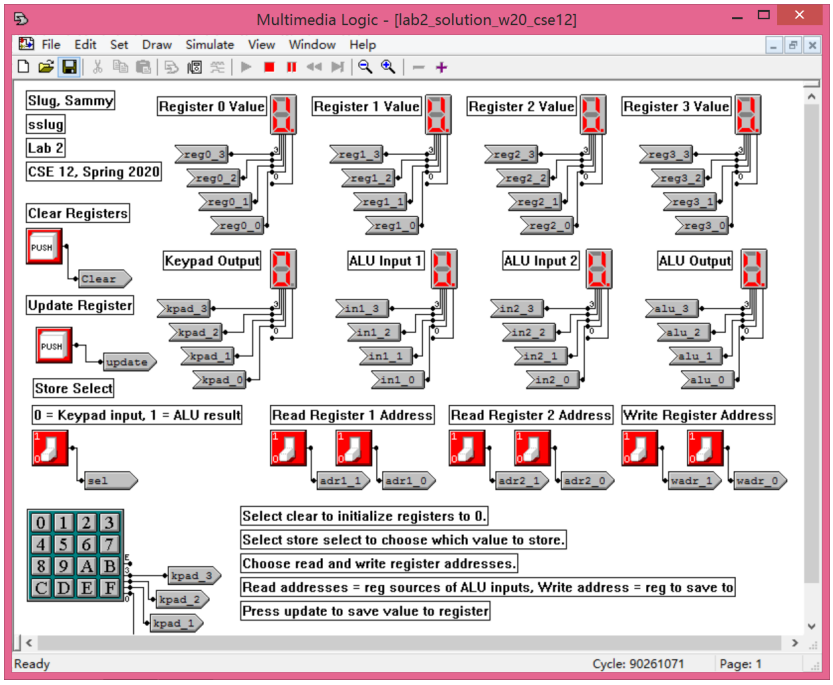
Grading Rubric (total 70 points)

```
 10 pt  simulates without errors

 25 pt  register file functionality
       4 pt  clear button works
       4 pt  register design
       5 pt  update button works
       12 pt  write register select logic

 20 pt  other functionality
       3 pt  ALU input 1 reads correct register
       3 pt  ALU input 2 reads correct register
       6 pt  ALU performs bitwise right rotation (doesn't use MML library ALU)
       8 pt  correct value saved

 15 pt  documentation
       3 pt  complete header comments on every page of schematic
       3 pt  useful & sufficient comments
       3 pt  clean visual structure / use of white space
       3 pt  README file complete
       3 pt  Google form complete with at least 150 words

-15 pt  if first page of template is modified, or if the template isn't used properly
-15 pt  incorrect naming convention (e.g .LGI or lab2.lgi)
```

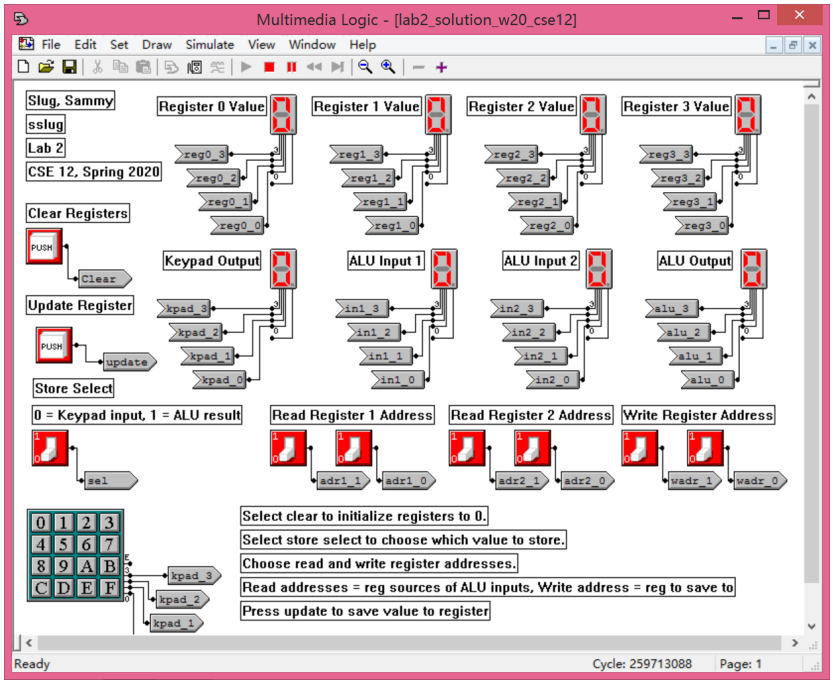
Appendix

Usage Example

1. Select simulate.

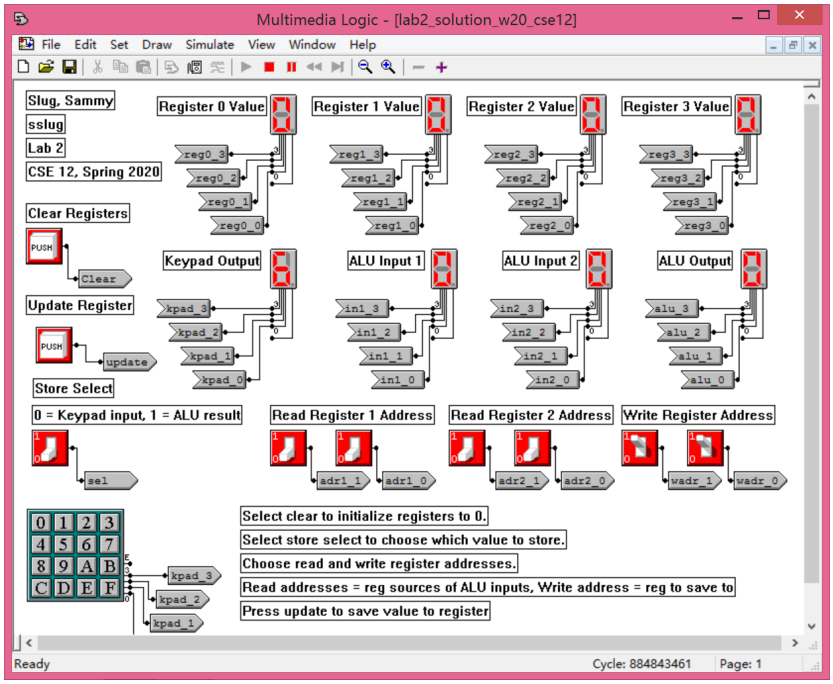


2. Press Clear Registers to reset all register values to 0.

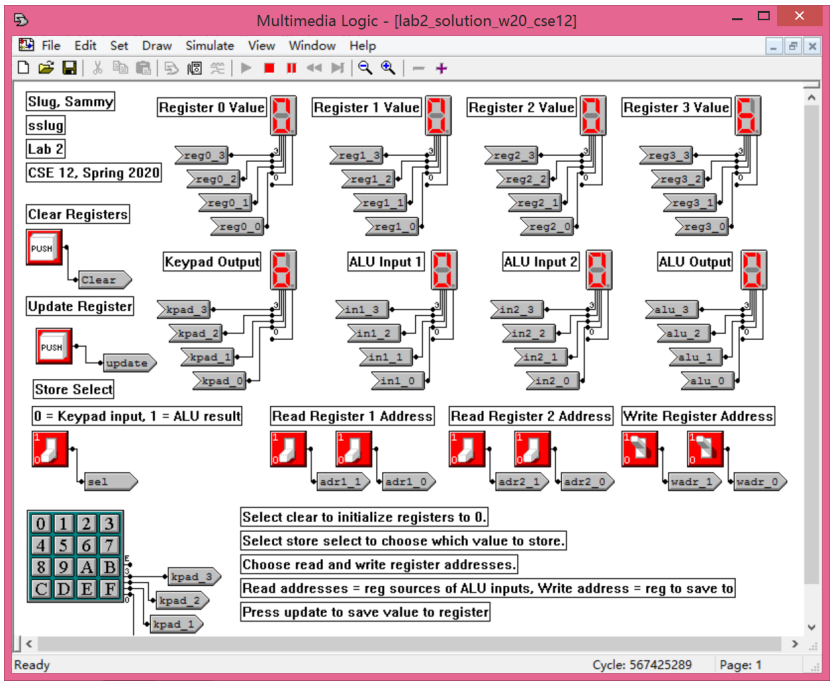


3. Select the following inputs to store the number 6 to register 3:

```
Store Select:            0
Keypad:                  6
Read Register Address 1: x (don't care)
Read Register Address 2: x
Write Register Address:  3
```

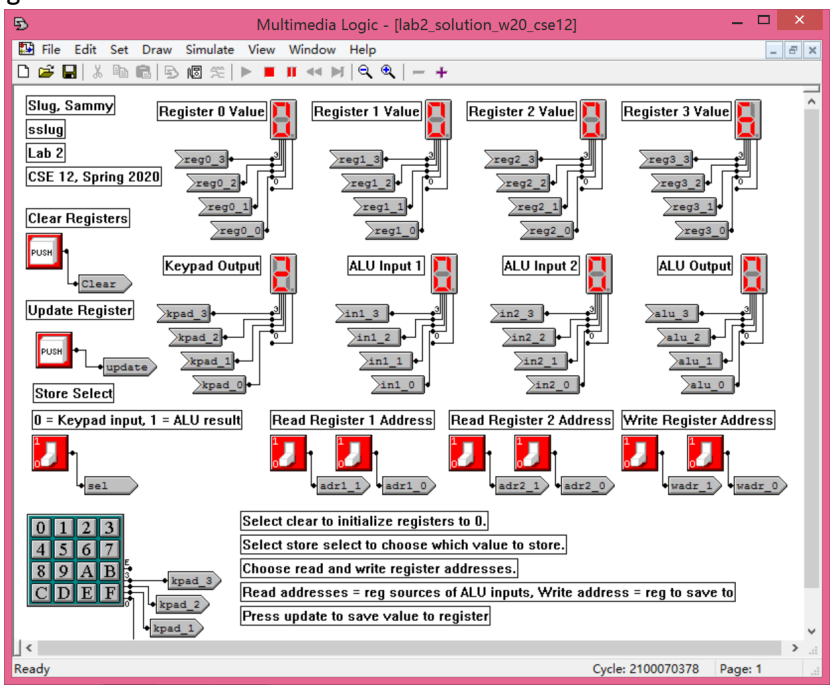


4. Press Update Register to save the keypad input to the destination register.

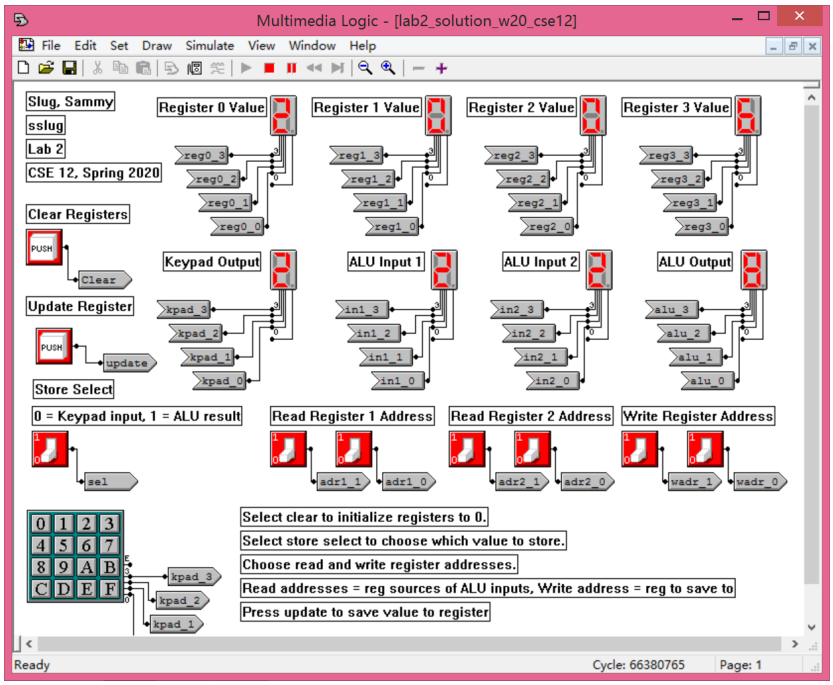


5. Select the following inputs to store the number 2 to register 0:

```
Store Select:            0
Keypad:                  2
Read Register Address 1: x
Read Register Address 2: x
Write Register Address:  0
```

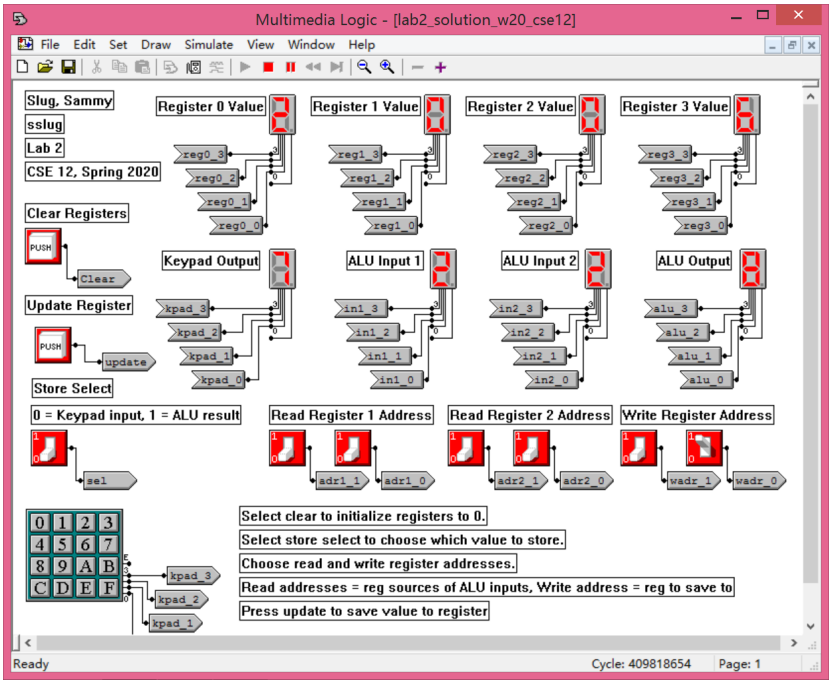


6. Press Update Register to save the keypad input to the destination register.

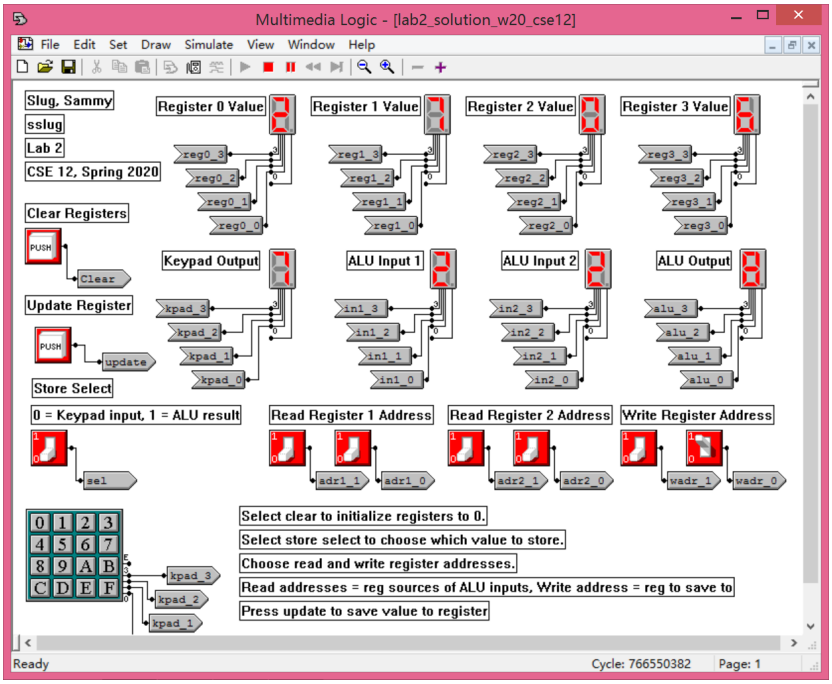


7. Select the following inputs to store the number 7 to register 1:

```
Store Select:            0
Keypad:                  7
Read Register Address 1: x
Read Register Address 2: x
Write Register Address:  1
```

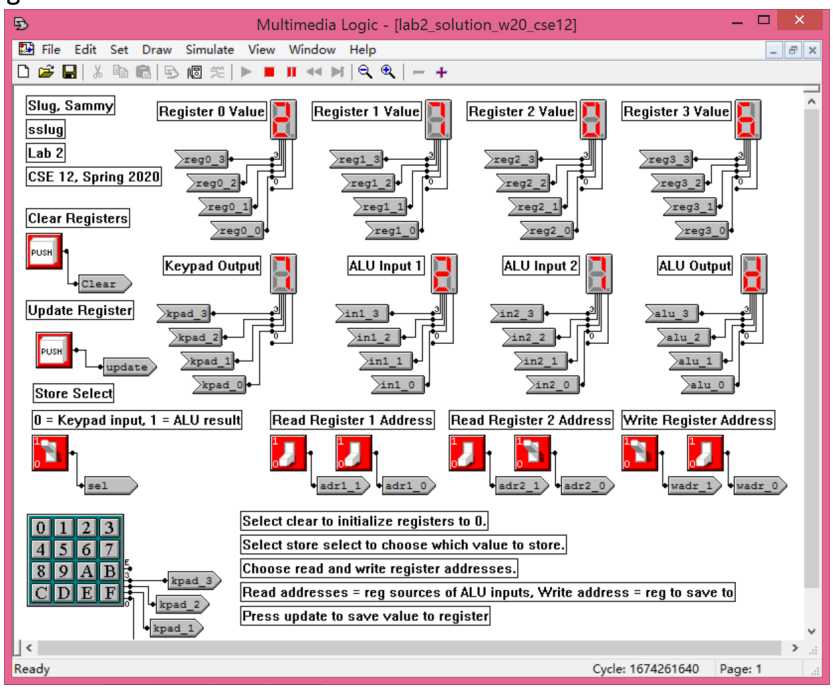


8. Press Update Register to save the keypad input to the destination register.



9. Select the following to store the rotation of registers 1 and 2 to register 0:

```
Store Select:            1
Keypad:                  x
Read Register Address 1: 0
Read Register Address 2: 1
Write Register Address:  2
```



10. Press Update Register to save the ALU output to the destination register.