**University of California, Santa Cruz**
**Board of Studies in Computer Engineering**

**CMPE012/L: Computer Systems and Assembly Programming**

# Lab 3: Logic Unit with Memory

## Minimum Submission Requirements:

- Create a lab3 folder
- Lab3.lgi in the lab3 folder
- README.txt in the lab3 folder
- Commit and Push your repo
- A Google form submitted with the Commit ID taken from your GITLAB web interface verifying that the above files are correctly named in their correct folders
- All of the above must be completed by the lab due date

## Lab Objective:

In this lab you will continue to use Multimedia Logic to connect individual gates into a larger structure that begins to implement a few functions of the ALU. In this case, you will be implementing a logic ALU, and the accompanying glue logic required to make it do something useful. This is an exploration of both combinational logic as well as sequential logic. In class you learned that combinational logic was just a function of current inputs and that sequential logic was a function not only of current input, but some past sequence of inputs.

You are going to use sequential and combinational logic to perform logical operations on a sequence of numbers. This lab will develop your ability to build systems in a modular way. You will design and test several components independently, before connecting those components into a larger system.

## Lab Preparation:

Read this document in its entirety carefully. Review basic logic gates (Chapter 3 of the Patt and Patel Reader, Appendix B of Patterson and Hennessy, up to B-4) and make sure you understand them. If you are having difficulty, supplement your reading with online tutorials on logic gates.

## Lab Specifications:

You will build a logic unit. The register stores a 4-bit binary number. The user uses a keypad to enter a 4-bit input number, along with two toggles switches that specify the operation. When the user presses a "store" button, the input number is either STOREd directly (0b00), ANDed (0b01) with the register and stored, ORed (0b10) with the register and stored, or the register is INVERTed (0b11) and stored in the register while ignoring the input number.

So, if the user has input "4" (0b0100), the stored number is "A" (0b1010), the circuit is set to OR (0b10) and the user presses the "Store" button, the stored number becomes "E" (0b1110), and the circuit is now waiting to perform the next operation. Similarly, if the user has input "4", the stored number is "A", the circuit is set to AND (0b01) and the user presses the "Store" button, the stored number becomes "0", and the circuit is now waiting to perform the next operation.

As usual, every file you turn in should have the following information:

- Your name and email@ucsc.edu
- Lab number and title
- Due date

- Your section (e.g.: 01A), your section TA/tutor

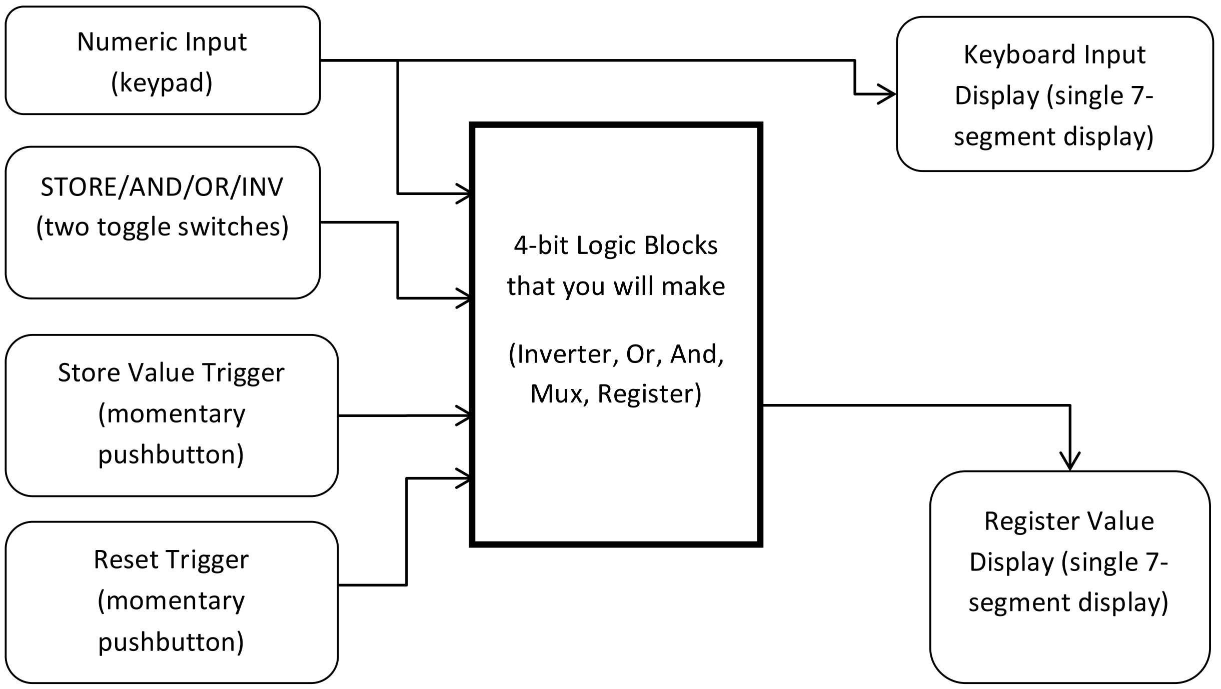
Subsequent pages should have a header indicating the part of the lab that the page contains (e.g., Mux).

**LAB OVERVIEW:**

To build the logic unit, we are going to need several parts. Refer to the block diagram below to see how it all fits together. Each of these should have its own page in your MML file, for a total of at least 4 pages. We *strongly* recommend that you test each part before attempting to integrate it with the other parts.

We'll need:

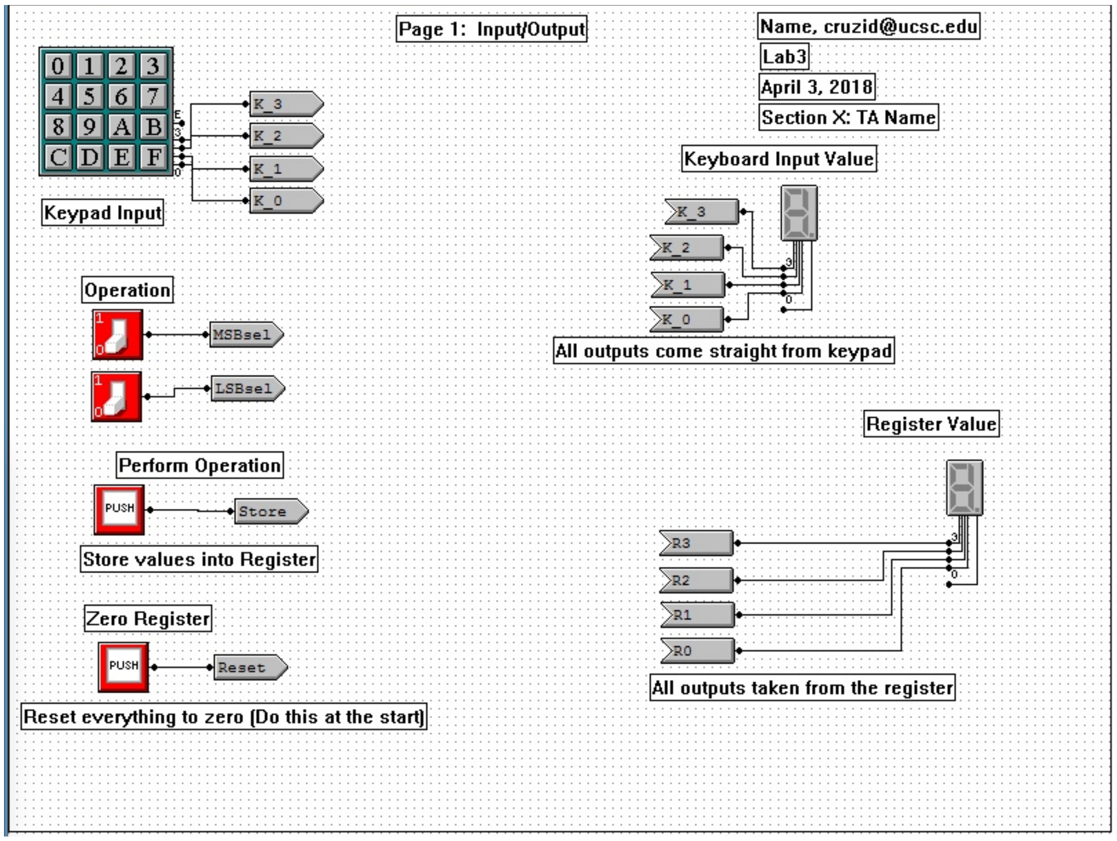
- Various devices for user input/output.
- A 4-bit inverter, AND, OR.
- A 4-bit, 4-way MUX (selector)
- A 4-bit register, to store our value.


Numeric Input (keypad)
STORE/AND/OR/INV (two toggle switches)
Store Value Trigger (momentary pushbutton)
Reset Trigger (momentary pushbutton)
4-bit Logic Blocks that you will make
(Inverter, Or, And, Mux, Register)
Keyboard Input Display (single 7-segment display)
Register Value Display (single 7-segment display)

## Input/Output:

Your Input/Output page must be the first page of the lab. Use the template file provided on CANVAS and immediately change your name, email, and section information and save it.

There are four inputs to this lab: A numeric keypad, two toggle switches to select the operation, a button perform the operation and store the result, and a button to clear the value in the register.



The keypad lets the user enter one hexadecimal digit, and outputs the corresponding 4-bit binary number.

The "Perform Operation" and "Zero Register" buttons are momentary pushbuttons. Perform operation completes the operation and captures the value into the register. The zero register button is used to force the register to all zeros (note that MML treats flip-flops correctly and their internal states are unknown at startup).

The outputs on the front page includes two seven-segment displays to show the keypad input and the contents of the register in hexadecimal.

You are encouraged to include other outputs to other pages to aid in debugging, but you should not alter the inputs and outputs on this page.

**You must follow the "Missing wire best practices" described in Lab2 to ensure that your submission will work peroperly.**

## Register:

You will need to build a register to store the current value. Recall that a register is several flip-flops, each of which is capable of "remembering" or storing a single bit of information. Setting up a flip-flop in MML is a bit complicated. Refer to the “starter_parts.lgi” file on CANVAS to see an example of a working flip-flop. You’ll have to adjust our example to connect it to the rest of your circuit.

The register will be 4 bits wide. This means that you will need 4 flip-flops for storage. These flip-flops will be controlled by the user inputs. The “Perform Operation” input should update the register’s contents, and the “Zero Register” input should clear the register’s contents (that is, set each bit to zero).

## MUX SELECT:

You will also need to build a 4-bit, 4-way multiplexor. We ***very strongly*** recommend building a 1-bit, 4-way multiplexor and testing it thoroughly, and only then copy it 4 times. Give yourself some debug tools and connect the inputs and outputs up to LEDs or 7 segments so that you can verify its operation.

Your full 4-bit mux should have 18 inputs – one for each of the 4 bits of the logic operations, plus two for the select signals. It is important to note the MSB (most significant bit) and LSB (least significant bit) of the select according to the operations defined earlier. 0b10 has 1 for the MSB and 0 for the LSB.

## LOGIC OPERATIONS:

We will perform each operation either on the input only (STORE), the stored number only (INVERT) or the input and the stored number (AND, OR).

It is suggested to make separate page for each of the three logic operations (INVERT, AND, OR). The AND and OR design will have two operands of 4-bits, so 8 input bits and 4 output bits. The INVERT design will have 4 inputs and 4 outputs.

## LAB WRITEUP:

Please use our README.txt template on Canvas to make grading easier. Be sure your write-up contains:

- Appropriate headers
- Describe what you learned, what was surprising, what worked well and what did not.
- Discuss issues you had building the circuit.
- Describe what you added to each module to make debugging easier.
- What is the difference between a bit-wise and reduction logic operation? What operations did we implement? Why might we want to use the other type of logic operations?

To alleviate file format issues we want lab reports in plain text. Feel free to use a word processor to type it up but please submit a plain text file and CHECK IT ON GITLAB to make sure it is readable.

## BUMPS AND ROAD HAZARDS:

This lab is significantly more difficult than the previous three. This, in essence, is your first “real” lab. Make sure you start this early, and get your files checked in and push early and often. We will be looking at your commit trees as part of the grading, and want to see you incrementally developing your solution to this lab.

You have a longer time to complete the lab, but it is also a whole lot more to do. Budget your time accordingly and don’t wait until the last minute to start the lab.