UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE012/L: COMPUTER SYSTEMS AND ASSEMBLY PROGRAMMING

LAB 3: RIPPLE ADDER WITH MEMORY

MINIMUM SUBMISSION REQUIREMENTS:

- 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 6 bit ripple adder, 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 sum (or subtract) a sequence of numbers together. 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 running summer. The summer stores a six-bit binary number. The user uses a keypad to enter a 4-bit input number, along with a toggle switch that specifies whether the circuit is adding or subtracting by using the additive inverse (2SC). When the user presses a "store" button, the input number is either added to or subtracted from the currently stored number (running sum).

So, if the user has input "4", the stored number is "A", the circuit is set to subtract and the user presses the "Store" button, the stored number becomes "6", 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

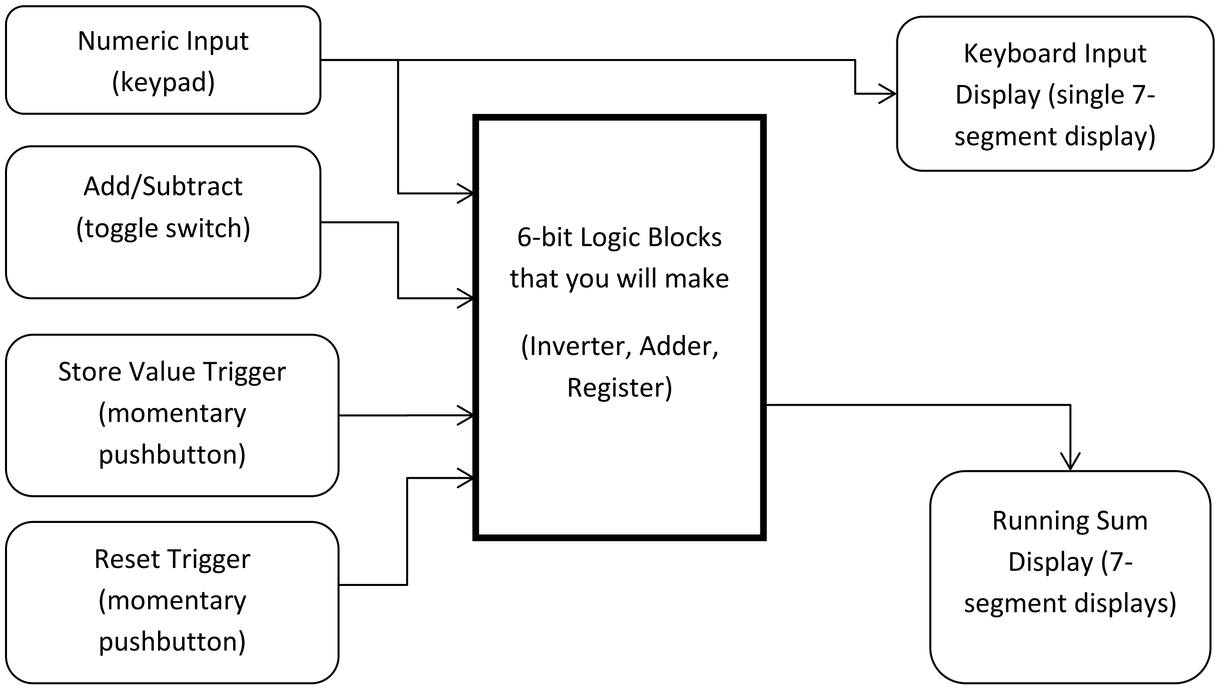
Subsequent pages should have a header indicating the part of the lab that the page contains (e.g.: Additive Inverse).

Lab Overview:

To build the adder 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.

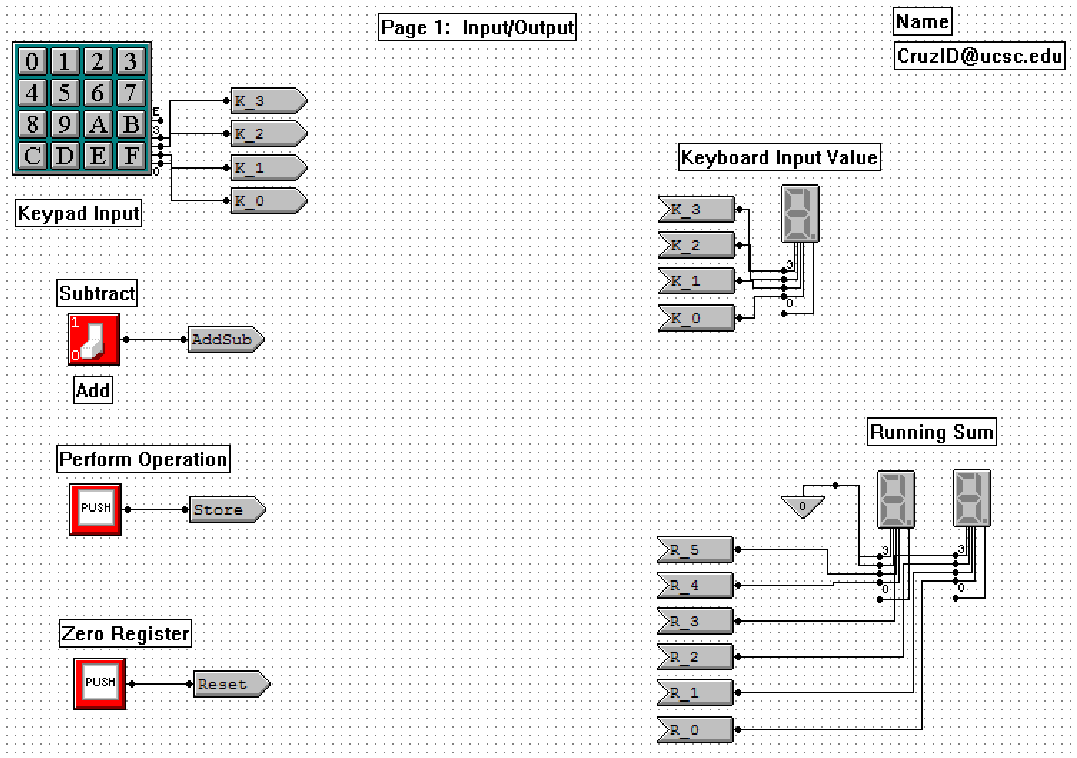
We'll need:

- Various devices for user input/output.
- A 6-bit inverter, to take the 2's complement of the user's input, when subtracting.
- A 6-bit ripple adder, to perform the addition operation.
- A 6-bit register, to store our running sum.


Numeric Input (keypad)
Add/Subtract (toggle switch)
Store Value Trigger (momentary pushbutton)
Reset Trigger (momentary pushbutton)
6-bit Logic Blocks that you will make
(Inverter, Adder, Register)
Keyboard Input Display (single 7-segment display)
Running Sum Display (7-segment displays)

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 and CruzID@ucsc.edu and save it.



There are four inputs to this lab: A numeric keypad, a toggle switch to select addition or subtraction, a button to perform the calculation and store the result, and a button to clear the value in the register.

The keypad lets the user select 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 addition/subtraction and latches the value into the register. Zero register 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.

Register:

You will need to build a register to store the running sum. 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 6 bits wide. This means that you will need 6 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).

RIPPLE ADDER:

You will also need to build a 6-bit ripple adder. We ***very strongly*** recommend building a one-bit full adder, testing it thoroughly, and only then copy it six 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 6-bit adder should have 13 inputs – one for each of the 6 bits of the 2 summands, plus one ones-place carry bit. That is the carry in bit for the one's column for doing the 2's complement for subtraction.

INVERTER:

We will perform subtraction by finding the 2's compliment of the input number from the keypad, then adding that result to the stored number.

To do so, we need to invert our input and add one to the result. This module performs the "flip the bits" step (the "add one" step is handled using the carry in bit of the adder).

When *adding* this module should leave the user's input unchanged; when *subtracting* you need to invert the number and add one. That is, there is some conditional operation going on here, and you will need to spend some time figuring out how to implement it.

If you aren't sure how to perform a conditional inversion, write a truth table and implement the result with gate-level logic.

Note: This module takes 4 bits of input, but outputs 6 bits. That means you need to extend the length of the user input in this module. Recall the sign extension issues that were covered in lecture in order to handle the extra bits.

LAB WRITEUP:

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 happens when you subtract a larger number from a smaller number? Does the result make sense? What happens when you add two numbers that won't fit in 6 bits?

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 pushed 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.