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

CMPE12/L: COMPUTER SYSTEMS AND ASSEMBLY LANGUAGE

Lab 2: Adding and Subtracting
Worth 65 points (60 lab + 5 report)

Lab Objectives:

You now have had a chance to play around with some basic combinational logic circuits. Let's move on to something a little more interesting, sequential logic. From 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. We are going to use sequential and combinational logic that allows us 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.

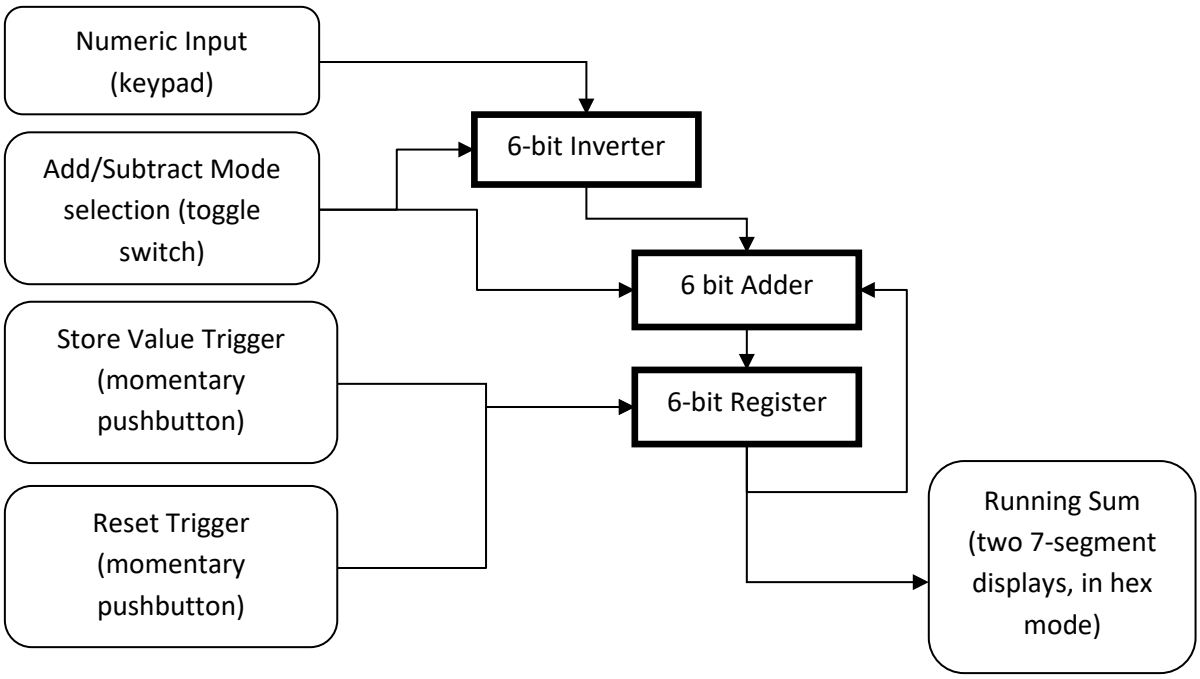
Lab Overview:

In this lab you will build a running summer. The summer stores a six-bit binary number (initially zero). The user can enter a four-bit hex number, and an additional bit to indicate a negative number. Each time the summer is triggered, its logic will add the user's input to the stored number, and store the result.

Have you ever used an older calculator that allows you to enter something like "7" followed by "+", then press "=" repeatedly for 14, 21, 28, etc? (Windows' default calculator has this behavior, so you can try it for yourself). This summer will work in the same way, but it will only add and subtract, and you can only input in the range 0 to 15 (in hex, 0 to F).

To build our summer we are going to need several parts. Refer to the block diagram below to see how it all fits together. We'll need:

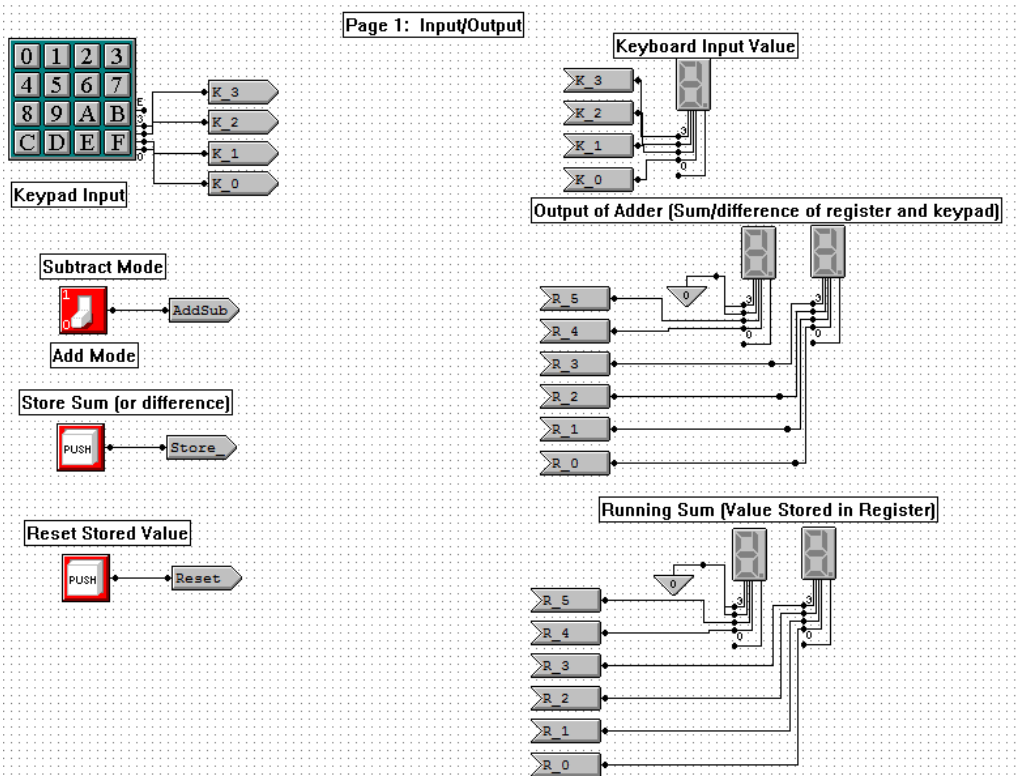
- Various devices for user input/output.
- A 6-bit conditional inverter, to help us take the 2's complement of the user's input, when appropriate.
- A 6-bit adder, to perform the addition operation.
- A 6-bit register, to store our running sum.



General Instructions:

Please name your file Lab2.lgi.

Make an overview page containing the input and output for your circuit along with the control buttons. Think of this page as the user interface, containing everything a user needs to operate and interact with your circuit. It should look more or less like this (note that this image includes some additional numerical displays):



Each other part of your running summer (the inverter, adder, and register) should get its own page (or two, if needed). Test these parts separately to make sure they work before integrating them with the rest of the circuit.

You need to use the "signal sender/receiver" components to route signals between pages. In order to function, the sender and receiver need to have EXACTLY the same name – be careful about capitalization and trailing whitespace!

There is a file called "Starter_Parts.lgi" which includes various pre-built pieces of your circuit, which you may copy-paste into your file if you wish.

You are encouraged to add LEDs or other devices to your schematic to make it easier to debug.

Part 1: Input/Output

There are four inputs to this lab: A numeric keypad, a toggle switch to select addition or subtraction, a button to store a new value in the register, 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 “Store” and “Reset” buttons should be momentary buttons. Be sure to set them to the correct initial state (that is, should they output 1 or 0 when you are not pressing the button? Check the documentation on flip-flops if you aren’t sure which is appropriate).

There is only one required output: Two seven-segment displays to show the contents of the register in hexidecimal. (Question: why do you need two seven-segment displays?)

You are encouraged to include other outputs to aid debugging.

Part 2: Register

In this part we need to build our register file. This will store our running sum and will display it to the user as a hex number. Recall that a register is several flip-flops, each of which is capable of “remembering” or storing a single bit of information.

The register will be 6 bits wide. This means that you will need 6 flip-flops for storage (the “D” latch is the easiest to use, so use these). Setting up flip-flops in MML is a bit complicated, so we’ve included one in the starter_parts.lgi file that is connected correctly, except for the inputs and outputs.

Connect all the CLR inputs to the “Reset” button. When the output of this button is 0, it will clear all of our Flip-Flops – that is, turn their stored values to zero.

Connect all the “clock” inputs to the “Store” pushbutton. In a real-time computer, registers are updated regularly on a clock cycle. In this case, however, we only want to update when the user is ready, so that user input will be our “clock”. When the switch is pressed the data will be stored in our flip flops. Until this button is pressed, the value in the flip-flips should not change.

Hook up the outputs of the Flip-Flops to 7 segment LEDs so you can see the register’s contents. You will find that each of the 7-segment LEDs won’t work correctly until all four of its inputs are connected. That means you’ll need to ground two of the inputs by connecting them to a “ground” component. (Question: Of the 8 inputs, which two should you ground?)

Part 3: Addition

Build a 6-bit full adder. We highly recommend building a one-bit full adder, testing it thoroughly, and only then copying it six times. Give yourself some debug tools and connect the inputs and outputs up to LEDs or 7 segments so that you can see what is going on.

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. (Question: Why do you need to include the ones-place carry bit?)

Part 4: Subtraction (or Inverting and Adding One)

Instead of building a dedicated hardware to for subtraction we will use the additive inverse (that is, we will take the 2's compliment of the user input). To do so, we need to invert our input and add one to the result. Build the circuit that allows you to toggle between the normal and the inverted number with a switch. (Question: how can we easily add one to the result?)

If you aren't sure how to perform a conditional inversion, here are a couple of ideas: You could find a gate-level implementation by writing up a truth table. Alternatively, you could use a "Mux" component to select between the original bit and its inverse.

You also need to extend the length of the user input from the 4-bit keypad output to the 6-bit adder. Think carefully about how to handle the extra bits!

Lab Write-up

As always, be sure your write-up contains:

- Appropriate headers
- Sensible lab writeup structure (eg: purpose, methods, results, analysis)
- Describe what you learned, what was surprising, what worked well and what did not
- Readability

Discuss issues you had building the circuit. Describe what you added to the minimum specifications 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 if you like to type it up but please submit a plain text file.

Collaboration: You are allowed to discuss this lab with other students on this lab *but all the work must be your own.*

Deliverables:

2 files required in lab2 folder in repository with commit id submitted to the google form:

- Lab2.lgi
- README.txt

Point Breakdown

5 pts: Input

10 pts: Inverter

20 pts: Adder

20 pts: Register

5 pts: Output

5 pts: Write up (README.txt)