Prologue

The exercises contained in the simulation portion of this digital-design course are meant to lead you step-by-step through the construction of an elementary microprocessor. At this point in your digital-design course, this may seem like an immense feat, but don't worry, we'll take it one step at a time, and before you know it, you'll have constructed the complete simulation. Some people feel overwhelmed by the prospect of having to "build a complete microprocessor." Let me assure you that more than 4,000 students, with no knowledge of digital circuits, have started these laboratory exercises (while taking a course in digital design) and completed them successfully – and you will too!

To complete the microprocessor design, you will need to complete four simulation labs. In each of the four laboratory exercises, you will either build pieces that will make up your microprocessor, or put the pieces together to form major sub-units of your microprocessor. In **Simulation Lab 1 and 2**, you will systematically design and simulate some elementary functions from AND, OR, NAND, NOR, and NOT gates that will be building blocks for subsequent exercises. In **Simulation Lab 3**, you will combine the circuits built in Simulation Lab 1 and 2 to form an Arithmetic and Logical Unit (ALU), which handles all the data manipulation operations in a microprocessor. In **Simulation Lab 4** you will first add memory and other simple circuitry to your ALU, to form a 'brainless' microprocessor. You will act as the brains of this microprocessor and control it to perform actions on stored data. Then you will add control circuitry to complete your microprocessor design.

As you work your way through these simulation exercises, notice that a large fraction (greater than 95%) of all the devices used are AND/NAND, OR/NOR, or NOT gates. At the completion of these exercises it is hoped that you will realize that all digital computers are merely complex combinations of these simple primitive logic devices.

In these exercises you are asked to build a simulation of a microprocessor rather than a hardware realization. Building a hardware realization of the microprocessor as prescribed in these exercises from discrete IC's (integrated circuits) requires several hundred IC's. The complexity of wiring together several hundred IC's puts a hardware realization beyond the scope of an introductory laboratory exercise. Further, when any circuit of this degree of complexity is to be constructed, a simulation of it is always completed first. Simulating a design before building it with hardware allows the conceptual design to be easily, reliably, and quickly built, tested, and modified. In practice, all complex systems are simulated before they are built. It is hoped that as you build the simulations in the subsequent laboratory experiments you will define for yourself the role that simulation plays in the construction of any large complex system.

Simulation Lab 1: Half Adder, Increment & Two's Complement Circuit

Prerequisites: Before beginning this laboratory experiment you must be able to:

- Use Logisim. (Have watched Logisim tutorial videos and completed Simulation Lab 0.)
- Interpret and construct schematic diagrams from Boolean algebraic expressions using AND/OR/NOT logic gates.

Equipment: Personal computer and Logisim.

Objective: In this experiment, you will gain some experience building and debugging combinational logic circuits using Logisim.

Outcomes: When you have completed the tasks in this experiment you will be able to:

- Describe the truth tables that characterize the addition of two single bit numbers.
- Write the Boolean algebraic expressions that characterize the sum and carry functions for the half adder.
- Build and debug a simulation of a circuit that will perform the half-adder operation.
- Build and debug a simulation of a circuit that will perform the 4-bit increment operation.
- Build and debug a simulation of a circuit that will perform the 2's-complement operation.

Introduction

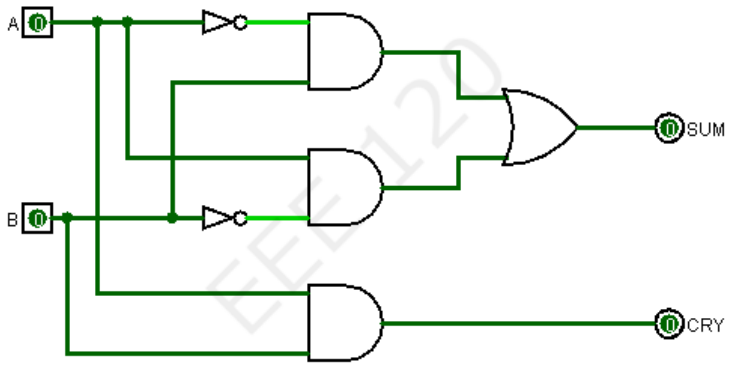
In this laboratory exercise you will be building three circuits from primitive logic gates that will provide part of the functionality of a complete microprocessor. These circuits will allow you add two 1-bit numbers (i.e., the half-adder function), increment a 4-bit number, and perform the two's complement operation on a 4-bit number. In this exercise, you will also be modularizing the circuits you build so that they can be easily used and their functions easily understood. Creating modules (a.k.a. subcircuits) from complex circuits and using these modules to create more complex modules is a powerful strategy that will allow you to create complex circuits whose modularized diagrams can easily be interpreted and used.

Task 1-1: Build and Test the 1-Bit Half-Adder

In lecture, you learned how to build a half adder. The truth table of a 1-bit half-adder is shown in Figure 1(a). Now use Logisim to build and test the 1-bit half-adder circuit as shown in Figure 1 (b).

A	B	CRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table



(b) Circuit diagram

Figure 1. 1-bit half adder

To test this circuit, toggle the input pins through every (A, B) combination and compare the SUM and CRY outputs observed with those prescribed by the truth table of Figure 1. Record the results of these tests in your lab template.

If the outputs your circuit generates agree with the outputs specified by the truth table, then your circuit is working correctly. If they do not agree, you have a problem with your circuit and you need to check that the connections have been made correctly. Once you are convinced that your circuit is working properly, save your file. Give the file a meaningful name such as **ha_1.circ**.

Task 1-2: Imbed the 1-Bit Half Adder in a Subcircuit

After you have built and tested the 1-bit half adder circuit, the next task is to create a subcircuit symbol as shown in Figure 2. Label your subcircuit 'HA_1'. 'HA' stands for 'half adder' and the '1' indicates that the circuit operates on 1-bit operands. (In subsequent laboratory exercises you will build circuits that operate on binary numbers with multiple bits; hence, it is important to specify the bit length in labeling each subcircuit so that the length of the operands accepted by each subcircuit can be determined simply by observing the label.)

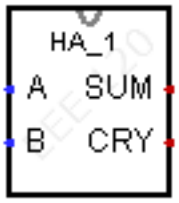


Figure 2. Subcircuit symbol for 1-bit half adder

Creating subcircuits is an important technique universally used in the creation of digital circuits. A subcircuit allows us to suppress the details of how a circuit operates and instead understand the subcircuit at the function level. By suppressing the internal details, subcircuits also greatly simplify schematic diagrams. We will see that as our circuits grow in complexity, using subcircuits is not only important but also necessary. For complex circuits, getting an overview of how the circuit functions without the use of subcircuits is next to impossible. The subcircuit symbols we create in this exercise will be used in schematics in subsequent laboratory exercises. If you keep your notation the same as we use here, you will have no problem using and combining your subcircuits using the schematic diagrams in subsequent laboratory exercises. So, for your own sake, follow the notation used in this lab instruction.

Task 1-3: Build a 4-Bit Increment Circuit

An arithmetic operation that our microprocessor will use in its program-counter circuitry is the increment operation. The increment circuit we will need must add 1 to a 4-bit number. A 2-bit increment circuit built from half-adder subcircuits is shown in Figure 3. Justify to yourself that the circuit will increment a 2-bit number by 1 if INC input is set to 1, and the circuit will increment the 2-bit number by 0 (basically keep the 2-bit number unchanged) if INC input is set to 0.

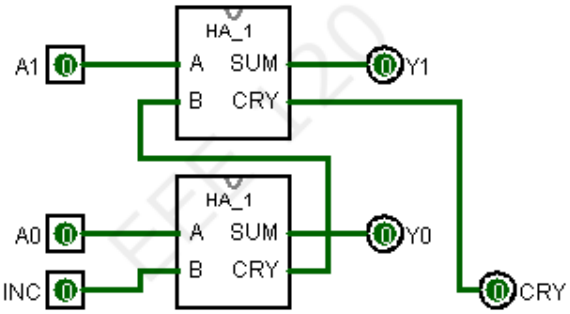


Figure 3. 2-bit increment circuit

In a similar fashion, we can use four 1-bit half adders to implement a 4-bit increment circuit. A 4-bit increment circuit will accept a 4-bit binary number as its operand and be controlled by the increment control, INC. (Label the increment control as INC in your design). Your circuit will produce a 4-bit binary number and a carry as output. If the input control, INC, is low, the output number will be the same as the input number. If the input control is high, the output will be one more than the input.

Using your understanding of the increment function and the 1-bit half adder, build a **4-bit increment circuit** using four 1-bit half adders. For naming inputs and outputs, see Figure 4 for reference. (Once you finish constructing the increment circuit using half-adder subcircuits, imagine what your circuit would look like if you did not construct it from subcircuit blocks. Your design is a good example of how we will use simple subcircuits to create more complicated circuits.)

Task 1-4: Imbed 4-Bit Increment Circuit in a Subcircuit

After creating the 4-bit increment circuit, put it in a subcircuit using the symbol shown in Figure 4, label it "INC_4".

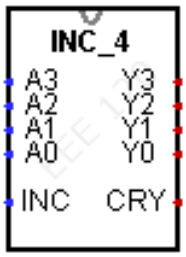


Figure 4. 4-bit increment circuit subcircuit symbol

Task 1-5: Test the 4-Bit Increment Subcircuit Using Hex Numbers

Let's test the 4-bit increment circuit you constructed in Task 1-3 and Task 1-4. Determining the minimum number of tests needed to guarantee that a circuit is working properly is a difficult problem – and beyond the scope of an introductory course. An alternative testing technique is to observe the output for **every** input combination. Successful completion of such an exhaustive test applied to any circuit will guarantee that it is operating correctly.

Test your 4-bit increment subcircuit by connecting the four inputs of your 4-bit increment circuit to a 4-bit input pin, and the four outputs of your 4-bit increment circuit to a hex digit display as shown in Figure 5. You will need to use buses and the splitter tool. Now verify that the 4-bit increment circuit works in the desired fashion using hex arithmetic to check your results. Note that this requires 32 tests. (Why?) Record your test results and include them in your lab template.

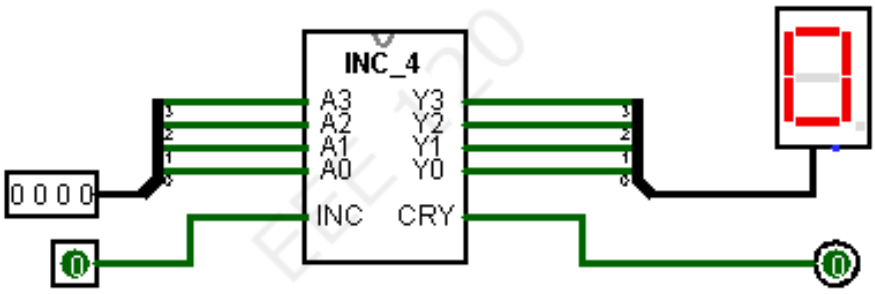


Figure 5. 4-bit increment circuit subcircuit testing set up

Task 1-6: Use INC_4 to Perform Two's-Complement Operation

In lecture, you learned about using the two's-complement number system to represent both positive and negative binary numbers. In the two's-complement number system, the sign of the number becomes an integral part of the

representation of the number. Most computer arithmetic is performed using numbers in two's-complement form – our microprocessor will be no different. In our microprocessor, we will want the capability of negating two's-complement numbers. Negation of two's-complement numbers is handled by complementing each bit of the number and adding 1 to it. The circuit shown in Figure 6 uses the increment subcircuit you created in Task 1-4 along with four inverters to perform the two's-complement operation.

Note that by connecting the 'INC' input in Figure 6 to +5 Volts, the circuit will always perform the two's-complement operation. Build the circuit in Logisim and verify that it performs the two's complement operation. In our earlier test of the increment circuit, we exhaustively tested every input combination. As you gain experience building and testing circuits you will realize that once a circuit works correctly for certain input combinations, it is **likely** to work correctly for all input combinations. It is left to you to decide what constitutes a sufficient set of tests to make it likely that the two's-complement circuit is operating correctly. The test should be thorough enough to prove that the circuit works beyond reasonable doubt. One set of test inputs is not enough to test the circuit. Justify in your lab template why successful completion of your tests makes it likely that your circuit is operating correctly. (Hint: Be sure to test the carry). Remember to record the details of your tests in your lab template.

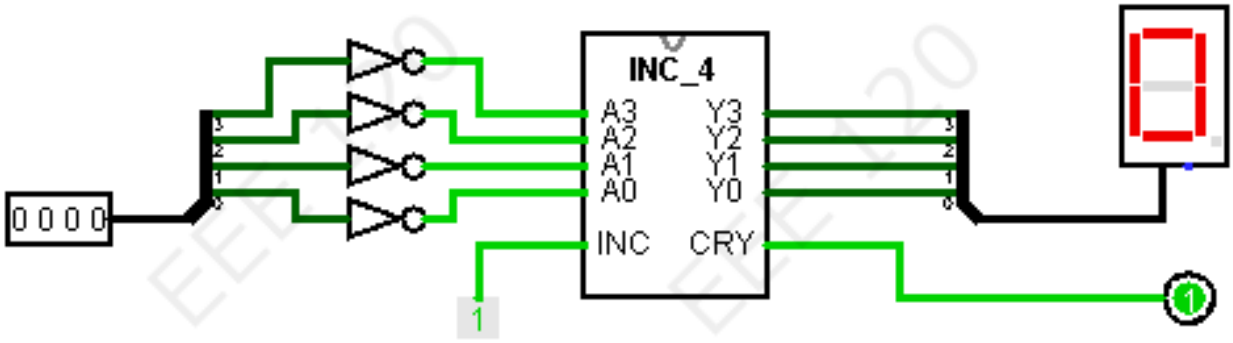


Figure 6. A circuit that performs the two's complement operation including testing set up

Once you have tested this circuit and have shown that it works correctly, save it. In Simulation Lab 3 you will be modifying this circuit and using it as one part of the arithmetic and logic unit for our microprocessor.

Task 1-7: Backup Your Files

Over the years many students have had their libraries and files destroyed by many causes. Because these simulation exercises build upon one another, some students have needed to redo three or more simulation labs just to be able to perform the most recent one assigned. Take a moment and back up you libraries and circuit files to a new directory and name it 'SimLab 1'. Do this using different media after every lab and you just may save yourself much unnecessary work.