COMP 273, Fall 2020, Assignment 2

School of Computer Science, McGill University
Available On: Friday, October 2nd, 2020.
Due Date: Friday, October 16th, 2020 by 11:59 pm.

By handing in your solutions using *mycourses*, you declare that you have followed the assignment submission instructions at the end of this assignment. **Late policy: 10% off of the total marks, per day late, for up to 2 days. If submitted 48 hrs or more after the deadline, your assignment will not be accepted.**

Question 1: Full 4-Bit Adder-Subtractor (25 points)

You should design your circuit using the provided template circuit file (*Four_Bits_Add_Sub.circ*)

In class, we saw a circuit diagram for a 1-bit full adder. In this question you will build a simple **4-bit adder-subtractor** that implements two different functions (addition and subtraction). You must build your circuit using only the basic gates provided in the built-in library, specifically, **AND, OR, NOT, XOR and XNOR**. You can change the properties of these gates as necessary (e.g, you may use 3-input gates if you wish).

You will also need to use wiring, such as splitters, to organize your implementation. To complete the objectives of this assignment, you must organize your solution into sub-circuits using the names and labels specified below (leave the main circuit empty).

1. You will need to also edit the appearances of some sub-circuits to better organize your solution. But, be careful not to change the sub-circuits' names, and input/output labels in the starter project file.

2. You are free to create additional sub-circuits with custom appearances and then use them in the starter sub-circuits. The sub-circuits for the main objectives, and in some cases the inputs and outputs, are already set up for you in the starter project file. Make sure that you are filling the starter project and its corresponding sub-circuits.

(A) Warm up (5 points):

Implement a one bit full adder in the "**Add_1Bit**" sub-circuit that takes **A, B** and **Cin** as single-bit inputs and produces the **Sum** and **Carry-out (Cout)** functions as outputs. Note that the sub-circuit appearance has already been created for you in the starter code, where Sum is labeled S for short.

(B) Build a 4-bit adder/subtractor (20 points):

You are given the starter sub-circuit ("**Add_Sub_4Bits**") which you will build on, according to the instructions below.

- You already have implemented a 1-bit adder in the sub-circuit "**Add_1Bit**". Implement a **4-bit adder-subtractor** in "**Add_Sub_4Bits**" by using four instances of "**Add_1Bit**" and additional circuitry. **(15 points)**

  Note that there is a control input signal, "Add_Sub". Whenever it is asserted to '0', the circuit ("**Add_Sub_4Bits**") should perform 4-bit addition (A + B). Whenever it is set to '1', the circuit should perform binary subtraction (A - B). In both cases the result should be on the output "R". Note that the inputs are both 2's complement numbers.

- Also, your implementation must be able to handle overflow[1] and zero[2]. **(5 points)**

- As an example, if A = +1 = 0001, B = -7 = 1001, and we perform the addition operation, then R = -6 = 1010, Overflow = 0, and Zero = 0.

[1]When overflow happens in the operation (add/sub), the "Overflow" output signal, in your circuit, should be asserted to '1'; otherwise it should be '0'

[2]When the result ("R") is all zeros, "0000", the "Zero" output signal should be asserted to '1'; otherwise it should be '0'.

Question 2: 16 Nibble RAM (50 points)

You should design your circuit in the provided template circuit file (RAM_16_Nibbles_Read_Write.circ).

In this question you will build a simple 16 Nibble RAM that implements two different functions (**read** and **write**). A Nibble is a group of 4 bits. You must build your circuit in the logisim-evolution using only the basic gates provided in the built-in library, specifically, **AND, OR, NOT, XOR, XNOR, and D flip-flops**.

You may set the properties on these gates as you wish, such as changing the the number of inputs, or the number of data bits. However, you must implement your solution in the starter project template we have provided, following the instructions below. You must organize your solution into sub-circuits using the names and labels specified below (leave the main circuit empty).

1. You will need to edit the appearances of some sub-circuits to better organize your solution. But, be careful not to change the sub-circuits' names, and input/output labels in the starter project file!
2. You are free to create additional sub-circuits with custom appearances as you see fit and then use them in the starter sub-circuits. Be sure to use the starter project sub-circuits though.
3. Do not use more complicated built-in modules from the logisim-evolution library (such as Mux, Decoder, Registers, Counters, RAM, etc). If you need any such functionality you should implement it by yourself (preferably in sub-circuits).

(A) Build a 4-bit register (nibble) (15 points):

Implement the nibble (4 bit) register using flip-flops. The corresponding starter sub-circuit (nibble) is provided, with proper input /output ports. Your circuit should perform read and write operations in one clock cycle, as follows. When writing is enabled (**Read_Write = 1**, and **En = 1**), the value of Data should be saved in the flip-flops.

This value should appear in the **Nibble_Out** within one clock cycle. However, if **En = 0**, no write operation should be performed. In other words, the last value of Nibble should be presented as **Nibble_Out**.

When **Read_Write = 0** the value of the Nibble should be observed on the **Nibble_Out** no matter what the value of **En** is. You may add a Clock source in the nibble circuit to check/test its functionality. Do not forget to remove it later. Eventually Nibble will receive its Clock signal from the higher module (16-nibble RAM).

(B) Build a 16-nibble RAM (35 points):

Here you will use the nibble register you build in part A. You must use the starter sub-circuit ("**RAM_16_Nibbles**") and add additional circuitry, to implement the following functionality.

- Your circuit should perform read and write operations in one clock cycle. In other words, if we want to read from RAM (**Read_Write = 0**), with the next Clock tick, the value of the nibble that the address bits indicate should be observed in **RAM_Out**.

- Your RAM circuit should read a 4-bit piece of data as input and set this as the value of the nibble that is defined by the address bits. For example, if Data = 1000 and Address = 0011, then in the next Clock cycle the value of 4th nibble of the RAM should become 1000, and also this value should be displayed on the output.

- The entire circuit should have just one single Clock signal, that is located in **RAM_16_Nibbles** module. If the nibble module requires a Clock signal to work properly, pass the Clock signal from **RAM_16_Nibbles** to the Nibble sub-circuit as an input.

Question 3: Midpoint and Slope of a Line Between Two Points (MIPS) (25 points)

You should solve this question by completing the *points.asm* template in the Assignment 2 folder

This is a warm-up question to introduce you to MIPs. More advanced questions will follow in future assignments.

Given the coordinates of two points (x1, y1) and (x2, y2) in a plane, the midpoint between the two points can be calculated as:

$$midpoint = \frac{x1 + x2}{2}, \frac{y1 + y2}{2}$$

and the slope calculated using the formula:

$$slope = \frac{y2 - y1}{x2 - x1}$$

You are to write a program that prompts the user to enter 4 integers $x1, y1, x2$ and $y2$. The program should then calculate the midpoint between the two points $(x1, y1)$ and $(x2, y2)$ and also the slope. You can use the *div* instruction. You don't have to create a separate subroutine for this question, just use the "main" that is provided. However, please comment your code.

The output of your program should be in the following format:

The midpoint is: a,b

The slope is: c

ASSIGNMENT SUBMISSION INSTRUCTIONS

Each student is to submit his or her own unique solution to these questions, electronically, in *mycourses*. By handing in this assignment, you declare that the work you are submitting is your own.

1. You have been given starter code in the Assignment 2 folder. Modify this code to answer the questions on this assignment. The logisim circuits you submit must be saved as logisim-evolution files, so we can test them. Your solution to the MIPs question should be submitted as a text file (points.asm) file, which should compile and run in MARS. If you wish, you can also add a PDF document to explain your designs for questions 1 and 2.

2. The logisim circuit(s) must run under logisim-evolution, to be graded. We will assume that you have tested it.

3. Zip your PDF, the two logisim-evolution and the points.asm assembly files into a single file and rename it with your student ID number, e.g., 260763964.zip. Ensure that you use only the *zip* format and no other compression software e.g., .rar, .7z, etc. Make sure that you submit a single file (the zipped file), not many files.

4. Submit this single compressed file on mycourses under Assignment 2.

5. Hints, suggestions and clarifications may be posted on the discussion board on *myCourses* as questions arise. Even if you don't have any questions, it is a good idea to check the discussion board.

6. Once you have submitted your assignment, download the zip file you uploaded and check that it is indeed what you intended us to grade. This step is critical because a non-trivial number of you will submit the wrong zip file, or a corrupted version. You cannot submit a corrected file later, i.e., after the submission deadline and the two day "late" window have passed.