CSE 390B Midterm Exam, May 6th, 2021

You will have 50 Minutes to complete the exam.

Rules:

- Except for the reference sheet given as a separate file, the exam is closed-note, closed-book.
- There are **100** points distributed unevenly across **5** questions.

Tips:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all the questions.
- If you have questions, ask via a private chat message to Porter, Eric, or Margot.
- Try to relax and take deep breaths.

1. (**25** points) In this problem, you will build Boolean circuits with three inputs and one output. You may use two-input And and Or gates, and single-input Not gates.

   Traffic lights are used to automatically control traffic at road intersections using three lights (usually, red, yellow, and green). Red indicates stop, yellow indicates slow down, and green indicates go. In some areas around the world, the red and yellow lights can be on at the same time to indicate stop, but prepare to go. The following are valid traffic light combinations:

Red: stop

Red and yellow: stop, but prepare to go

Yellow: slow down

Green: go

   Traffic lights can occasionally malfunction, putting vehicles and pedestrians in danger. Design a circuit called `TrafficLightValid` that outputs 1 if the traffic light combination is valid and a 0 if the traffic light combination is invalid. An input of 1 indicates that the light is on and an input of 0 indicates that the light is off.

   a. Write a truth table for the circuit. You can call the inputs r for red, y for yellow, g for green, and the output v for if the traffic light combination is valid.

r	**y**	**g**	**v**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

b. Based on the truth table you filled out, write a boolean expression for the output.

Two possible expressions:

v = (~r & ~y & g) | (~r & y & ~g) | (r & ~y & ~g) | (r & y & ~g)

v = (g & ~r & ~y) | (~g & r) | (~g & y)

c. Implement the boolean expression by either completing the following HDL template or drawing a circuit diagram using the standard symbols. You do not need to do both, either will suffice.

```
CHIP TrafficLightValid {
    IN r, y, g; // red, yellow, green
    OUT v;      // 1 if the traffic light combo is valid

    PARTS:
    // Your code here:
    Not(in=r, out=notr);
    Not(in=y, out=noty);
    Not(in=g, out=notg);
    And(a=g, b=notr, out=gandnotr);
    And(a=gandnotr, b=noty, out=out1);
    And(a=notg, b=r, out=out2);
    And(a=notg, b=y, out=out3);
    Or(a=out1, b=out2, out=out4);
    Or(a=out4, b=out3, out=v);

}
```

2. (**10** points) Explain at a high level in 1-2 English sentences how you would build a circuit that takes a two's-complement 16-bit number and outputs 1 if and only if it is equal to 1. In this problem, you can use only And, Or, and Not gates. Your And and Or gates can take any number of inputs. You may include a picture if you find it helpful.

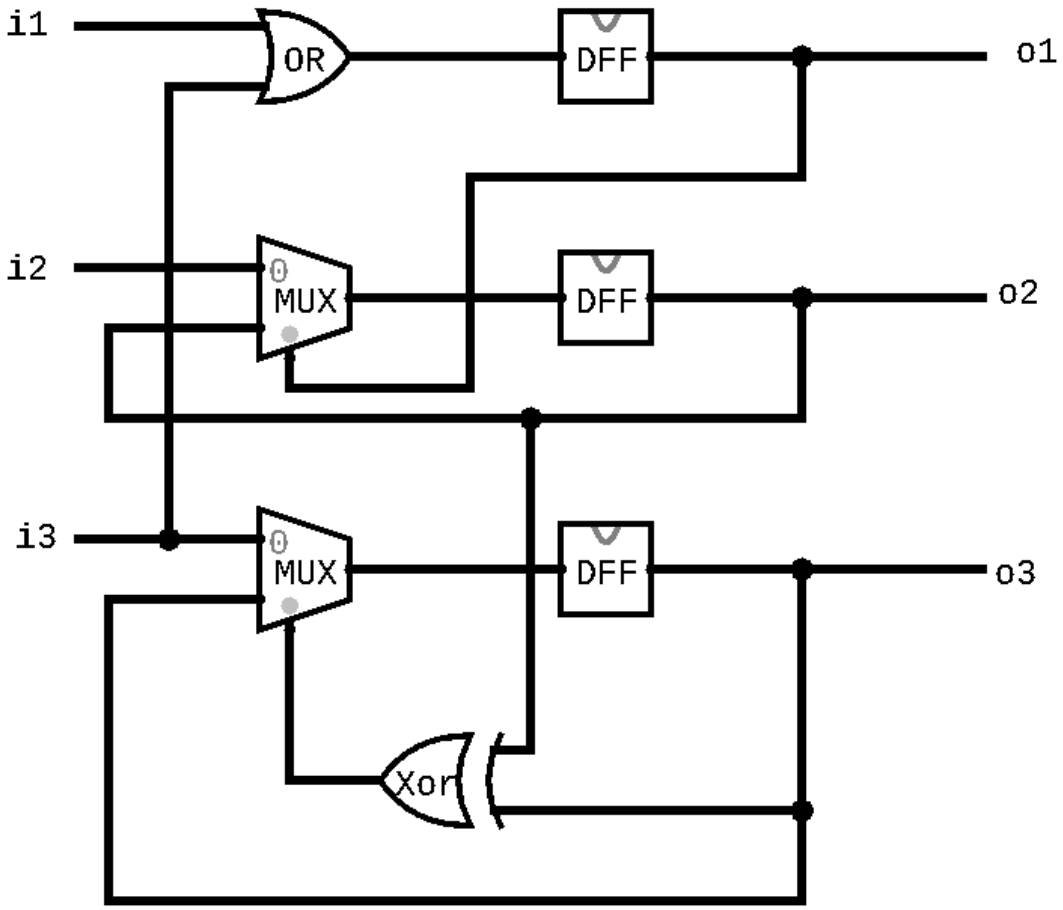
   Use an Or gate on the 15 most significant bits to see if any of them are 1. Then Not the output of that Or gate, such that it is 1 if they are all 0 and 0 if they are all 1. Finally, And the output from the Not gate with the least significant bit to determine if the value is equal to 1.

   Note: could also Not all of the 15 most significant bits individually and then connect them with an And gate instead of using the Or/Not gates.

3. (**25** points) In this problem, use only two-input muxes, basic flip-flops, and combinational logic.

   Draw the following circuit specification using conventional notation, omitting implicit clock signals.

   - It takes three data inputs (`i1`, `i2`, and `i3`)
   - It has three outputs (`o1`, `o2`, and `o3`)
   - Each output at time `t + 1` is defined as follows:
     - `o1(t + 1) = i1(t) | i3(t)`
     - ```
       if(o1(t)): o2(t + 1) = i2(t)
       else:      o2(t + 1) = o2(t)
       ```
     - ```
       if(o2(t) == o3(t)): o3(t + 1) = i3(t)
       else:               o3(t + 1) = o3(t)
       ```



4. (**20** points) Here is a sample program given in high-level pseudo code:

```
if (R0 > 0 || R1 > 0) {
    R2 = R0 - R1
} else if (R0 == 0) {
    R2 = R1
} else {
    R2 = R2 + 1
}
```

Write an equivalent Hack assembly program using the virtual registers R0, R1, and R2 (remember that these correspond to the slots in memory at address 0, 1, and 2, respectively).

```
    @R0
    D = M
    @IF
    D; JGT
    @R1
    D = M
    @IF
    D; JGT
    @R0
    D = M
    @ELSEIF
    D; JEQ
(ELSE)
    @R2
    M = M + 1
    @END
    0; JMP
(IF)
    @R1
    D = M
    @R0
    D = M - D
    @R2
    M = D
    @END
    0; JMP
(ELSEIF)
    @R1
    D = M
    @R2
    M = D
(END)
    @END
    0; JMP
```

5. (**20** points) Here is a buggy Hack assembly program:

```
01.       @1
02.       M = 0
03.       @2
04.       M = 1
05.   (L0)
06.       @0
07.       D = M
08.       @2
09.       // PART I
10.       D = M - D
11.       @END
12.       D; JGE
13.       @2
14.       D = M
15.       M = M + 1
16.       // PART II
17.       @1
18.       M = D + M
19.       @L0
20.       0; JMP
21.   (END)
22.       @END
23.       0; JMP
```

Here is the state of part of memory ***before*** the above code runs (we will use this to answer the questions below):

Address	**Value**
0	3
1	10
2	95

a. Walk through the code starting with the state of memory given above. Indicate the value of the registers M, A, and D, at each of the following locations commented "PART #" *the first time you reach that location when executing the code*.
   i. Values of M, A, and D when first reaching comment w/"PART I"
   M=1, A=2, D=3
   ii. Values of M, A, and D when first reaching comment w/"PART II"
   M=2, A=2, D=1

b. Again starting with the state of memory given above, what are the values stored at address 0, address 1, and address 2 in memory after the above code runs to completion, (enters the END infinite loop)?
   i. Value stored in address 0: **3**

   ii. Value stored in address 1: **3**

   iii. Value stored in address 2: **3**

c. The above code attempts to sum the numbers 1 through the value stored in memory at address 0 (inclusive) and store the result in memory at address 1. It attempts to be roughly equivalent to the following pseudo code:

```
ram[1] = 0
for (i = 1; i <= ram[0]; i++) {
  ram[1] += i
}
```

   As stated above, the program is buggy, but it turns out it can be fixed by changing one line! Give the line # for the line of code that you would change to fix the program and what you would change it to.
   i. Line # that should be fixed: **12**

   ii. New line of code: **D; JGT**