CSE 390B Midterm Exam, February 11th, 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 **6** 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, Melissa, or Margot.
- Try to relax and take deep breaths.

1. (**20** 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.

   Leslie's cat Kaelo will only eat what's on his plate if it is one of the following: 1) only apples and cheetos 2) only butterfingers 3) only butterfingers and cheetos. Kaelo will refuse any other combination of those foods served to him. Design a circuit `WillKaeloEat` that outputs 1 if he eats and 0 if he refuses for all combinations of apples, butterfingers, and cheetos (for each of these inputs, 1 indicates the item is on the plate, and 0 indicates it is not on the plate).

   a. Write a truth table for the circuit. You can call the inputs `a` (apples), `b` (butterfingers), and `c` (cheetos), and the output `d` (will Kaelo eat).

a	**b**	**c**	**d**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

   b. Based on the truth table, determine a boolean expression for `d`.

      d = (~a & b) | (a & ~b & c)

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 WillKaeloEat {
    IN a, b, c; // apples, butterfingers, cheetos
    OUT d;      // whether or not Kaelo will eat

    PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=nota, b=b, out=notaandb);
    And(a=a, b=notb, out=aandnotb);
    And(a=aandnotb, b=c, out=aandnotbandc);
    Or(a=notaandb, b=aandnotbandc, out=d);
}
```

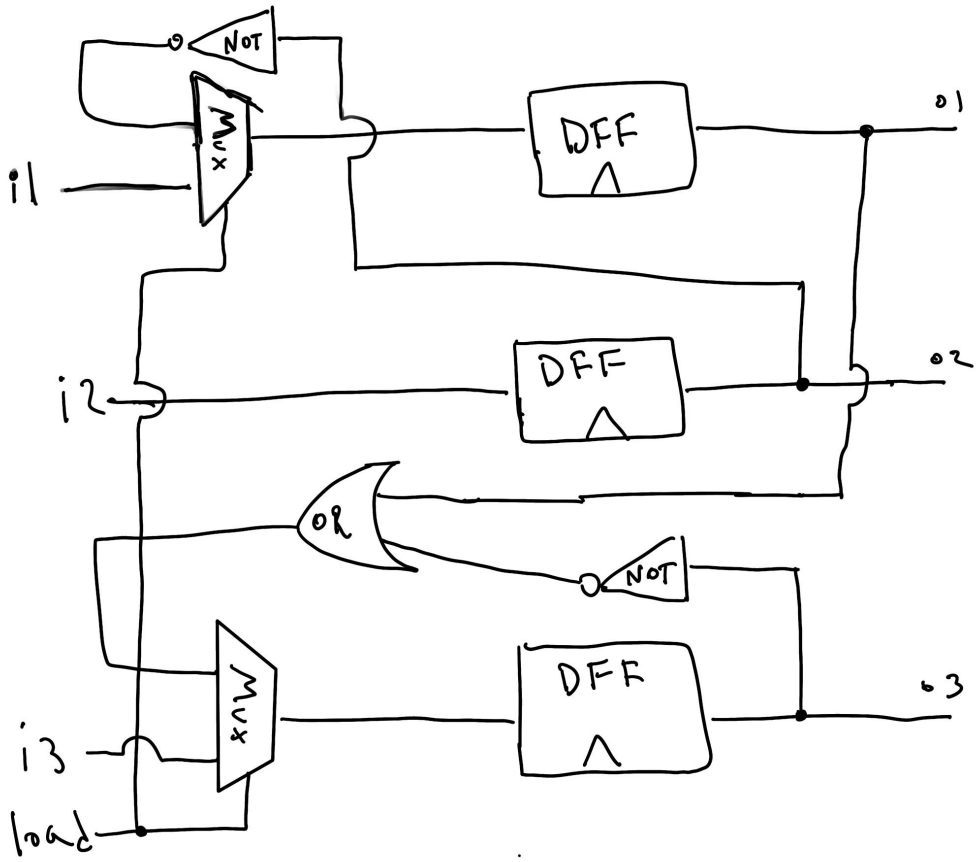
2. (**5** points) Explain at a high level in 1-2 English sentences how you would build a circuit that takes a two's-complement 16-bit number and outputs 1 if and only if the number is divisible by 2 (negative numbers can be divisible by 2, e.g. -6, and 0 is also considered divisible by 2). In this problem, 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. (Hint: write out the binary of different numbers if you're stuck).

   Take the least significant bit (index 0) of the 16-bit number, which is 0 when the number is divisible by 2 and 1 otherwise, and pass it through a Not gate, so that it is 1 when the number is divisible by 2 and 0 otherwise.

3. (**20** 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 four inputs: three data inputs (`i1`, `i2`, and `i3`) and a control signal `load`.
   - It has three outputs (`o1`, `o2`, and `o3`)
   - When the control signal `load` is set, the output at time `t + 1` should be the input at time `t`, with `o1` matching `i1`, `o2` matching `i2`, etc.
   - When the control signal `load` is not set, the output at time `t + 1` is defined as follows:
     - `o1(t + 1) = ~o2(t)`
     - `o2(t + 1) = i2(t)`
     - `o3(t + 1) = o1(t) | ~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 = R0 & R1
} else {
  R2 = (-R0) | R1
}
```

Write an equivalent Hack assembly program using the virtual registers R0, R1, and R2.

```
// One possible solution
    @R0
    D = M
    @ELSECASE
    D; JGE
    // IF or ELSEIF case (R0 < 0)
    @R1
    D = M
    @ELSEIFCASE
    D; JGE
    // IF CASE (R1 < 0)
    @R0
    D = D + M
    @R2
    M = D
    @END
    0; JMP
(ELSEIFCASE)
    @R0
    D = D & M
    @R2
    M = D
    @END
    0; JMP
(ELSECASE)
    D = -D
    @R1
    D = D | M
    @R2
    M = D
(END)
    @END
    0; JMP
```

5. (**15** points) Here is a buggy assembly program that takes two inputs, R0 and R1, performs a calculation, and stores the outputs in R2 and R3.

```
    @R2
    M = 0
    @R3
    M = 0
(L1)
    @R1
    D = M
    @R0
    D = M - D
    // PART I
    @L2
    D; JLT
    @R0
    M = D
    @R2
    M = M + 1
    // PART II
    @L1
    0; JMP
(L2)
    @R0
    D = M
    @R3
    M = D
(END)
    @END
    0; JMP
```

   a. Walkthrough the code with the input values R0 = 9 and R1 = 4. Indicate the value of the registers M, A, and D at each of the 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 = 9, A = 0, D = 5
      ii. Values of M, A, and D when first reaching comment w/"PART II"
      M = 1, A = 2, D = 5
   b. For the same input values R0 = 9 and R1 = 4, what are the values of R2 and R3 when the program finishes (enters the END infinite loop)?
      R2 = 2, R3 = 1

c. Unfortunately this program is buggy and in some cases and does not terminate (i.e. loops infinitely *before* reaching the END infinite loop). Give 2 inputs R0 and R1 for which this program does not reach the END label.

Multiple possible answers. The code goes into an infinite loop when R0 >= 0 and R1 == 0.

6. (**20** points) For each of the following changes to the Hack architecture, give one benefit and one downside to the change:

   a. This change would provide hardware implementations for the following operations: Multiply, Divide, and Xor.
      i. One benefit
         Multiple possible answers. One benefit is that the hardware implementations are likely to be faster than the software implementations in Hack.
      ii. One downside
         Multiple possible answers. One downside is that the ALU will likely have to be heavily redesigned and less elegant to accommodate the new operations (for instance, can't implement Multiply and divide using the zx, nx, zy, ny, f, no trick).

   b. Recall that in Hack there are two types of instructions, A instructions and C instructions. Also recall that our devices, Screen and Keyboard, are memory-mapped (controlled by writing to particular regions in RAM). This change would separate the Keyboard and Screen into their own individual chips (i.e. no longer a part of our RAM address space) and add 2 new types of instructions, one for interacting with the screen and one for interacting with the keyboard.
      i. One benefit
         Multiple possible answers. One benefit is that normal writes to memory cannot accidentally mess with the screen and keyboard anymore because they don't use the same instruction type.
      ii. One downside
         Multiple possible answers. One downside is that new instruction specifications have to be introduced for the new screen and keyboard instructions, which makes things more complicated both for the hardware implementers and assembly programmers.