CSE 390B: Building Academic Success Through Bottom-Up Computing 2024 Winter

Midterm Solutions February 9th, 2024

Name: _____

UW NetID: _____

Instructions:

- Make sure you have included your name (first & last) and your UW NetID on this page.
- When you finish the exam, turn in your exam to the course staff.
- You will have 60 minutes to complete the exam.
- Questions are not necessarily in order of difficulty.
- This exam is closed-note, closed-book (except for the reference sheet).
- This exam contains 100 points distributed unevenly among five questions (some with multiple parts).

Advice:

- Read each question carefully. Understand a question before you start writing.
- When applicable, elaborate on your answer, explain your thought process, and write down the intermediate steps for possible partial credit. However, clearly indicate what your final answer is.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all the questions.
- If you have questions, please raise your hand, and the course staff will get to you shortly.

Technical Details:

- You may specify Boolean operations using symbols or words (e.g., for the And gate, you may use the symbol & or “And”).
- The ^ symbol refers to an Xor gate.
- When using a Mux or DMux gate, explicitly show or describe the select bits that the inputs are connected to (e.g., the a input of the Mux is connected to the select bit of 0).

Question	1	2	3	4	5	Total
Possible Points	25	10	20	20	25	100

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

   Consider the hexadecimal number 0x390B. Your task is to design a circuit called DigitsOf390B that returns 1 if the three-bit binary input in unsigned binary matches the three right-most bits from any of the hexadecimal digits from 0x390B in unsigned binary. For example, the first digit from 0x390B is 3. 3 in unsigned binary is 0b011, so the output for the input 0b011 in the truth table would be 1. If any of the hexadecimal digits exceeds the number 7 in unsigned binary, use that number modulo 8 in unsigned binary (modulo means remainder; e.g., 13 modulo 8 is 5 because 13 divided by 8 is 1 with a remainder of 5) as the value to compare to the input bits in each row of the truth table.

   a. Given the specification of DigitsOf390B described in the paragraph above, fill in the output of the truth table for the circuit with three inputs and one output.

a	b	c	out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

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

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

c. Implement the Boolean expression you came up with from part b for the out output by completing the following HDL template or drawing a circuit diagram. You do not need to do both.

```
CHIP DigitsOf390B {
      // a, b, and c represent the binary value in unsigned
      // binary
      IN a, b, c;

      // out is the result of whether the binary input in
      // unsigned binary equals any of the hexadecimal
      // digits from 0x390B in unsigned binary
      OUT out;

      PARTS:
      // Your code or circuit diagram here:

      // Setting up nots of the input bits
      Not (in=a, out=nota);
      Not (in=b, out=notb);
      Not (in=c, out=notc);

      // Row 1 Boolean expression
      And (a=nota, b=notb, out=nota-notb);
      And (a=nota-notb, b=notc, out=row1);

      // Row 2 Boolean expression
      And (a=nota, b=notb, out=nota-notb);
      And (a=nota-notb, b=c, out=row2);

      // Row 4 Boolean expression
      And (a=nota, b=b, out=nota-b);
      And (a=nota-b, b=c, out=row4);

      // Combining all the Boolean expressions using Or
      Or (a=row1, b=row2, out=row1-or-row2);
      Or (a=row1-or-row2, b=row4, out=out);




}
```

2. (10 points) Free response questions. Describe the answers to the following questions in a paragraph.

    a. Explain what a Mux gate is, including what the inputs and output are and what it is used for in a circuit. Describe how you have used any Mux gate to implement a component of any one of the chips in this class, including what the inputs and output of the Mux gate are connected to within the chip you built and why a Mux gate is necessary for building this chip.

    A Mux gate is used for making decisions in hardware, similar to how an if/else statement works in software. A Mux gate has a select bit that chooses between two inputs and one output that outputs the input corresponding to the select bit.

    (Answers will vary for the second question). The Mux gate was used in the ALU to decide whether the function should be Add16 or And16. One input is the result of adding x and y, and the other input is the result of x and y. The select bit is the function control bit, passed as an input to the ALU. The output is passed further along in the ALU for further output processing. A Mux gate is necessary for this component of the ALU because we need to make a decision about which function to perform between x and y, and the Mux gate is used for this purpose.

    b. Explain why sequential logic is necessary for a physical, hardware circuit in the real world. There are two kinds of components in a circuit which cause limitations and therefore sequential logic to be necessary—be sure to address what these components are and what their limitation is.

    Sequential logic is necessary in real world circuits because real world circuits contain timing delays. In combinational logic, we assumed that any circuit could produce an output given a set of inputs instantly with no delay, but this assumption does not hold in reality. The two components of a circuit that cause limitations and therefore require sequential logic are wires and logic gates in physical hardware. Wires cause propagation delays (it takes a non-instantaneous time for a signal to travel from point a to point b on a wire), and logic gates cause propagation delays (it takes time for any single logic gate to calculate its output).

3. (20 points) In this problem, you may only use two-input Mux gates, DFFs, and combinational logic gates.

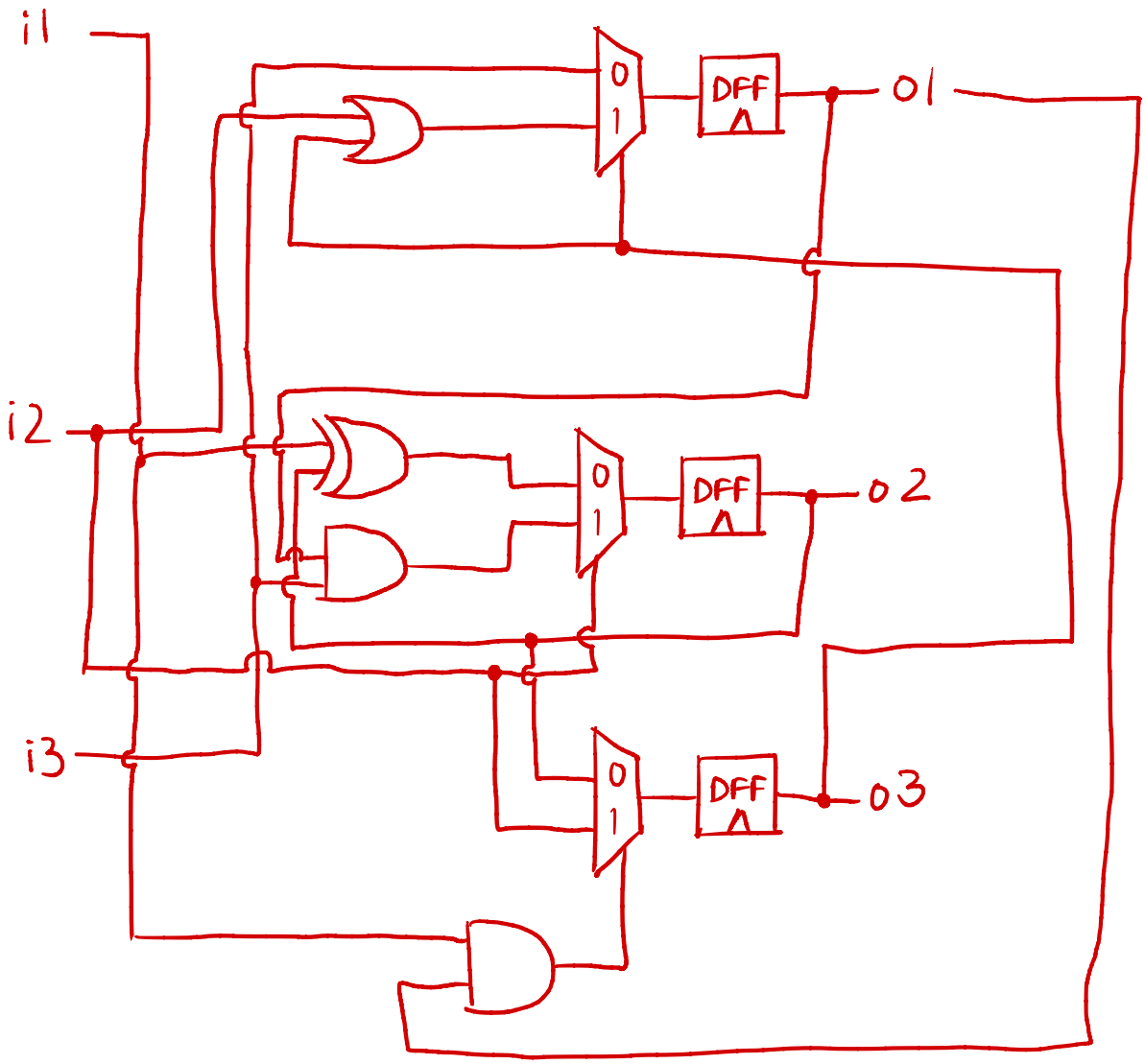
   Draw the following circuit specification using conventional notation, omitting implicit clock signals.

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

     ```
     ○ if (o3):                o1(t + 1) = o3 | i2
       else:                   o1(t + 1) = i3

     ○ if (i2):                o2(t + 1) = o1 & i3
       else:                   o2(t + 1) = i1 ^ o2

     ○ if (i1 & o1):           o3(t + 1) = i2
       else:                   o3(t + 1) = o2
     ```



4. (20 points) Below is a sample program written in high-level pseudocode. Write an equivalent Hack Assembly program using the virtual registers R0, R1, R2, and R3, each of which corresponds to the values in memory at addresses 0, 1, 2, and 3, respectively.

```
if (R1 == 3 | R2 >= -2) {
      R0 = R3
} else if (R1 + R3 < 5) {
      R0 = R1 & R2
} else {
      R0 = -2
}
```

```
// Branch 1 condition
@R1
D = M
@3
D = D - A
@BRANCH1
D; JEQ
@R2
D = M
@2
D = D + A
@BRANCH1
D; JGE
// Branch 2 condition
@R1
D = M
@R3
D = D + M
@5
D = D - A
@BRANCH2
D; JLT
// Branch 3 statement
@2
D = A
@R0
M = -D
@END
0; JMP
```

```
(BRANCH1)
   @R3
   D = M
   @R0
   M = D
   @END
   0; JMP
(BRANCH2)
   @R1
   D = M
   @R2
   D = D & M
   @R0
   M = D
(END)
   @END
   0; JMP
```

5. (25 points) Below is a Hack Assembly program with a bug.

```
01.     (START)
02.         @2
03.         D = A
04.         @R1
05.         M = D
06.     (LOOP)
07.         @R1
10.         D = M
11.         @8
12.         D = D - A
13.         // PART I
14.         @END
15.         D; JGE
16.     (CHECK_ZERO)
17.         @R1
18.         A = M
19.         D = M
20.         @UPDATE_INDEX
21.         D; JEQ
22.         @R1
23.         A = M
24.         D = 0
25.     (UPDATE_INDEX)
26.         @R1
27.         M = M + 1
28.         // PART II
29.         @LOOP
30.         0; JMP
31.     (END)
32.         @END
33.         0; JMP
```

Here is the memory state before the Hack Assembly code to the left runs (we will use this to answer later parts of this problem):

Address	**Value**
0	0
1	7
2	4
3	0
4	8
5	1
6	16
7	3
8	5

a. Trace through the code starting with the state of memory given in the table. Indicate the value of the registers A, D, and M at each of the following locations commented with “PART #” the first time you reach that location when executing the code.

   i. Values of A, D, and M when first reaching comment with “PART I”

      A = 8

      D = -6

      M = 5

   ii. Values of A, D, and M when first reaching comment with “PART II”

      A = 1

      D = 0

      M = 3

b. Starting with the state of memory given in the table, what are the values stored at address 0, address 1, and address 2 in memory after the Hack Assembly code runs to completion (i.e., enters the END infinite loop)?

      Value at address 0 = 0

      Value at address 1 = 8

      Value at address 2 = 4

c. The Hack Assembly code should scan through the memory addresses from address 2 to 7, inclusive, and set every memory value that is not equal to 0 to the value 0. The Hack Assembly program above attempts to be equivalent to the following pseudocode:

```
1.      R0 = 0
2.      for (i = 2; i < 8; i++) {
3.          if (RAM[i] != 0)
4.              RAM[i] = 0
5.      }
```

We can fix the bug in the Hack Assembly code by **modifying** a single line of Hack Assembly. Circle the section of code indicated by the symbols in the Hack Assembly program in which we should modify the line to fix the bug.

START **LOOP** **CHECK_ZERO** **UPDATE_INDEX** **END**

d. What line number would you modify to fix the bug in the Hack Assembly program, and what should the line of code be instead?

i. Line number: 24

ii. Line of Hack Assembly to fix the bug: `M = 0`

e. Does the buggy Hack Assembly program return the correct output (according to the pseudocode above) given the initial memory state shown in the table?

No, the buggy Hack Assembly program does not return the correct output. The bug is that memory values with a non-zero value do not actually get changed to zero. All the non-zero values should be modified to be zero when the buggy code does not do this.