CSE 390B: Building Academic Success Through Bottom-Up Computing 2023 Autumn

Midterm November 9th, 2023

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”).
- 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.

   Design a circuit diagram called IdentifyOnes in which the output is 1 if the binary number represented by the row equals 1 or -1 in any of the binary number representations we have learned in this class (i.e., unsigned binary, signed binary, and Two's Complement). For example, 0b1 would have an output of 1 because using the unsigned number representation, 0b1 is 1 in decimal.

   a. Given the specification of IdentifyOnes 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	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

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

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 IdentifyOnes {
      // a, b, and c represent the binary value in unsigned
      // binary, signed binary, or Two's Complement
      IN a, b, c;

      // out is the result of whether the binary input in
      // unsigned binary, signed binary, or Two's Complement
      // equals 1 or -1
      OUT out;

      PARTS:
      // Your code or circuit diagram here:









}
```

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

    a. Describe the process for converting a number from decimal to binary. Include an example of converting the number 26 into binary (please show your work).

    b. Describe what the critical path of a circuit is and how it relates to determining the length of a clock cycle.

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 (i2):              o1(t + 1) = i1 | o3
       else:                 o1(t + 1) = i3 & o2

     ○ if (o3):              o2(t + 1) = o1
       else:                 o2(t + 1) = o2 ^ i3

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

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 == -1) {
      R0 = R1 & R2
} else if (R2 | R3 < 2) {
      R0 = 3
} else {
      R0 = R3
}
```

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

```
01.     (START)
02.         @R0
03.         M = 0
04.         @2
05.         D = A
06.         @R1
07.         M = D
08.     (LOOP)
09.         @R1
10.         D = M
11.         @8
12.         D = D - A
13.         // PART I
14.         @END
15.         D; JGE
16.     (CHECK_INDEX)
17.         @R1
18.         A = M
19.         D = M
20.         D = D - A
21.         @UPDATE_INDEX
22.         D; JEQ
23.         @R0
24.         M = 1
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	0
2	2
3	3
4	4
5	5
6	6
7	7

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 =

      D =

      M =

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

      A =

      D =

      M =

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 =

      Value at address 1 =

      Value at address 2 =

c. The Hack Assembly code is supposed to check whether the elements stored in memory addresses R2 to R7 contain any negative values. The result is 1 if any of the values at the specified memory addresses contain any negative values and 0 otherwise. The result is stored in address R0. 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 (i == RAM[i])
4.              R0 = 1
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_INDEX** **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?

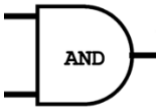
i. Line number:

ii. Line of Hack Assembly to fix the bug:

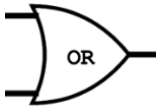
e. Does the buggy Hack Assembly program return the correct output (according to the pseudocode above) given the initial memory state shown in the table? If so, how could you change the initial memory state for the bug to appear? If not, how could you change the initial memory state for the buggy assembly program to produce the correct output?

CSE 390B Midterm Reference Sheet

Fundamental Combinational Logic Gates

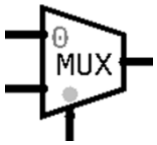


a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



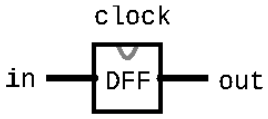
a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

in	out
0	1
1	0



a	b	sel	out
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Fundamental Sequential Logic Gate



`out(t) = in(t-1)`

- Triangle indicates implicitly connected to hardware clock
- **`out`** only changes on clock signal boundaries

HDL

Syntax:

- Basic Format: **`ChipName (in1=w1, in2=w2, ..., out=w3);`**
- Example: **`Mux (a=w1, b=w2, sel=w3, out=w4);`**
- Multiple wires connected to single output:
  `Mux (a=w1, b=w2, sel=w3, out=w4, out=w5);`

Multi-Bit Buses:

- Accessing Single Bit: **`w1[2]`**
- Slicing Multiple Bits: **`w1[0..3]`** (indices inclusive)
- Multi-Bit Input / Output Declaration: **`IN a[16];`**

Special Values:

- **`true`** is an any-width bus of all 1s, **`false`** of all 0s

Hack Assembly Language

Machine Characteristics:

- Two physical registers: **`D`**, **`A`**
- Pseudoregister **`M`** accesses memory at address **`A`**
- **`RAM`** and **`ROM`** have different, 0-indexed address spaces

Existing Symbols:

- **`R0...R15`** are "virtual registers": symbols bound to addresses 0 … 15 of **`RAM`**
- **`SCREEN`** is symbol bound to address at start of screen memory map
- **`KBD`** is bound to address of keyboard memory map (immediately after screen)

Label: **(*`LABELNAME`*)**

- Binds symbol ***`LABELNAME`*** to line number of instruction after it

A-Instructions: ***`@VALUE`***

- Loads ***`VALUE`*** into A register

C-Instructions: **`DEST=COMP;JUMP`**

- **`DEST`** or **`JUMP`** optional
- Performs **`COMP`**, result is stored in **`DEST`**, and if the result satisfies **`JUMP`** the PC jumps to address in A register

C-Instructions: Options for Fields

`COMP`

`0`
`1`
`-1`
`D`
`A`
`!D`
`!A`
`-D`
`-A`
`D+1`
`A+1`
`D-1`
`A-1`
`D+A`
`D-A`
`A-D`
`D&A`
`D|A`
`M`
`!M`
`-M`
`M+1`
`M-1`
`D+M`
`D-M`
`M-D`
`D&M`
`D|M`

`DEST`

`(empty)`
`M`
`D`
`A`
`MD`
`AM`
`AD`
`AMD`

`JUMP`

`(empty)`	`No jump`
`JGT`	`Jump if out > 0`
`JEQ`	`Jump if out = 0`
`JGE`	`Jump if out >= 0`
`JLT`	`Jump if out < 0`
`JNE`	`Jump if out != 0`
`JLE`	`Jump if out <= 0`
`JMP`	`Always jump`