

Midterm Exam

May 4<sup>th</sup>, 2023, at 2:30pm

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. Feel free to skip around. Do your best to get to all the questions.
- If you have a question, please raise your hand, and the course staff will get to you shortly.
- Take deep breaths and relax. Remember that you are here to learn.

**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 and Or gates and single-input Not gates.

A palindrome is a sequence of characters that is the same backward as is it is forward. For example, the words “kayak” and “Hannah” are palindromes because when you reverse the characters in that word, the word ends up being the same as the original word. Similarly, a sequence of binary bits is a palindrome if the bits are in the same order in reverse as its original order. For example, “10101” is a palindrome while “10111” is not a palindrome. For the problem below, determine if the sequence of bits in the order {a, b, c} represents a palindrome.

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

<b>a</b>	<b>b</b>	<b>c</b>	<b>isPalindrome</b>
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 isPalindrome.

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

```
CHIP IsPalindrome {  
    // a, b, and c represent the bits in binary  
    IN a, b, c;  
  
    // isPalindrome is 1 if the result of the binary bits  
    // in the order {a, b, c} is a palindrome, 0 otherwise  
    OUT isPalindrome;  
  
    PARTS:  
    // Your code or circuit diagram here:
```

```
}
```

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

  - a. List the two ways to update the A register in the Hack computer and give an example of each. Why are there two separate ways to update the A register as opposed to just one option?
- b. Describe in your own words how the Hack computer determines whether a **conditional jump** should be performed given the output control bits of the ALU (*ng* and *zr*) and the jump bits of the C-instruction (*j1*, *j2*, and *j3*). Give an example of the values of *ng*, *zr*, *j1*, *j2*, and *j3* that would cause a jump to occur and explain why in 2-3 sentences.

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:

○ <code>if (i2   i3):</code>	<code>o1(t + 1) = ~o2</code>
<code>else:</code>	<code>o1(t + 1) = i1 ^ o3</code>
○ <code>if (i1 ^ o3):</code>	<code>o2(t + 1) = o1 &amp; o2</code>
<code>else:</code>	<code>o2(t + 1) = i3</code>
○ <code>if (o1 &amp; o2):</code>	<code>o3(t + 1) = ~(o1 &amp; i2)</code>
<code>else:</code>	<code>o3(t + 1) = i2   i3</code>

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 (-R0 - R2 != 13) {  
    R0 = -3  
} else if (R2 | R3 <= 20) {  
    R0 = -R1  
} else {  
    R0 = R2 + 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.      @END
14.      D; JGE
15.      (CHECK_EQUAL_ONE)
16.      @R1
17.      A = M
18.      D = M + 1
19.      // PART I
20.      @UPDATE_INDEX
21.      D; JNE
22.      @R0
23.      M = M + 1
24.      (UPDATE_INDEX)
25.      @R1
26.      M = M + 1
27.      // PART II
28.      @LOOP
29.      0; JMP
30.      (END)
31.      @END
32.      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	1
1	1
2	1
3	6
4	9
5	-1
6	4
7	5

- a. Trace through the code starting with the state of memory given in the table. Indicate the value of the registers D, A, 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 D, A, and M when first reaching comment with "PART I"

D =

A =

M =

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

D =

A =

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 count the number of elements stored in memory addresses R2 to R7 that equal 1 and store the result in R0. The Hack Assembly program above attempts to be equivalent to the following pseudocode:

```
1.      R0 = 0
2.      for (R1 = 2; R1 < 8; R1++) {
3.          if (RAM[R1] == 1) {
4.              R0 += 1
5.          }
6.      }
```

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\_EQUAL\_ONE      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?
- Line number:
  - Line of Hack Assembly to fix the bug:
- e. Does the buggy Hack Assembly program return the correct output given the initial memory state shown in the table? If so, how could you change the initial memory state for the bug to appear in the output? If not, how could you change the initial memory state for the buggy assembly program to produce the correct output?