# Lecture Final Exam COMP 122/L

**Summer 2020**

**Score breakdown:**

From last exam: 22 Points

Circuits: 12 Points

Simplifying Boolean formulas: 66 Points

Total: 100 Points

There are 9 questions in all; there is no need to complete them in order.

# Ground Rules

1. You can use the course slides
2. You may have a calculator in front of you that can handle exponentiation.
3. You **may not** communicate with anyone else. **Violations of this rule will result in a 0 on the exam.** This exam is purely individual effort.

# From Last Exam

1.) (5 pts) What is the two’s complement number 1010 1110 in decimal? The space in between is only for readability.

2.) (5 pts) What is decimal -4 in two’s complement notation? Represent your answer using 8 bits.

3.) (6 pts) Consider the following Java-like code:

int number = <<read number from user>>; int mask = MASK;

int result = number OP mask;

if (result != 0) { print(“Bit 12 was set”);

}

The above code is supposed to print “Bit 12 was set” if bit 12 of number was set to 1. If bit 12 was not set, then this code should not print anything. What **hexadecimal** value should MASK be, and what bitwise operation should OP be, for the above code to work correctly?

4.) (6 pts) Consider the following code, which sets up a .data section:

.data label1:

.asciz “xy” label2:

.word 255 label3:

.word 8

Assuming the .data section starts at address 0, how does this look in memory? Use the following table as a template. Each value should be represented with a two-digit hexadecimal number, and the preceding 0x may be omitted. As a hint, the ASCII values of ‘x’and ‘y’in hexadecimal are 0x78 and 0x79, respectively. If the value at a given index is unknown, mark the position with ?.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

# Circuits

5.) (12 pts) Draw a circuit corresponding to the following Boolean formula. Do not simplify the formula.

F = AB + BC + !A!C

# Simplifying Boolean Formulas

6.) (12 pts) Simplify the following Boolean formula using Boolean algebra and possibly De Morgan’s laws. Show all steps.

F = (!A + !B)(A + C)

7.) (12 pts) Consider the following Boolean formula:

F = AB + !C

Write out a truth table corresponding to this formula. As a hint, you may add additional columns in the output corresponding to subformulas.

8.) (22 pts) Consider the following truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **R** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Draw a Karnaugh Map corresponding to this truth table, where **R** is the output. Draw boxes around the appropriate bits, following the appropriate rules for doing so. For full credit, you must draw boxes in a way that guarantees that **both** the number of products and the number of sums is minimized. **In addition**, write out the optimized sum-of-products equation corresponding to the boxes you drew.

9.) (20 pts) Consider the following truth table, which includes *don’t cares*:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **R** |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | X |

Draw a Karnaugh Map corresponding to this truth table, where **R** is the output. Draw boxes around the appropriate bits, following the appropriate rules for doing so. For full credit, you must draw boxes in a way that guarantees that **both** the number of products and the number of sums is minimized. **In addition**, write out the optimized sum-of-products equation corresponding to the boxes you drew.

Don’t cares in a Karnaugh map, or truth table, may be either **1**s or **0**s, as long as we don’t care what the output is for an input condition we never expect to see. We plot these cells with an X, among the normal **1**s and **0**s.

When forming groups of cells, treat the don’t care cell as either a **1** or a **0**, or ignore the don’t cares.

This is helpful if it allows us to form a larger group than would otherwise be possible without the don’t cares. There is no requirement to group all or any of the don’t cares. Only use them in a group if it simplifies the logic.