# Lecture Final Exam COMP 122/L

**Fall 2017**

**Score breakdown:**

From last exam: 20 Points (~17%) Circuits: 15 Points (~13%)

Simplifying Boolean formulas: 56 Points (~49%) Finite State Machines: 24 Points (~21%)

Total: 115 Points

There are 11 questions in all; there is no need to complete them in order.

Please write your name below, and **wait until told to begin:**

# Ground Rules

1. You may have up to three 8 1/2 x 11 inch sheets of paper in front of you containing handwritten notes covering both sides of each sheet.
2. You may have the combined ARM reference card/SWI instruction handout in front of you.
3. You may have a calculator in front of you that can handle exponentiation.
4. If you have a question, raise your hand and I will come to you.
5. 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.) (4 pts) What is the two’s complement number 1010 1110 in decimal? The space in between is only for readability.

2.) (4 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.) (7 pts) Draw a circuit corresponding to the following Boolean formula. Do not simplify the formula.

F = AB + BC + !A!C

6.) (8 pts) Define a register file containing two single-bit registers. The register file takes the following inputs:

|  |  |
| --- | --- |
| **Input Name** | **Input Description** |
| WE | Short for “Write Enable”; set to 1 if we are writing to a register, else 0 |
| CLK | The system clock |
| R | Which register we are reading from / writing to; 0 for the first register and 1 for the second register |
| I | Bit to write. Ignored if WE = 0. |

Given the above inputs, the register file produces the following outputs:

|  |  |
| --- | --- |
| **Output Name** | **Output Description** |
| U | The bit read from the selected register |

For this task, you may use **only** the following components, in unlimited supply:

* AND, OR, and NOT gates
* 2-input multiplexers
* D flip-flops

# Simplifying Boolean Formulas

7.) (10 pts) Simplify the following Boolean formula using Boolean algebra. Show all steps.

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

8.) (8 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.

9.) (20 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.

10.) (18 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.

# Finite State Machines

11.) (24 pts total) Write a finite state machine that will match the input string 1101. Inputs are read one bit at a time via input **I**. Outputs are written out on output **U**. To illustrate the expected behavior, the finite state machine should show the following sort of output pattern given the following inputs. Note that there is a delay of one clock tick between **U** being set and the output being matched.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I** | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| **U** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

11.a.) (10 pts) Write out a state diagram for this machine.

11.b.) (2 pts) If you were to implement this as a circuit, how many D flip-flops would you need?

(Questions continue on the next page.)

11.c) (12 pts) Convert your state diagram to a truth table. Use *don’t cares* where appropriate.