COSC-175: Systems-I
Fall 2026
Mid-term exam — Solutions

1. Question: [Low] Provide brief answers to each of the following questions:

   (a) We have proven that NAND is a *universal logic operator*. Why, then, do we not implement all of our circuits using only NAND gates?

   (b) What is the range of values that can be represented by a 6-bit two's-complement integer?

   (c) For a *sequential circuit* (e.g., a counter), what function does the *clock* input serve? Why is it necessary?

   (d) What is the *critical path* of a circuit? Why does it matter?

   Answers:

   (a) Efficiency! Requiring two gates for every AND and three for every OR implies more gates, more wires, and longer delays through those extra gates.

   (b) 2^6 bit patterns, split equally between negative and non-negative values, implies a range of -2^5 to $2^5 - 1$, or -32 to 31.

   (c) The clock controls the rate at which the counter advances. It is required to control when the memory elements allow the next value to pass through and become the current value.

   (d) The *critical path* is a path from an input to a circuit to an output for which the sequence of gate delays will require the greatest time before the output has settled to its final value.

   Observations: Most of the answers here were good! The most typical mistake was simply to misremember what a *critical path* was, yielding some creative attempts at "alternative definitions".

2. QUESTION: [LOW] Suppose we discover an alien technology that uses *trits* instead of *bits*. Each trit can take on value 0, 1, or 2. Now consider a logic gate that takes in two trits A and B and outputs a single trit Y. **How many (unique) gates of that form are possible?**

   ANSWER: Given two input trits, there are $3^2 = 9$ possible input patterns. Each unique way of filling in the 9 output values that define Y is a distinct function. Since each of those 9 output values is a trit (0, 1, or 2), then there are 3^9 functions.

   OBSERVATIONS: Few got this one. Many determined that the number of possible 2-trit input values was $3^2 = 9$. However, few recognized that each of those 9 inputs corresponded to an independent trit output that could take on one of the 3 values.

3. QUESTION: [HIGH] First, consider the truth table for an S-R latch:

Q	S	R	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Second, consider the *cross-coupled NOR-gate structure* that we have used to implement this device. **Prove algebraically** that the cross-coupled NOR-gates are equivalent to the truth table shown.

ANSWER:

$$
\begin{array}{rl}
Q = \bar{Q}S\bar{R} + Q\bar{S}\bar{R} + QS\bar{R} & \text{Canonical Disjunctive Normal Form} \\
= \bar{R}(\bar{Q}S + Q\bar{S} + QS) & \text{Distributivity} \\
= \bar{R}(\bar{Q}S + Q\bar{S} + QS + QS) & \text{Idempotency} \\
= \bar{R}(\bar{Q}S + QS + Q\bar{S} + QS) & \text{Commutativity} \\
= \bar{R}((\bar{Q}S + QS) + (Q\bar{S} + QS)) & \text{Associativity} \\
= \bar{R}(S(\bar{Q} + Q) + Q(\bar{S} + S)) & \text{Distributivity} \\
= \bar{R}(S1 + Q1) & \text{Idempotency} \\
= \bar{R}(S + Q) & \text{Identity} \\
= \overline{\overline{\bar{R}(S + Q)}} & \text{Involution} \\
= \overline{\bar{\bar{R}} + (\overline{S + Q})} & \text{DeMorgan's} \\
= \overline{R + (\overline{S + Q})} & \text{Involution} \\
= \overline{R + \bar{Q}}, \text{ where } \bar{Q} = \overline{S + Q} & \text{Substitution, given by circuit}
\end{array}
$$

OBSERVATIONS: Many people got part-way through this problem, but not all of the way. The requirement that the proof be algebraic was often violated, shifting to a proof via truth-table or to a set of cases (where each case was presented as an argument).

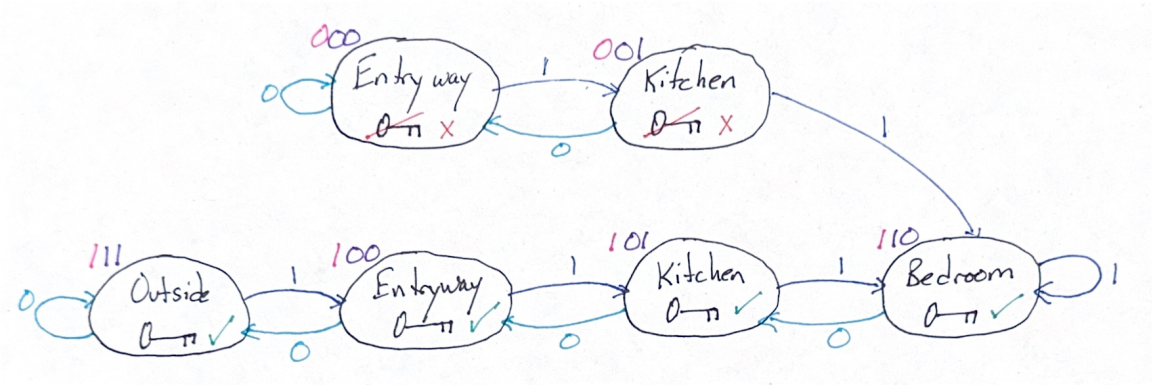
Almost nobody thought to label each step of their algebraic transformations with the *names* of the axioms and theorems being applied. A proper proof is the application of a specific rule at each step; there should be no guesswork in determining how two expressions are related.

4. QUESTION: You are playing the world's least interesting video game: *You are standing in the entryway of a narrow house. In front of you, further into the house, is a kitchen, beyond which is a bedroom. Behind you is a door leading outside, and it is locked. It is only possible to unlock it with a key, and that key is in the bedroom. Your goal is to retrieve the key (by entering the bedroom) and leave the house.*

   The controller for this primitive game has a single switch, labeled *forwards* (towards the bedroom) and *backwards* (towards the outside). There is also a button, labeled *move*, which will try to move you in the direction chosen on the switch. You cannot move from the entryway to the outside unless you first get the key to the door.

   Design a circuit that implements this game. Each of the four locations (bedroom, kitchen, entryway, outdoors) should be assigned a number, and the circuit should show where the player is by displaying that number (on LED's). The player should begin in the entryway, and the game ends when the player moves from the entryway to the outdoors (which should be possible only if the player has the key—obtained by having been in the bedroom!).

   ANSWER: We represent the possible states of the game as a *finite state machine*:[1]



   Each state is numbered, where:

   - The most significant bit indicates whether the player *holds the key* (0 = no key; 1 = key).

   - The other two bits indicate the *current location* (00 = entryway; 01 = kitchen; 10 = bedroom; 11 = outdoors).

[1] This particular depiction of a FSM is not the only valid one. The placement of states and transitions can vary. So can the numbering of the states. The numbering presented here split the bits to represent information about the state (i.e., *location* vs. *key posession*). So long as each state has a unique number, and that the initial state is 0, then the numbering would be valid.

Likewise, the transitions are labeled 0 to go *backwards*, and 1 to go *forwards*. The FSM can therefore be implemented by:

(a) **One 3-bit register (flip-flops)** to store the current state (`Q2, Q1, Q0`).

(b) **One switch** to represent the player's direction of movement (`M`).

(c) **One button** to trigger movement (`Clock`).

(d) **Transitional logic** comprising gates for each logic function.

With the directional input (`M`), the transitional logic is:

Q2	Q1	Q0	M	D2	D1	D0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	0	1	1	1	1	0
0	1	0	0	x	x	x
0	1	0	1	x	x	x
0	1	1	0	x	x	x
0	1	1	1	x	x	x
1	0	0	0	1	1	1
1	0	0	1	1	0	1
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	0	0

OBSERVATIONS: This was a large problem with many opportunities to lose the thread. Some failed to lay out the game as a finite state machine. Those who did sometimes failed to identify the possession (or non-possession) of the key as an element of being in a particular state. Some state machines had invalid trasitions (e.g., two outgoing arrows, both labelled 0).

Given a state machine, the transitional logic table was sometimes a bit messy (especially if the key-holding component was not clearly represented in the states), but often had the right basic structure for controlling the state-to-state movement based on the directional input.

The logic, once expressed, was a bit messy. What was most critical in showing the circuit was the presence of memory elements, clocked by the *move* input, feeding back into themselves through the transitional logic. This basic structure of a sequential circuit was important, and sometimes not drawn.

5. QUESTION: [MEDIUM] Consider addition of 4-bit, two's-complement values $A+B=Y$, where C represents the carry bits:

$$\begin{array}{cccccc} & C_3 & C_2 & C_1 & C_0 \\ & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ \hline C_4 & Y_3 & Y_2 & Y_1 & Y_0 \end{array}$$

Prove that *overflow* is indicated if and only if $C_3 \neq C_4$

ANSWER: Given that *overflow* occurs when either two negative values sum to a positive value, or two positive values sum to a negative value, we can prove that $c_3 \oplus c_4$ indicates overflow in exactly the same cases that $(a_3 = b_3)(a_3 \neq y_3)$ via *exhaustive enumeration* (a.k.a., via truth-table):

c_3	a_3	b_3	c_4	y_3
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The two highlighted cases are the exact ones for which the two expressions indicating overflow are true:

- When $a_3 = b_3 = 1$ and $y_3 = 0$, $c_3 = 0$ and $c_4 = 1$.
- Where $a_3 = b_3 = 0$ and $y_3 = 1$, $c_3 = 1$ and $c_4 = 0$.
- In all other cases, $c_3 = c_4$.

OBSERVATIONS: There were a number of correct answers to this problem, but few of them employed the truth table as a form of proof. The correct alternative, although lengthier to present, was to break down the possibilities into cases and then show, algebraically, the equivalence. The danger with this approach (other than the additional time and complexity required) was that *all* of the cases needed to be covered: some addresses the cases that *did* indicate overflow, but failed to show correctness for cases that *did not* trigger overflow.

Some attempted to present arguments in prose, while others attempted to demonstrate equivalence with a couple of examples. These approaches are insufficient as proof of equivalence.