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

1. Question: (Low) **Prove** that NOR is a *universal logic operator*.

   Answer: *Disjuctive normal form*:

   (a) Can be used to express any possible logic function, and

   (b) Uses only AND, OR, and NOT as its logic functions.

   Therefore, AND/OR/NOT are *universal*. NOR is universal if those three (AND/OR/NOT) can be expressed using only NOR:

NOT	$\overline{A}$	$= \overline{A + A}$	Idempotence

OR	$A + B$	$= \overline{\overline{A + B}}$	Involution
		$= \overline{\overline{A + B} + \overline{A + B}}$	Idempotence

---

AND	AB	$= \overline{\overline{AB}}$	Involution
		$= \overline{\overline{A} + \overline{B}}$	DeMorgan's
		$= \overline{\overline{A + A} + \overline{B + B}}$	Idempotence

---

Observations: The most prominent missing element missed by nearly everyone was establishing that AND/OR/NOT are, as a trio, themselves universal. Without establishing and justifying that fact, showing equivalence to those three operators wasn't fully grounded.

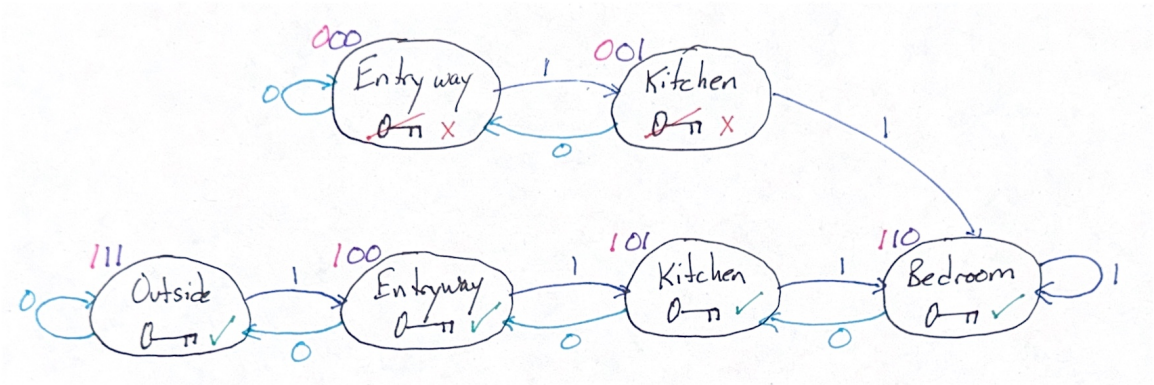
While nearly everyone showed correct constructions (via logic or via gates), a proof requires a step-by-step connection from one expression to another, elucidating the exact algebraic rules used. Some wrote statements of equivalence that required multiple algebraic steps, but did not indicate which steps those were.

2. QUESTION: (HIGH) 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 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!). **Show your work:** Show how you are representing the player's current position, and how you are keeping track of whether the player has the key.

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



   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).

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) **One ROM** with a 4-bit address input (`A3, A2, A1, A0`) and a 3-bit data value output (`V2, V1, V0`).

If the address comprises the current state (`A3, A2, A1 = Q2, Q1, Q0`) and the directional input (`A0 = M`), then the output of the ROM that implements the transitional logic provides the next state (`V2, V1, V0 = D2, D1, D0`). The transitional logic is:

Q2	Q1	Q0	M	D2	D1	D0
A3	A2	A1	A0	V2	V1	V0
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: Most answers did employ a finite state machine. However, many did not identify "holding the key" as part of the *state*. Without noticing that the key was part of the state itself—that *in the kitchen without the key* was a fully distinct state from *being in the kitchen with the key*—those solutions needed to apply some kind of logic to the transitions between states. While doing was sometimes good enough to produce a workable sequential logic circuit, the storage of the "holding the key" bit was separated from the others, and then passed through contorted logic.

A subspecies of this mistake was to assume that this large word-problem was really just masking a rigid sequence *a la* Lab-4. Doing so led to FSM's that only allowed a strict progress through the states without allowing the player to wander back and forth through the rooms (which was valid). The FSM and thus the transitional logic was then too constrained.

For some, the difference between a *register* (a.k.a., a multi-bit flip-flop) and a *ROM* led to malformed circuits. The ROM would sometimes be clocked (and ROMs are never clocked), and the distinction between storing the *current state* vs. storing the *transitional logic* was lost, becomming muddled and commingled.

3. QUESTION: (MEDIUM) For our 4-bit ripple-carry adder/subtracter, **prove** that $c_3 \oplus c_4$ (i.e., the *carry-in* and *carry-out* for the most significant full-adder) correctly indicates *overflow*.

   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:

   (a) When $a_3 = b_3 = 1$ and $y_3 = 0$, $c_3 = 0$ and $c_4 = 1$.

   (b) Where $a_3 = b_3 = 0$ and $y_3 = 1$, $c_3 = 1$ and $c_4 = 0$.

   OBSERVATIONS: There were a number of correct answers to this problem, but vanishingly 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.

4. QUESTION: (HIGH) Update our multiplier:

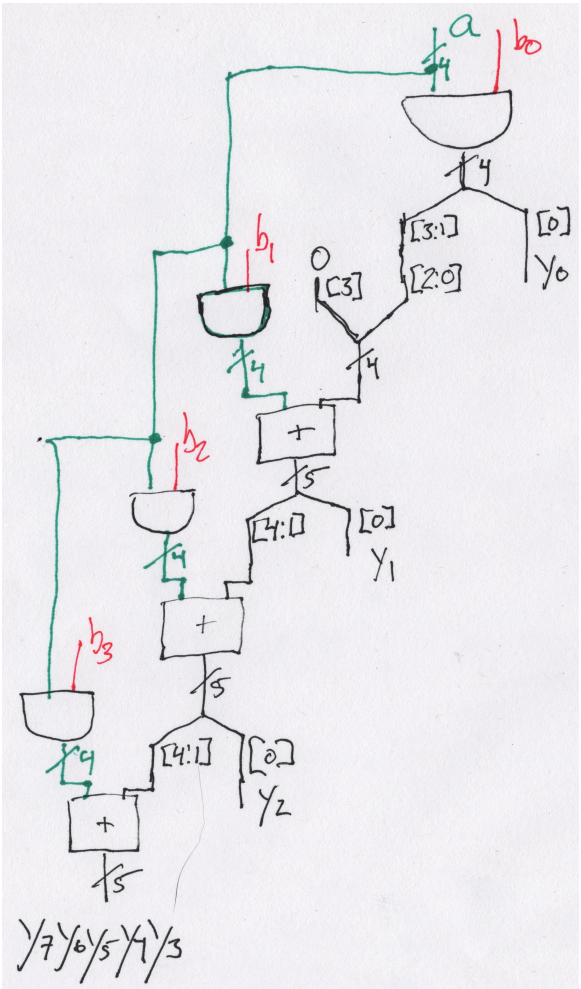
   (a) **Design and draw** a *combinational circuit* that multiplies two 4-bit unsigned integers, producing an 8-bit product.

   (b) Given that combinational multiplier, now alter/augment it to multiply *signed* (two's complement) integers.

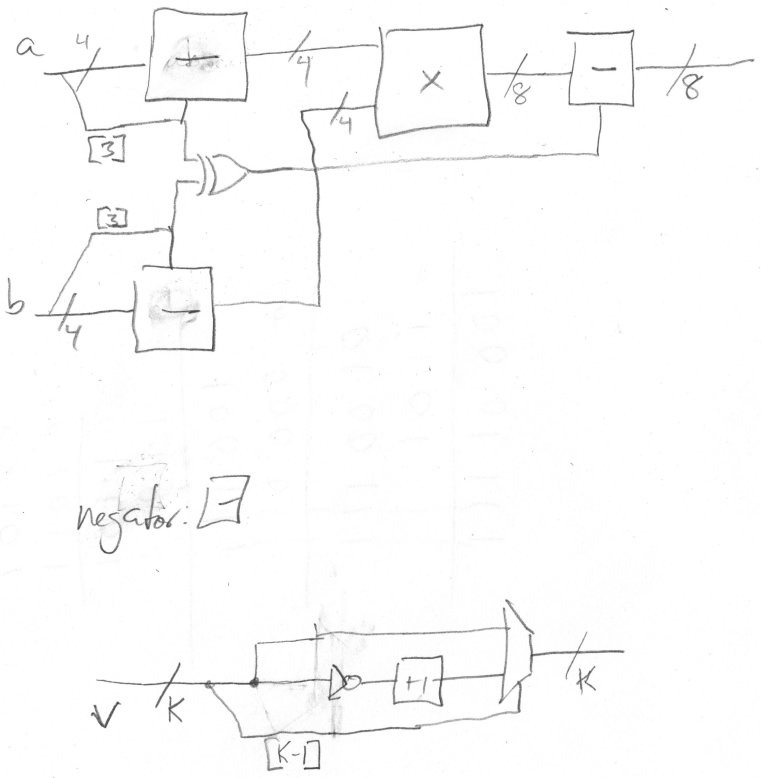
   You may use high-level components such as adders, multiplexers, etc.

   ANSWER: Here is a *combinational multiplier*, where the input and output bits are:

$$\begin{array}{cccccccc} & & & & a_3 & a_2 & a_1 & a_0 \\ \times & & & & b_3 & b_2 & b_1 & b_0 \\ \hline y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \end{array}$$



And here is a circuit that handles *signed* integers by implementing the following observation: *If the signs of the multiplier and multiplicand match, then the product is non-negative; if the input signs differ, the product is non-negative.*



OBSERVATIONS: For part (a), the incorrect answers failed to remove the sequentiality from our Lab-5 solution. That is, they continued to use registers and a clock, and thus did not produce a combinational result. Those who were able to break away from the sequential structure largely got this question right with some depiction of the cascade of adders.

Part (b) was likewise feast or famine: most answers were either largely similar to the one shown above, or they did not present any coherent approach to handling two's complement values.

Consequently, this was the question on which people lost the most points. Given that you either conceived of a workable answer or just didn't, the answers were mostly correct or mostly incorrect.