Systems I — Fall 2011 — Mid-term exam

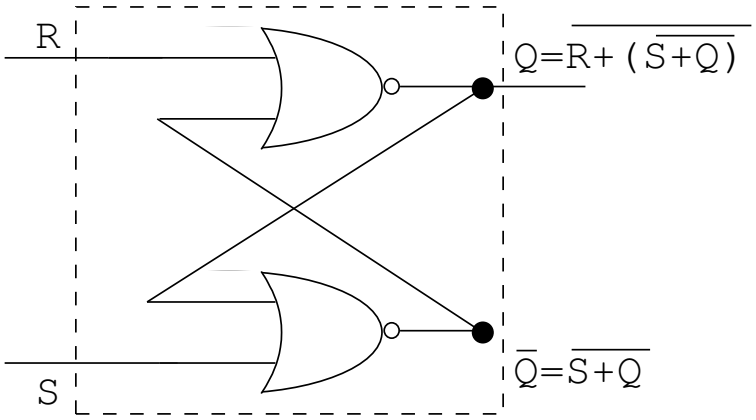
This is a closed-book, closed-note exam. Answer all of the questions **clearly, completely, and concisely**. You have 50 minutes, so be sure to use your time wisely. All work should be written in your blue book. If a question's intent is ambiguous, then **don't make assumptions; rather, ask me to clarify the question** for you.

1. QUESTION: Recall the following feedback function that we developed in class, when first devising a function capable of "remembering" a value, such that its output Q would be **set** ($Q = 1$ when $S = 1$), **reset** ($Q = 0$ when $R = 1$), or retain its old value when neither S nor R is asserted (Q is whatever it was):

$$Q = \bar{R}S + Q\bar{R}\bar{S}$$

I then claimed that this function was equivalent to a pair of cross-coupled NOR gates. **Show, algebraically, that the function above is equivalent to the cross-coupled NOR gates**.

ANSWER: Given the structure of the cross-coupled NOR gates ...



... we get the following formulae from the outputs:

$$\bar{Q} = \overline{S + Q}$$
$$Q = \overline{R + \bar{Q}} = \overline{R + \overline{S + Q}}$$

We then derive, step by step, the formula given by the question:

$Q =$	$\bar{R}S + Q\bar{R}\bar{S}$	given
	$\bar{R}(S + \bar{S}Q)$	factor $\bar{R}$ and conjunctive commutativity
	$\bar{R}(S1 + \bar{S}Q$	conjunctive identity preservation
	$\bar{R}(S(1 + Q) + \bar{S}Q$	disjunctive identity elimination
	$\bar{R}(S + SQ + \bar{S}Q$	distributive
	$\bar{R}(S + Q(S + \bar{S})$	factor Q
	$\bar{R}(S + Q1)$	disjunctive complementarity
	$\bar{R}(S + Q)$	conjunctive identity preservation
	$\overline{R + \overline{S + Q}}$	DeMorgan's

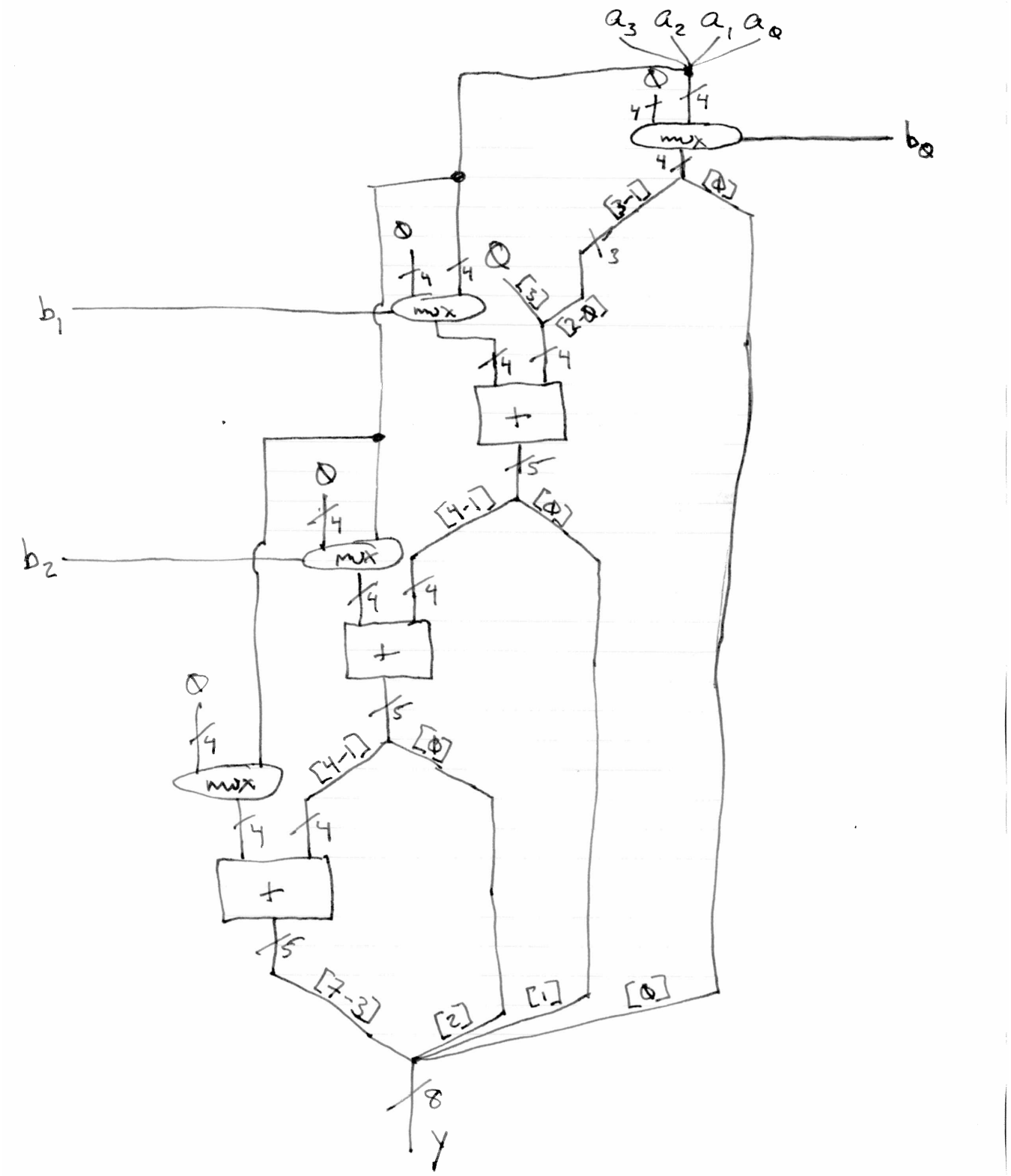
Discussion: The solution given above is pedantically step-by-step, naming each small transformation. I did not expect property names from anyone. Moreover, everyone combined some steps when their combination was clear, which was perfectly acceptable. The most common problem of only modest consequence was the skipping of **non-**obvious steps. For example, some would transform $Q\bar{R}\bar{S}$ to $Q\bar{R}$ by improperly noting, "Remove redundant terms." A key difficulty in this problem was providing the algebraic foundation for such "obvious" simplifications.

Some errors were more fundamental. Using a truth table was explicity forbidden. More commonly, some transformed the expressions in each direction, but never attempted to get them to meet in the middle—that is, to transform each into a common, identical expression.

2. QUESTION: In a recent lab, you built a multiplier that was based on *sequential logic*—that is, a clocked sequence of logic operations calculated the product of two 4-bit, unsigned values. Now, you must **design a combinational multiplier**. That is, you must **draw a circuit** that uses no registers and no clock, but rather computes the product of two 4-bit, unsigned values via a circuit that takes an input and calculates its output in a single, direct step.

   Assume that you have, at your disposal, high-level components such as 4-bit adders, multiplexers, etc.

   ANSWER: Handle the multiplication as a series of three 4-bit addition operations, where the inputs for the additions are "shifted" into the correct positions of significance.


a_3 a_2 a_1 a_0
mux
b_0
b_1
b_2
+
[4-1]
[3-1]
[3]
[2-0]
[0]
[7-3]
[2]
[1]
8
y

DISCUSSION: A number of people came up with minor variations on the solution shown above. Others came up with more elaborate solutions that involved a larger number of adders, but ultimately produced a correct result. One person went truly brute-force, breaking the problem into 16 cases (one for each possible multiplicand value); for each case, the correct number of adders were chained in sequence, adding the multiplier to itself that many times; the correct case was then selected by multiplexer. This solution used a spectacular number of gates, and would scale poorly if applied to larger inputs, but it had the benefit of being conceptually straightforward, making its correctness easy to verify.

Errors occurred in many ways for this problem. Some tried to chain together a collection of addition operations, but failed to correctly account for the handling of carry values. Others failed to read or understand the problem correctly, and simply presented a sequential-logic solution that used memory elements, in spite of the question's explicit prohibition of them. Still others simply ran out of time, working out some of the logic, but not enough to develop a comprehensible design. Any solution that attempted to solve the problem by cascading adders received a goodly portion of the available credit.

3. QUESTION: Your friends have been breaking into your dorm room and pulling pranks on you: stealing all of your clothes while you were in the shower was the last straw. Locking the door doesn't work, since one of them has a lock-pick set and knows how to use it. You want to add a deadbolt, but Physical Plant won't let you drill a hole in their nice door. So, you've decided to go the homebrew electronic route.

   You have powerful magnets that can hold the door shut. You want to design your own circuit that will allow you to enter a passcode to disable the magnets, thus unlocking the door when you need to get in. You mount, outside your door, a single toggle switch (labeled 0 and 1) and a single button. To unlock the door, you want a circuit that will require a person to enter a sequence of 1-bit numbers, where entering a number requires setting the switch to 0 or 1 and then pressing the button.

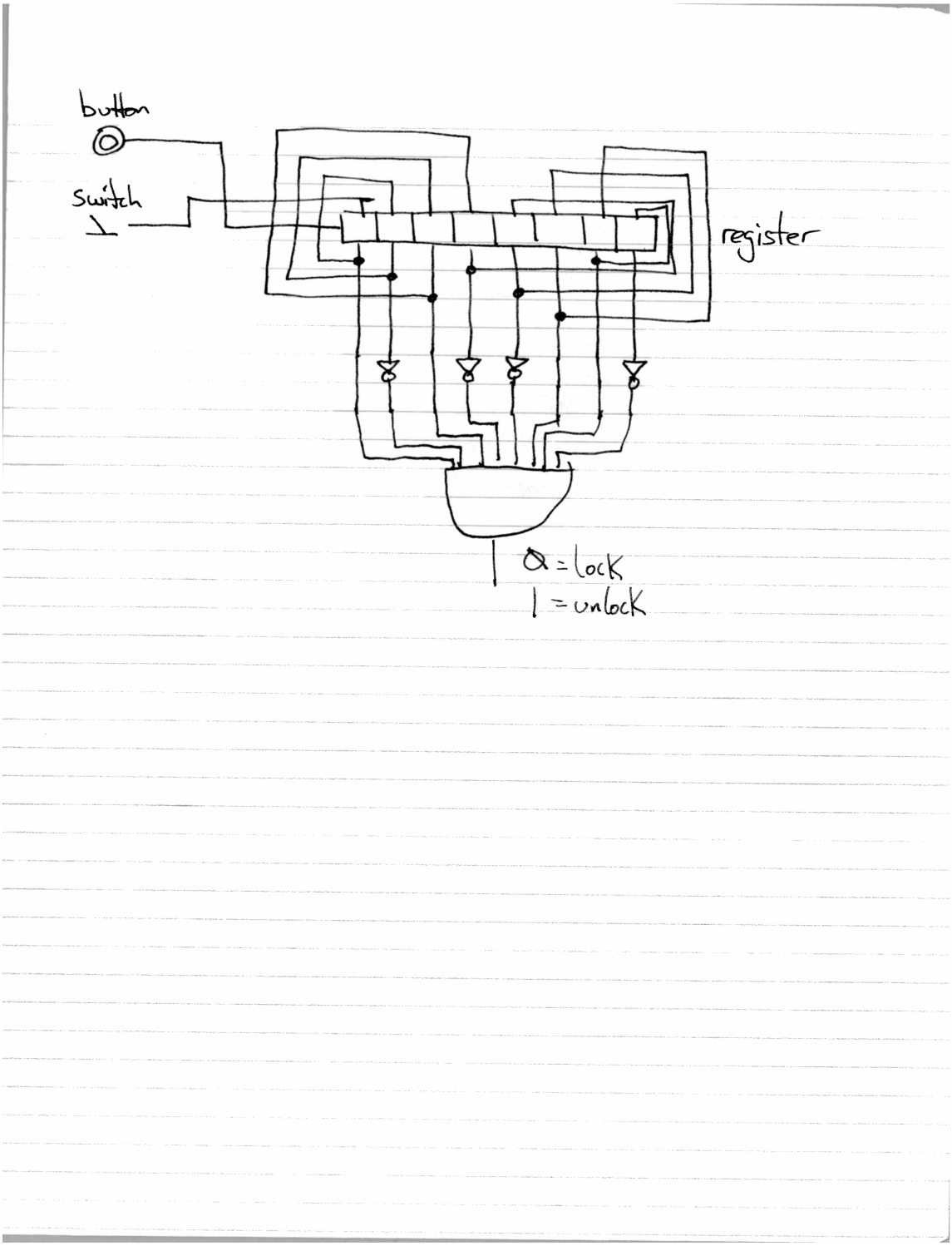
   Specifically, when your circuit emits a 0, the magnets remain active and the door remains locked. To unlock the door, you want a person to have to enter a sequence of k 1-bit values. As soon as the k^{th} value is correctly entered, your circuit sets its output to 1, disabiling the magnets. You choose $k = 8$, and specifically choose the sequence: $0, 1, 1, 0, 0, 1, 0, 1$.

   (a) Design and draw this circuit, based on the description above. Again assume that you have appropriate high-level components available to use.

   (b) Is $k = 8$ big enough for this locking mechanism to really help you? If not, what would make a better k?

   ANSWER:

   (a) The following circuit implements a solution to this problem:



(b) There are $2^8 = 256$ different sequences that someone could enter. While doing so may take some time, a persistent adversary could accomplish the task before long. Each additional bit would provide a doubling of the number of combinations. A $k = 16$ would imply $2^16 = 32,768$ possibilities, which is likely too many for anyone to attempt in practice.

DISCUSSION:

(a) The essential feature for any viable solution here—and there were significant variations—was some kind of 8-bit memory that would accept the inputs, one bit at a time. Most people tried to solve this problem not by shifting the bits (as in the solution above), but by using a 3-bit counter and a multiplexer so that each bit would be deposited, in turn, into a different position in the register. This solution mostly works, but someone using lock would have no way to reset it, ensuring that first bit was stored into the first position; worse, the user cannot determine what the state of the counter is, and if the counter doesn't begin at 000, then the user may enter the correct value but not successfully unlock the door.

It is worth noting that even my solution above has a failing, although one that I would not expect you to have noticed. In particular, when the flip-flops of the register are clocked, they may not all settle to their new values at **exactly** the same moment. Thus, there may be an intermediate time, between when the clock button is pushed and the outputs of the register settles, that the outputs may temporarily be exactly those needed to trigger a 1 output from the AND gate, even though the user never input the correct sequence. How would you fix this problem?

(b) I accepted nearly any answer that acknowledged that there are 256 possible 8-bit seuqences that a user could enter to, via *brute force enumeration*, crack this lock. Whether that would suffice depended on your characterization of your friends' tenacity.

Note, however, that someone **could** attempt every possible sequence that this lock would accept without entering all 256 possibilities as full 8-bit values to be entered one bit at a time. Specifically, after entering some 8-bit value, an adversary could enter just 1 additional bit and test a different 8-bit value. Enter 1 more bit, test a third 8-bit value. A bit of careful combinatorics can determine just how many individual bits someone would have to enter in order to test all 256 8-bit values. It is that result that should determine whether you think an 8-bit lock of this kind is likely to hold your friends at bay for long.