Systems I
Fall 2013
Mid-term exam solutions

1. Question: (20 points) Prove that NOR is a *universal operator*. That is, prove that any logic function is equivalent to some expression that uses only the NOR operator. You may demonstrate equivalence of NOR with other operators using Boolean algebra or (labeled) circuit diagrams.

   Answer: Since a DNF expression can be formed for any function, and since DNF employs only AND, OR, and NOT, we infer that this trio of operations is, together, universal. If we can calculate these three using only NOR, then NOR is also universal.

   (a) NOT:

   $$\bar{x} = \overline{x + x} \quad \text{(self disjunction)}$$

   (b) OR:

   $$\begin{aligned} x + y &= \overline{\overline{x + y}} && \text{(double negation)} \\ &= \overline{(\overline{x + y}) + (\overline{x + y})} && \text{(self disjunction)} \end{aligned}$$

   (c) AND:

   $$\begin{aligned} xy &= \overline{\overline{xy}} && \text{(double negation)} \\ &= \overline{\bar{x} + \bar{y}} && \text{(DeMorgan's)} \\ &= \overline{(\overline{x + x}) + (\overline{y + y})} && \text{(self disjunction)} \end{aligned}$$

   Discussion: The most common source of error was inattention to detail. Many people failed to note **why** it would be useful to show the equivalence to the AND/OR/NOT operators. Others either used truth tables or (largely) unlabeled circuit diagrams. The former was explicitly not offered as an option for showing equivalence for this question; the latter was acceptable only because it required a demonstration of at least the critical portions of the relevant algebra.

2. QUESTION: (20 points) Consider the logic function described by the following truth table:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

(a) Write this function in *disjunctive normal form*.

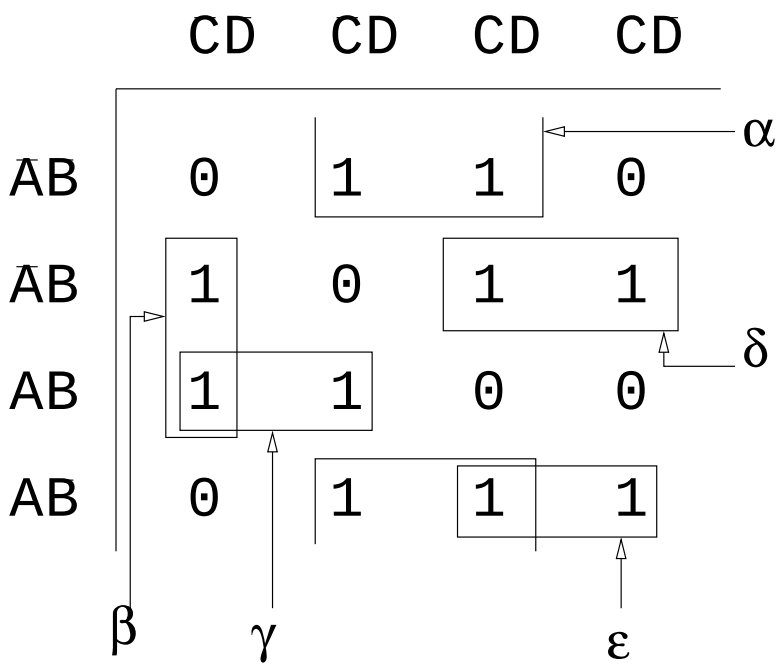
(b) Simplify this expression by using a *Karnaugh map*.

ANSWER:

(a) Disjunctive Normal Form:

$$\begin{aligned} Y = \quad & \bar{A}\bar{B}\bar{C}D + \\ & \bar{A}\bar{B}C\bar{D} + \\ & \bar{A}\bar{B}CD + \\ & \bar{A}B\bar{C}\bar{D} + \\ & \bar{A}BC\bar{D} + \\ & \bar{A}BCD + \\ & A\bar{B}\bar{C}D + \\ & A\bar{B}C\bar{D} + \\ & A\bar{B}CD + \\ & AB\bar{C}\bar{D} + \\ & AB\bar{C}D + \end{aligned}$$

(b) Karnaugh map:



$$\begin{aligned} Y &= \alpha + \beta + \gamma + \delta + \epsilon \\ &= \bar{B}D + B\bar{C}\bar{D} + AB\bar{C} + \bar{A}BC + A\bar{B}C \end{aligned}$$

Discussion: The mistakes on this question were typical ones: misremembering the layout and/or ordering of each axis; mis-transcribing the correct values into the map; creating rectangles of invalid sizes. These errors lost a some points, but not the majority. More significant errors involved grossly misshapen maps, the use of algebra instead of Karnaugh maps for simplification, and grander failures to understand that question sought a single, simplified logic expression.

3. QUESTION: (30 points) Create a circuit that calculates $a < b$, where a and b are 4-bit, two's-complement values. This circuit should have a 1-bit output that is 1 when $a < b$, and 0 otherwise. **Hint:** Note that $a < 0 \Leftrightarrow a - b < 0$, implying that subtraction is likely a useful operation here.
Extra challenge: If you use subtraction to solve this problem, then you must consider the behavior of your solution when the subtraction yields overflow. Can your circuit provide the correct answer for all values of a and b, even in the presence of overflow? If so, how? If not, why not?

ANSWER: A *1-bit full adder* is a device that accepts 3 inputs (c_I, a, and b), and emits 2 outputs (c_O and r). Define those outputs as:

- $r = c_I \oplus a \oplus b$
- $c_O = c_I a + c_I b + ab$

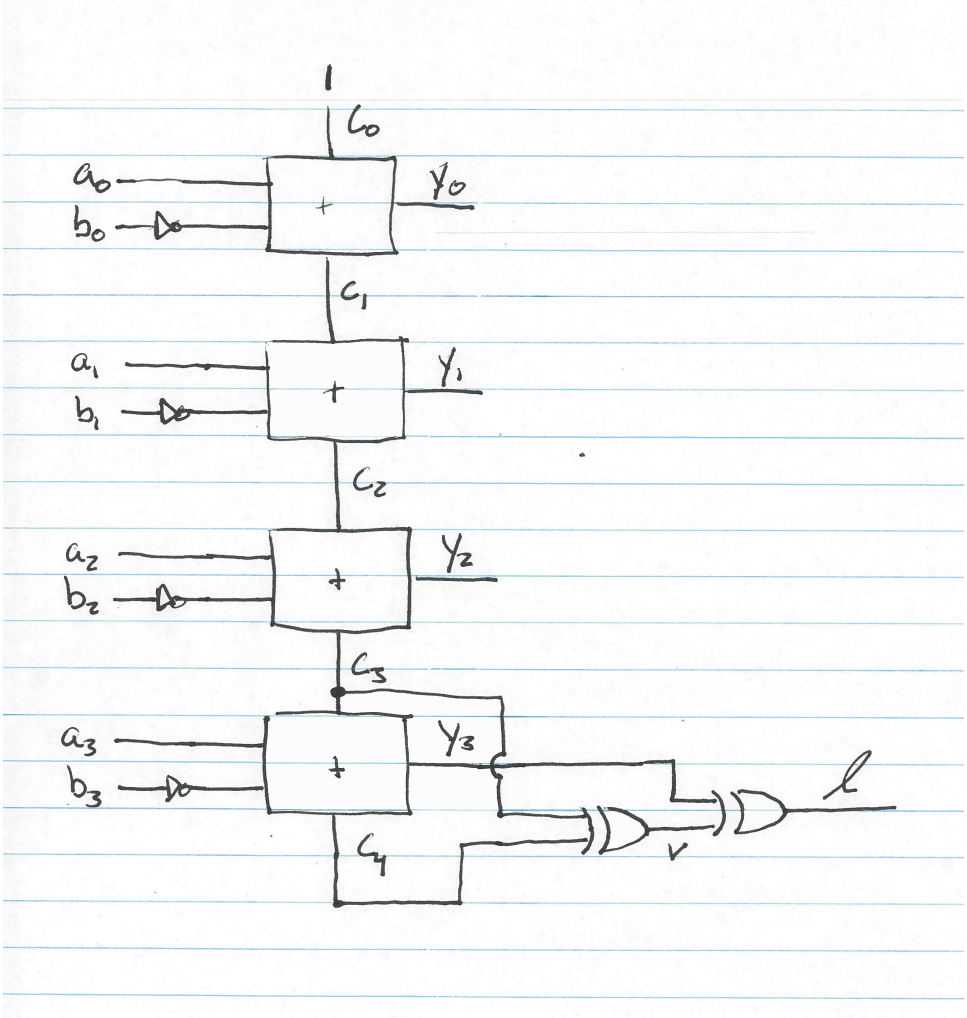
The question observes that $a < b \Leftrightarrow a - b < 0$. So, we can perform this comparison by:

(a) Negating b.

(b) Summing a and $-b$.

(c) Examining the sign of the result of that summation, where a negative result implies that the comparison is $true$, and a non-negative result implies that the comparison is $false$.

Consider a 4-bit ripple-carry subtracter, shown below. First, the sign of the result can be determined by y_3, which is the most significant bit and is 1 *iff* y is a negative value. We must also consider overflow, which can be determined by $v = c_3 \oplus c_4$. Since overflow occurs when the sign of the result is incorrect, we know that, when $v = 1$ then y_3 is the opposite of what it should be. Thus, to restore the correct sign only when overflow occurs, $l = y_3 \oplus v$, where l now indicates the result of $a < b$ even in the presence of overflow.



Discussion: First, there was no good excuse for missing this problem. We developed this exact solution in class. More reasonable errors seemed to be the result of studying the sample exam a bit too carefully without thinking about the underlying semantics of the answer. That is, many people explicitly tested the subtraction's result to determine if it was 0. On the sample exam, where the goal was to calculate $a > b$, combining the result of $a < b$ with $a = b$ was a useful strategy for which detecting equality (found by $a - b = 0$) was necessary. For the problem of finding $a < b$, evaluating $a = b$ was at best superfluous, and at worst an erroneous addition to your circuit.

Some built a circuit that correctly determined the less-than result in the absence of overflow, which required only to remember how subtraction was performed, as well as the examination of the y_3 output. Others tried to incorporate overflow into their solutions, but did so incorrectly. To that end, I will reiterate for what I hope is the last time: The final carry out (c_4) is **not itself an indication of overflow**. Be sure that you understand why before the final exam.

4. QUESTION: **Draw a circuit** that emits the following repeating 2-bit sequence:

$$00, 01, 00, 10, 00, 11 \ldots$$

Your circuit should have 2 bits of output, as well as incoming CLOCK and CLEAR inputs. Every 6 cycles of the clock should progress through this sequence, which should "wrap around" and repeat with the next 6 clock cycles. You may use basic memory devices such as *flip-flops* (which may be aggregated into *registers*).

ANSWER: The sequence has a length of 6, so we use a 3 bit register to count from 0 to 5, repeatedly, to keep track of the current position in the sequence. Each position number is then mapped to its corresponding output pattern. In the following truth table, Q is the current position number in the sequence, D is the next position number in the sequence, and Y is the value at that point in the sequence. In other words, Q and D are a standard counting sequence, used internally to track which of the in the 2-bit output values Y in the sequence to show at that moment.

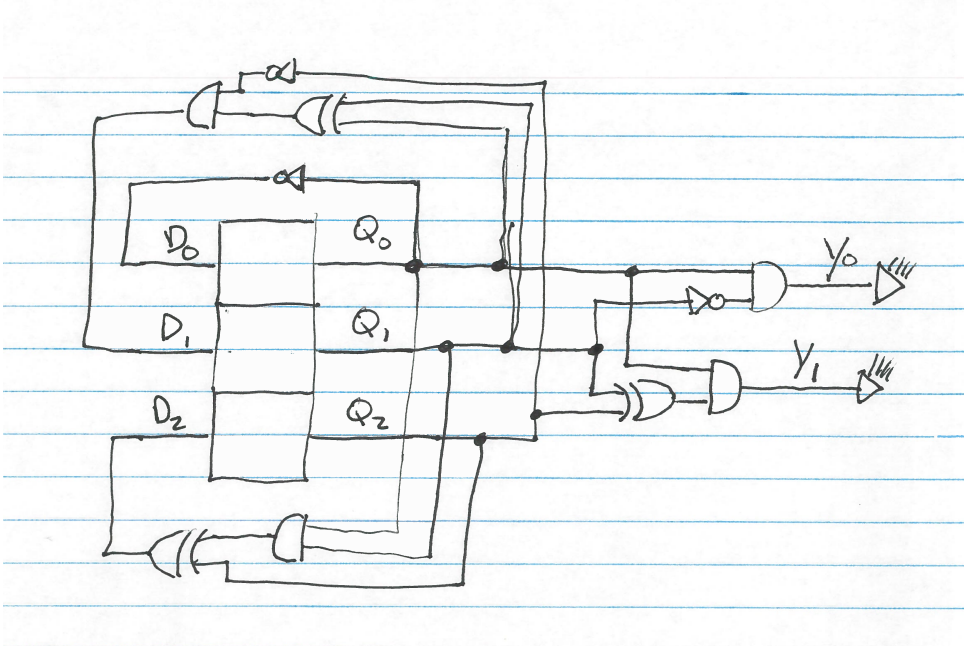
D_2	D_1	D_0	Q_2	Q_1	Q_0	Y_1	Y_0
0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	0
1	0	0	0	1	1	1	0
1	0	1	1	0	0	0	0
0	0	0	1	0	1	1	1

From this truth table, we derive the following logic functions for D (which determines the next state) and for Y (which determines the output):

$$\begin{aligned} D_2 &= Q_2 \oplus Q_1 Q_0 \\ D_1 &= \bar{Q}_2 (Q_1 \oplus Q_0) \\ D_0 &= \bar{Q}_0 \\ Y_1 &= (Q_2 \oplus Q_1) Q_0 \\ Y_0 &= \bar{Q}_1 Q_0 \end{aligned}$$

Given a 3-bit register driven by a clock input[1], with a *clear* input that is permanently disabled (not clearing the register), the sequential logic described above yields the following circuit:

[1] Accidentally not shown.



DISCUSSION: This question was the trickiest of the bunch. The insight was to notice that, although the sequence had two bits per position, two bits alone could not possibly encoding the current position in the sequence. Some used ad-hoc arrangements of four flip-flops, with two bits used to store the current value in the sequence, and two bits to store the previous value. Taken together, those four bits could correctly determine, with some properly mapped logic, the next state. This approach was slightly inefficient bit largely correct.

Some other were just thrown for a loop by the question, and created incomprehensible circuits. I'd my best to follow them, but sometimes it was hopeless. The most partial credit was given for any sequential logic circuit that used more than two bits to store the current state, and to separate the contents of the memory elements from the eventual output itself.