HW 5

Due: 23:00, Wednesday, June 19, 2019

CS ID 1: d6a2b

CS ID 2: b9i2b

Instructions:

1. Do not change the problem statements we are giving you. Simply add your solutions by editing this latex document. To make it easier for the TAs to find your solutions, please use the **soln** environment we provided as follows:

```
\begin{soln}
My solution is here.
\end{soln}
```

Your solution will then appear in blue, and be easier to differentiate from the questions.

2. Include formatting to clearly distinguish your solutions from the given problem text. Improperly or insufficiently typeset submissions will receive a penalty.

3. If you need more space, add a page between the existing pages using the `\newpage` command.

4. Export the completed assignment as a PDF file for upload to Gradescope.

5. On Gradescope, upload only **one** copy per partnership. (Instructions for uploading to Gradescope are posted on the assignments page of the course website.)

6. During submission, for each question, please link ALL pages on which your solution appears. Submissions with several linking errors will incur a small penalty.

7. Late submissions will be accepted up to 24 hours past the deadline with a penalty of 20% of the assignment's maximum value

Academic Conduct: I certify that my assignment follows the academic conduct rules for of CPSC 121 as outlined on the course website. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff.

Version history:

- 2019-06-16 11:58 – Bug corrected in Q6 ALU diagram, Q2 hint updated
- 2019-06-11 18:28 – Initial version for release

tikz

Questions 1 and 2 ask you to design a Deterministic Finite-State Automaton (DFA). You are also required to indicate the meaning of each state. Here is an example to show you what we mean by this, taken from an old assignment. The question was:

Design a deterministic finite-state automaton (DFA) that accepts exactly the strings over the alphabet {A, B, . . . , Z} *that contain at least one* C*, at most two O, and where every D comes before every E. For instance, your DFA should accept the strings*

- *ACCORDEON*
- *CHOCOLATE*
- *CROCODILE*
- *SCALD*
- *SCALE*

but not the strings

- *ADRIAN (it doesn't contain the letter* C*)*
- *COLONIZATION (there are three* Os*)*
- *MANACLED (there is an* E *before a* D*)*
- *BEWARETHEKOMODODRAGON (all of these at the same time)*

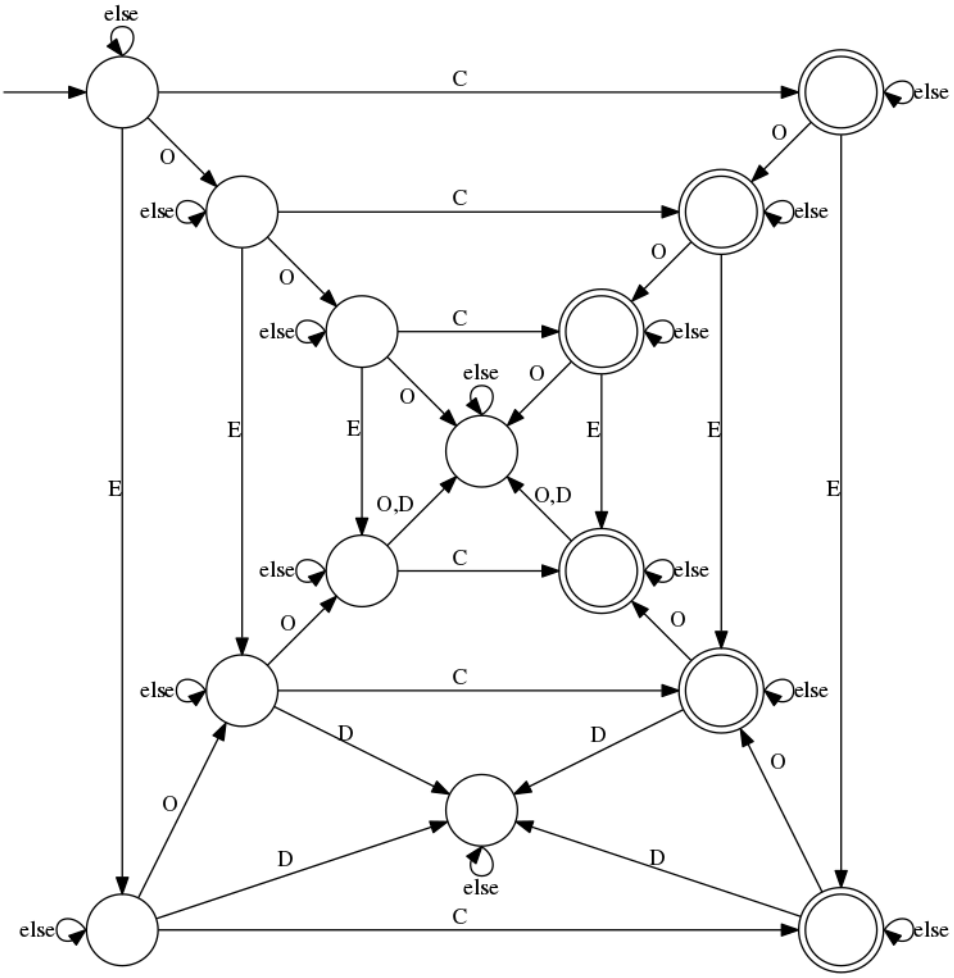
Clearly indicate the meaning of each state. Hint: there are three separate conditions accepted strings must meet; states will need to encode whether or not they are met. You can label an edge with the word "else" to indicate it would contain every character that does not appear on another edge leaving from the same state. For instance, if a state has outgoing edges labeled "A", "B", "E", "P" and "else", then "else" stands for "C, D, F, G, H, I, J, K, L, M, N, O, Q, R, S, T, U, V, W, X, Y, Z".Our solution uses 13 states.

The solution, and the explanation about the meaning of each state are listed below:

Here is a DFA that works. We used 14 states instead of 13 because having two "garbage" states allows the DFA to be drawn without edges crossing. The idea is that we have three separate conditions to check for:

- *Whether or not a "C" was seen. The states in the left half of the picture are those for which no "C" has yet been seen. The states in the right half of the picture are those for which a "C" has been seen.*
- *Whether or not a "E" was seen. The states in the top half of the picture are those for which no "E" has yet been seen. The states in the bottom half of the picture are those for which a "E" has been seen (and hence seeing a "D" will lead the DFA to one of the garbage states).*
- *Whether we have seen no "O", one "O" or two "O"s. These states are drawn in "layers": the outside rectangle contains the states for which no "O" has been seen, the middle rectangle contains the states for which a single "O" has been seen, and the inner rectangle contains the states for which two "O"s have been seen. Once the DFA reaches a state where two "O"s have been seen, any further "O" leads it to a garbage state.*

Twelve of the fourteen states encode all possible combinations of these three conditions. The remaining two are garbage states that are used when one of the requirements are violated (three or more "O"s, or a "D" after an "E") and it is no longer possible for the DFA to accept the input string.


else
C
else
O
O
else
C
else
O
O
else
C
else
else
O
O
E
E
E
E
E
E
O,D
O,D
else
C
else
O
O
else
C
else
D
D
O
O
D
D
else
else
C
else

1. **[6 marks]** To prevent himself from eating too many late-night snacks while preparing lecture notes, Geoff installed an electronic lock on his refrigerator door. The lock comes with an alphabetic keypad, and the door remains locked unless the last eleven letters typed form the word "CHIMICHANGA". Geoff loves chimichangas!

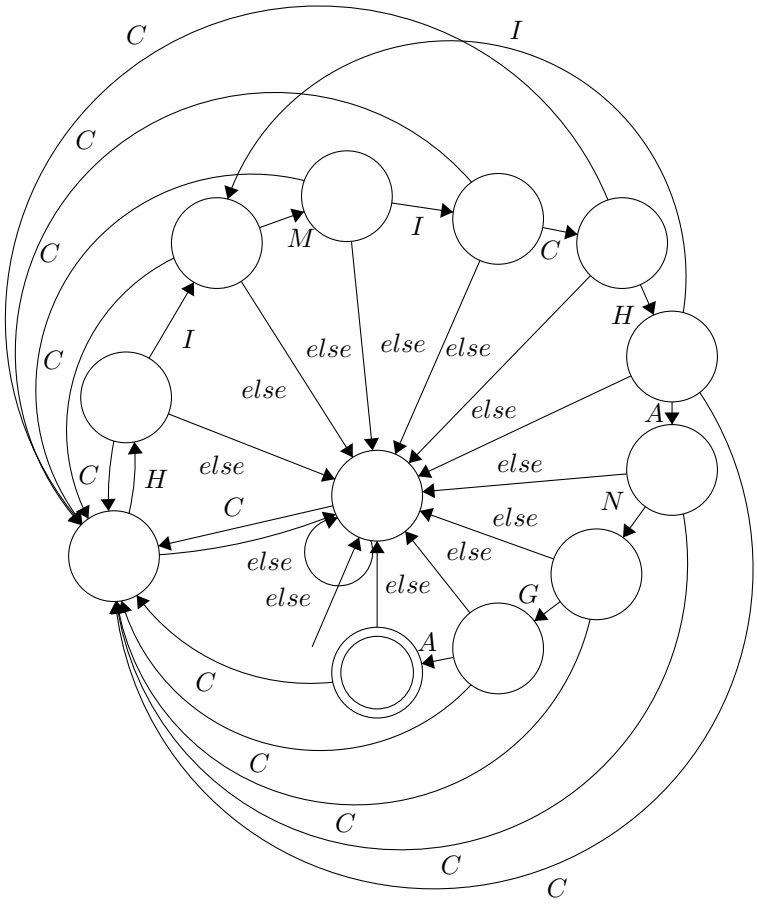
   Design a Deterministic Finite-state Automaton (DFA) which takes in a string s of alphabetic uppercase characters, and accepts s if and only if it ends with CHIMICHANGA. You **must** describe the meaning of each state in your DFA.

   You can label an edge with the word "else" to indicate it would contain every character that does not appear on another edge leaving from the same state. For instance, if a state has outgoing edges labeled "A", "B", "E", "P" and "else", then "else" stands for "C, D, F, G, H, I, J, K, L, M, N, O, Q, R, S, T, U, V, W, X, Y, Z". Hint: our solution has 12 states.

   Here is a DFA that works. We used 12 states just like the hint states. Here are the ideas:

   - We know that it can have any strings before the word CHIMICHAGA, so unless it starts to spell the word, there can be an initial state where the rest of the strings can be in. This is the state in the center of the diagram below.
   - The last letters of the string has to be CHIMICHANGA, in this order, so the state after A has to be an accepting state and redirects to the initial state if there's anymore letters input. These are the states circling around the initial state in the diagram below.
   - If the letters don't spell CHIMICHANGA exactly until the end, then it has to redirect to the initial state to restart spelling the word again, since we want that exact order. These are the lines drawing back to the state in the center in the diagram below.
   - The word can be restarted at any point in the string, so all the states redirect to the first C state if "C" is detected as input at any point.
   - The second "CH" in CHIMICHANGA could also redirect back with "I" as input (for example: CHIMICHIMICHANGA is a valid input).

   We need 11 states to check for the exact order of the letters in the word CHIMICHANGA, and 1 initial state for all the letters before the word CHIMICHANGA to be accepted.


C
I
C
C
M
I
C
H
C
I
else
else
else
else
else
C
H
else
C
else
N
else
else
else
else
G
A
C
C
C
C
C

2. **[8 marks]** Design a Deterministic Finite-state automaton (DFA) that accepts exactly the strings over the alphabet $\{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$ that

   - must contain "NG",
   - does not end with Y,
   - any I it contains must be followed by a S (after any number of other letters including another I).

   For instance, your DFA should accept the strings

   - STRINGS
   - VAINGLORIOUS
   - ENGINEERS
   - MONOPHTHONG

   but not the strings

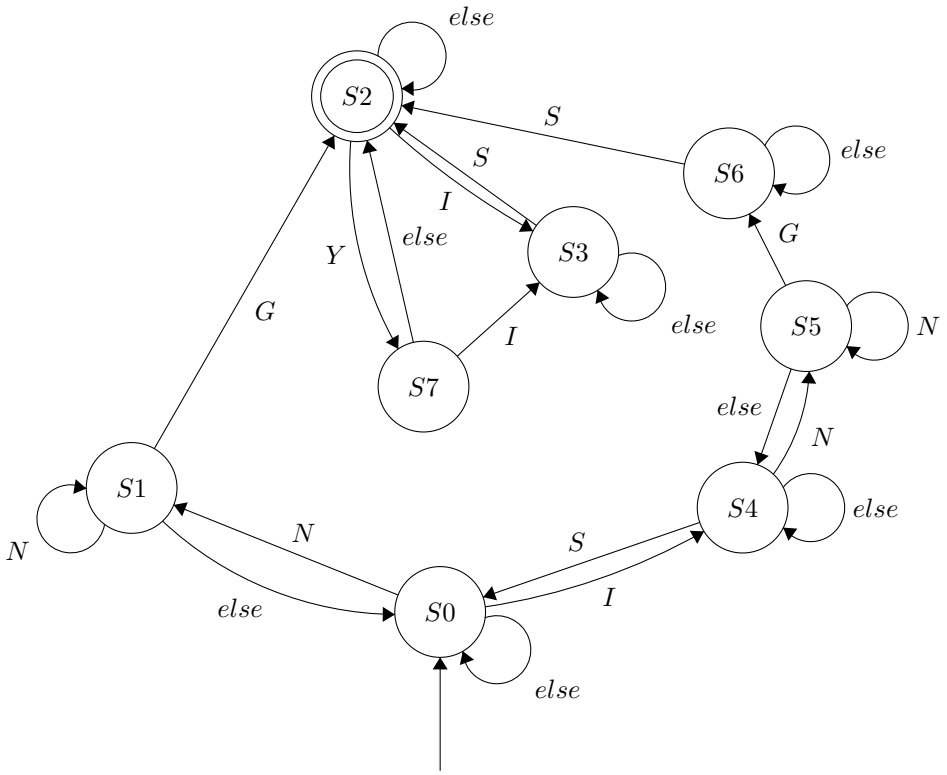
   - TREPIDATIOUS (does not contain NG).
   - INDISTINGUISHABLY (ends with a Y).
   - FINGERPRINT (contains an I that is not followed by S).
   - INQUISITIVELY (violates all rules).

   You **must** describe the meaning of each state in your DFA.

   Hint: there is a solution with 14 states, although having additional garbage states (non-accepting states where every outgoing edge loops back to that state) may help with readability.

   Here is a DFA that works. We used 12 states just like the hint states. We have three conditions to check for. Here are the ideas:

   - If there is "Y" as input after it's reach accepting state and no input afterwards, then it goes to a non-accepting state. This is represented by the "Y" arrow from S2 to S7. If the string is not already at accepting, then it's not accepting whether or not it ends with "Y", so there doesn't need to be a Y state for all individual states. Additionally, there is no way to get into accepting with "Y", so we don't have to worry about that.
   - "NG" must be in this particular order, so if an input after "N" is not "G", we must go back to the previous state. This is represented by the "else" arrows from S5 and S1, where the input to get to that state is "N".
   - If there's "I", then "S" has to exist. Since "NG" has to exist to get to accepting state, it can be before any "I" then "S" [1], in between "I" and "S" [2], after "I" then "S" [3], or "I" then "S" can not exist at all [4]. The first three cases can have overlaps. For example, you can overlap [1] and [2] by after "I" then "S" after and bofore "NG". We can break these 4 cases into parts of the diagram. The order of the states presented are according to which state will be traveled to first. In explained order, [1] is represented by S0, S4, S1, S2; [2] by S0, S4, S5, S6, S2; by [3] is S0, S1, S2, S3; [4] by S0, S1, S2.


else
S2
S
S6
else
S
I
else
S3
G
Y
G
else
S5
N
S7
I
else
else
N
S1
S4
else
N
N
S
else
S0
I
else

3. **[8 marks]** Using only three (3) D flip-flops and any strictly combinational devices available in Logisim's Gates, Plexers, and Arithmetic component libraries, construct a synchronous device with no external control inputs that continually generates the sequence of unsigned values: 16, 13, 27, 17, 12, 26, (repeat from 16).

   Hint: Observe the least significant 3 bits of each unsigned value, and use patterns from those to construct the complete numbers.

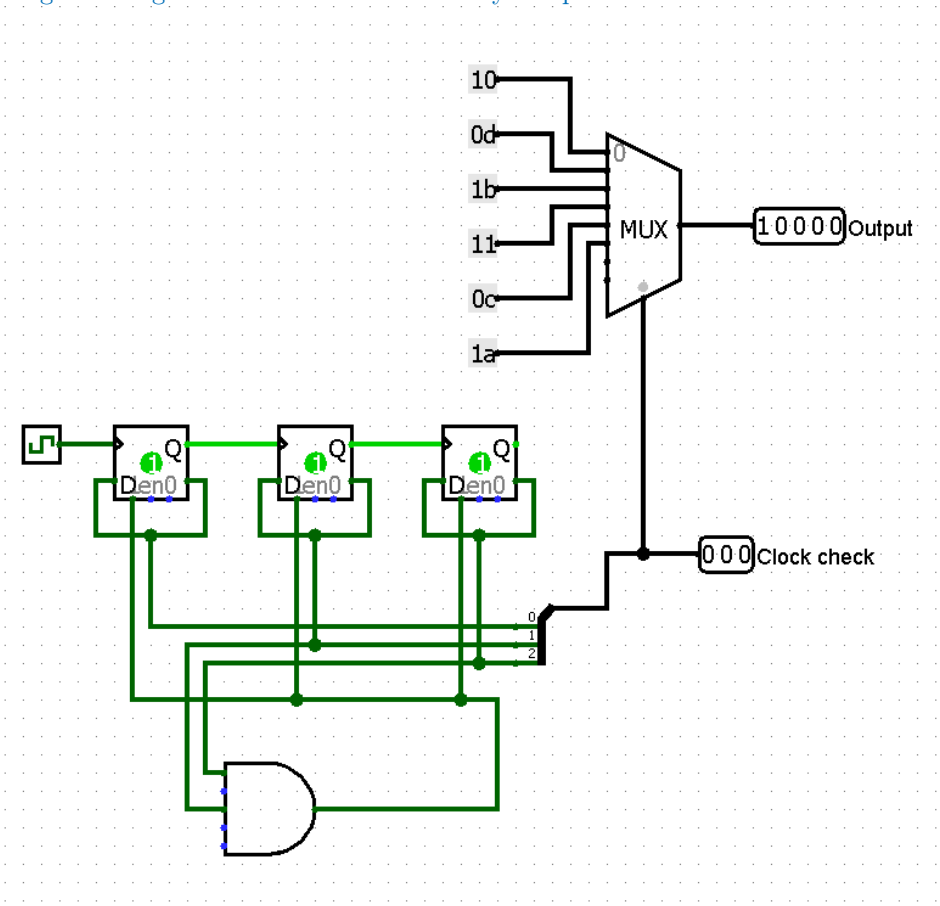
   Since we can use any plexers and 3 D-flip-lops to store at most 8 states, we can make a plexer into a look-up table and make the flip-flop values reset to 0 when it reaches 6 (so we have state 0,1,2,3,4,5, in total 6 states). In other words:

   MUX table:

$Q2$	$Q1$	$Q0$	Output in hex
0	0	0	10
0	0	1	0D
0	1	0	1B
0	1	1	11
1	0	0	0C
1	0	1	1A
1	1	0	X
1	1	1	X

   Flip flop reset condition = $Q2 \wedge Q1$

   Note that there is a way to do this without MUX, with flip flop outputs as inputs to digital circuits where output is each binary digits. That requires constructing 5 3-inputs k-maps. However, the output values from MUX are constants, so they wouldn't be counted as external control inputs, so doing it through a circuit adds unneccessary complications.



4. **[8 marks]** Using mathematical induction, prove that

$$\sum_{i=1}^{n}(i-1)2^i = (n-2)2^{n+1} + 4$$

Proof:

Part 1: Base case

$$(n-1)2^n = (n-2)2^{n+1} + 4 \tag{1}$$
$$(1-1)2^1 = (1-2)2^{1+1} + 4 \tag{2}$$
$$0 = 0 \tag{3}$$

Part 2: Induction step

Assuming that n case is true, we have to prove that this statement is true:

$$\sum_{i=1}^{n+1}(i-1)2^i = ((n+1)-2)2^{(n+1)+1} + 4 \tag{4}$$

To prove that [4] is true, we say that the sum of n+1 terms is equals to the sum of n terms plus the n+1th term, and simplify from there:

$$\sum_{i=1}^{n+1}(i-1)2^i = \sum_{i=1}^{n}(i-1)2^i + ((n+1)-1)2^{n+1} \tag{5}$$
$$((n+1)-2)2^{(n+1)+1} + 4 = ((n-2)2^{n+1} + 4) + ((n+1)-1)2^{n+1} \tag{6}$$
$$(n-1)2^{n+2} + 4 = (n-2)2^{n+1} + n2^{n+1} + 4 \tag{7}$$
$$(n-1)2^{n+2} = (n-2)2^{n+1} + n2^{n+1} \tag{8}$$
$$n2^{n+2} - 2^{n+2} = (n-2)2^{n+1} + n2^{n+1} \tag{9}$$
$$n2^{n+2} - 2^{n+2} = n2^{n+1} - 2^{n+2} + n2^{n+1} \tag{10}$$
$$n2^{n+2} - 2^{n+2} = n2^{n+1} - 2^{n+2} + n2^{n+1} \tag{11}$$
$$n2^{n+2} - 2^{n+2} = 2n2^{n+1} - 2^{n+2} \tag{12}$$
$$n2^{n+2} - 2^{n+2} = n2^{n+2} - 2^{n+2} \tag{13}$$

Steps 5-7 is substituting terms. Step 8 cancels 4 from both sides. Steps 9-10 expands (n-1) and (n-2) terms by distributive property. Steps 11-13 combine like terms to get the final equality.
Thus, we have proved that when n case is true, n+1 case is true as well.

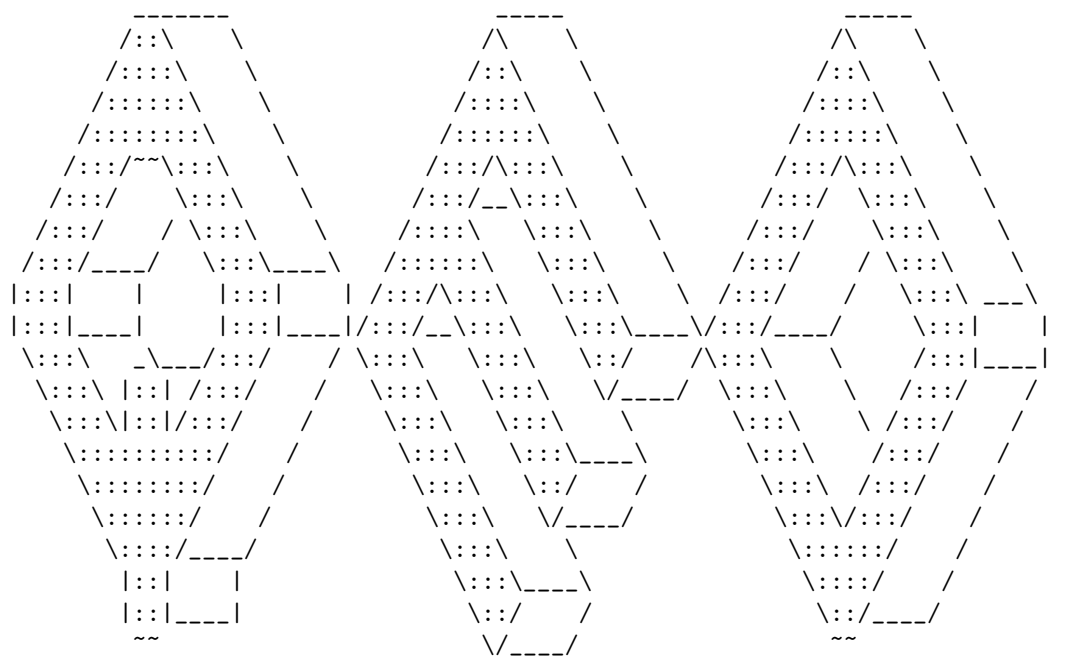
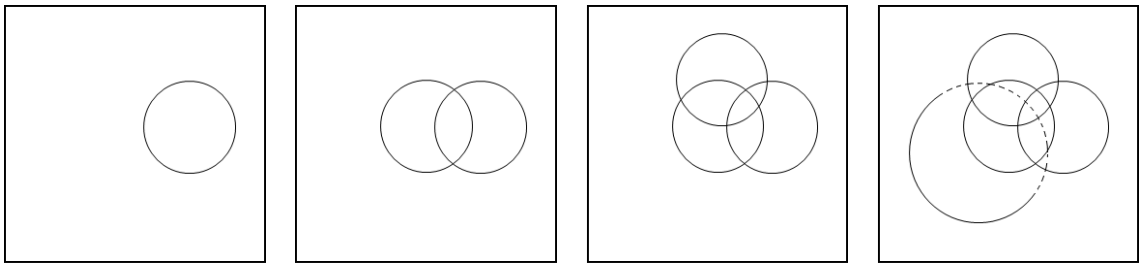


Figure 1. A blessed QED to end a blessed proof

5. **[9 marks]** As soon as the grades for CPSC 121 have been submitted to the department, Geoff will celebrate the start of his summer break by using his car to draw perfect circles of melted rubber in the parking lot across the street from ICCS. He draws his doughnuts according to the following rules:

   - No circle can touch or intersect the (rectangular) boundaries of the parking lot, and
   - Each new circle must intersect with the other circles such that it produces the maximum number of separate regions in the parking lot.

   The maximum number of regions produced by n circles (including the outside of the circles) for some small values of n are shown below:



$R(1) = 2$ $R(2) = 4$ $R(3) = 8$ $R(4) = ?$

   a. **[1 marks]** What is the value of $R(4)$? You should trace the dotted portion of the circle to see exactly how regions are created. If your answer does not seem to fit the obvious pattern, you are probably on the right track.

   R(4) = 14

   b. **[2 marks]** The arrangement of circles that achieves the maximum number of regions has the property that every pair of circles intersects twice. Given this fact, complete the following recurrence relation for $R(n)$:

   $R(1) = 2$
   $R(n) = R(n-1) + 2(n-1)$ for $n > 1$

   c. **[6 marks]** Geoff has determined (by techniques he learned in CPSC 221) that the formula for the maximum number of circles is $R(n) = n^2 - n + 2$.
   Using mathematical induction and your recursive definition found in part (5b), prove that this formula is correct for all $n \geq 1$.

   Proof:

   Part 1: Base case
   Using formula: $R(2) = 2^2 - 2 + 2 = 4$
   Using recursive definition: $R(2) = R(1) + 2 * (2 - 1) = 2 + 2 = 4$

   Part 2:
   Assuming $R(n) = n^2 - n + 2$ and $R(n) = R(n-1) + 2(n-1)$ are true:

   $R(n+1) = (n+1)^2 - (n+1) + 2$ (14)
   $R(n+1) = R(n) + 2(n+1-1) = n^2 - n + 2 + 2(n)$ (15)
   $(n+1)^2 - (n+1) + 2 = n^2 - n + 2 + 2(n)$ (16)
   $(n+1)^2 - (n+1) + 2 = n^2 + n + 2$ (17)
   $n^2 + 2n + 1 - (n+1) + 2 = n^2 + n + 2$ (18)
   $n^2 + n + 2 = n^2 + n + 2$ (19)

Line 14-15 states out the two forms of R(n+1) equations. Line 16 equates the two forms. Line 17 combines like terms of RHS. Line 18-19 expands exponents and combine like terms of LHS.

We have proved that the base case is true and n+1 case is true when n case is true, therefore the induction proof is complete.

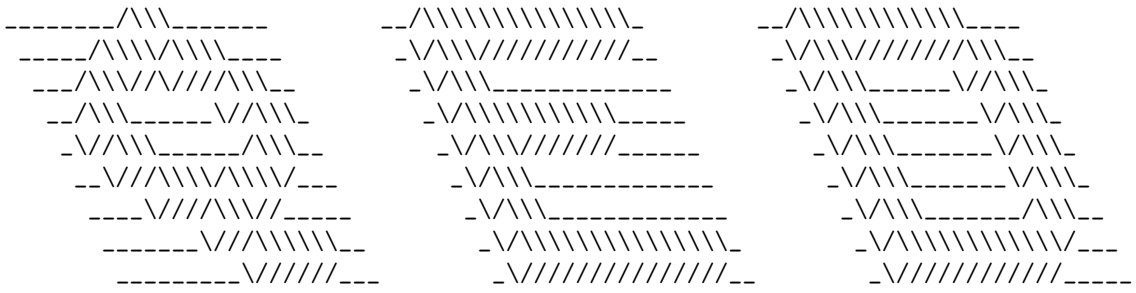
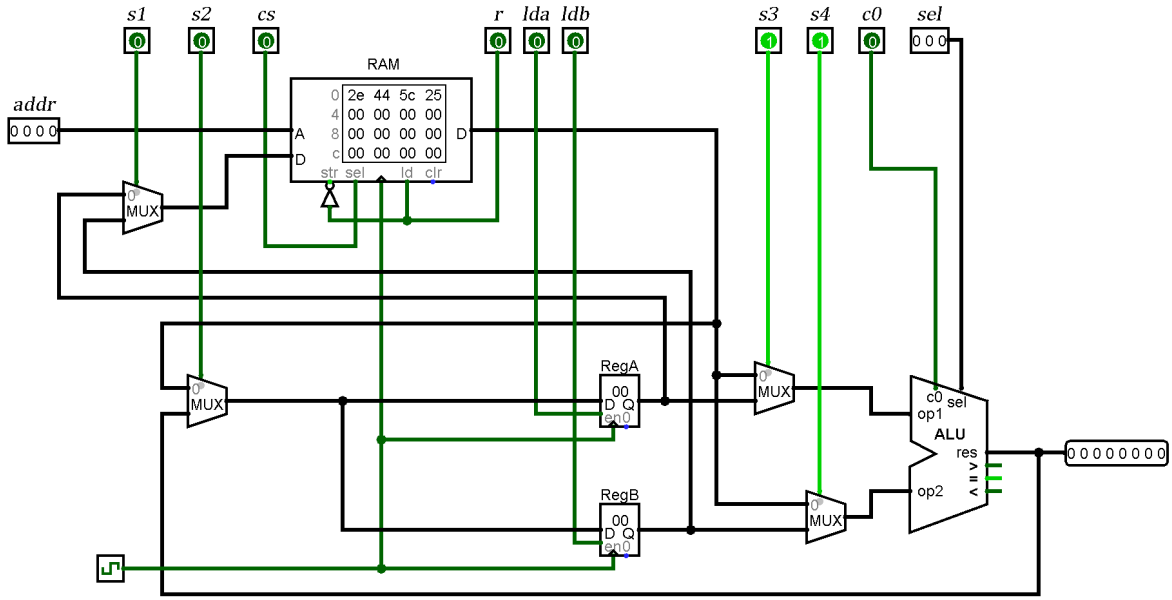


Figure 2. Another blessed QED to end another blessed proof

6. **[16 marks]** Consider the partial CPU datapath given here (found in hw5_datapath.circ):



This datapath contains two 8-bit parallel load registers A and B, a $2^4 \times 8$ RAM, and an 8-bit ALU with the function table below (NOTE - this is more or less the ALU described during the combinational logic lecture):

sel	Function	*sel*	Function
0 0 0	$res = op1 + c0$	1 0 0	$res = c0$
0 0 1	$res = op1 + op2 + c0$	1 0 1	$res = op2 + c0$
0 1 0	$res = op1 + \sim op2 + c0$	1 1 0	$res = \sim op2 + c0$
0 1 1	$res = op1 \wedge op2$	1 1 1	$res = op1 \vee op2$

For the next four parts of this question, you are to analyze the circuit to determine a sequence of appropriate assignments to every control input of the datapath (including *addr*), so that the final result of the required computation is stored into RAM at the specified address(es).

Some notes and guidelines:

- Each 8-bit word in RAM is shown as hexadecimal (e.g. $\texttt{2e}_{16} = 46_{10}$). Assume that negative numbers are represented using two's complement.
- Carefully study the diagram to observe the flow of 8-bit data values to/from registers and RAM.
- If any synchronous device is not involved in one of your intended operations, ensure that the device does not overwrite its contents during the operation (i.e. disable the device using *ld_* or *cs* appropriately).
- In most cases, it is better practice to move needed values from RAM into the work registers for computation, and once a work register contains a result (or intermediate result), move it back to RAM unless you can use it again immediately.
- You may not overwrite your given initial data values in RAM.
- In completing the provided tables, write a 0 or 1 if the control input *must* receive a 0 or 1 for the current operation. If the control input can receive either 0 or 1 without affecting the intended operation, write X (don't care).
- In the "Operation" column of the table, use register transfer notation to write a reasonable summary of the general operation.
- If testing your mini-program on the provided Logisim file, operations combining RAM retrieval with register storage and ALU *might* require 2 clock cycles (not accurate to real-world microcontrollers, but it's the best Geoff could do in Logisim with the built-in RAM and register devices and this datapath configuration). Make sure your register load control is only active on the second clock cycle.

(a) **[4 marks]** Assume that $RAM[0]$ contains 46_{10}, $RAM[1]$ contains 68_{10}, $RAM[2]$ contains 92_{10}, and $RAM[3]$ contains 37_{10}. Other entries are unknown and can be overwritten. Determine a sequence of control input assignments to perform $(46 + 68) \uparrow (92 - 37)$ and store the final result into $RAM[15]$. The first operation is completed for you as an example.

Operation	*addr*	s_1	s_2	*cs*	*r*	*lda*	*ldb*	s_3	s_4	*c0*	*sel*
$RegA \leftarrow RAM[0]$	0000	X	0	1	1	1	0	X	X	X	XXX
$RegB \leftarrow RegA + RAM[1]$	0001	X	1	1	1	0	1	1	0	0	001
$RAM[4] \leftarrow RegB$	0100	1	X	1	0	0	0	X	X	X	XXX
$RegB \leftarrow RAM[3]$	0011	X	0	1	1	0	1	X	X	X	XXX
$RegB \leftarrow RAM[2] + \neg RegB + 1$	0010	X	1	1	1	0	1	0	1	1	010
$RegB \leftarrow RAM[4] \wedge RegB$	0100	X	1	1	1	0	1	0	1	X	011
$RegB \leftarrow \neg RegB$	XXXX	X	1	0	X	0	1	X	1	0	110
$RAM[15] \leftarrow RegB$	1111	1	X	1	0	0	0	X	X	X	XXX

(b) **[4 marks]** Assume that $RAM[3]$ contains 73_{10}. Other entries are unknown and can be overwritten. Determine a sequence of control input assignments, so that -70_{10} is stored into $RAM[15]$.

Operation	*addr*	s_1	s_2	*cs*	*r*	*lda*	*ldb*	s_3	s_4	*c0*	*sel*
$RegA \leftarrow RAM[3]$	0011	X	0	1	1	1	0	X	X	X	XXX
$RegB \leftarrow 0$	XXXX	X	1	0	X	0	1	X	X	0	100
$RegA \leftarrow RegA + \neg RegB$	XXXX	X	1	0	X	1	0	1	1	0	010
$RegA \leftarrow RegA + \neg RegB$	XXXX	X	1	0	X	1	0	1	1	0	010
$RegB \leftarrow RegA + \neg RegB$	XXXX	X	1	0	X	0	1	1	1	0	010
$RegB \leftarrow \neg RegB + 1$	XXXX	X	1	0	X	0	1	X	1	1	110
$RAM[15] \leftarrow RegB$	1111	1	X	1	0	0	0	X	X	X	XXX

(c) **[4 marks]** Assume that $RAM[0]$ contains 16_{10}, $RAM[1]$ contains 38_{10}, and $RAM[2]$ contains 61_{10}. Other entries are unknown and can be overwritten. Determine a sequence of control input assignments, so that $(38 + 61) \bmod 16$ is stored into $RAM[15]$.

Operation	*addr*	s_1	s_2	*cs*	*r*	*lda*	*ldb*	s_3	s_4	*c0*	*sel*
$RegA \leftarrow RAM[2]$	0010	X	0	1	1	1	0	X	X	X	XXX
$RegA \leftarrow RegA + RAM[1]$	0001	X	1	1	1	1	0	1	0	0	001
$RegB \leftarrow RAM[0]$	0000	X	0	1	1	0	1	X	X	X	XXX
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RegA \leftarrow RegA + \neg RegB + 1$	XXXX	X	1	0	X	1	0	1	1	1	010
$RAM[15] \leftarrow RegA$	1111	0	X	1	0	0	0	X	X	X	XXX

(d) **[4 marks]** All entries are unknown and can be overwritten. Store 00_{16} into $RAM[0]$ and $RAM[2]$, and store $\texttt{ff}_{16}$ into $RAM[1]$ and $RAM[3]$.

Operation	*addr*	s_1	s_2	*cs*	*r*	*lda*	*ldb*	s_3	s_4	*c0*	*sel*
$RegB \leftarrow 0$	XXXX	X	1	0	X	0	1	X	X	0	100
$RAM[0] \leftarrow RegB$	0000	1	X	1	0	0	0	X	X	X	XXX
$RAM[2] \leftarrow RegB$	0010	1	X	1	0	0	0	X	X	X	XXX
$RegB \leftarrow \neg RegB$	XXXX	X	1	0	X	0	1	X	1	0	110
$RAM[1] \leftarrow RegB$	0001	1	X	1	0	0	0	X	X	X	XXX
$RAM[3] \leftarrow RegB$	0011	1	X	1	0	0	0	X	X	X	XXX

You may have noticed that some of the operations you are asked to do (e.g. NAND), are not directly supported by the ALU. It is the job of the **compiler** to turn a high-level instruction into a sequence of ALU-supported low-level instructions that produce the required result. Thank you for your effort in this course, good luck with the rest of your studies!