# Homework #2

(Deadline: 11:59PM PDT, February 11, 2020)

Name (Last, First): Arora, Gurbir

Student Id #: 105178554

**INSTRUCTIONS**

**This homework is to be done individually. You may use any tools or refer to published papers or books, but may not seek help from any other person or consult solutions to prior exams or homeworks from this or other courses (including those outside UCLA). You're allowed to make use of tools such as Logisim, WolframAlpha (which has terrific support for boolean logic) etc.**

**You must submit all sheets in this file based on the procedure below. Because of the grading methodology, it is much easier if you print the document and answer your questions in the space provided in this problem set. It can be even easier if you answer in electronic form and then download the PDF. Answers written on sheets other that the provided space will not be looked at or graded. Please write clearly and neatly - if we cannot easily decipher what you have written, you will get zero credit**

**SUBMISSION PROCEDURE: You need to submit your solution online at Gradescope (https://gradescope.com/). Please see the following guide from Gradescope for submitting homework. You'd need to upload a PDF and mark where each question is answered.**
**http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf**

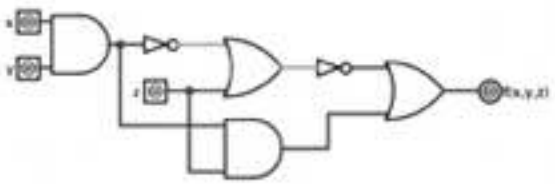
**Simulation #1**

(a) Draw, in Logisim, the schematic of the following unsimplified logic equation

$$f(x, y, z) = \overline{(x \cdot y)} + z + (x \cdot y \cdot z)$$

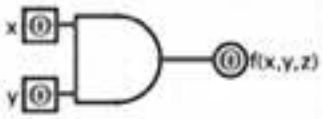
(b) Simplify the function.

(c) Draw the simplified function in Logisim, and show that it is equivalent to the unsimplified one.



b) $f(x,y,z) = \overline{(\overline{x \cdot y}) + z} + (x \cdot y \cdot z) = \overline{(\bar{x} + \bar{y}) + z} = x \cdot y \cdot \bar{z} + xyz$

$= x \cdot y(\bar{z} + z) = xy(1) = \boxed{xy}$




| x | y | fxyz |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| x | y | z | fxyz |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Simulation #2**

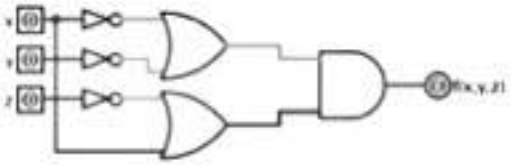
Implement the logic function

$f(x, y, z) = (\overline{x} + \overline{y}) \cdot (x + \overline{z})$

Test the output of function $f(x, y, z)$ using different input combinations in Logisim and fill the following truth table.

| x | y | z | f (x, y, z) |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

Verify the Logisim outputs using the results calculated from Boolean algebra.



| x | y | z | fxyz |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

**Problem #1**

Consider the following logic of 4 inputs, *A[3:0]*, and 1 output, Y. The output is a 1 when *A[3:0]* is a multiple of 3 (including 0) and don't care when A[3:0] is a multiple of 4 (excluding 0).

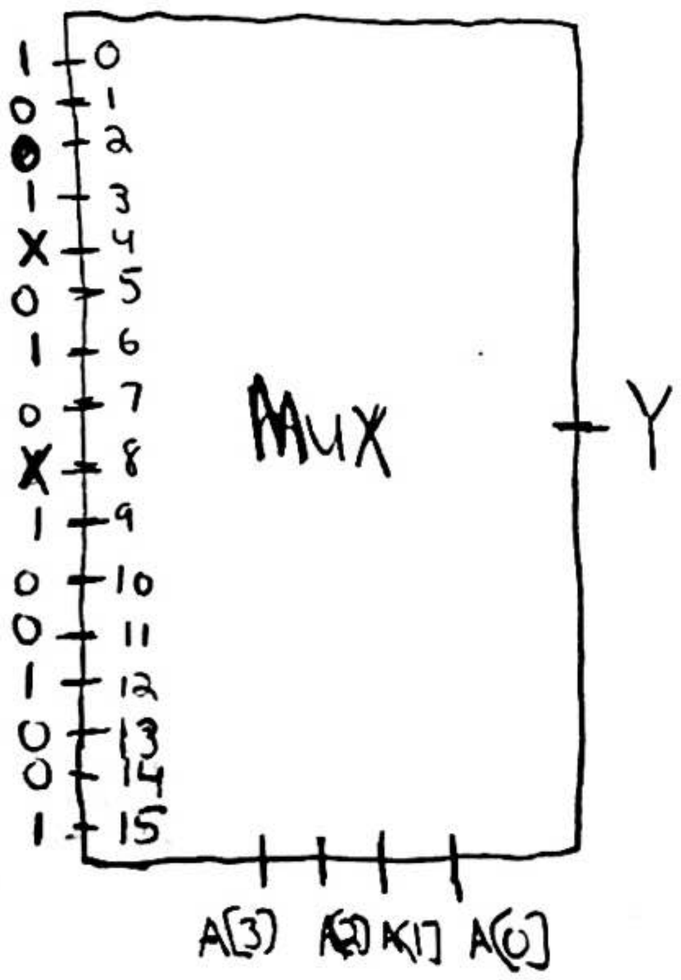
(a) Use a 16-input multiplexer to produce the output with *A[3:0]* as the select inputs.

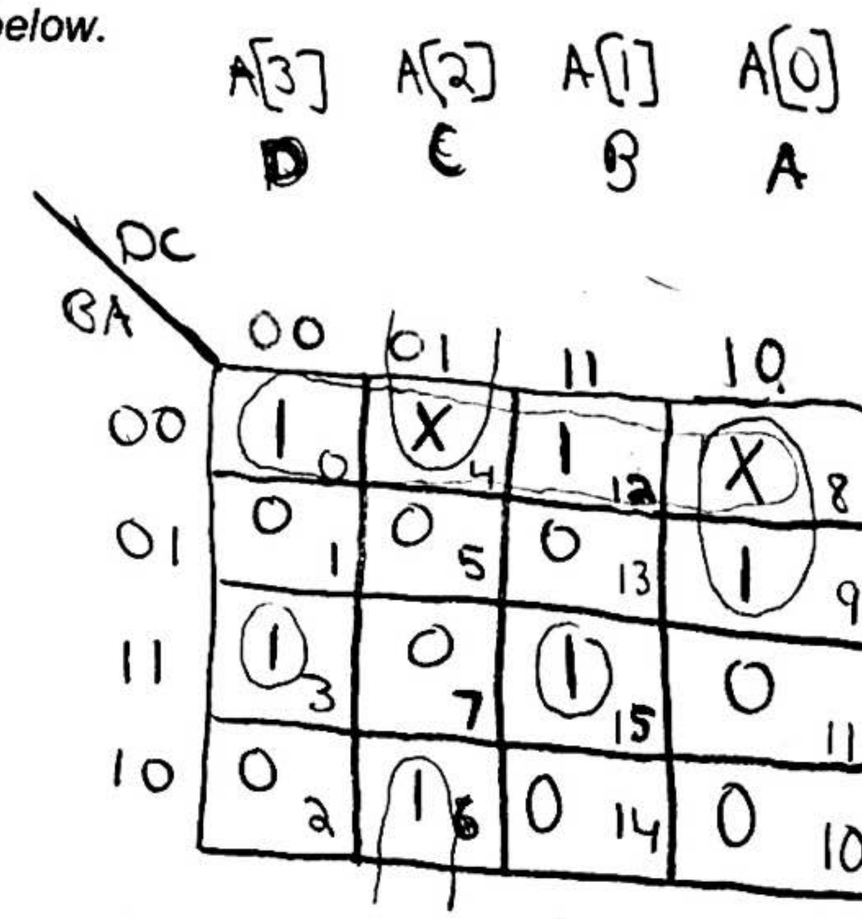
(b) Now assuming that *A[3]* is allowed to be a data input to the multiplexer (as opposed to being one of the select signals). Implement the same logic using an 8-input multiplexer.

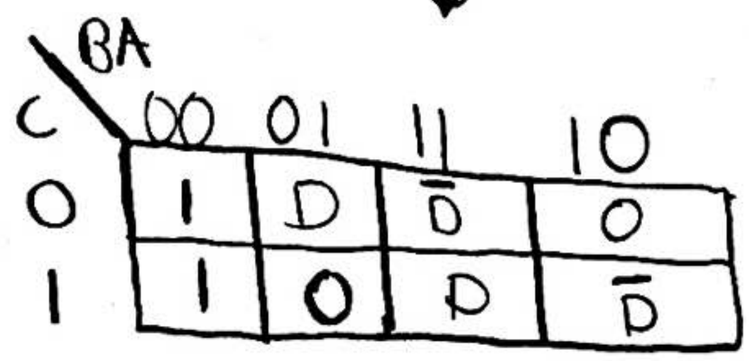
(c) Implement the same logic as (b) assuming that you can only use a tree of 2-input multiplexers. Similar to (b), *A[1]* are the select inputs of the 2-input multiplexers closest to the inputs, *A[0]* controls the next set of multiplexers, and *A[2]* controls the multiplexer closest to the output. Since don't care can be either a 1'b1 or 1'b0, minimize the number of 2-input multiplexers.

*Answer the question for all parts in the space below.*

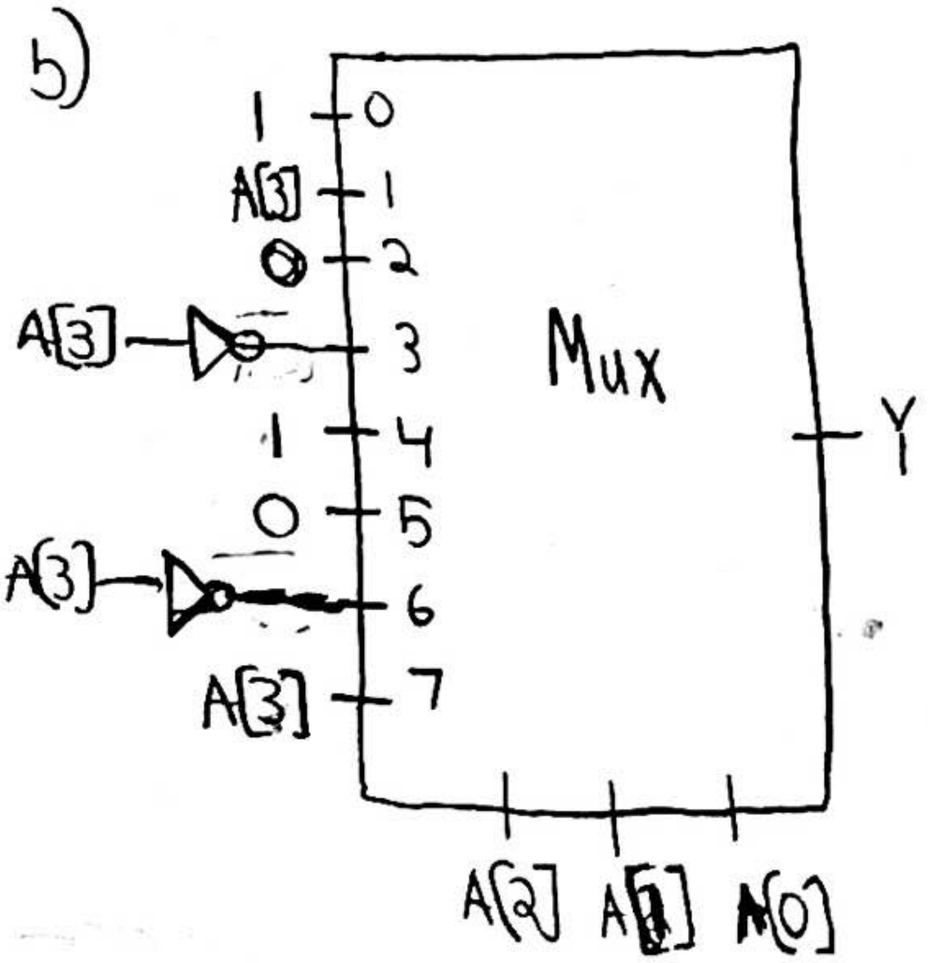
a)



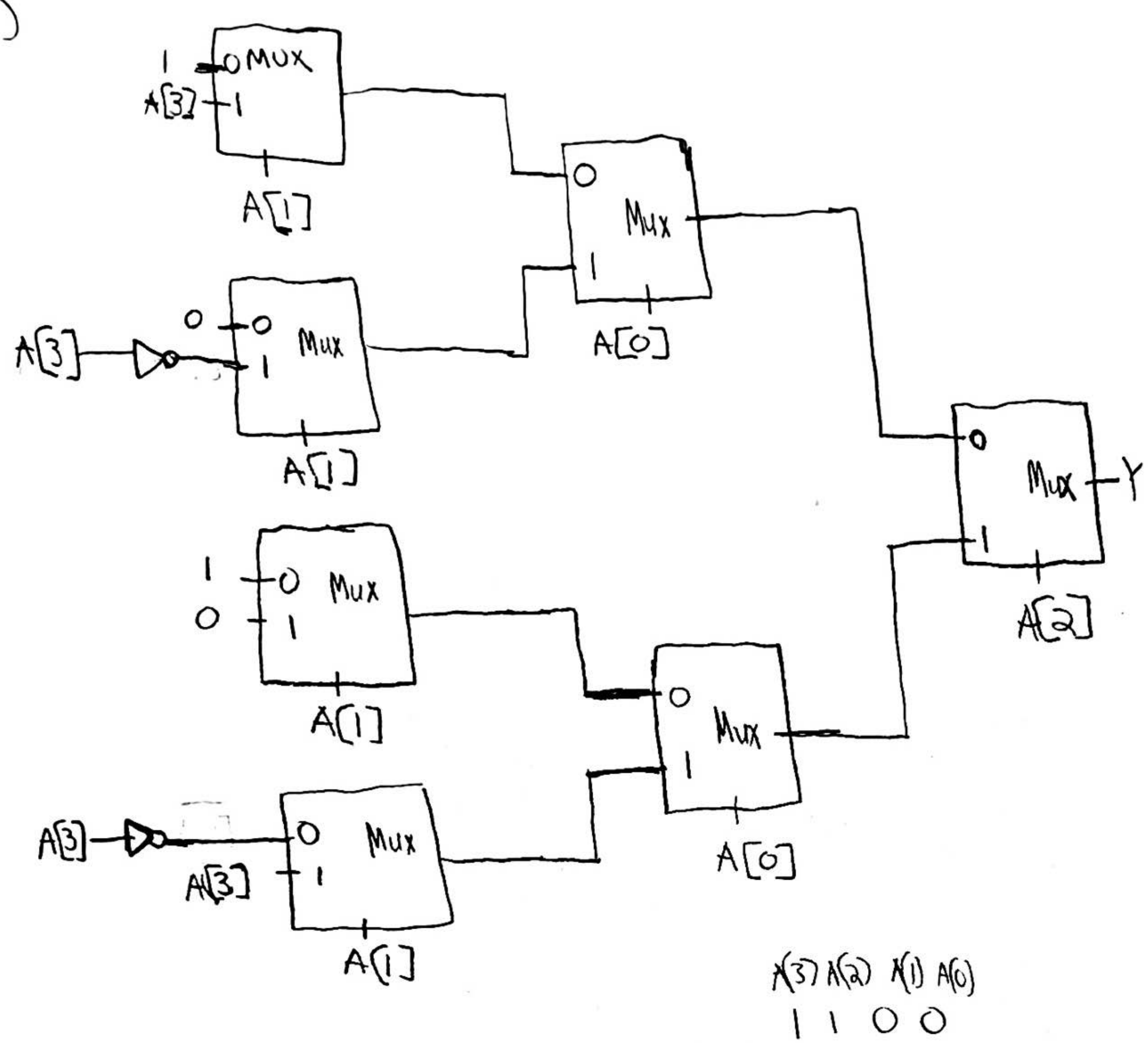




b)



1c)




A(3) A(2) A(1) A(0)

1 1 0 0

**Problem #2**

Assume you have 3 binary inputs of 2-bits each, *a[1:0]*, *b[1:0]*, and *c[1:0]*. We add these numbers together to produce an output.

(a) If this is an unsigned addition, how many bits, *r*, is the output, *result[r-1:0]*?

(b) Determine the logical expression, as simplified a boolean expression as possible or minimal sum-of-product, for the MSB, *result[r-1]*. Incidentally, using this type of dedicated logic to calculate the value of a more significant bit position is the principle behind carry-lookahead adders.

(c) If you implement the summation using 1-bit adders (with 3 inputs, a sum output and a carry output), show the design using the fewest adders.

*Answer the question for all parts in the space below.*

a) XX + XX + XX

max = $11_2 + 11_2 + 11_2 = 9_{10}$, which requires 4 bits, so r=4, results[3:0]

b)

| 8 | 4 | 2 | 1 | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | 10 |
| 1 | 0 | 1 | 1 | 11 |
| 1 | 1 | 0 | 0 | 12 |
| 1 | 1 | 0 | 1 | 13 |
| 1 | 1 | 1 | 0 | 14 |
| 1 | 1 | 1 | 1 | 15 |

MSB
results[3] results[2] results[1] results[0]

XX + XX + XX = XXXX

AB + CD + EF = XXXX
a[1:0] + b[1:0] + c[1:0]

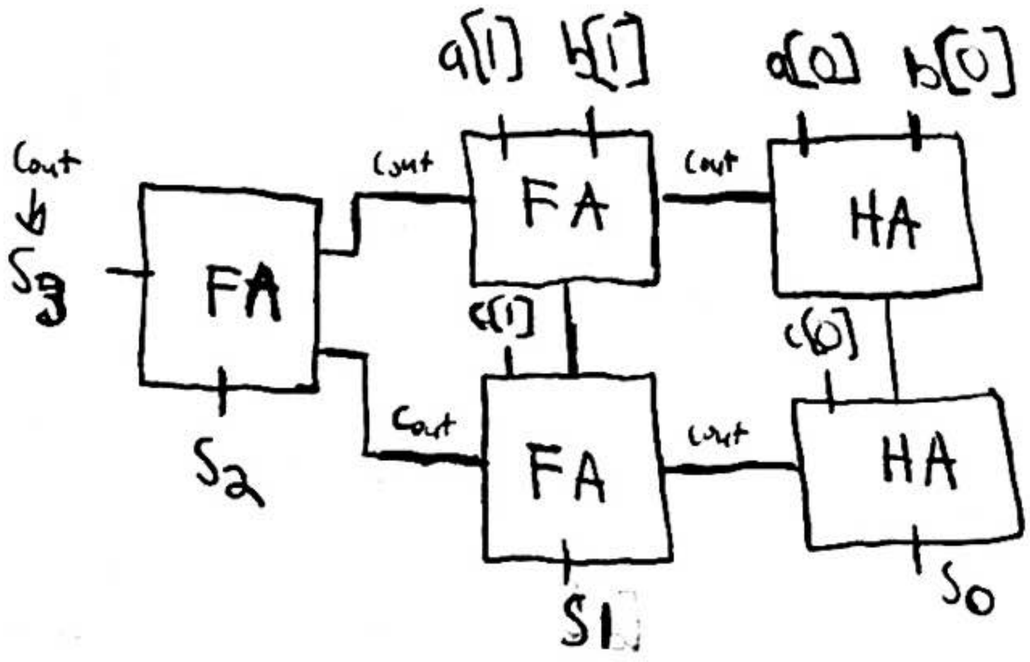
ACE(BD + DF + BF)

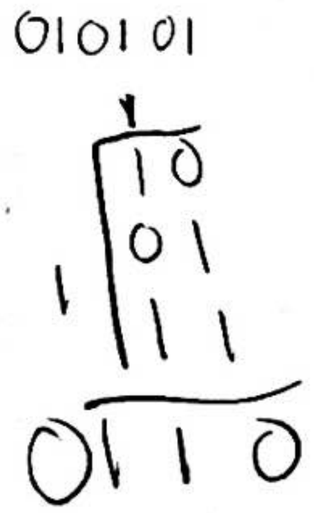
A = a[1], B = a[0], C = b[1], D = b[0],

E = c[1], F = c[0]

MSB = ACE(BD + F(D+B))

c)



## Problem #3

Recall in Homework #1, we represented the 4 DNA nucleotides using binary encoding. G=2'b00, C=2'b11, A=2'b10, and U=2'b01. For this problem, the input is a DNA segment that has 27 nucleotides (9 amino acids).

(a) We would like to count the number of nucleotides of each type. Use the building blocks approach and design the combinational logic. The output should be 4 binary counts corresponding to the 4 nucleotides of 5-bits each. Hint: You should consider converting the binary representation of the nucleotides into one-hot encoding.

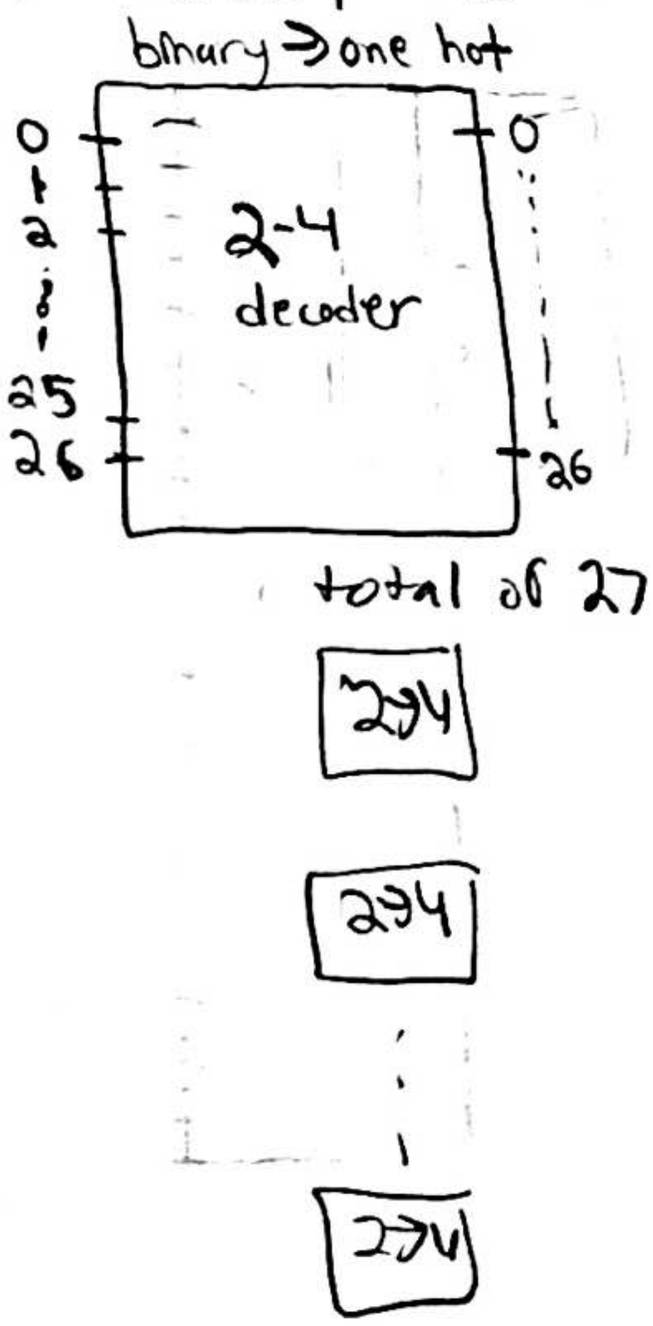
(b) Once we have the 4 binary numbers from part (a), we would like to indicate which nucleotides have the greatest count. The output should comprise of the highest count, *max_cnt[4:0]*, and four 1 bit signals, one for each nucleotide indicating if it is the greatest, *{gtT, gtA, gtC, and gtG}*. Note that since the counts can be equal, more than 1 of the greatest signals can be asserted 1'b1.

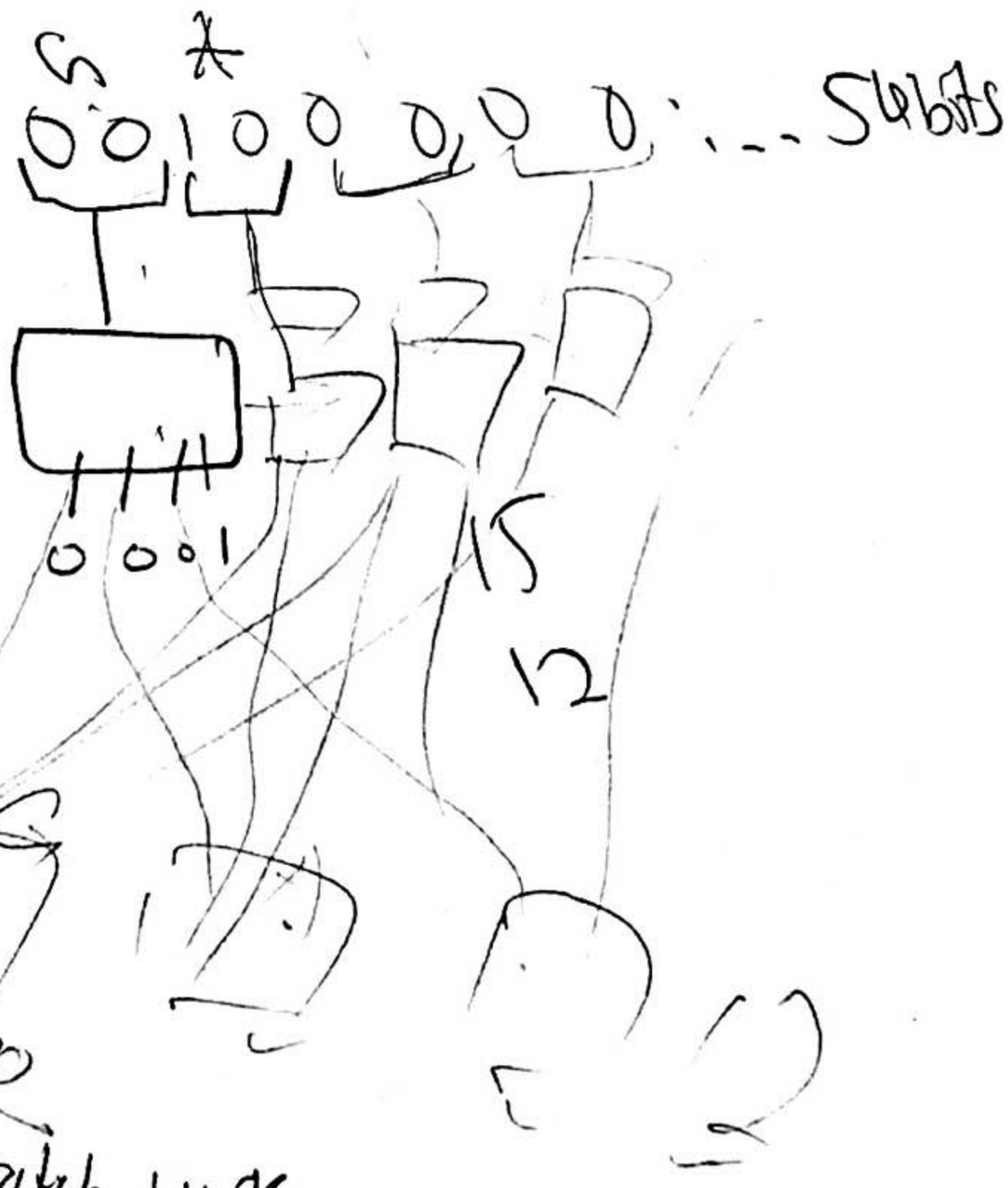
Within any part of your design, you may also use a block as a module that we have covered in class and not implement/draw the design down to the gate level. In cases where a block is not one we covered, you should show this design of the block down to the gate level (once) and then you can use the block as a module.

*Answer the question for all parts in the space below.*

G=2'b00 C=2'b11 A=2'b10 T=2'b01

G= 0001 C=1000 A=0100 T=0010



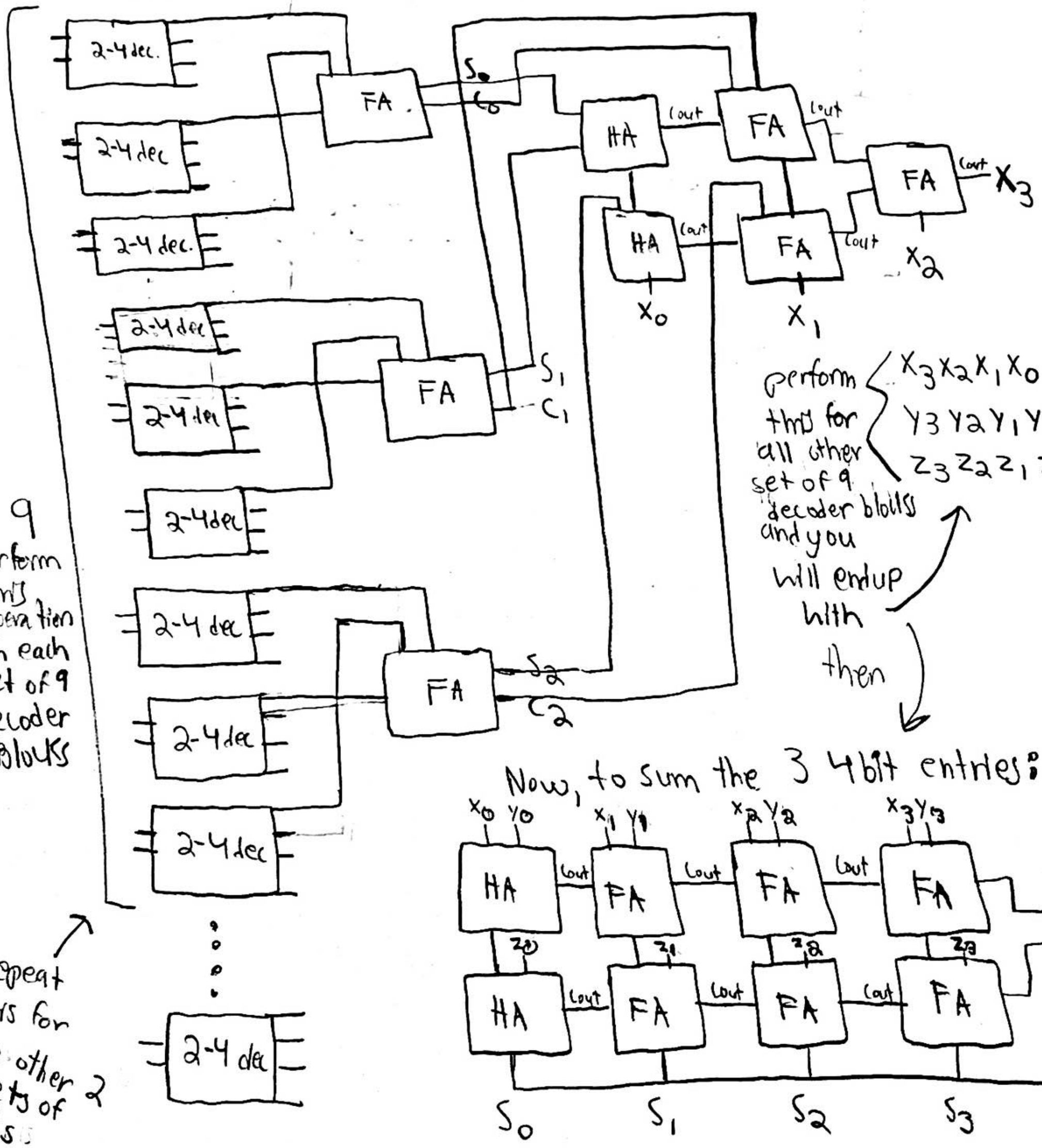




A idea/scratch work part a begins on next page

**(IMPORTANT: Keep this page in submission even if left unused)**

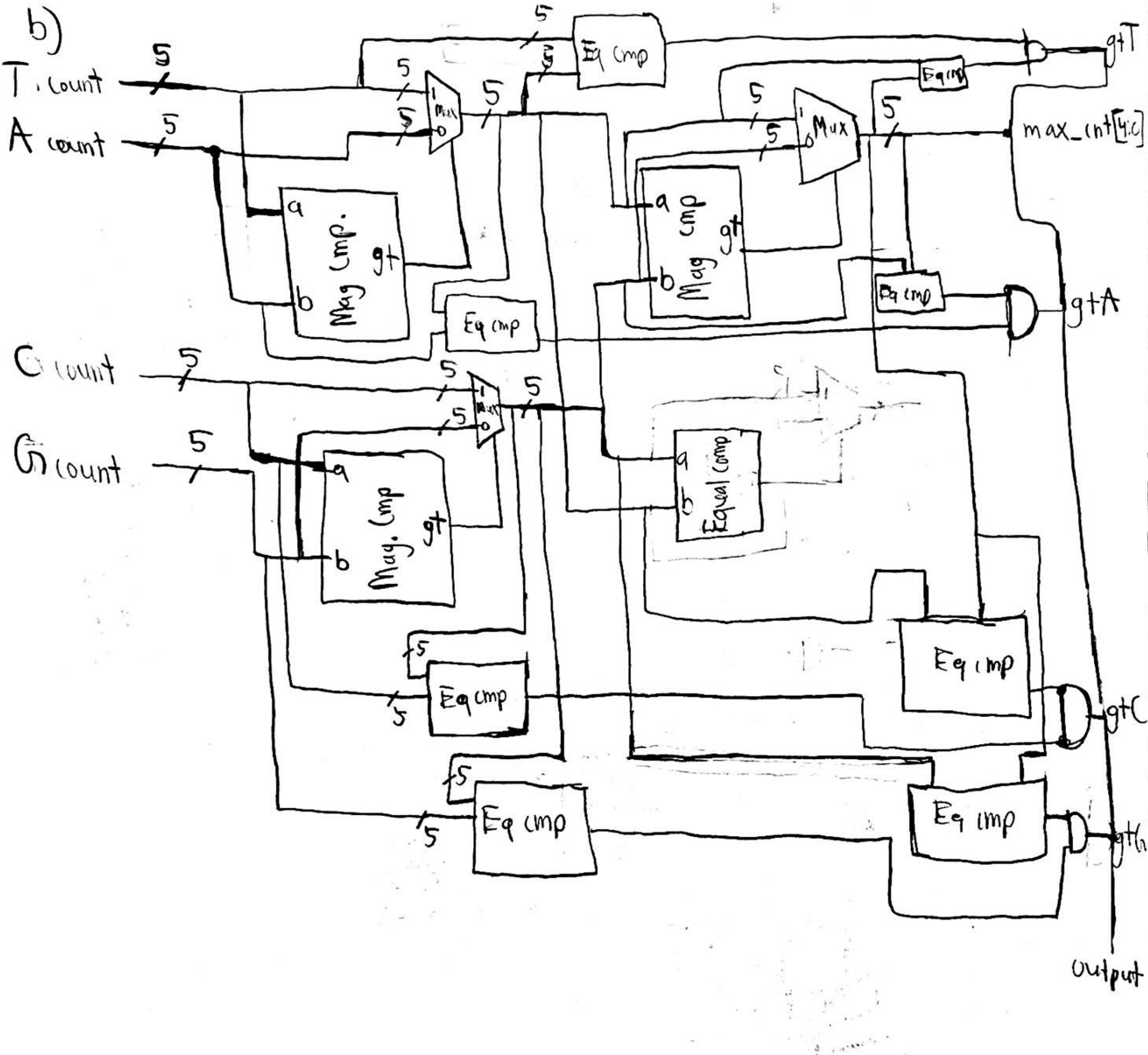
9)

decode 27 times



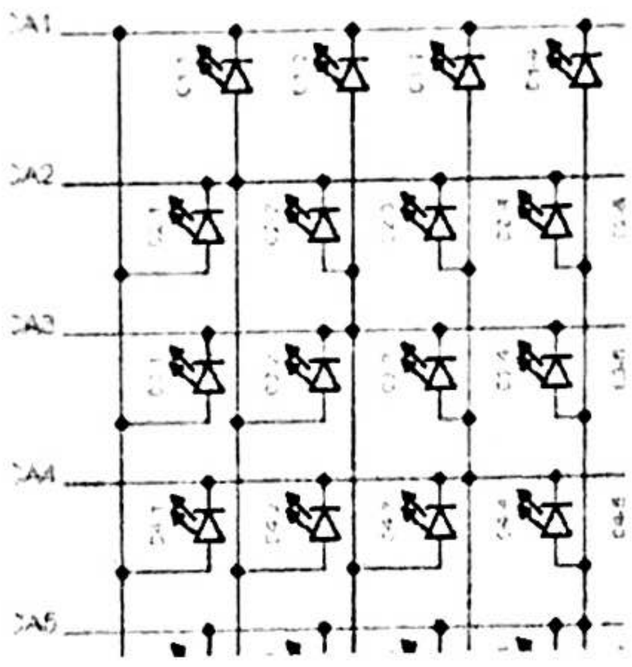

$S_4S_3S_2S_1S_0$ is the # of Cs that appear, in order to find the # of A, G, T s, use this same model, but on the 2nd, 3rd, 4th bits of the one hot output from the 2-4 decoder

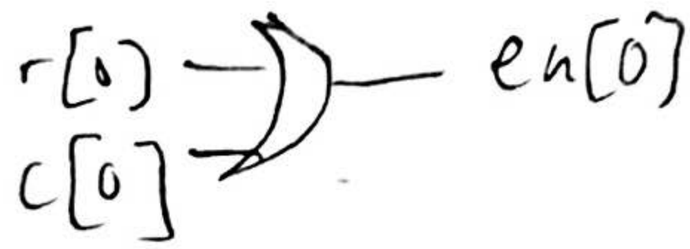
b)


T count
A count
C count
G count
5
Mux
Mag. Cmp.
gt
a
b
Eq cmp
Equal comp
gtT
gtA
gtC
gtG
max_cnt[4:0]
Output

## Problem #4

Arrays of LEDs are designed to be controlled by relatively few signals to minimize the wiring. One such control mechanism invented (perhaps not the very first) by Charlie Allen of Maxim Integrated Inc. is now known as "Charlieplexing". Each wire is driven by a tri-state logic gate whereby each wire can have a value of 0, 1, or z (high impedance). Below is an example of such an array of 4x4 LEDs.

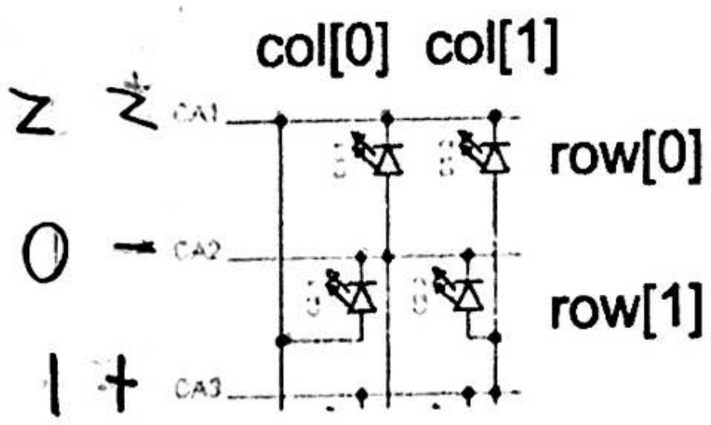



(a) If an array of *m x n* LEDs is provided where *m* is the columns and *n* is the rows. If *m<n*, what is the minimum number of wires needed to control this *m x n* array so that any single LED can be selectively turned on?

(b) A 2x2 array is shown below. Note that each wire CA1-3 that can be 1, 0, or z. What would the CA1-3 signals need to be to light up the LED at col[1] x row[1]?

(c) Each of the CAn wires are produced using a "tristate buffer" that accepts two signals, *en* (enable) and *ctrl* (control) as inputs. The signal, *en,* determines if the signal CAn is z or not, and the signal, *ctrl,* determines if the signal is 1 or 0 (when it is not z). Design a decoder (write the logic and draw any logic diagrams) that generates the *en[2:0]*, and *ctrl[2:0]* from from one-hot inputs *row_sel[1:0], col_sel[1:0]*, that indicates which LED in row/column format.




| c | d | f |
|---|---|---|
| 0 | 0 | z |
| 0 | 1 | z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(d) Let's extend part (b). For an 6x6 array with one-hot inputs *row_sel[5:0], col_sel[5:0]*, what would the CA1-7 signals need to be to light up the LED at col[4] x row[3]?

(e) You may use any building blocks or AND/OR/INV (2-input) gates as needed to implement the signals *en[6:0] and ctrl[6:0]* that are needed for the tri-state buffers driving *CA7-1*. Draw your design and describe how you completed the design. Note that there are many possible ways to do this design so your design process and thoughts should be conveyed.

*Answer the question for all parts in the space below.*

n = # of wires
r = 2



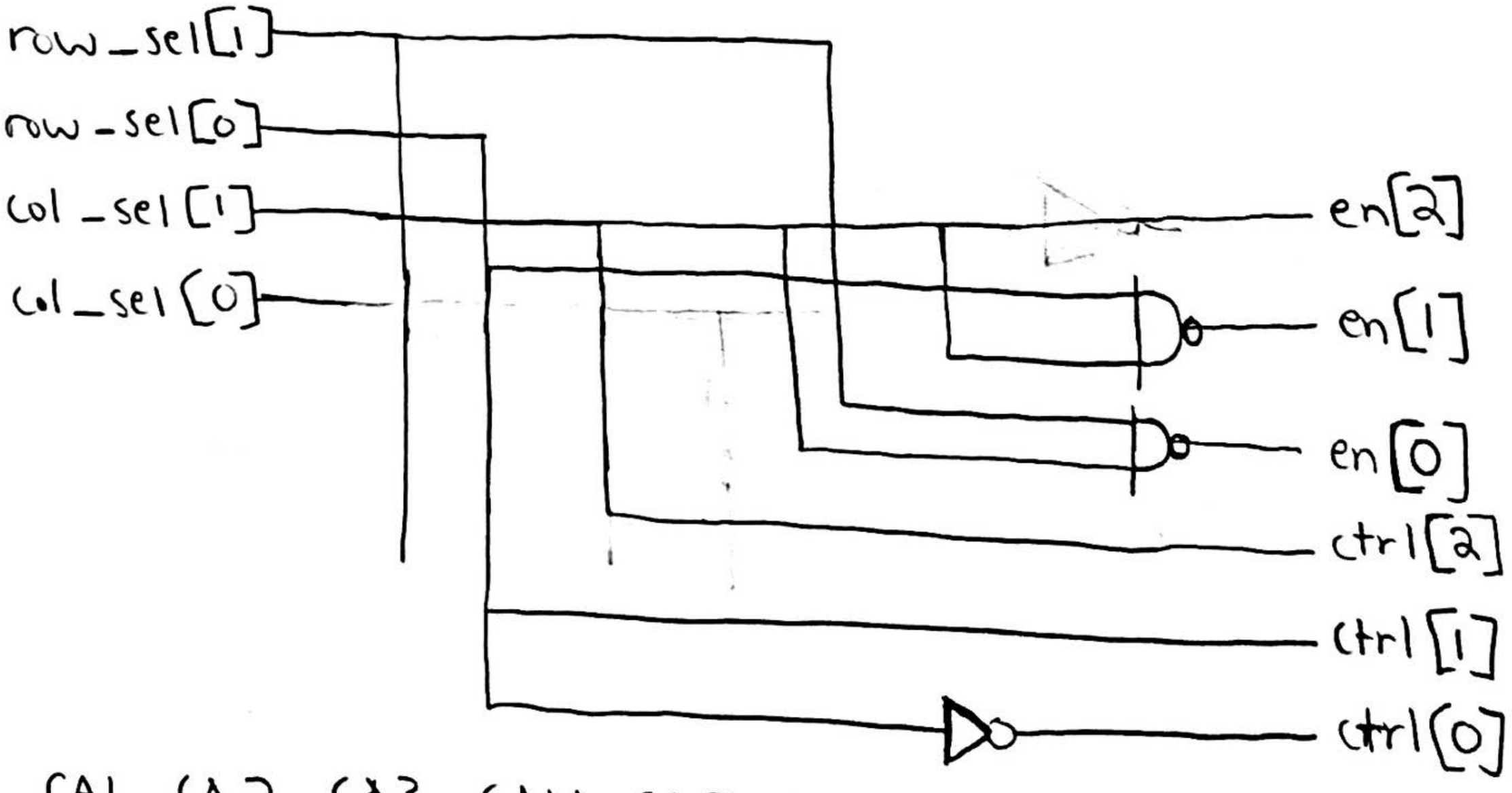
**(IMPORTANT: Keep this page in submission even if left unused)**

a) $P(n,r) = \frac{n!}{(n-r)!} = \frac{n!}{(n-2)!} = \frac{n(n-1)(n-2)!}{(n-2)!} = n^2 - n$

b) CA1: z CA2: 0 CA3: 1

c) row_sel [1:0] & col_sel[1:0]

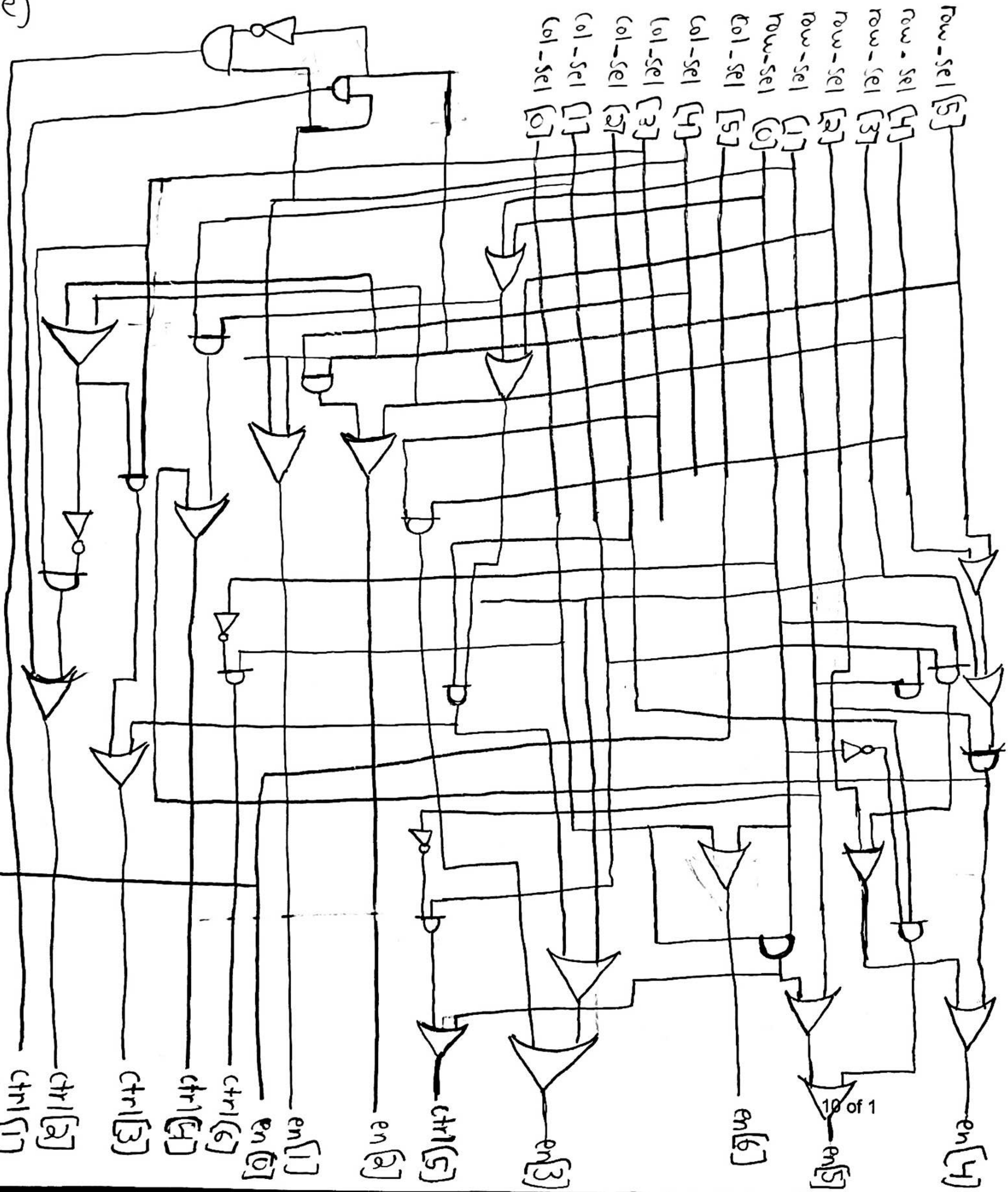
| row_sel[1:0] | col_sel[1:0] | en[2:0] | ctrl[2:0] |
|---|---|---|---|
| 01 | 01 | 011 | x10 |
| 01 | 10 | 101 | 1x0 |
| 10 | 01 | 011 | x01 |
| 10 | 10 | 110 | 10x |




d) CA1 CA2 CA3 CA4 CA5 CA6 CA7
z z 0 z 1 z z

**(IMPORTANT: Keep this page in submission even if left unused)**

e)



en[6] = row_sel[0] ∨ col_sel[0]

en[5] = row_sel[1] ∨ (row_sel[0] ∧ col_sel[0])
∨ (col_sel[2] ∧ ¬(row_sel[0]))

en[4] = row_sel[2] ∨ (row_sel[0] ∧ col_sel[1])
∨ (row_sel[1] ∧ col_sel[1]) ∨
(col_sel[2] ∧ (row_sel[3] ∨ row_sel[4] ∨ sel[5]))

en[3] = row_sel[3] ∨
(col_sel[3] ∧ (row_sel[0] ∨ r_sel[1] ∨ r_sel[2]))
∨ (col_sel[3] ∧ row_sel[4])

en[2] = row_sel[4] ∨ ¬(row_sel[4])
(col_sel[3] ∧ (row_sel[0] ∨ 1 ∨ 2 ∨ 3 ∨ 5)) ∨
(col_sel[4] ∧ row_sel[5])

en[1] = row_sel[5] ∨
(col_sel[4] ∧ (row_sel[0] ∨ 1 ∨ 2 ∨ 3 ∨ 4))

en[0] = col_sel[5]

ctrl[6] = col_sel[0] ∧ ! row_sel[0]

ctrl[5] = (col_sel[0] ∧ row_sel[0]) ∨
(col_sel[1] ∧ ¬(row_sel[1]))

ctrl[4] = (col_sel[1] ∧ (row_sel[0] ∨ row_sel[1]))
∨ (col_sel[2] ∧ (row_sel[5] ∨ 4 ∨ 3))

$$ctrl[3] = (col\_sel[2] \wedge (row\_sel(0) \vee 1 \vee 2))$$
$$\vee (col\_sel[3] \wedge (row\_sel(4) \vee row\_sel[5]))$$
$$ctrl[2] = (col\_sel[3] \wedge \neg(row\_sel[4] \vee row\_sel[5]))$$
$$\vee (col\_sel[4] \wedge row\_sel[5])$$
$$ctrl[1] = (col\_sel[4] \wedge \neg(row\_sel[5])$$
$$ctrl[0] = col\_sel[5]$$

| | row_sel[5:0] | col_sel[5:0] | en[6:0] | ctrl[6:0] |
|---|---|---|---|---|
| | 000001 | 000001 | 1100000 | 01XXXXX |
| | 000010 | 000001 | 1100000 | 10XXXXX |
| | 000100 | 000001 | 1010000 | 1XOXXXX |
| | 001000 | 000001 | 1001000 | 1XXOXXX |
| | 010000 | 000001 | 1000100 | 1XXXOXX |
| | 100000 | 000001 | 1000010 | 1XXXXOX |
| | 000001 | 000010 | 1010000 | 0XIXXXX |
| | 000010 | 000010 | 0110000 | XOIXXXX |
| | 000100 | 000010 | 0110000 | X1OXXXX |
| | 001000 | 000010 | 0101000 | X1XOXXX |
| | 010000 | 000010 | 0100100 | X1XXOXX |
| | 100000 | 000010 | 0100010 | X1XXXOX |
| | 000001 | 000100 | 1001000 | 0XX1XXX |
| | 000010 | 000100 | 0101000 | XOX1XXX |
| | 000100 | 000100 | 0011000 | XX1OXXX |
| | 001000 | 000100 | 0011000 | XX1XOXX |
| | 010000 | 000100 | 0010100 | XX1XXOX |
| | 100000 | 000100 | 0010010 | XX1XXXO |
| | 000001 | 001000 | 1000100 | 0XXX1XX |
| | 000010 | 001000 | 0100100 | XOXX1XX |
| | 000100 | 001000 | 0010100 | XXOX1XX |
| | 001000 | 001000 | 0001100 | XXXO1XX |
| | 010000 | 001000 | 0000110 | XXXX1OX |
| | 100000 | 001000 | 0000110 | XXXX1XO |
| | 000001 | 010000 | 1000010 | 0XXX1XX |
| | 000010 | 010000 | 0100010 | XOXX1XX |
| | 000100 | 010000 | 0010010 | XXOX1XX |
| | 001000 | 010000 | 0001010 | XXXO1XX |
| | 010000 | 010000 | 0000110 | XXXX1OX |
| | 100000 | 010000 | 0000110 | XXXX1XO |
| | 000001 | 100000 | 1000010 | 0XXX1XX |
| | 000010 | 100000 | 0100010 | XOXX1XX |
| | 000100 | 100000 | 0010010 | XXOX1XX |
| | 001000 | 100000 | 0001010 | XXXO1XX |
| | 010000 | 100000 | 0000110 | XXXX1OX |
| | 100000 | 100000 | 0000110 | XXXX1XO |

## Problem #5

The world ended about six and a half years ago… according to the Mayans. Mayans used a base-20 ("vigesimal") counting system. Accordingly, their calendar reflects this (mostly). A calendar date is written in the format *v.w.x.m.d* where each number is a single vigesimal digit, a "*vigit*".

(a) Like calendars in all cultures, calendars are notoriously irregular so *v* only counts up to 12, and *m* (month) only counts up to 17 (and increments *x* on 18). What is the maximum number of days represented in this Mayan calendar?

(b) Roughly how many years does the answer in (a) correspond to. Note that the Mayans believed that the world began on August 11, 3114 BCE. So, you can estimate which year the world should have ended.

(c) Instead of a calendar date, assume that the format simply corresponds to a 5-vigit number. This number does not have the counting constraints in (a), and the Mayans figured out complement counting system. How would you represent the most positive and most negative 20's-complement number? And what are these numbers in decimal?

(d) What is the 20's complement Mayan number, 10.11.12.13.14, in decimal?

*Answer the question for all parts in the space below.*

a) 12 19 17 19
V W x m d

12.19.19.17.19

$= 19 + 17\cdot 20 + 19\cdot 18\cdot 20 + 19\cdot 18\cdot 20^2 + 12\cdot 18\cdot 20^3$

$= 1{,}871{,}999$ days

b) August 11, 3114 BCE
+ 5129
= 2015

$\frac{1871999}{365} \approx 5129$ years

c) 0.19.19.19.19 pos. max $= 159{,}999_{10}$

19.0.0.0.0 neg mh $= -3{,}040{,}000_{10}$

d) $10.11.12.13.14 = -1{,}506{,}926_{10}$

**(IMPORTANT: Keep this page in submission even if left unused)**

## Problem #6

Let's take a 20-bit word, *W[19:0]*.

(a) If *W* is 2's complement, what's the most positive and most negative possible values in decimal?

(b) How do you represent 201903 and -1550 in 2's complement (binary representation)?

(c) If *W* is unsigned, what's the largest magnitude value in decimal?

(d) If *W* is hex, repeat (b).

(e) If *W* is 1's complement, repeat (b).

(f) If *W* is signed fixed point of 12.08 (2's complement), what is the largest and smallest magnitude negative number that is achievable? Show both in decimal integer value and the binary representation. What is the absolute accuracy of this representation?

(g) If *W* is an extension of the IEEE floating point format (incidentally, it doesn't actually exist for a 20-bit word) where the mantissa (or fraction) is 12 bits and the exponent is 7 bits, what is the bias? And what is the relative accuracy of this representation?

(h) Extending on (g), what is the largest and smallest magnitude (positive sign) achievable? Show both the decimal integer value and the binary representation. Recall that the exponent cannot be 7'b000_0000 or 7'b111_1111.

pg 2/4

(i) How do you represent the decimal, -2019.2019, in binary (closest value) if it is fixed point (f)? What about floating (g)?

(j) What is 0111_0111_1100_1100_1011 (binary number) in decimal if the number is fixed point (f)? What about floating (g).

***Answer the question for all parts in the space below.***

a) 10000000000000000000

01111111111111111111

$T_{min} = -524,288_{10}$

$T_{max} = 524287_{10}$

b) 00110001010010101111 = $201903_{10}$

00000000001100001110 = $1550_{10}$

11111110011110001

11111111110011111110010 = $-1550_{10}$

c) $U_{max}$ = 11111111111111111111 = $2^{20}-1 = 1,048,575_{10}$

d) 201903

-196608

5295

-4096

1199

-1024

175 | =15

-1550

| $\frac{3}{16^4}$ | $\frac{1}{16^3}$ | $\frac{4}{16^2}$ | $\frac{A}{16^1}$ | $\frac{F}{16^0}$ |
|---|---|---|---|---|

= 314AF$_{16}$

Part d on next page

**(IMPORTANT: Keep this page in submission even if left unused)**

d) $(201903)_{16} = (210355)_{10}$

$= 0010\ 0000\ 0001\ 1001\ 0000\ 0011_2$

$(-1550)_{16} = -(5456)_{10} = -(1010101010000)_2$

$(-1550)_{16} = (1110101011000000)_2$

e) $(201903)_{10} \Rightarrow (00110001010010101111)_2$

$(-1550)_{10} : 1550_{10} = 00000001100\ 0001110)_2$

$= (1111111110011\ 1110001)_2$

f) XXXX XXXX XXXX. XXXX XXXX

largest pos val $0111\ 1111\ 1111.1111\ 1111_2$

$= 2047.99609375_{10}$

Smallest neg val $1000\ 0000\ 0000.1111\ 1111_2$

$= -2048.99609375_{10}$

$.5 + .25 + .125 + .0625 + .03125 + .015625$ (1, 2, 3, 4, 5, 6)

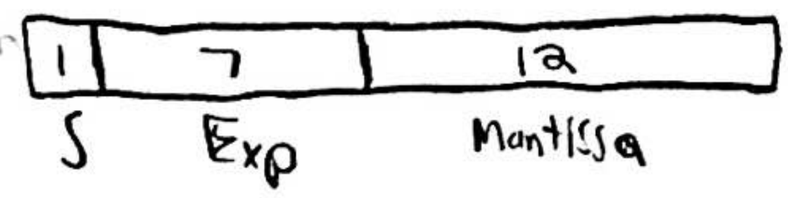
$.0078125 + .00390625 = .99609375$ (7, 8)

largest neg val $1111\ 1111\ 1111.1111\ 1111_2$

$= -1.99609375_{10}$

absolute accuracy $= \frac{.00390625}{2} = 0.001953125$

g)

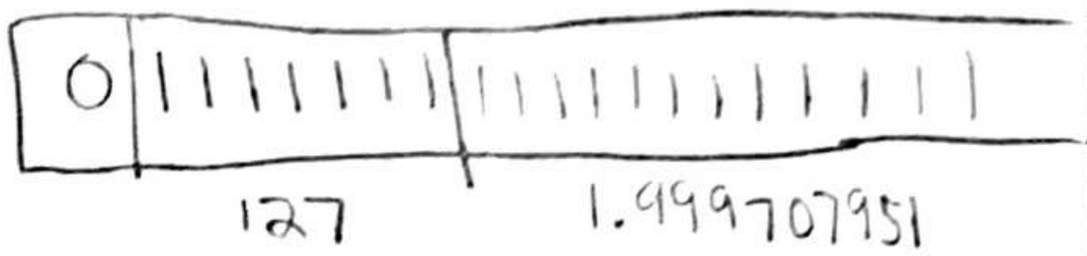



$bias = 2^{k-1} - 1 = 2^{7-1} - 1 = 2^6 - 1 = 63$

relative error: $e = \frac{|true\ val - approx\ val|}{|true\ val|}$ $\frac{.05 - .025}{.05} = .05$

$= 5\%$

h)


BIAS = 63

$(-1)^S M 2^E$ $E = Exp - 63$

$127 - 63 = 64$

$(-1)^0 (1.9997079S1)\, 2^{64}$

$= 3.68881007 9 \times 10^{19}$

: largest

Smallest: 1.000196232

i) -2019.2019

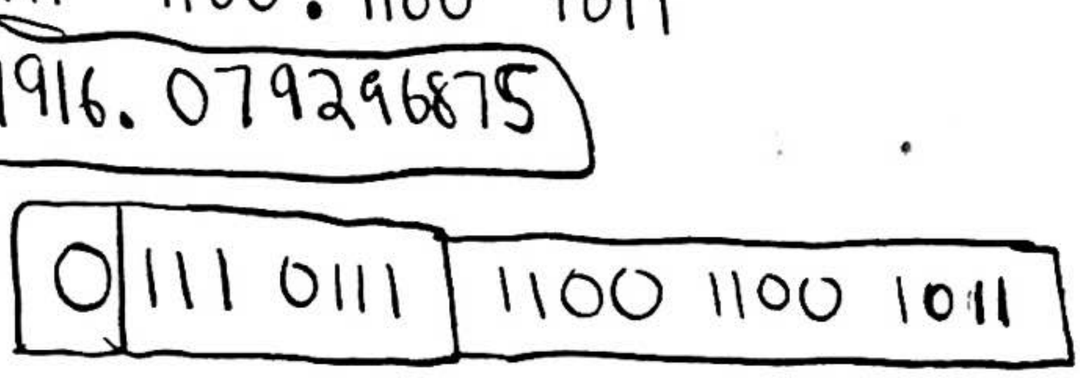
fixed) 1000 0001 1101.00110100

floating) 0 00001010 11111000 1001

j) 0111 0111 1100.1100 1011

fixed) 1916.079296875

floating)




119

$E = 119 - 63$

$= 56$

$(-1)^S = (-1)^0$

$(-1)^S M 2^E$

$M = 1.799580547$

$(1) M (2^{56})$

$= 1.296720033 \times 10^{17}$