CS 5

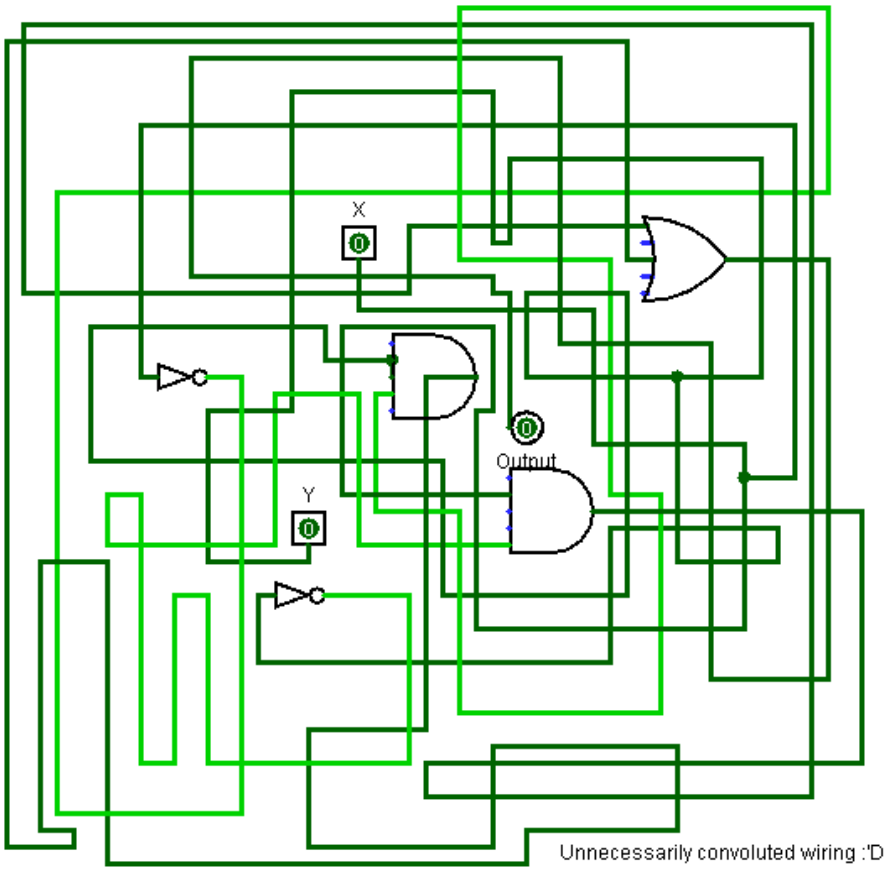
Python

Hmmm...

OUR CS PATH

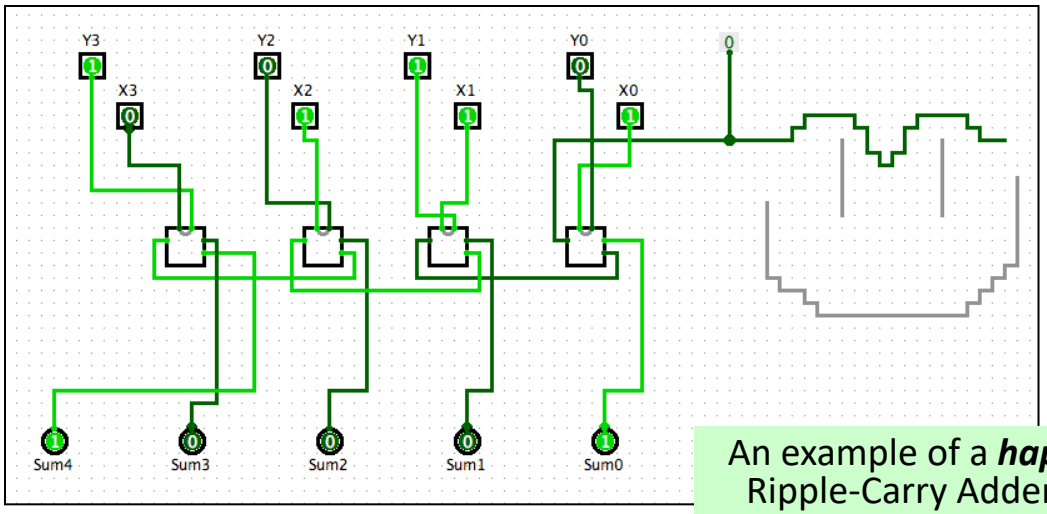
- computer
- main memory
- registers
- 1-bit memory: *flip-flops*
- arithmetic
- bitwise functions
- logic gates
- switches: ***transistors***

ABSTRACTION



Unnecessarily convoluted wiring :'D

taking the circuit *"canvas"* literally ...



An example of a ***happy*** Ripple-Carry Adder...

Things are awfully messy 'round here...

hw5 ~ *composing* circuits

3-bit **Full Adder**

4-bit *Ripple-Carry* **Adder**

Prime tester

using "minterm expansion"

4-bit **Multiplier**

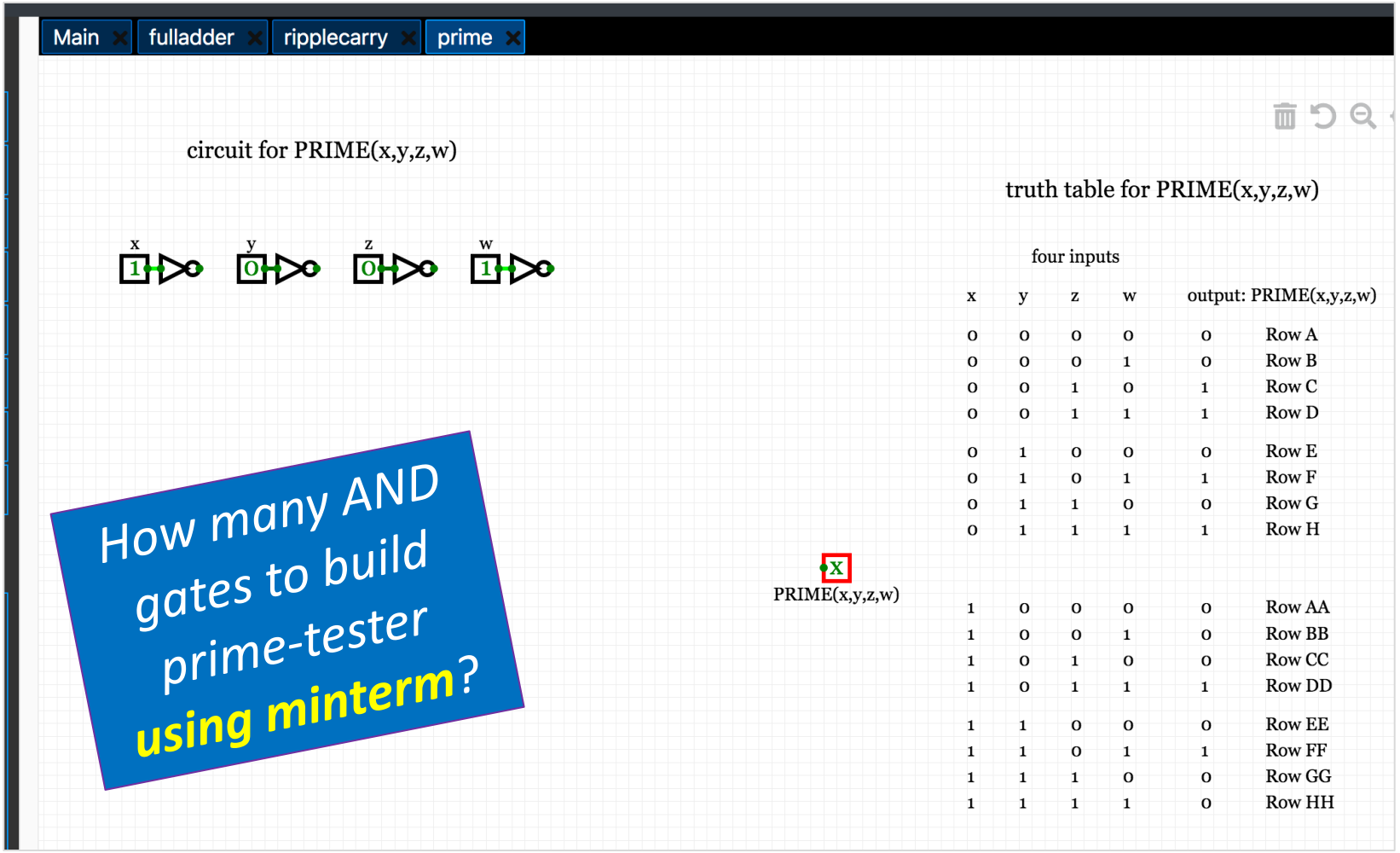
3x2-bit **Divider**

extra credit

12 nGbits of memory (RAM)

Optimized Prime

Prime-tester...



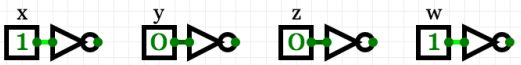
pure "minterm expansion": *one AND for each '1' output*

Prime-*optimizer*?!

This one is problem 5, EC

Main | fulladder | ripplecarry | prime

circuit for PRIME(x,y,z,w)



How ***few*** gates can correctly implement prime-tester?

this is an open problem in cs: "circuit complexity"

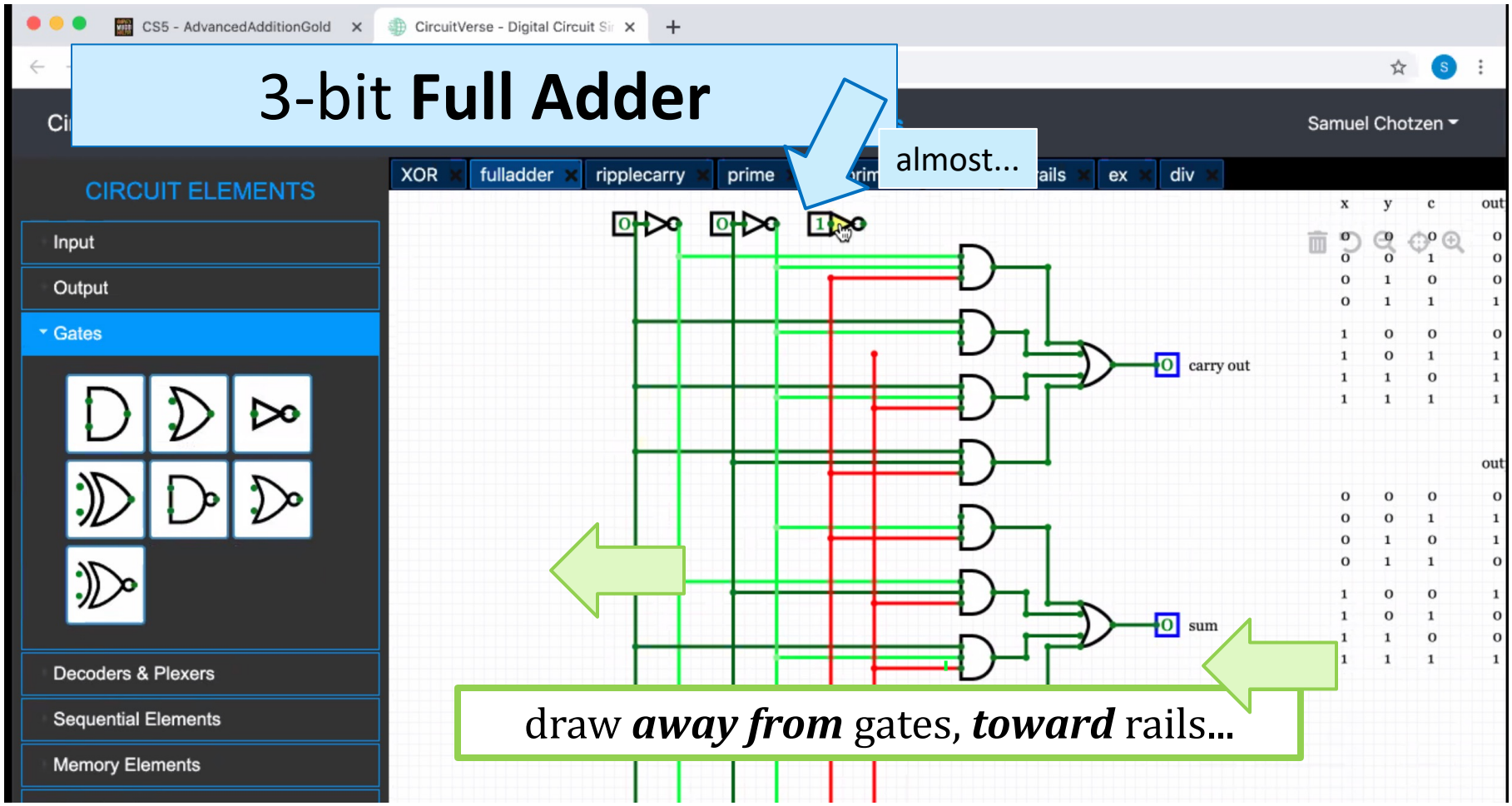
truth table for PRIME(x,y,z,w)

four inputs

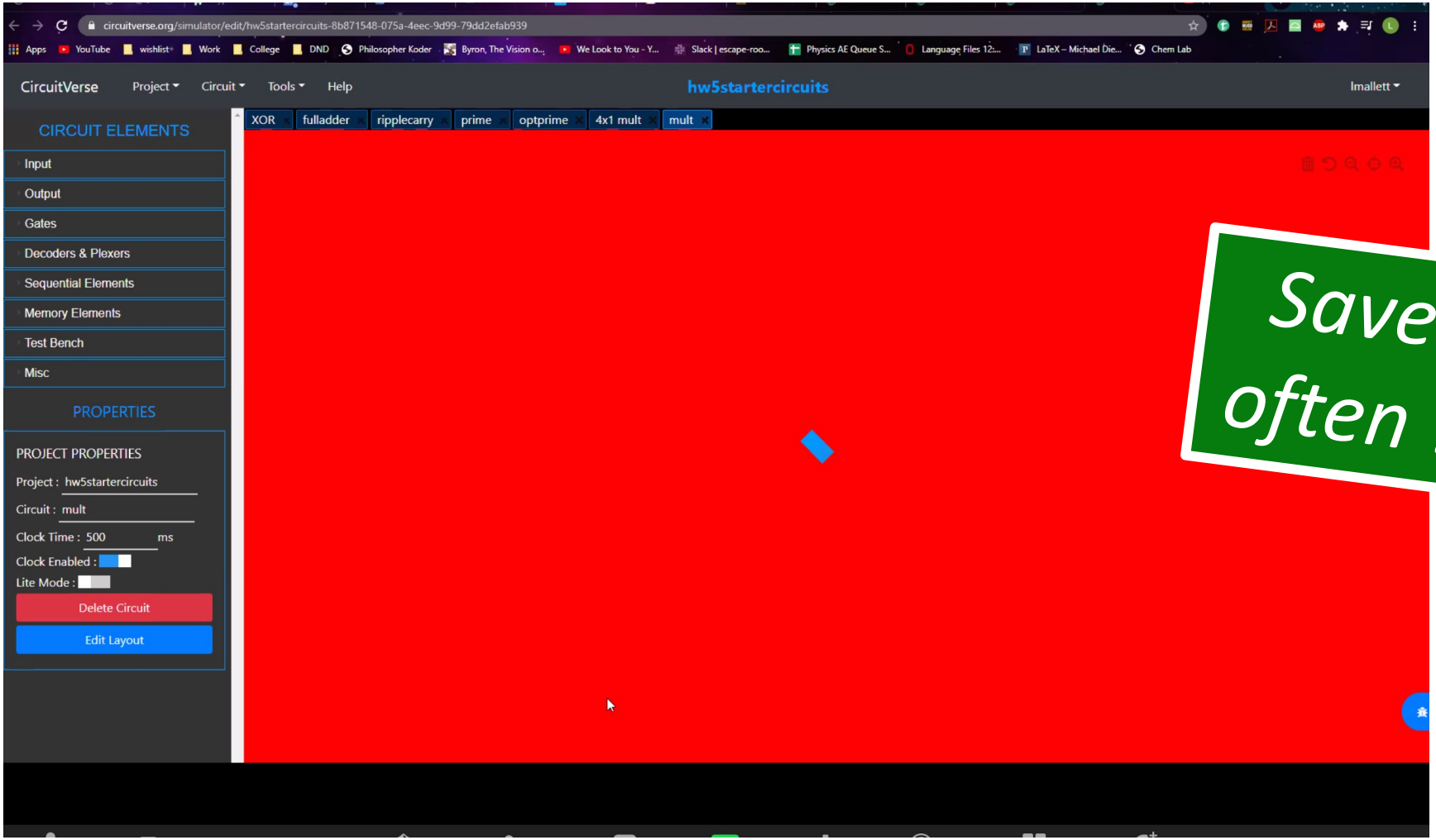
x	y	z	w	output: PRIME(x,y,z,w)	
0	0	0	0	0	Row A
0	0	0	1	0	Row B
0	0	1	0	1	Row C
0	0	1	1	1	Row D
0	1	0	0	0	Row E
0	1	0	1	1	Row F
0	1	1	0	0	Row G
0	1	1	1	1	Row H
1	0	0	0	0	Row AA
1	0	0	1	0	Row BB
1	0	1	0	0	Row CC
1	0	1	1	1	Row DD
1	1	0	0	0	Row EE
1	1	0	1	1	Row FF
1	1	1	0	0	Row GG
1	1	1	1	0	Row HH

pure "minterm expansion": *one AND for each '1' output*

Wiring tips…


CS5 - AdvancedAdditionGold
CircuitVerse - Digital Circuit Si
3-bit Full Adder
almost…
Samuel Chotzen
CIRCUIT ELEMENTS
Input
Output
Gates
Decoders & Plexers
Sequential Elements
Memory Elements
XOR
fulladder
ripplecarry
prime
ex
div
carry out
sum
x y c out
draw *away from* gates, *toward* rails…

Worst-case, ***so far***...

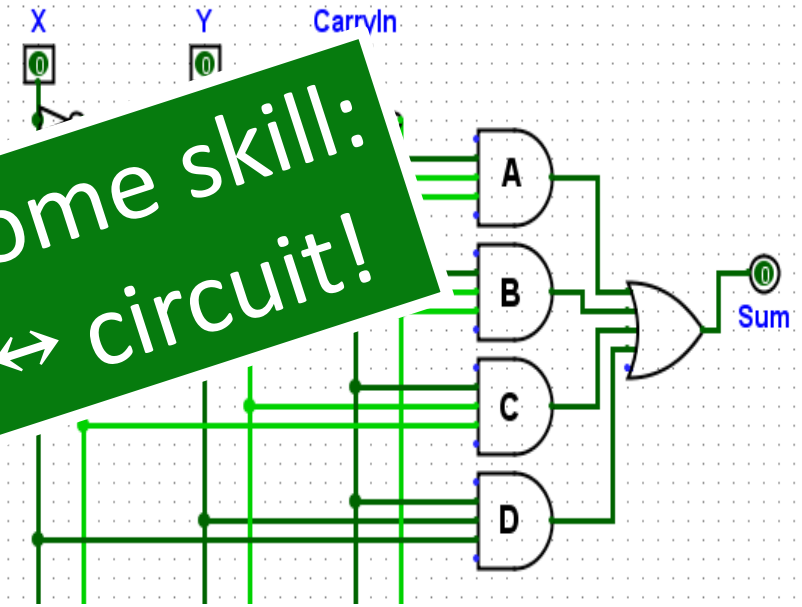


truth tables ↔ circuits

IN			OUT	
x	y	c		circuit output
0	0	0		0
0	0	1		0
0	1	0	B	1
0	1	1		
1	0	0		
1	0	1		
1	1	0		1
1	1	1	D	1

(the truth table)

hw5's take-home skill:
truth-table ↔ circuit!

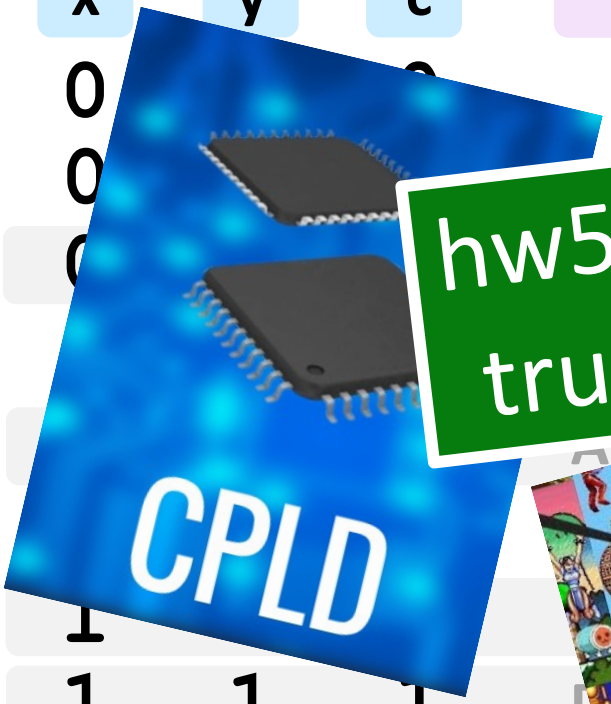


(the circuit)

truth tables ↔ circuits

IN			OUT
x	y	c	circuit output
0	0	0	0
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(the truth tab

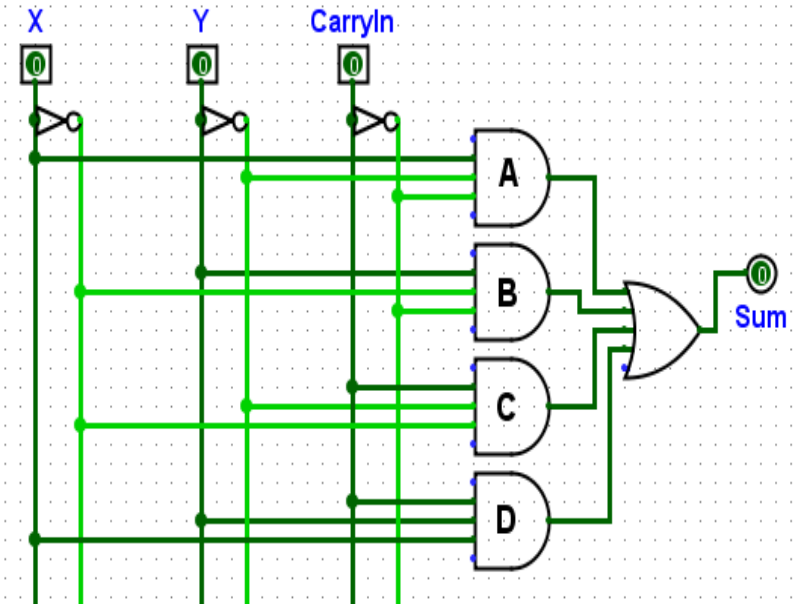


hw5's take-home skill:
truth-table ↔ circuit!

truth tables <-> circuits

IN			OUT	
x	y	c		circuit output
0	0	0		0
0	0	1		0
0	1	0	B	1
0	1	1		0
1	0	0	A	1
1	0	1		0
1	1	0	C	1
1	1	1	D	1

(the truth table)



(the circuit)

truth tables <-> circuits

Name(s) _____

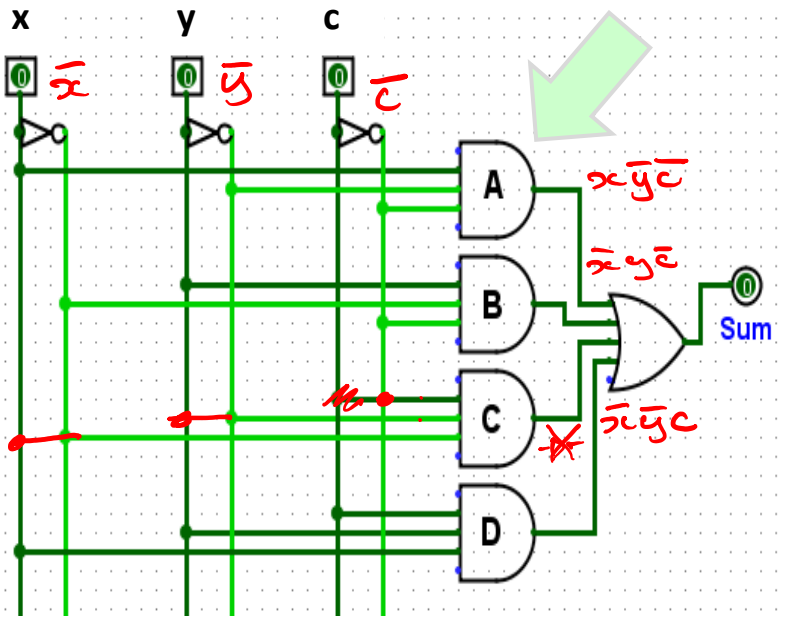
Challenge 1: There is a **mismatch** between this function (truth table) and circuit.

Challenge 2: Fix it on **BOTH** sides: Draw how to make the circuit match the table + how to make the table match the circuit

IN				OUT
x	y	c		circuit output
0	0	0		0
0	0	1		0
0	1	0	B	1
0	1	1		0
1	0	0	A	1
1	0	1		0
1	1	0	C	1
1	1	1	D	1

(the truth table)

AND gate for each **1** in the truth table's output



(the circuit)

Extra Challenge: This "minterm" approach can implement *any* function. *But functions miss most of what computers do!* ***What's missing?***

truth tables <-> circuits

Try this on the other page first...

Row **6** (110) has no AND gate!

Row **1** (001) does have an AND gate!

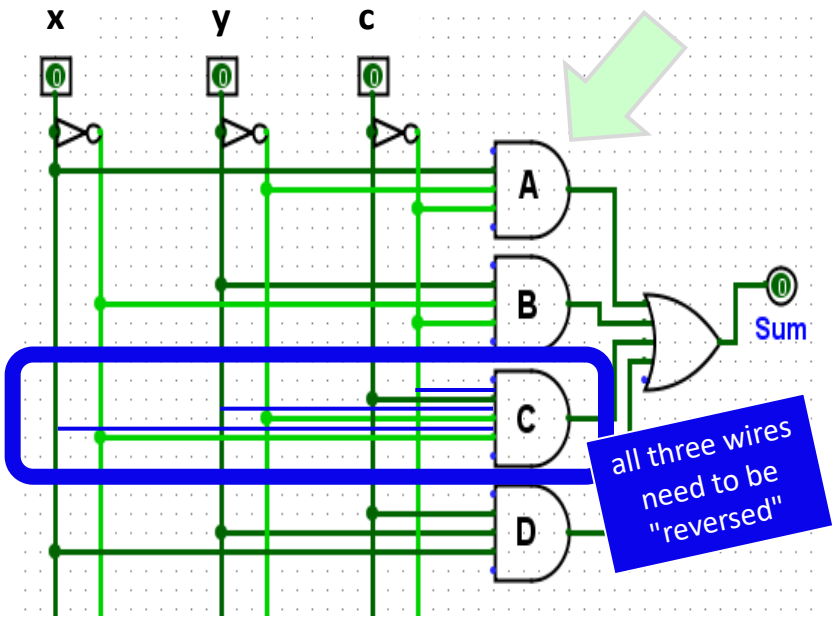
Challenge 1: There is a **mismatch** between this function (truth table) and circuit.

Challenge 2: Fix it on **BOTH** sides: Draw how to make the circuit match the table + how to make the table match the circuit

IN				OUT
x	y	c		circuit output
0	0	0		0
0	0	1	C	1
0	1	0	B	1
0	1	1		0
1	0	0	A	1
1	0	1		0
1	1	0	~~C~~	1 ~~0~~
1	1	1	D	1

(the truth table)

AND gate for each **1** in the truth table's output



(the circuit)

MEMORY!

Extra Challenge: This "minterm" approach can implement *any* function. *But functions miss most of what computers do!* ***What's missing?***

truth tables <-> circuits

Name(s) _____

Challenge 1: There is a **mismatch** between this function (truth

Challenge 2: Fix it on **BOTH** sides

tch the circuit

Pass those outdoorwards!

AND gate for
each **1** in the
ble's
ut

x
0
0
0
0
1
1
1
1

Sum

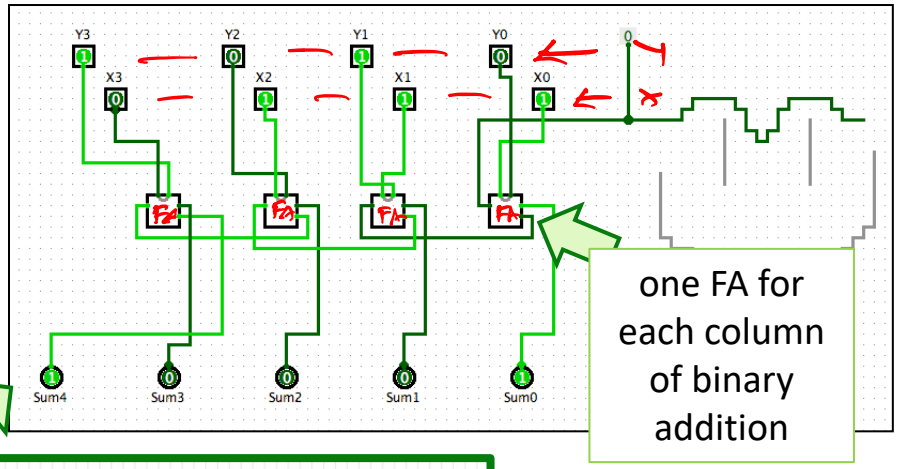
"It's not real unless it can be done in Minecraft."

Extra Challenge:

s missing?

Composing circuits

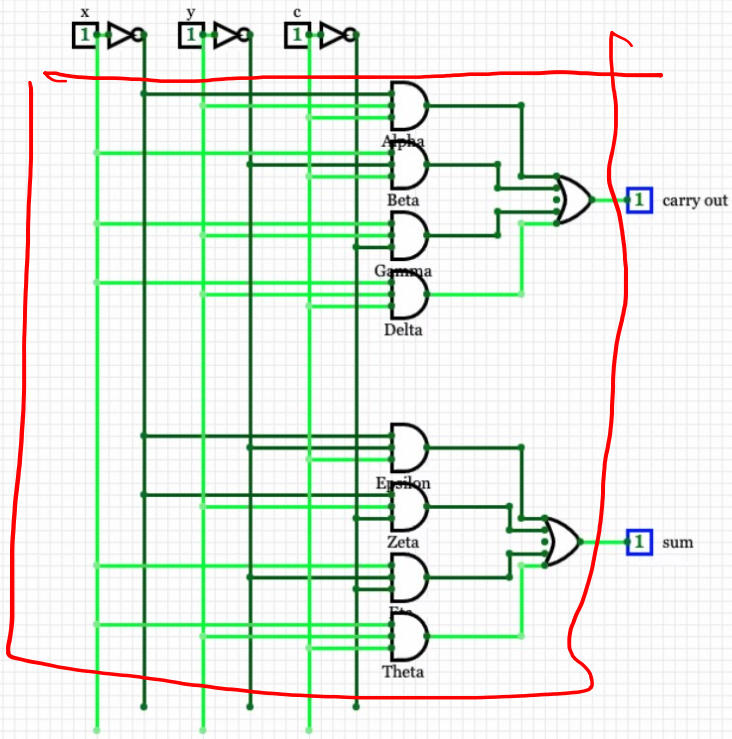
Full Adder
~ minterm



one FA for each column of binary addition

this FA gets "boxed up"

This will be a fulladder circuit (FA), implemented from fundan l logic gates.



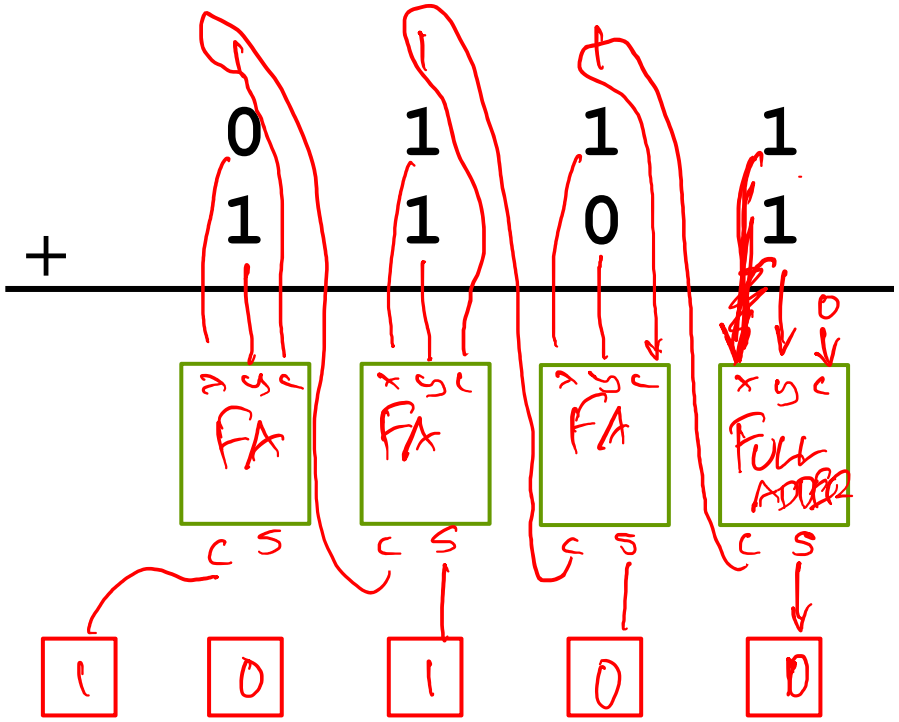
inputs

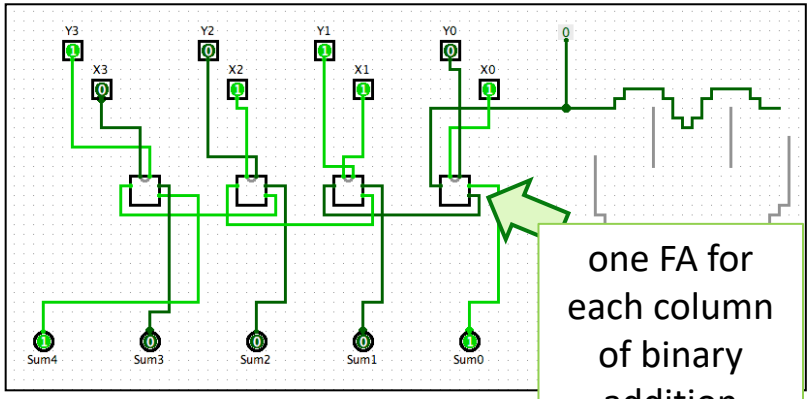
x	y	c	output: CARRY(x,y,c)		
0	0	0	0	Row A	
0	0	1	0	Row B	
0	1	0	0	Row C	
0	1	1	1	Row D	Alpha
1	0	0	0	Row E	
1	0	1	1	Row F	Beta
1	1	0	1	Row G	Gamma
1	1	1	1	Row H	Delta

			output: SUM(x,y,c)		
0	0	0	0	Row A	
0	0	1	1	Row B	Epsilon
0	1	0	1	Row C	Zeta
0	1	1	0	Row D	
1	0	0	1	Row E	Eta
1	0	1	0	Row F	
1	1	0	0	Row G	
1	1	1	1	Row H	Theta

Composing circuits

4-bit Ripple-Carry Adder





one FA for each column of binary addition

8 bits in

5 bits out

5 full adders

one FA for each column of binary addition

5 "sum" bits

4-bit Ripple-Carry Adder

keep abstracting!

$$
\begin{array}{rcccc}
 & Y_3 & Y_2 & Y_1 & Y_0 \\
+ & X_3 & X_2 & X_1 & X_0 \\
\hline
Z_4 & Z_3 & Z_2 & Z_1 & Z_0
\end{array}
$$

⇦ 8 bits in

⇨ 5 bits out

4-bit Ripple-Carry Adder

keep abstracting!

$$\begin{array}{cccccc} & Y_3 & Y_2 & Y_1 & Y_0 \\ + & X_3 & X_2 & X_1 & X_0 \\ \hline Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \end{array}$$

⇐ 8 bits in

⇨ 5 bits out

$$\begin{array}{cccccc} & Y_3 & Y_2 & Y_1 & Y_0 \\ + & X_3 & X_2 & X_1 & X_0 \\ \hline Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \end{array}$$

⇐ 8 bits in

⇨ 5 bits out

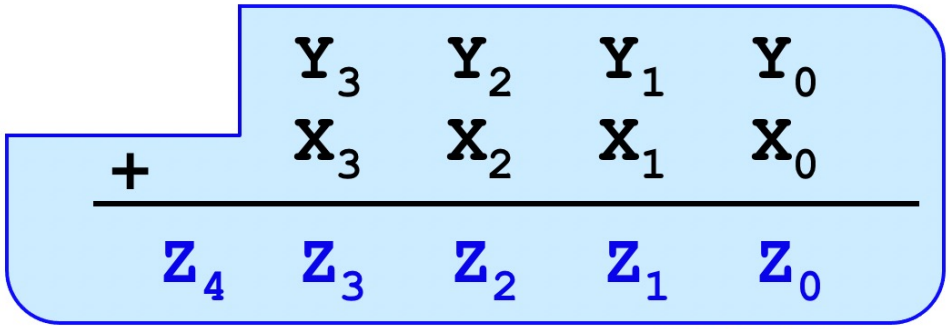
our ripple-carry schoolbus

a ripple-carry "bus"!

Composing circuits

keep abstracting!

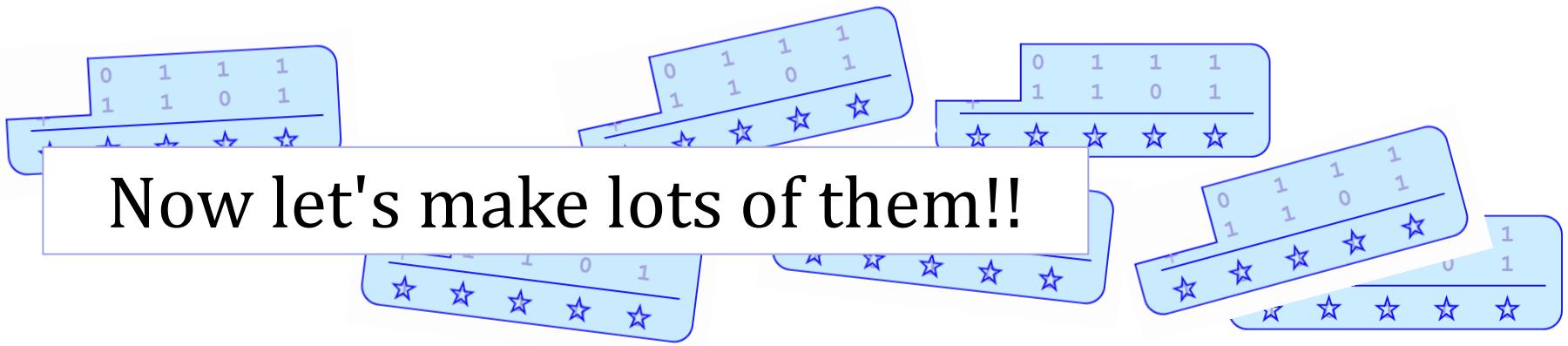
4-bit Ripple-Carry Adder



8 bits in

5 bits out

our ripple-carry schoolbus



Now let's make lots of them!!

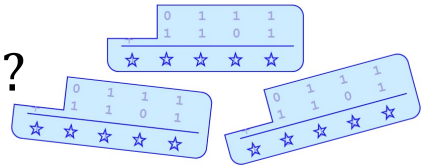
hw5pr3: A 4-bit *multiplier*

				1	1	0	1	Multiplicand	Y_3 Y_2 Y_1 Y_0
			x	0	1	1	0	Multiplier	X_3 X_2 X_1 X_0
			☆	0	0	0	0	4 partial products	
		+		1	1	0	1		
		+	1	1	0	1			
	+	0	0	0	0				
0	1	0	0	1	1	1	0	Final answer…	
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0		

Be sure you "see" the 16 bits you'll need to create!

Each bit will have a wire!

(Q3) How could **THREE** 4-bit ripple-carry adders help here?

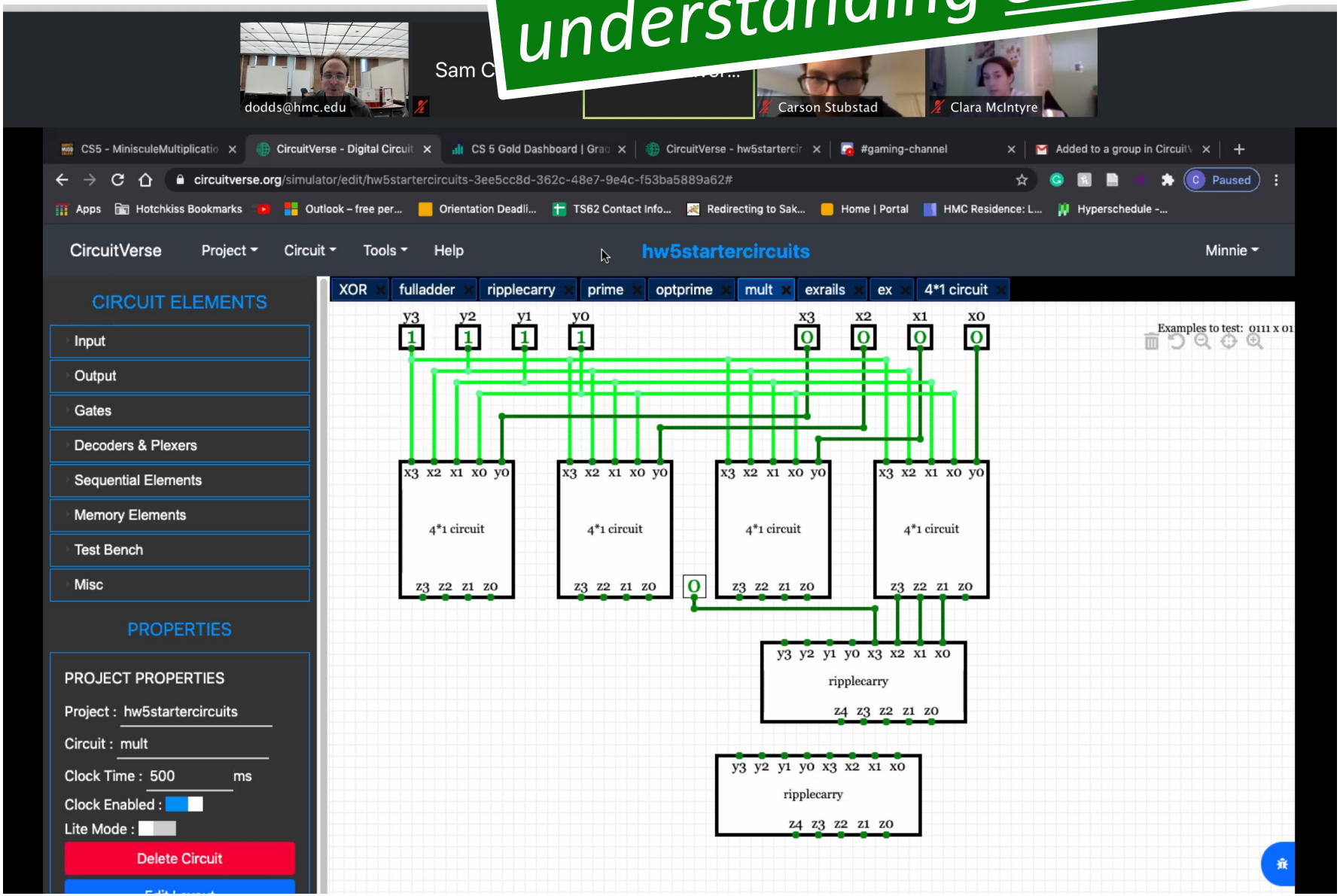


(Q2) What bit would be correct for the starred spot ☆ ?

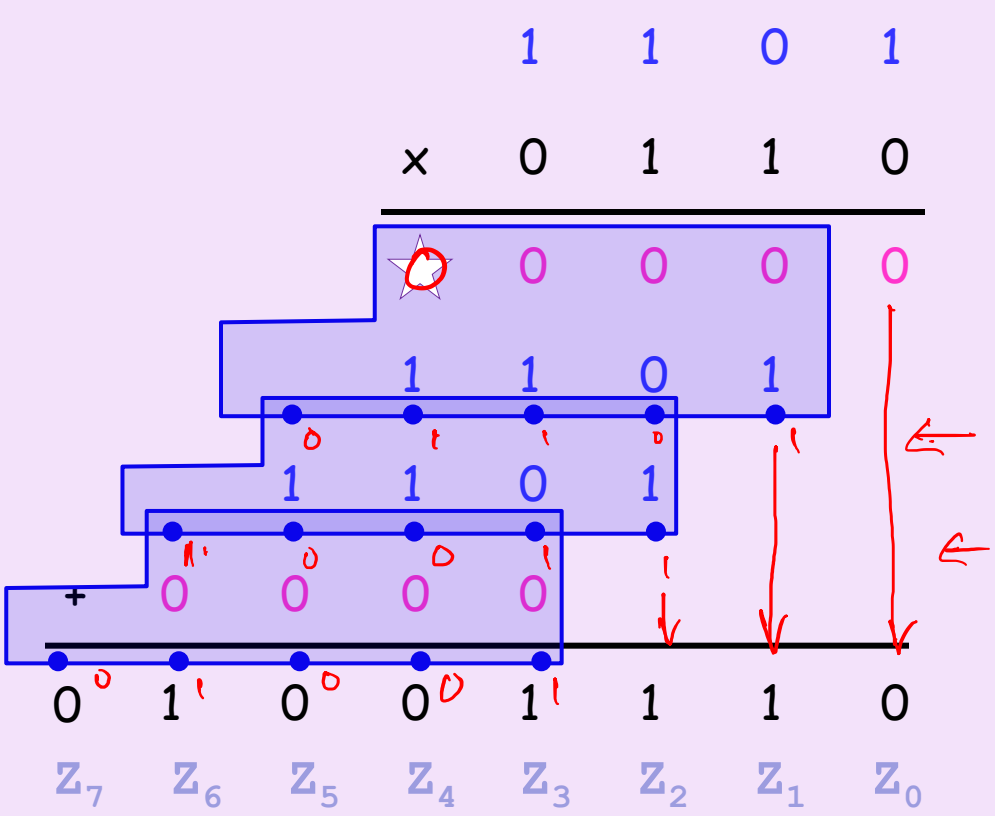
(Q1) What circuit could you use to create the four "partial products" ??

The Challenge...

understanding each bit!



hw5pr3: A 4-bit *multiplier*



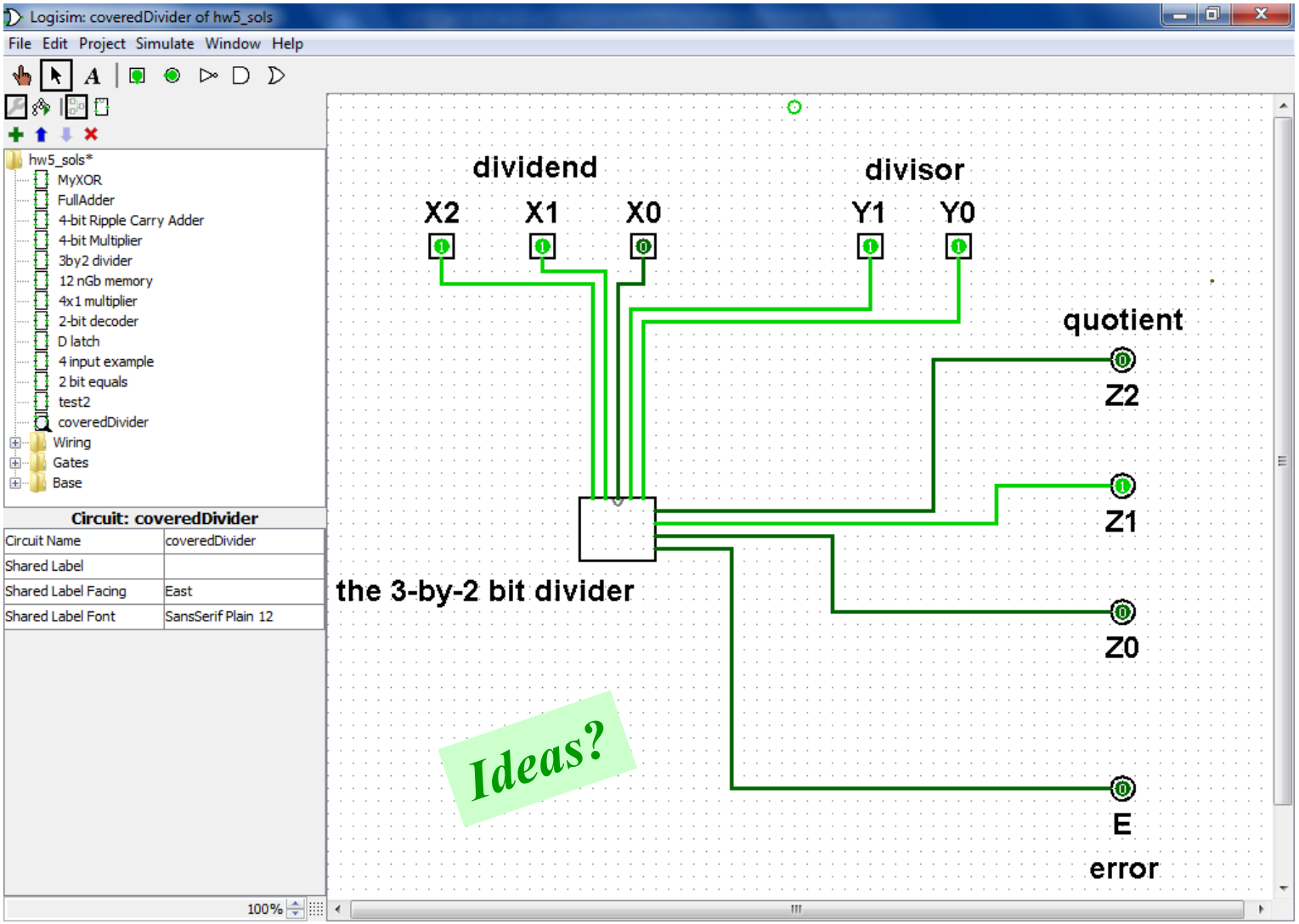
(A1) The AND gate is ***single-bit*** multiplication.

(A2) ☆ == 0

(A1) Use a 4x1-bit helper circuit to find the four *partial products*…

(A3) You need three (3) ripple-carry adders to finish: *see above*…

Division? hw5pr4



"principled" design

Minterm *Division*

(0) All computation can be expressed as bits...

(1) ***Any*** function of bits can be made a truth table

(2) Consider the output, *one bit at a time...*

(3) *The circuit will output 0 by default!*

(4) Are there *subcircuit patterns* to notice?

(5) *If not*, use an AND gate to **<u>select</u>** each input for which the output should be 1 (a minterm!)

To implement the red 1, how many inputs will its AND gate need??

How many **NOT**'ed?

What division *<u>is</u>* that line?

(6) OR the outputs from step (5) together.

(7) optimize your circuit later -- *or never*

INPUTS					OUTPUT			
Y2	Y1	Y0 (dividend)	X1	X0 (divisor)	Z2	Z1	Z0 (quotient)	E (error bit)
div. by 0: anything			0	0	anything			1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	1	0
0	1	0	0	1	0	1	0	0
0	1	1	0	1	0	1	1	0
1	0	0	0	1	1	0	0	0
1	0	1	0	1	1	0	1	0
1	1	0	0	1	1	1	0	0
1	1	1	0	1	1	1	1	0
div. by 1								
0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	1	0	0	0	1	0
0	1	1	1	0	0	0	1	0
1	0	0	1	0	0	1	0	0
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	1	1	0
1	1	1	1	0	0	1	1	0
div. by 2								
0	0	0	1	1	0	0	0	0
0	0	1	1	1	0	0	0	0
0	1	0	1	1	0	0	0	0
0	1	1	1	1	0	0	1	0
1	0	0	1	1	0	0	1	0
1	0	1	1	1	0	0	1	0
1	1	0	1	1	0	1	0	0
1	1	1	1	1	0	1	0	0

Circuit *Optimization*?

Welcome to Electrical and Computer Engineering!

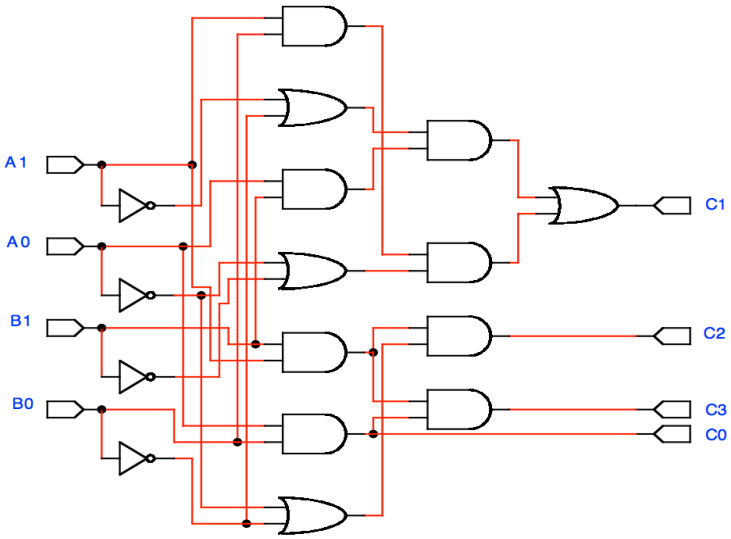


Figure 5: A 2-bit multiplier generated by a human designer

16 gates

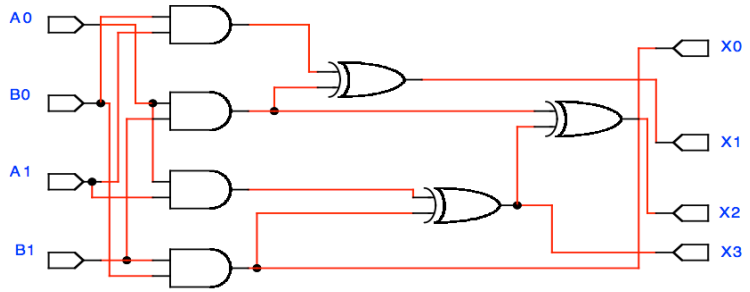


Figure 7: A 2-bit multiplier generated by a GA using our approach

7 gates

for *exploring* genetic algorithms...

Perhaps ***artistically*** optimized!

Optimizing *for what*?!

Time-optimized circuits: ***Carry lookahead adders***

The following circuit is called a **carry lookahead adder.**

By adding more hardware, we reduce the number of levels in the circuit and speed things up.

We can "cascade" carry lookahead adders, just like ripple carry adders.
We'd have to do carry lookahead *between* the adders too.

How much faster is this?

For a 4-bit adder, not much. There are 4 gates in the **longest path** of a carry lookahead adder, versus 9 gates for a ripple carry adder.

But if we do the cascading properly, a 16-bit carry lookahead adder could have only 8 gates in the longest path, as opposed to 33 for a ripple carry adder.

Newer CPUs these days use 64-bit adders. **That's 12 vs. 129 gates or 10x speedup!**

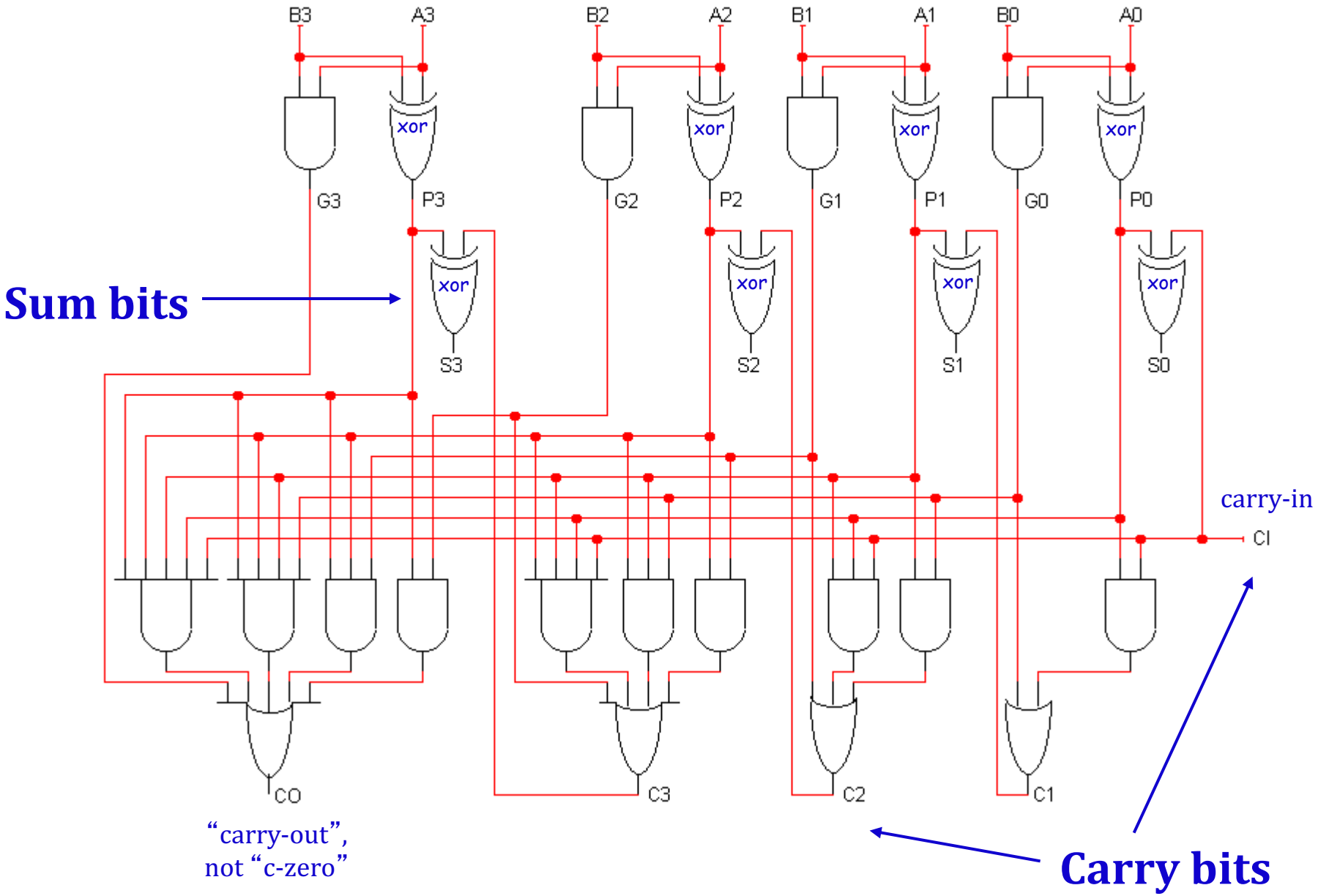
The delay of a carry lookahead adder grows *logarithmically* with the size of the adder, while a ripple carry adder's delay grows *linearly*.

The thing to remember about this is the **trade-off between complexity and performance**.

Ripple carry adders are simpler, but slower. Carry lookahead adders are faster but more complex.

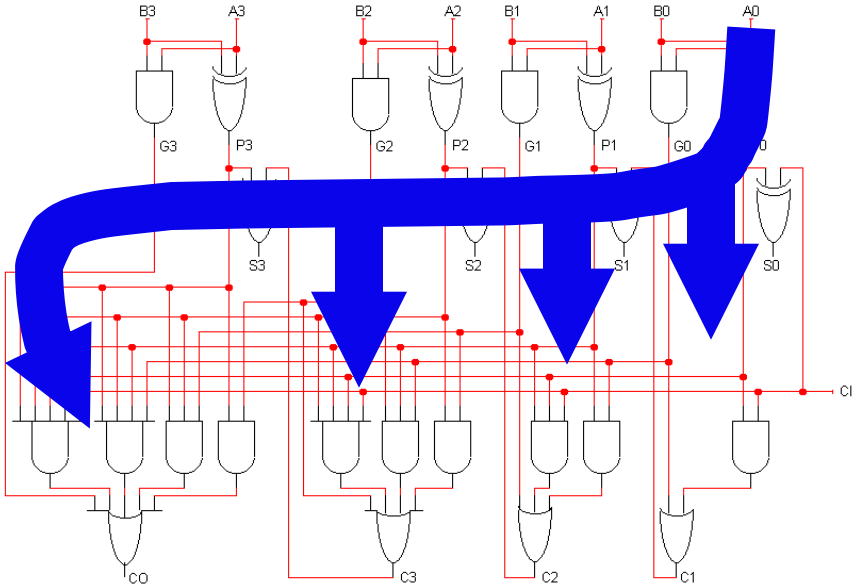
Details!

A 4-bit carry-lookahead adder circuit

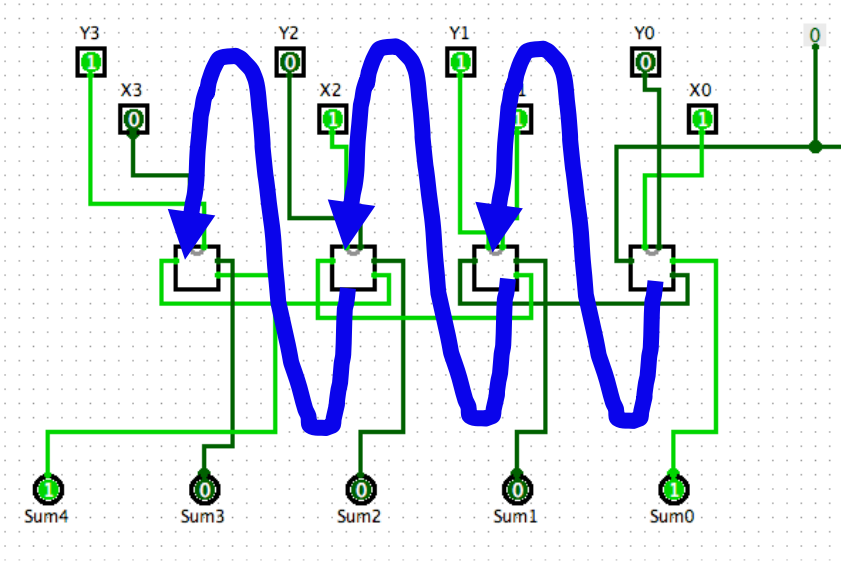


Flows!

A 4-bit **carry-lookahead** adder circuit

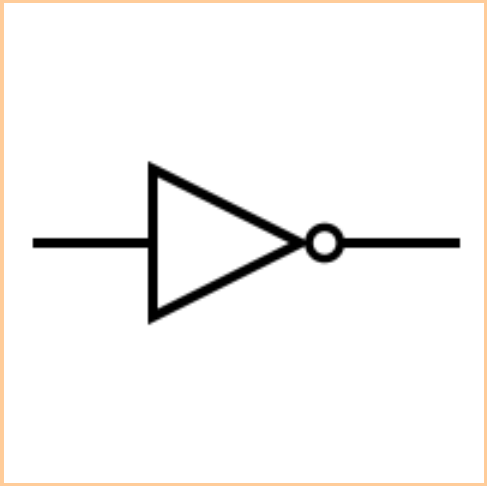


A 4-bit **ripple-carry** adder circuit



What information is needed? Where? How?

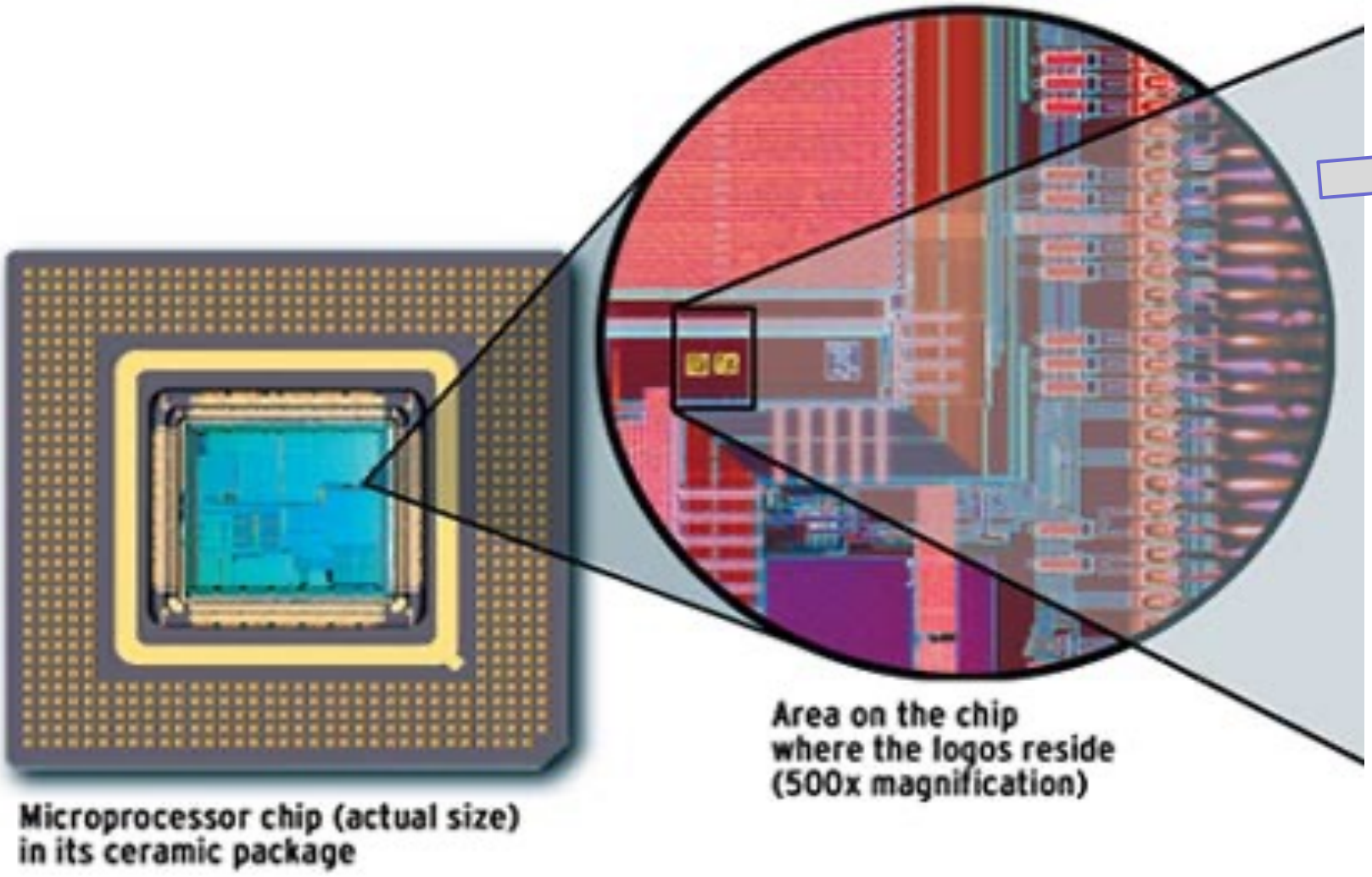
speed vs. complexity tradeoffs ~ the *"cs facets"* of engineering



What's ***inside*** gates?

What's the *other half* of computation?

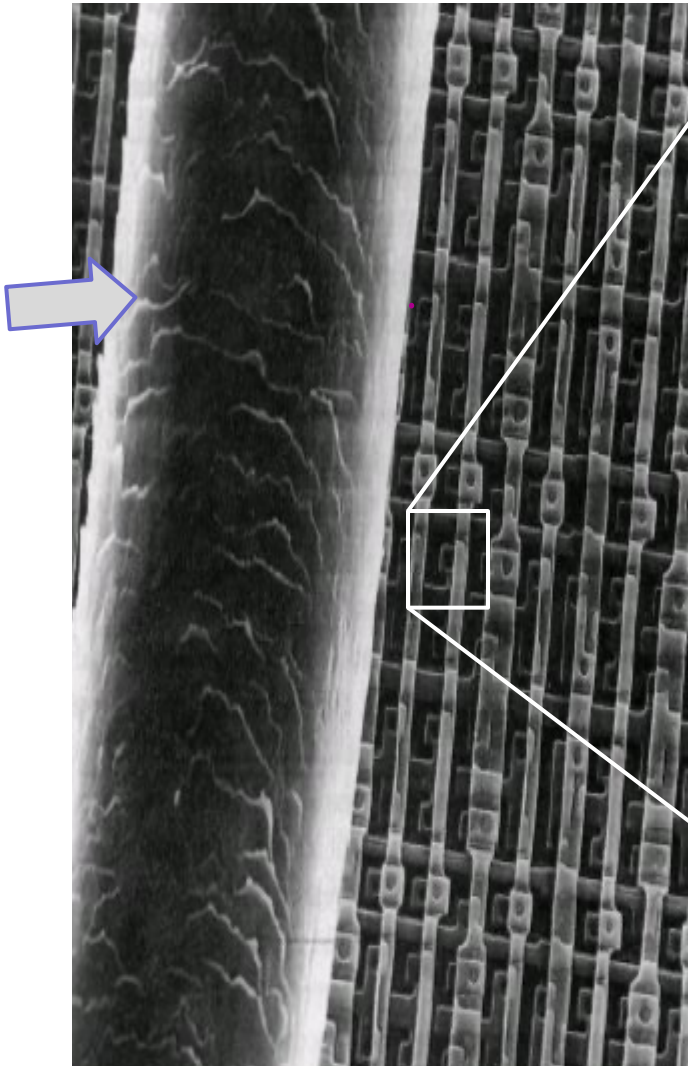
Today's gates?

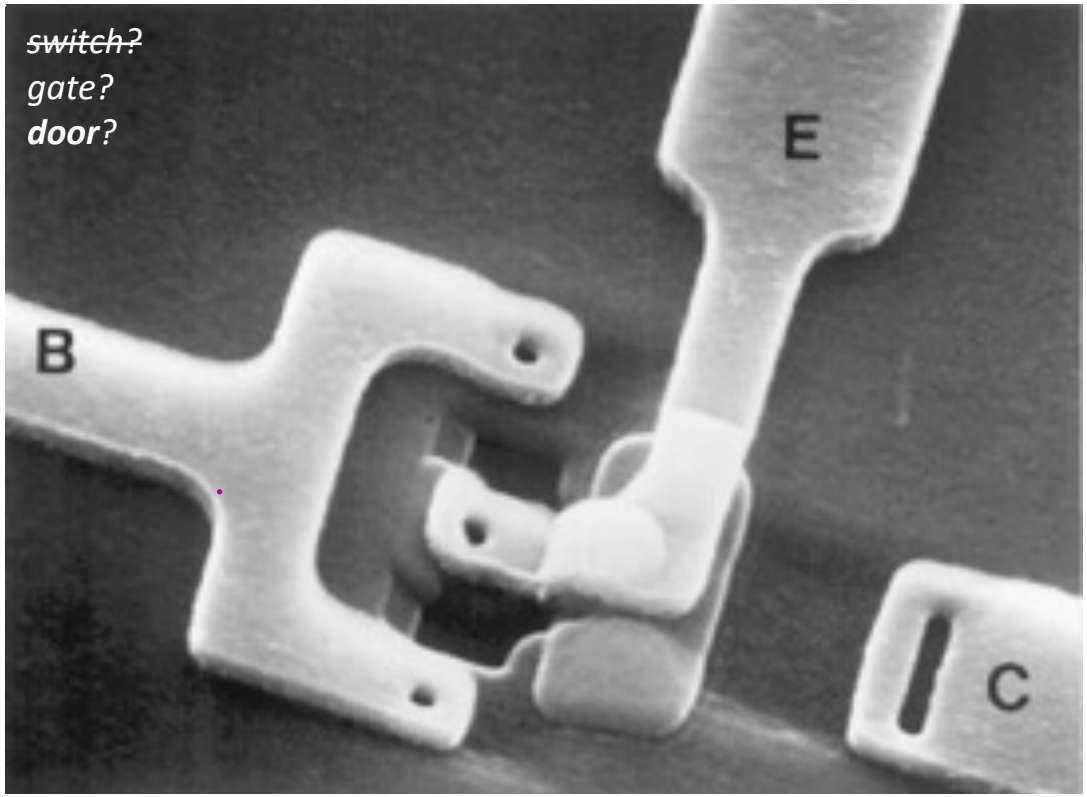


https://www.youtube.com/watch?v=2z9qme_ygRI
https://www.youtube.com/watch?v=Fxv3JoS1uY8

are from silicon-based switches ~ ***transistors***

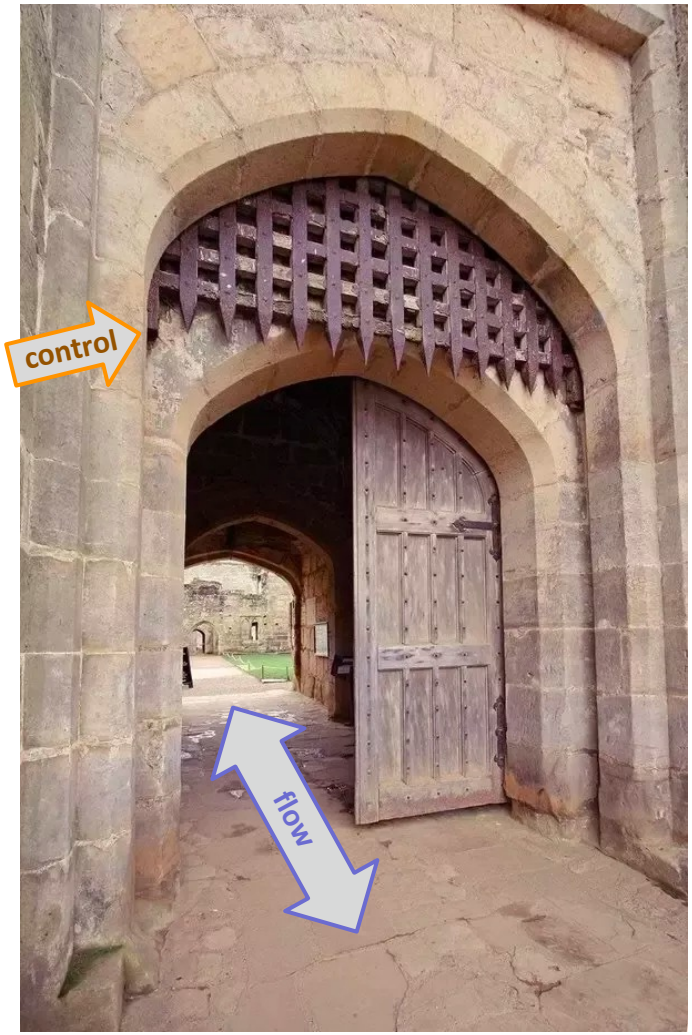
switches?

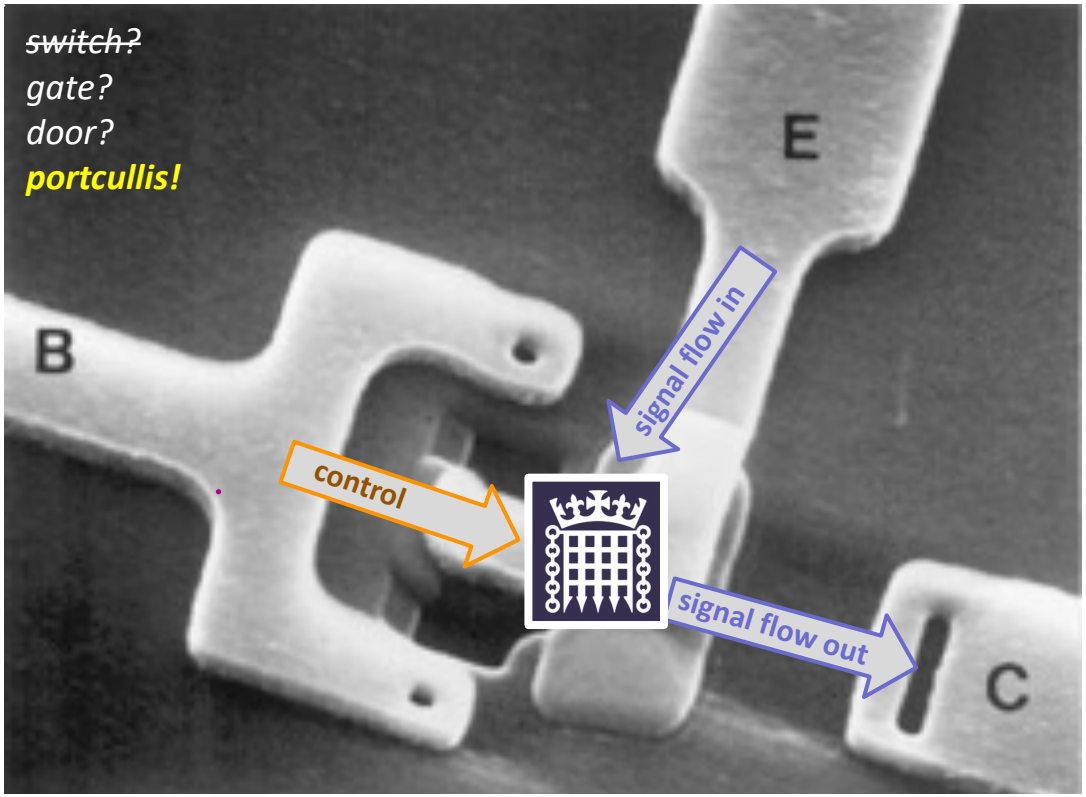




a single etched transistor labeled with base (b), emitter (e), and collector (c)

are from silicon-based switches ~ ***transistors***

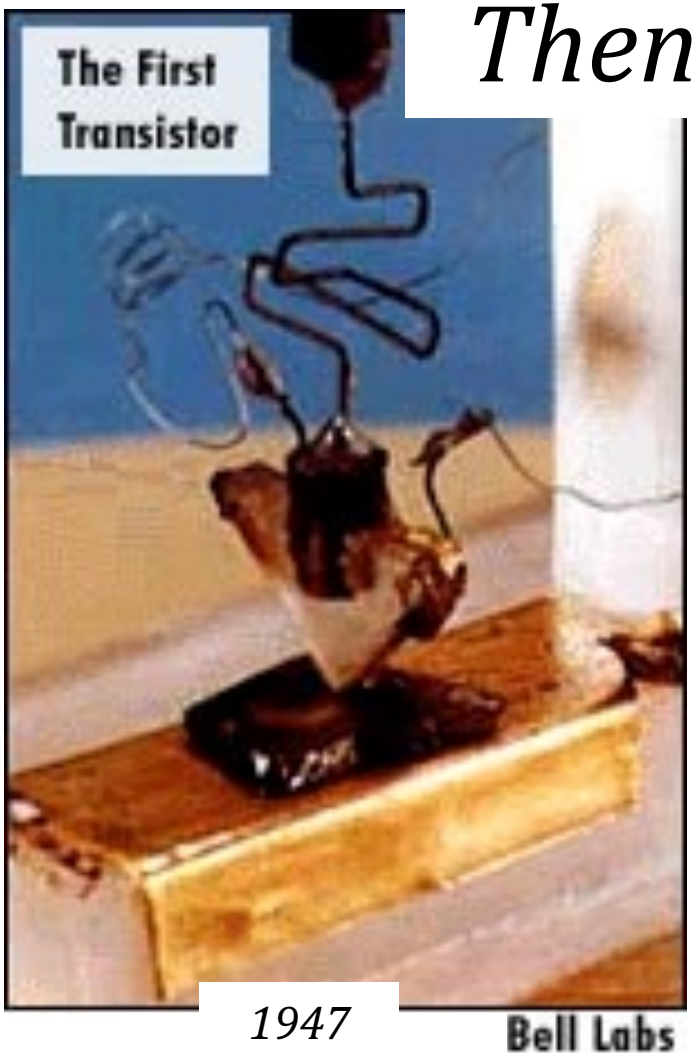




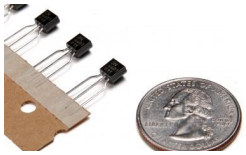
a single etched transistor labeled with base (b), emitter (e), and collector (c)

One transistor!

Then

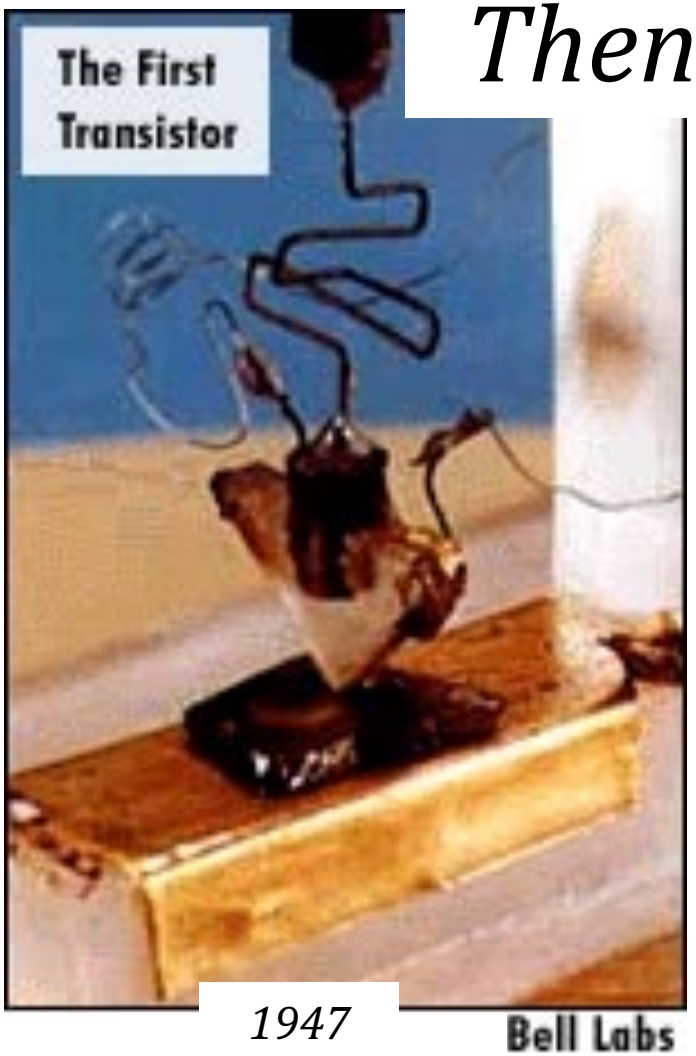


One transistor!



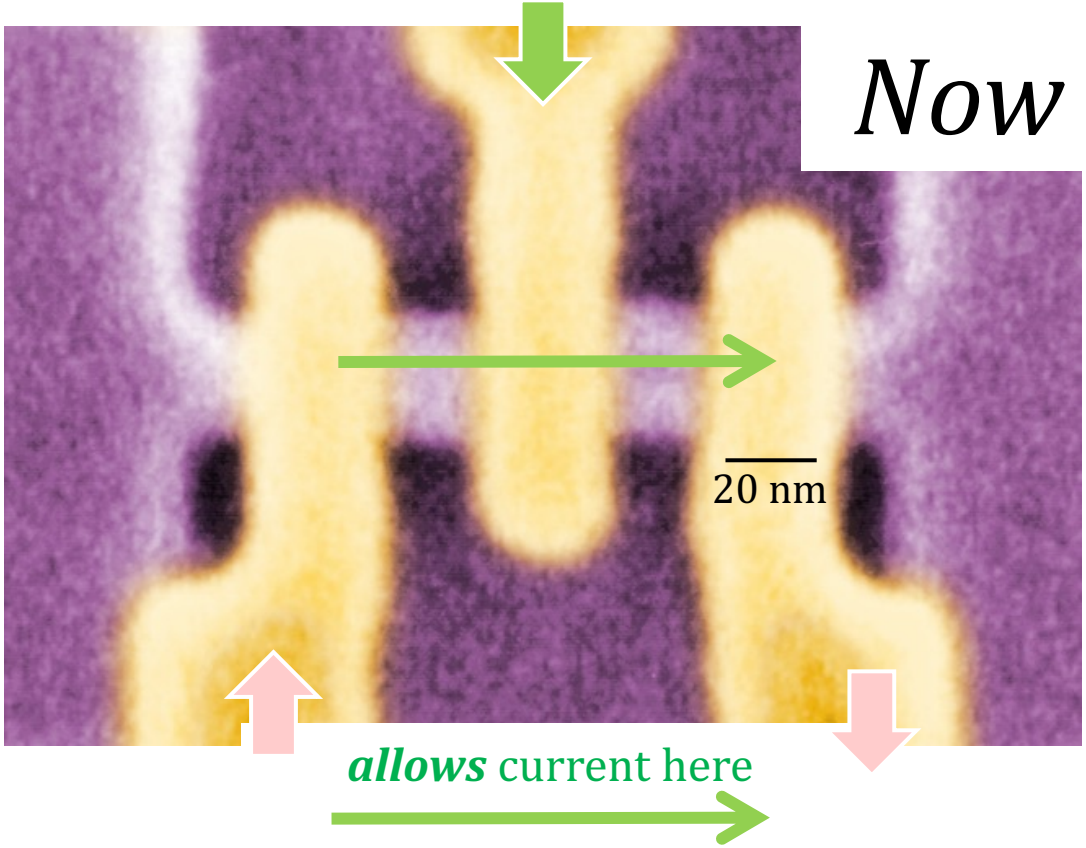
E85's transistors

Then



Now

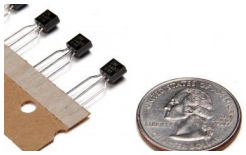
"high"
a +5v voltage here



open-on-high type transistor

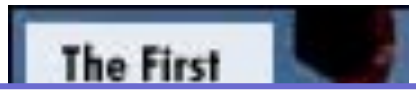
single-electron tunneling, or SET transistor

Lots of transistors!



E85's transistors

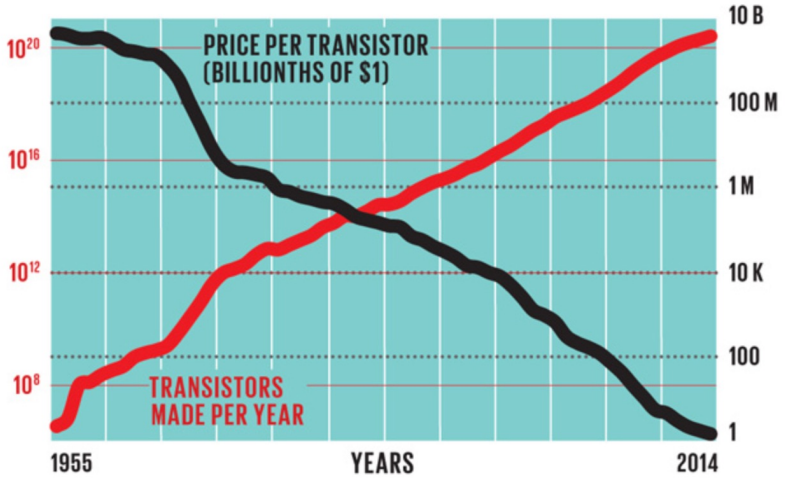
Then



Graphic: Transistor Production Has Reached Astronomical Scales

A look at Moore's Law in action

By **Dan Hutcheson**



1947 Bell Labs

Now

"high"

a +5v voltage here

https://en.wikipedia.org › wiki › Transistor_count

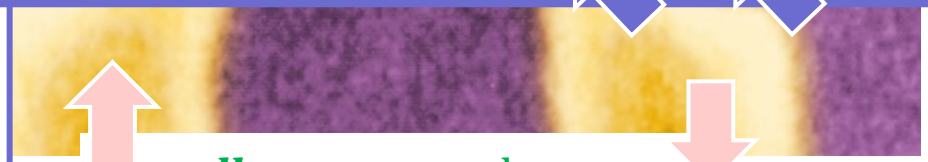
Transistor count - Wikipedia

About featured snippets • Fe

People also ask

How many transistors are produced each year?

Global production of transistors has surpassed 20 trillion per second—**hundreds of quintillions per year**. Apr 4, 2017



allows current here

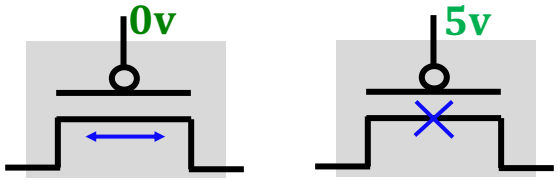
open-on-high type transistor

single-electron tunneling, or SET transistor

Two *types* of transistors...

Transistors are current ***gates***:

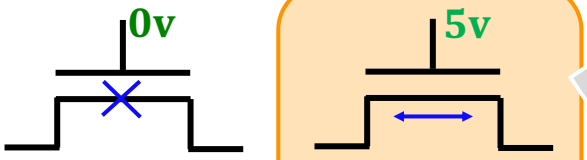
open-on-low (pmos)



0v "opens" this wire

5v "cuts" this wire

open-on-**high** (nmos)

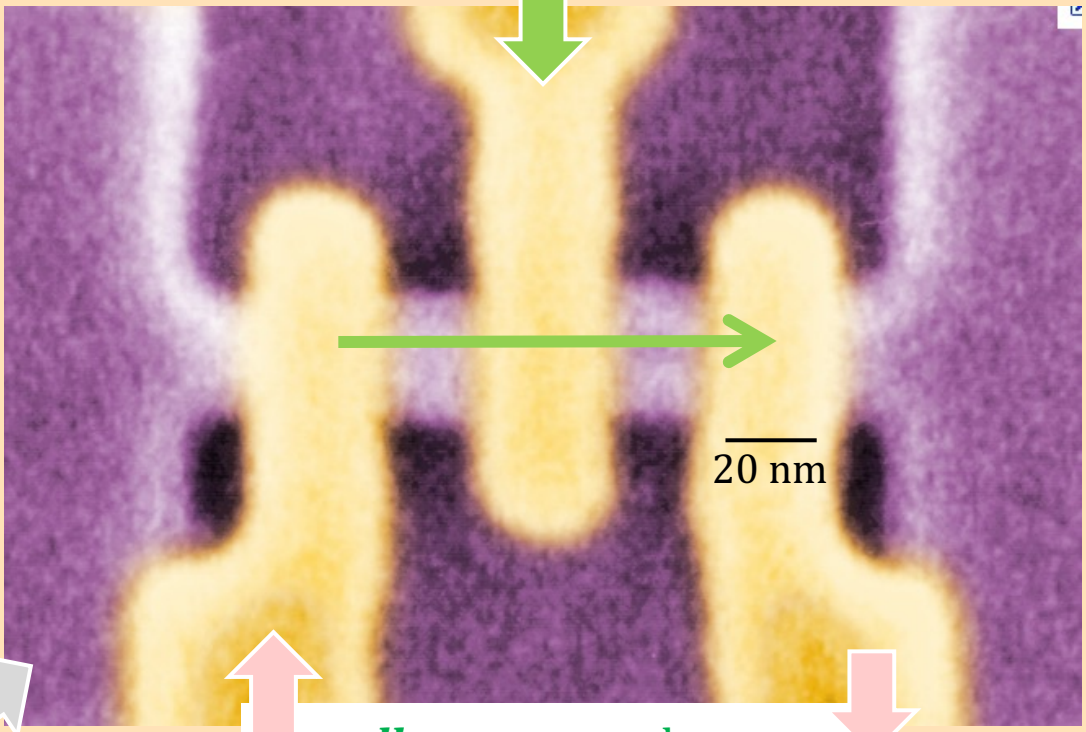


0v "cuts" this wire

5v "opens" this wire

"high"

a +5v voltage here



allows current here

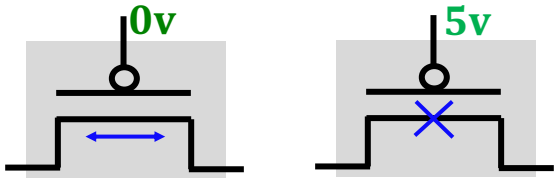
open-on-**high**-type transistor

*an input of **1** opens the portcullis...*

Two *types* of transistors...

Transistors are current ***gates***:

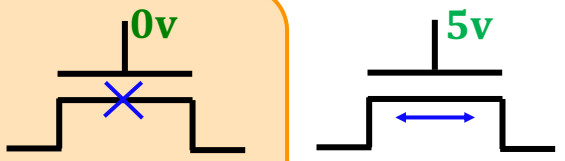
open-on-low (pmos)



0v "opens" this wire

5v "cuts" this wire

open-on-**high** (nmos)

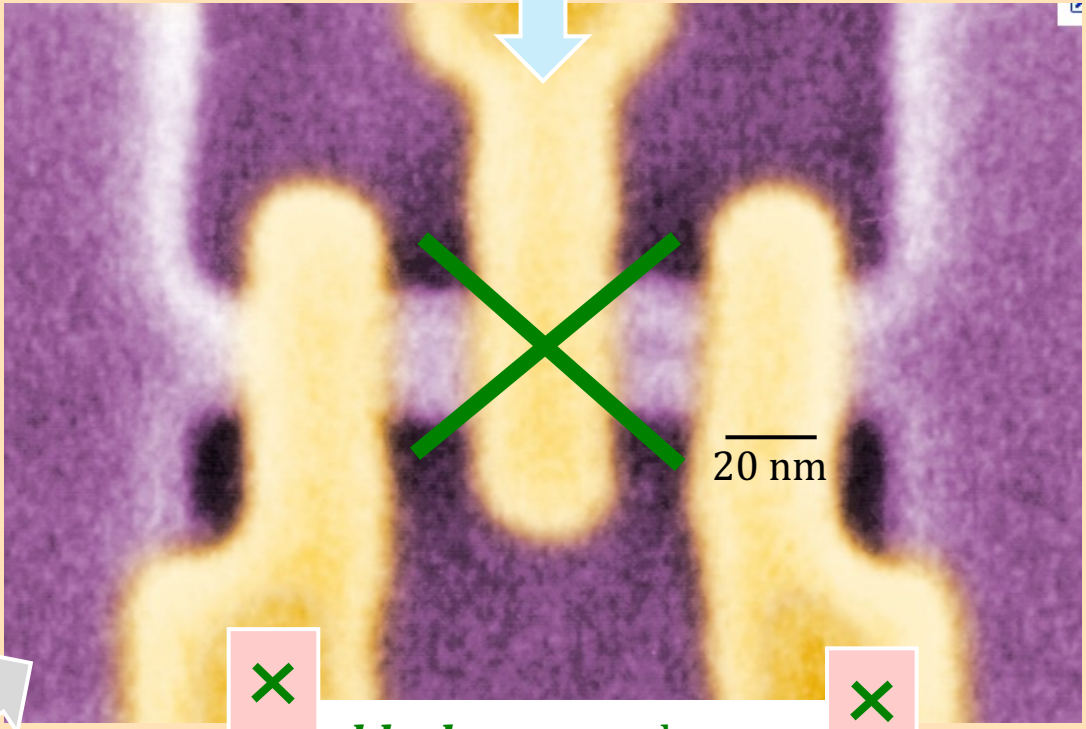


0v "cuts" this wire

5v "opens" this wire

"low"

a 0v voltage here



blocks current here

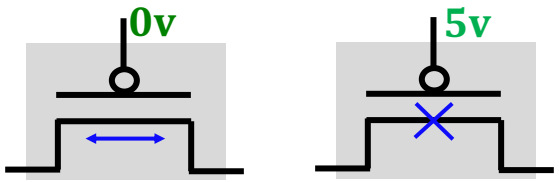
open-on-**high**-type transistor

*an input of **0** closes the portcullis!*

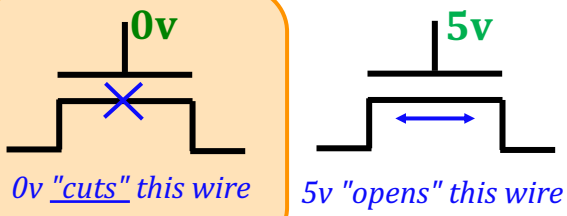
Rotations are common...

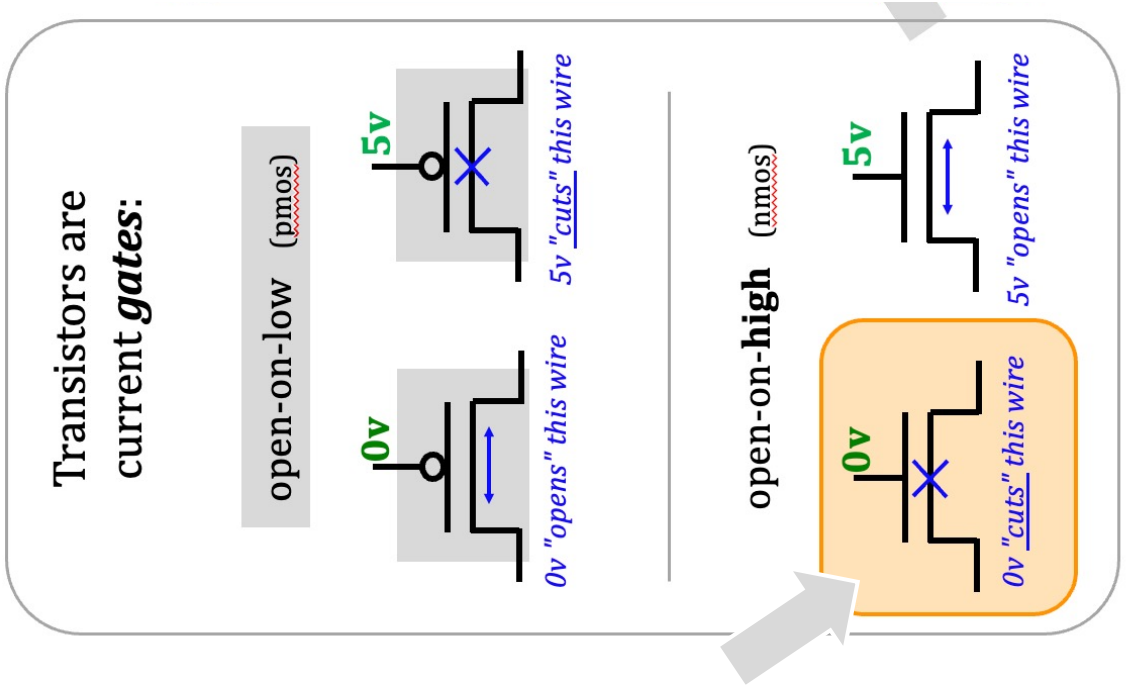
Transistors are current ***gates***:

open-on-low (pmos)



open-on-**high** (nmos)

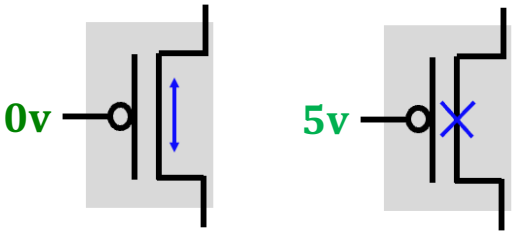




Building a NOT gate

Transistors are current ***gates***:

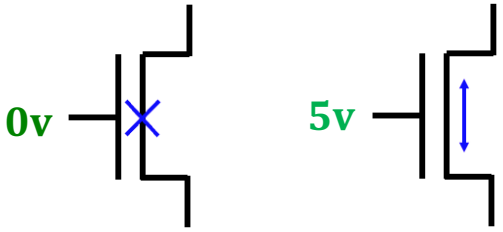
open-on-low (pmos)



0v "opens" this wire

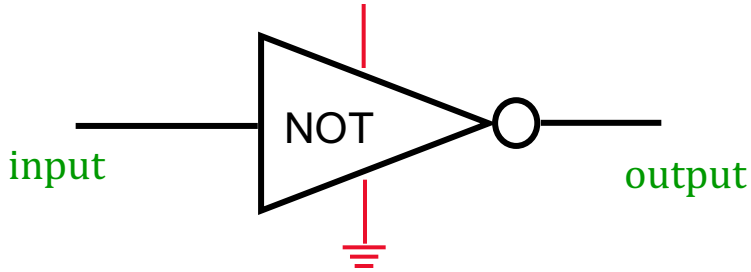
5v "cuts" this wire

open-on-high (nmos)

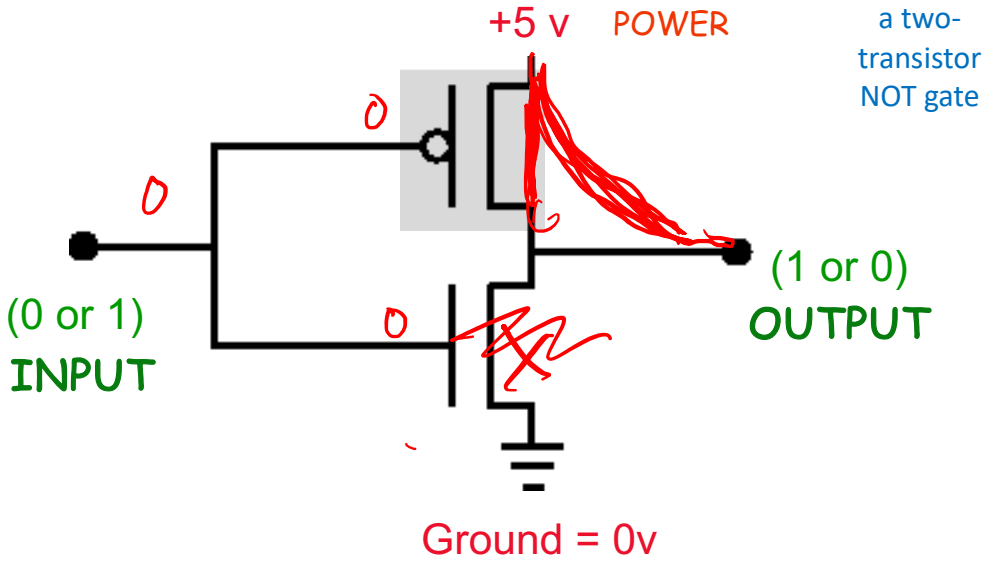


0v "cuts" this wire

5v "opens" this wire

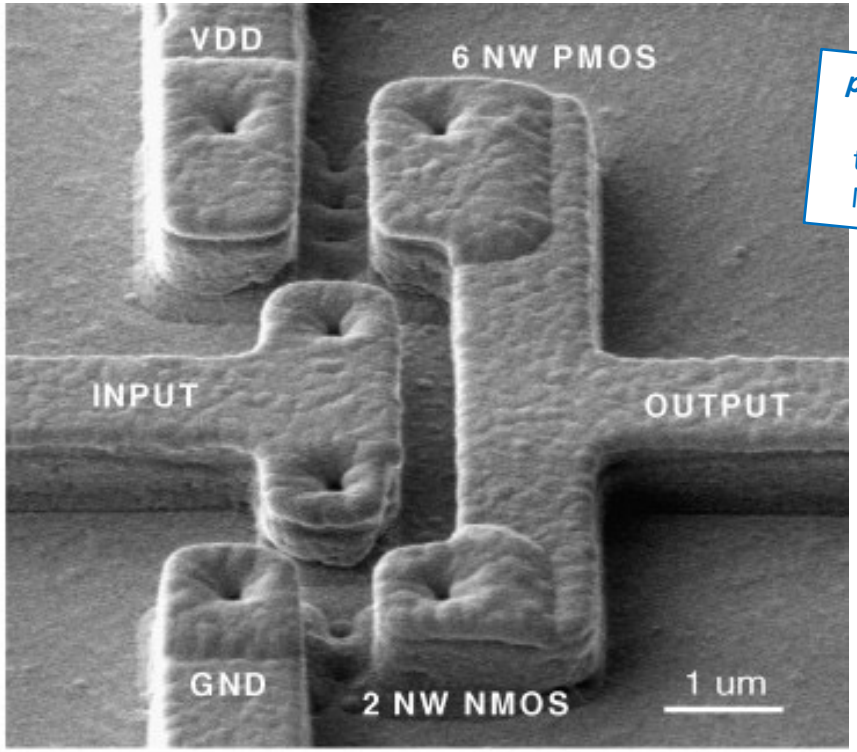


Building a **NOT** gate from transistors:



a NOT gate

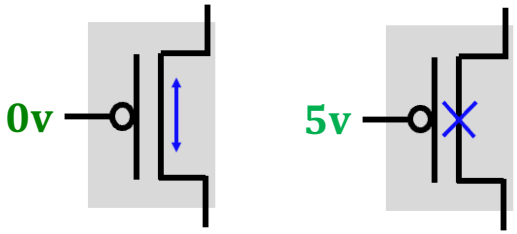
Implemented!



photograph of a two-transistor NOT gate

Transistors are current ***gates***:

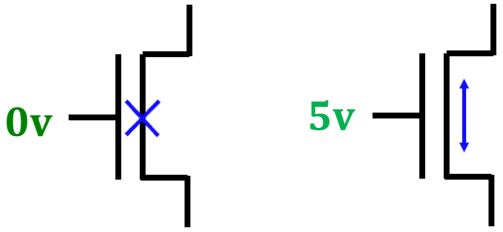
open-on-low (pmos)



0v "opens" this wire

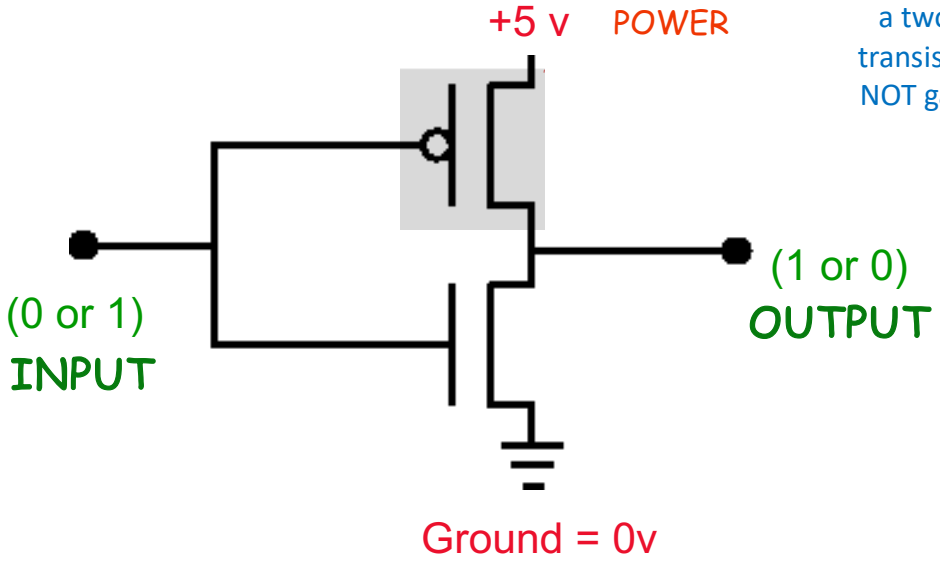
5v "cuts" this wire

open-on-high (nmos)



0v "cuts" this wire

5v "opens" this wire



a two-transistor NOT gate

Transistors!

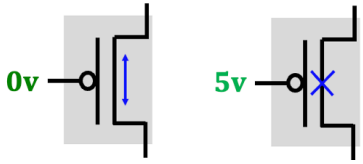
(1) What will be output?
Fill out the truth table to the right

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

(1) Fill in this circuit's truth table

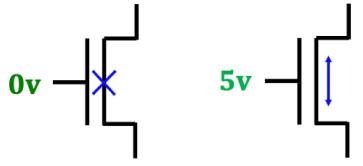
Transistors are current ***gates***:

open-on-low (pmos)

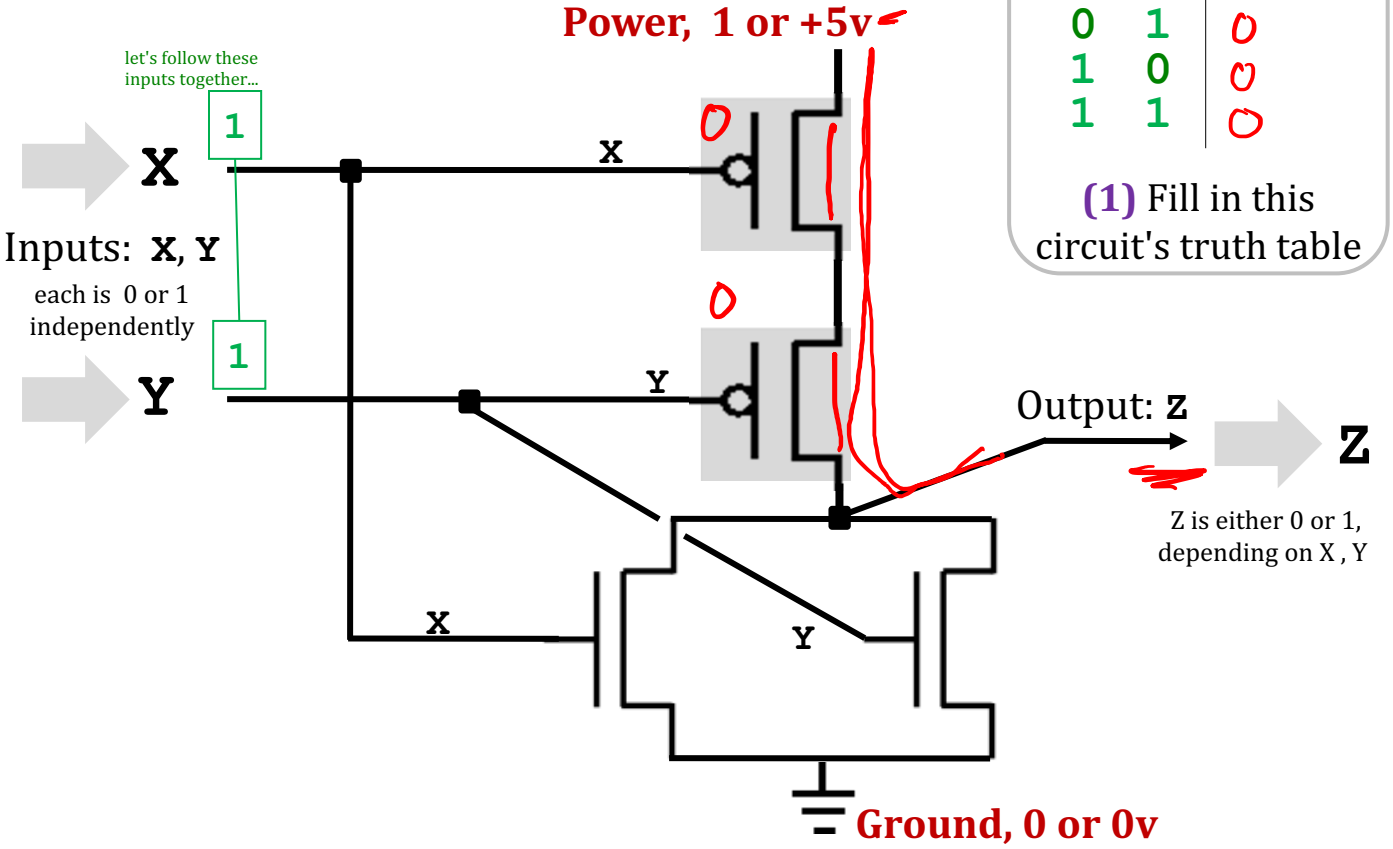


0v "opens" this wire *5v "cuts" this wire*

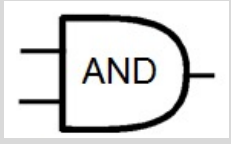
open-on-high (nmos)



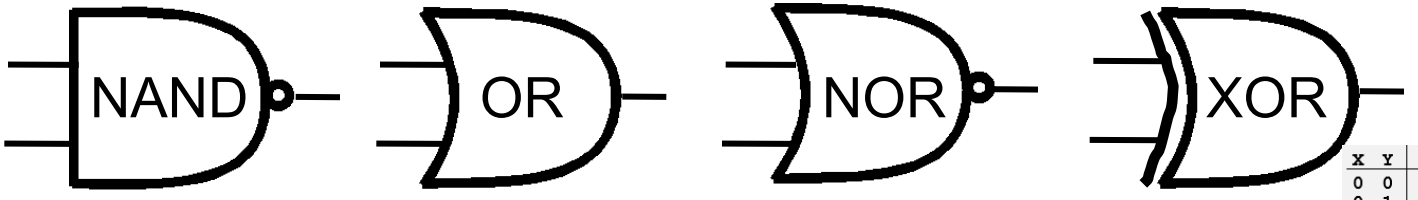
0v "cuts" this wire *5v "opens" this wire*



(3) Extra! How could you alter the *transistor-level* design to make the design above into an AND gate?



(2) Challenge: What gate is the above diagram? **It's one of these four:**



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Transistors!

Answers!

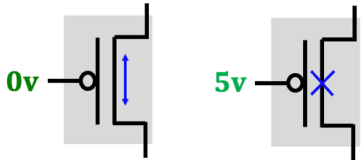
(1) What will be output?
Fill out the truth table to the right

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

(1) Fill in this circuit's truth table

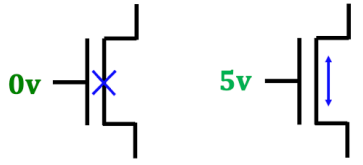
Transistors are current ***gates***:

open-on-low (pmos)



0v "opens" this wire *5v "cuts" this wire*

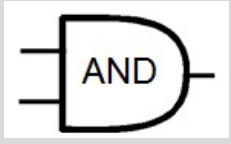
open-on-high (nmos)

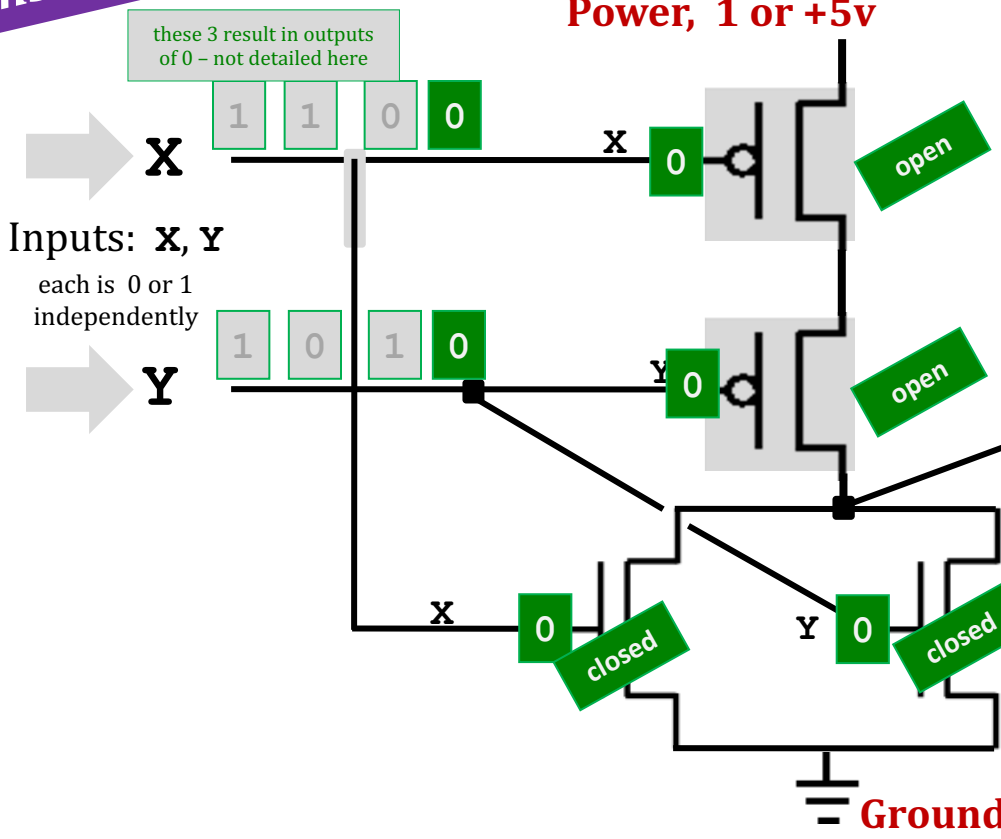


0v "cuts" this wi *"opens" this wire*

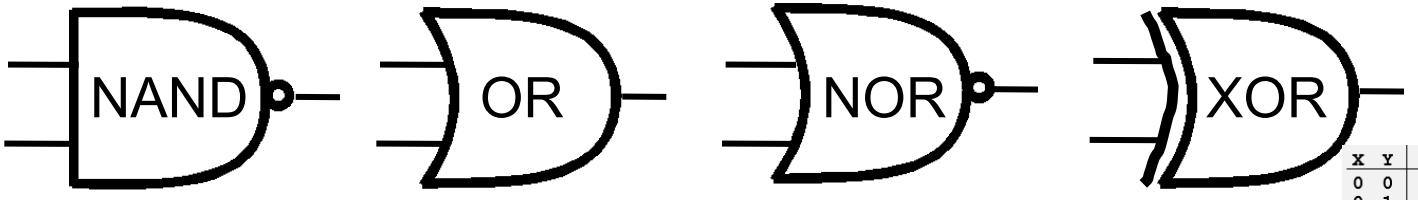
Swap the transistor TYPES!

(3) Extra! How c… you alter the *transistor-level* design to make the design above into an AND gate?



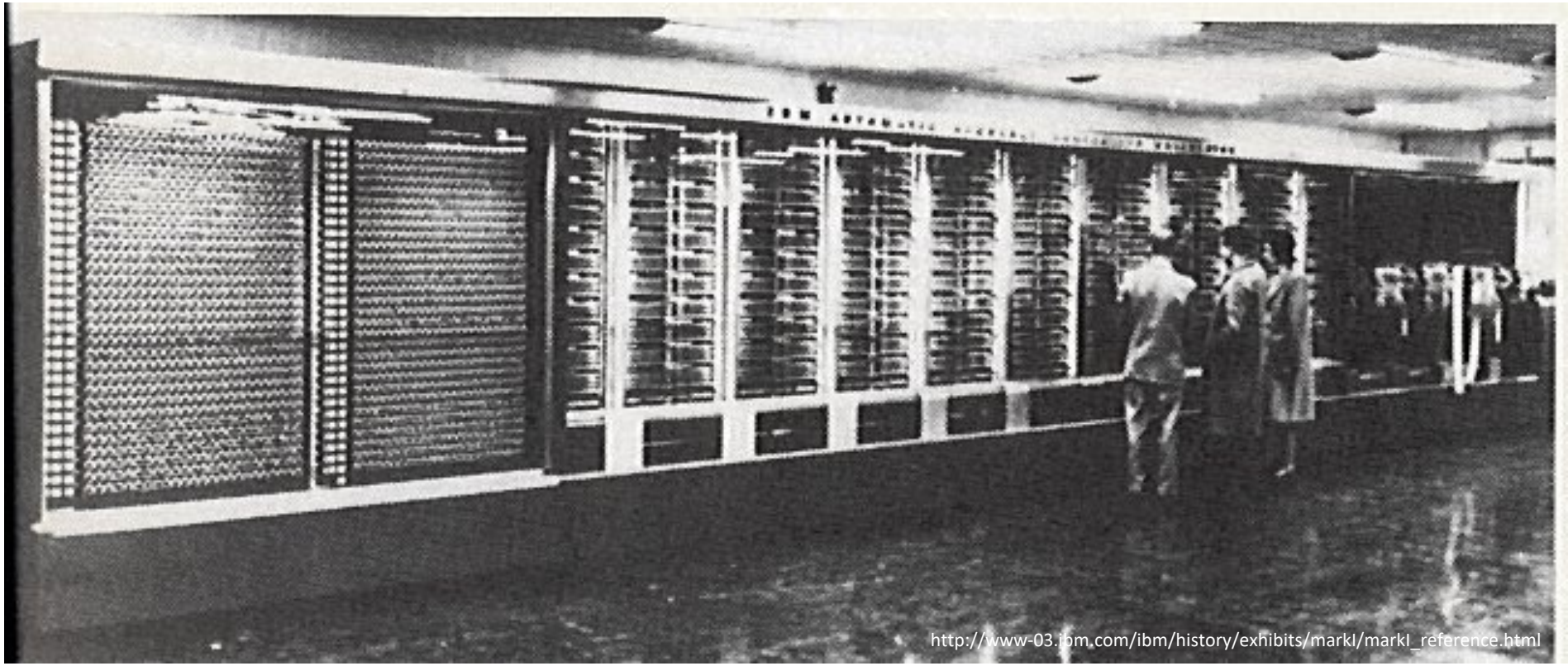


(2) Challenge: What gate is the above diagram? **It's one of these four:**



Their Mark 1

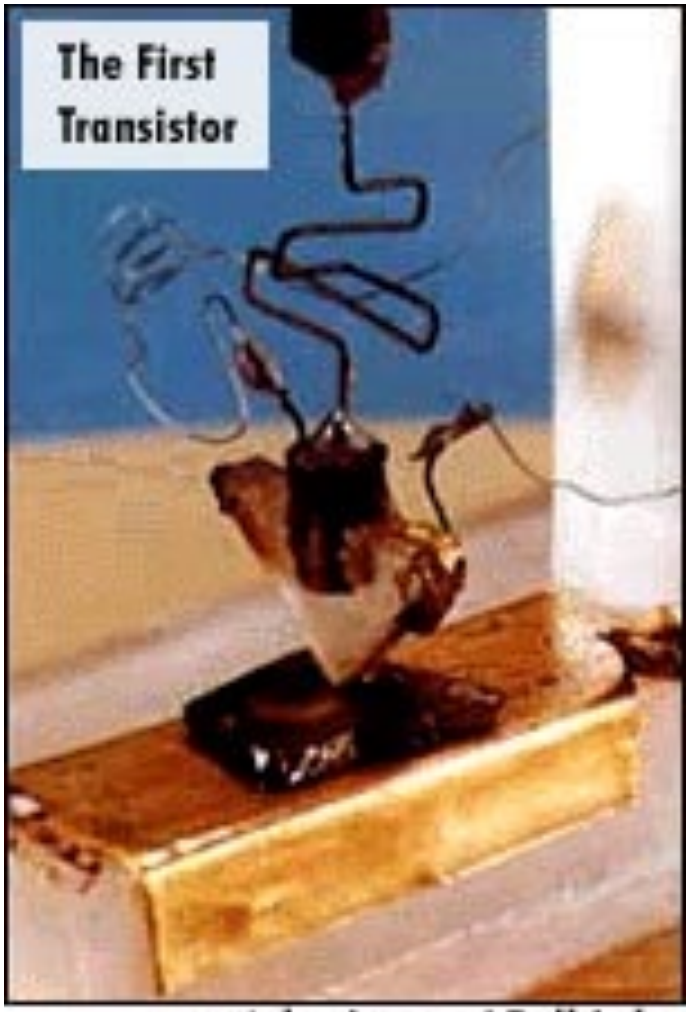
an early, relay-based computer



Grace Hopper + Howard Aiken, Harvard ~ 1944

Transistors as *disruptive* technology

point contact transistor



copyright: Lucent / Bell Labs

1947: **Bell Labs**

seeking better amplifiers for phone lines

team of physicists: W. Brattain, *W. Shockley*, and J. Bardeen

1948: **junction transistor**

much more robust design

1956: **Shockley Semiconductor** Co.

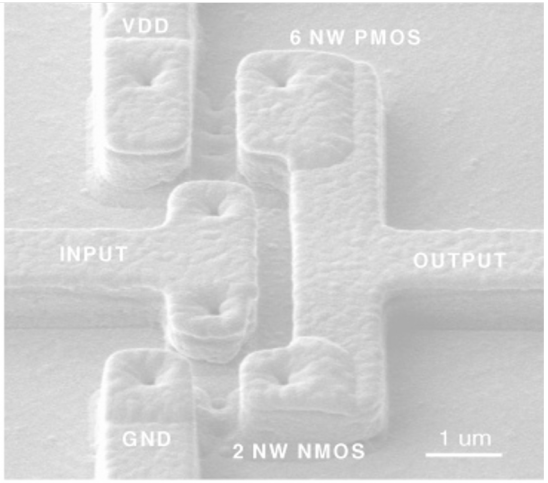
in hometown of Palo Alto...

in a few months...

the "traitorous eight" left to found

1957: **Fairchild Semiconductor** Co.

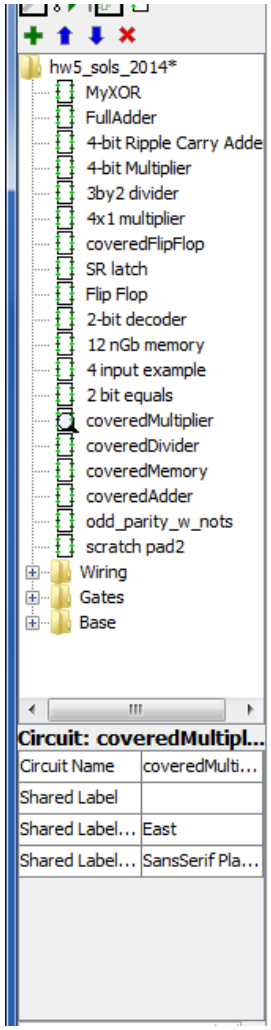
*... and so begins the valley's **siliconization***

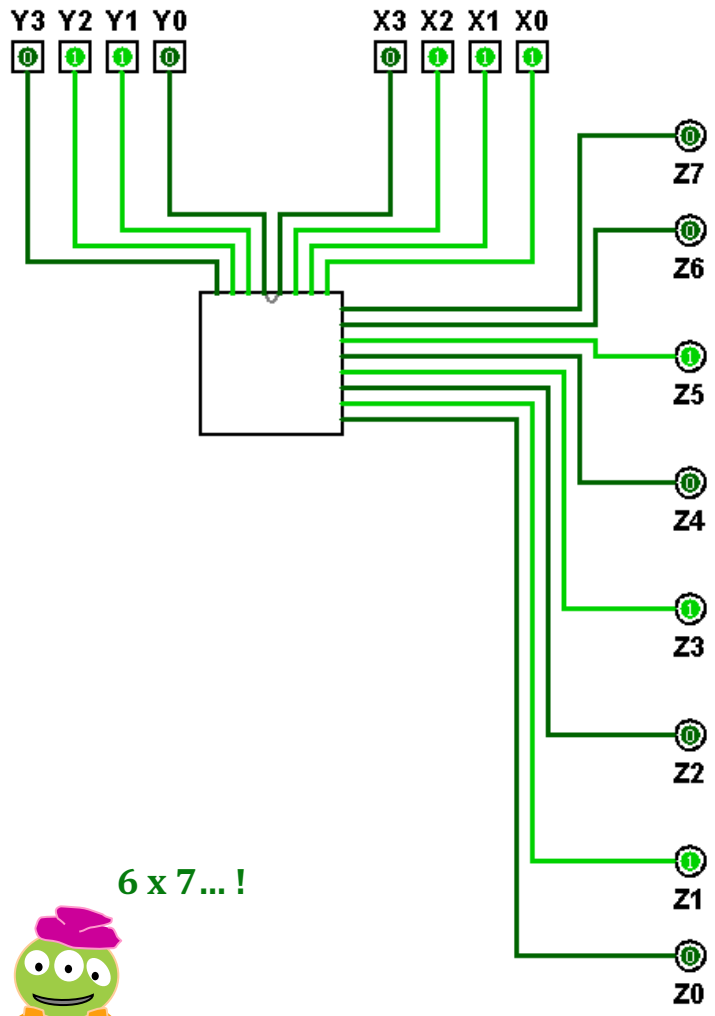


What's *inside* gates?

What's the *other half* of computation?

Half a computer: the **CPU**





6 x 7… !

transistors

↓

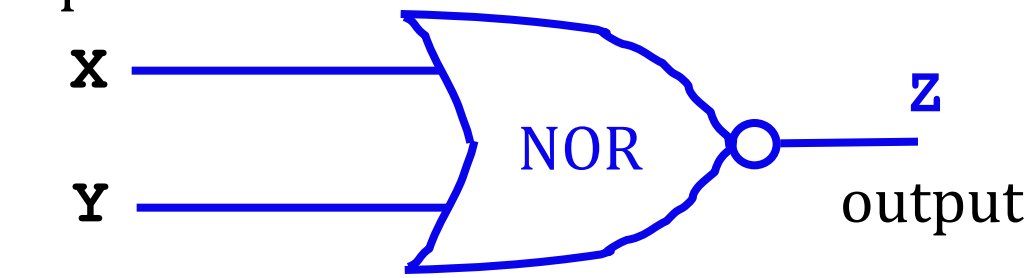
gates

↓

arithmetic

For systems, a face-lift is to add an edge that ***creates a cycle***, not just an additional node.

- also Alan Perlis



NOR's Truth Table

NOR

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Let's engineer this into 1 bit of memory!

Memory!

NOR's Truth Table

NOR		
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

- The circuit starts with R being **0** + S being **0**

  and Q starts at _0_ (the "loopback wire" from S to R will be 1)

- What if S stays **0** and R is set to **1**?

  Q is then set to __ _0_

- What happens if S stays **0** and R is set ***back*** to **0**?

  Q still stays (!) at __ _0_

- What happens if R is **0** and S is set to **1**?

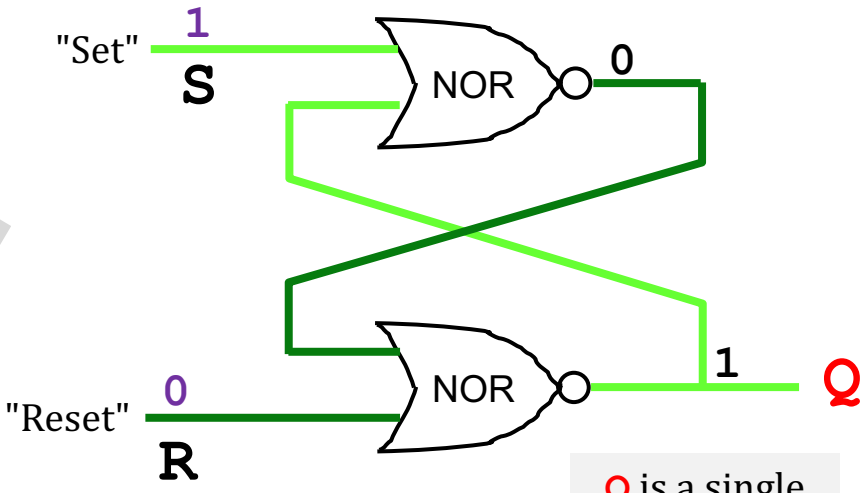
  Q is then set to __ _1_

- What happens if S is **0** and R is set ***back*** to **0**?

  Q still stays (!) at __ _1_

*Why does "S" stand for "**Set**" and R for "**Reset**" ?*

S "sets" Q to 1; R "resets" it back to 0.

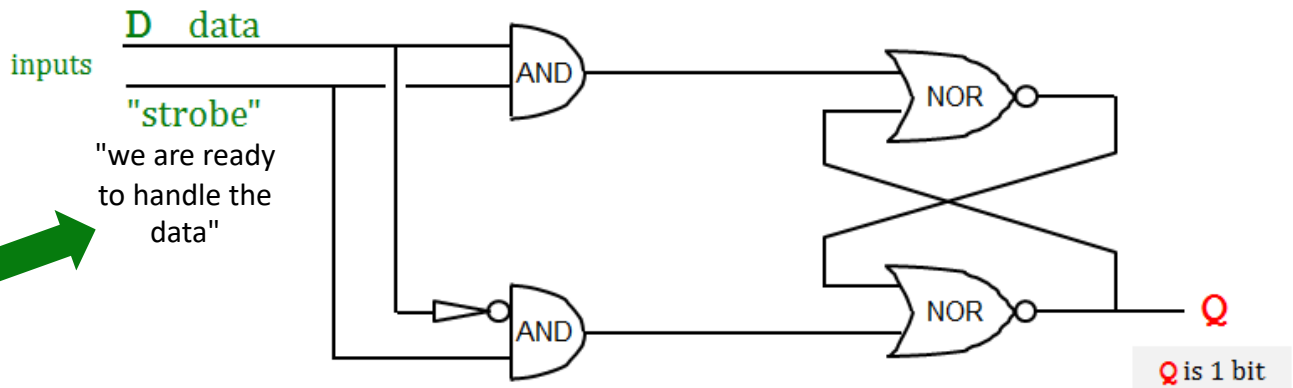


Q is a single bit of storage

Take a look at this circuit:

The D (**data**) line holds a single bit we want to store (either a 0 or a 1).

How does the strobe bit help store the bit D into Q?

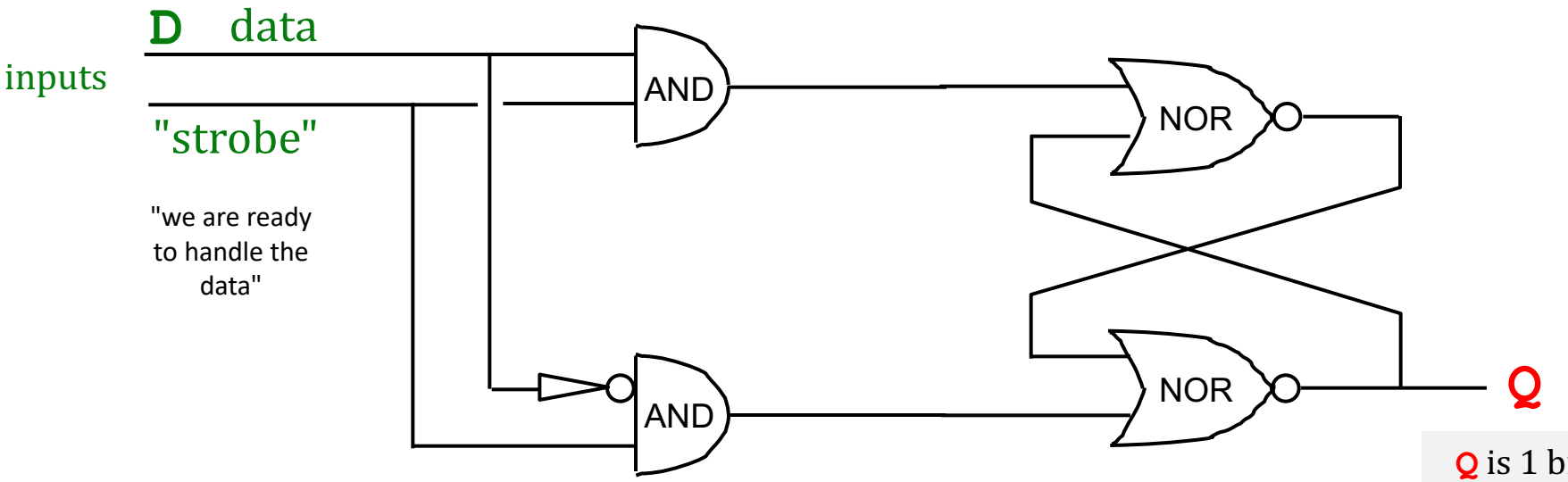


Q is 1 bit of storage

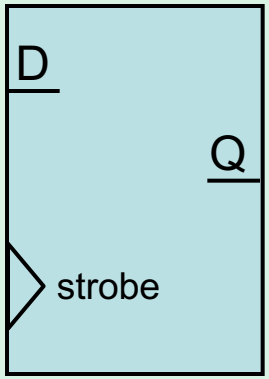
Hint: What happens when the "strobe" is 1?

The flip-flop

Demo!



Q is 1 bit of storage

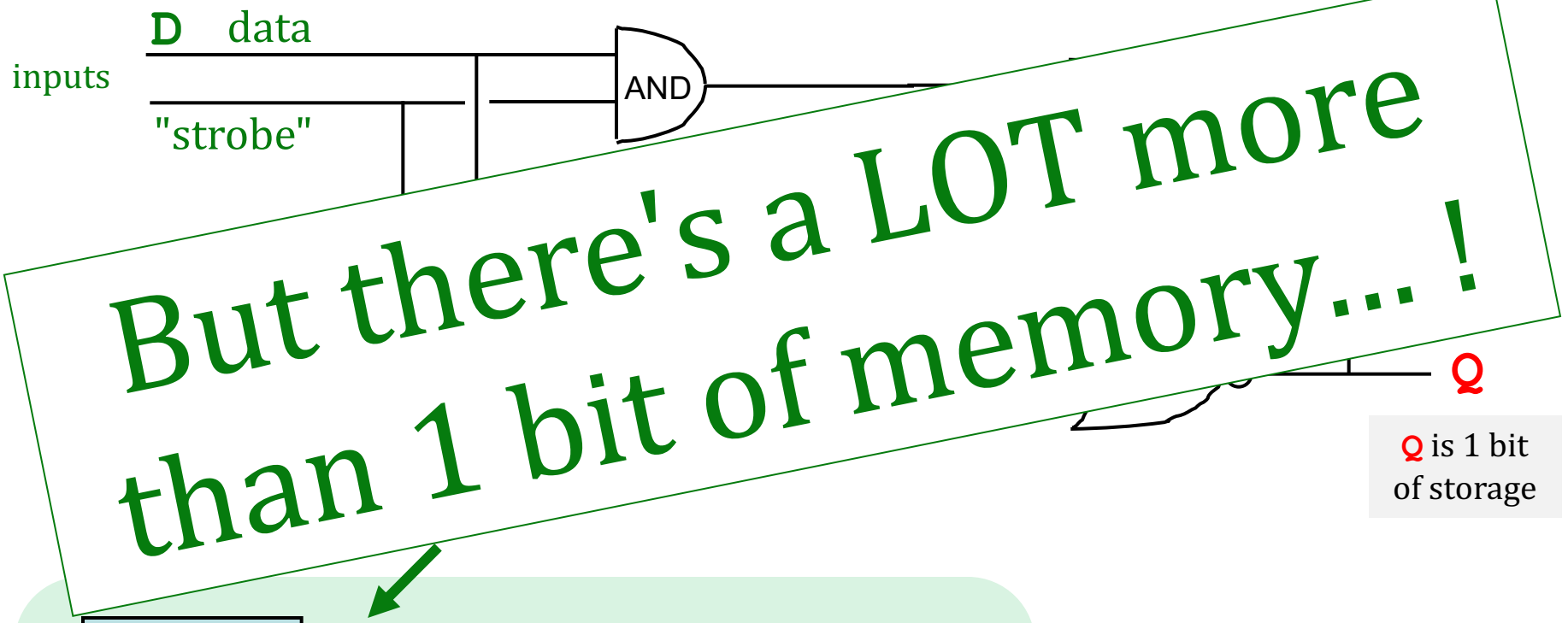


the flip-flop's diagram

1 bit of memory!

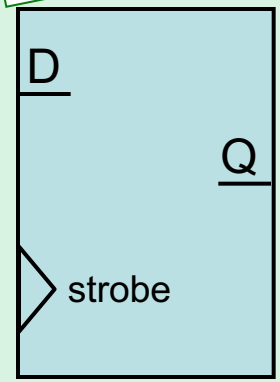
The flip-flop

Demo!



But there's a LOT more than 1 bit of memory... !

Q is 1 bit of storage



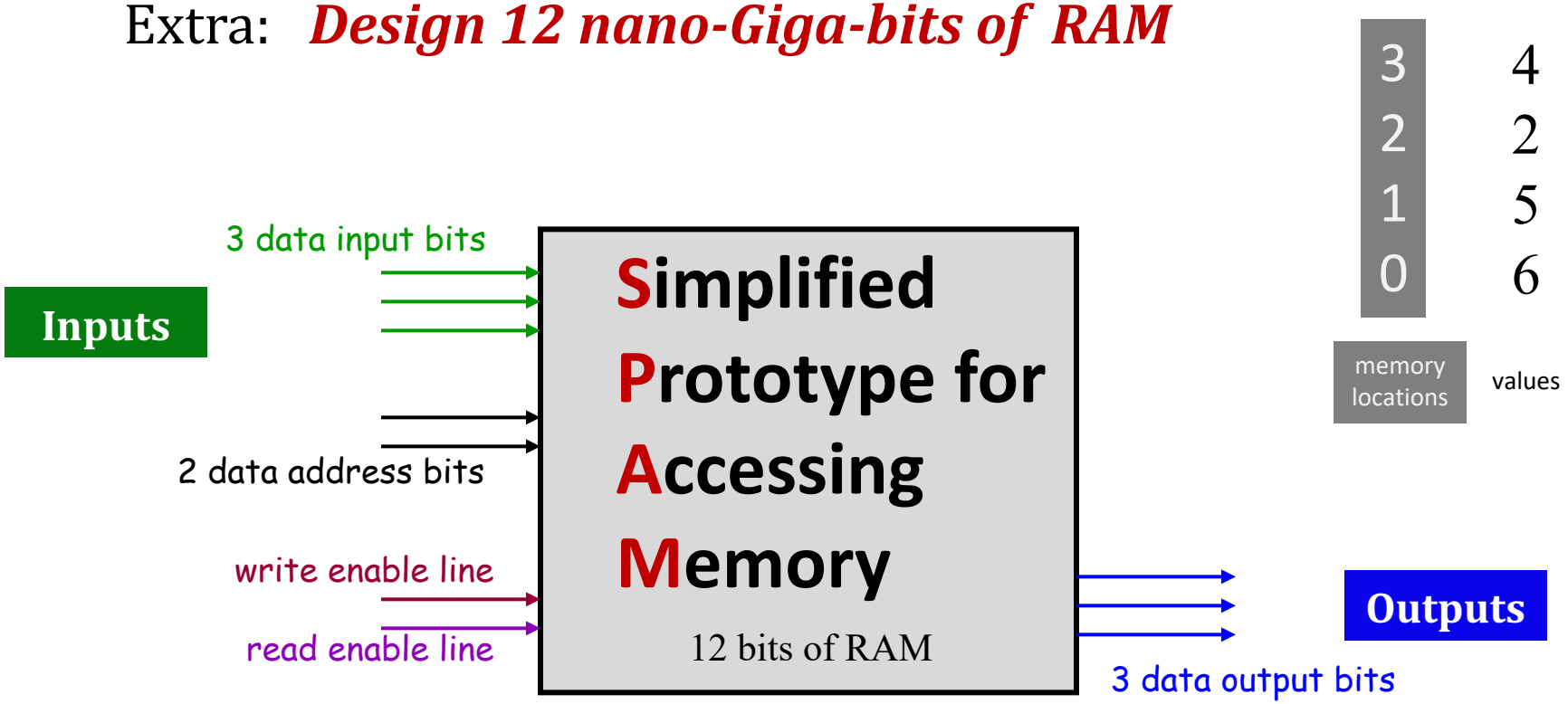
the flip-flop's diagram

1 bit of memory!

Random Access Memory

Demo!

Extra: ***Design 12 nano-Giga-bits of RAM***



3 bits stored at location 00
3 bits stored at location 01
3 bits stored at location 10
3 bits stored at location 11

Ex Cr

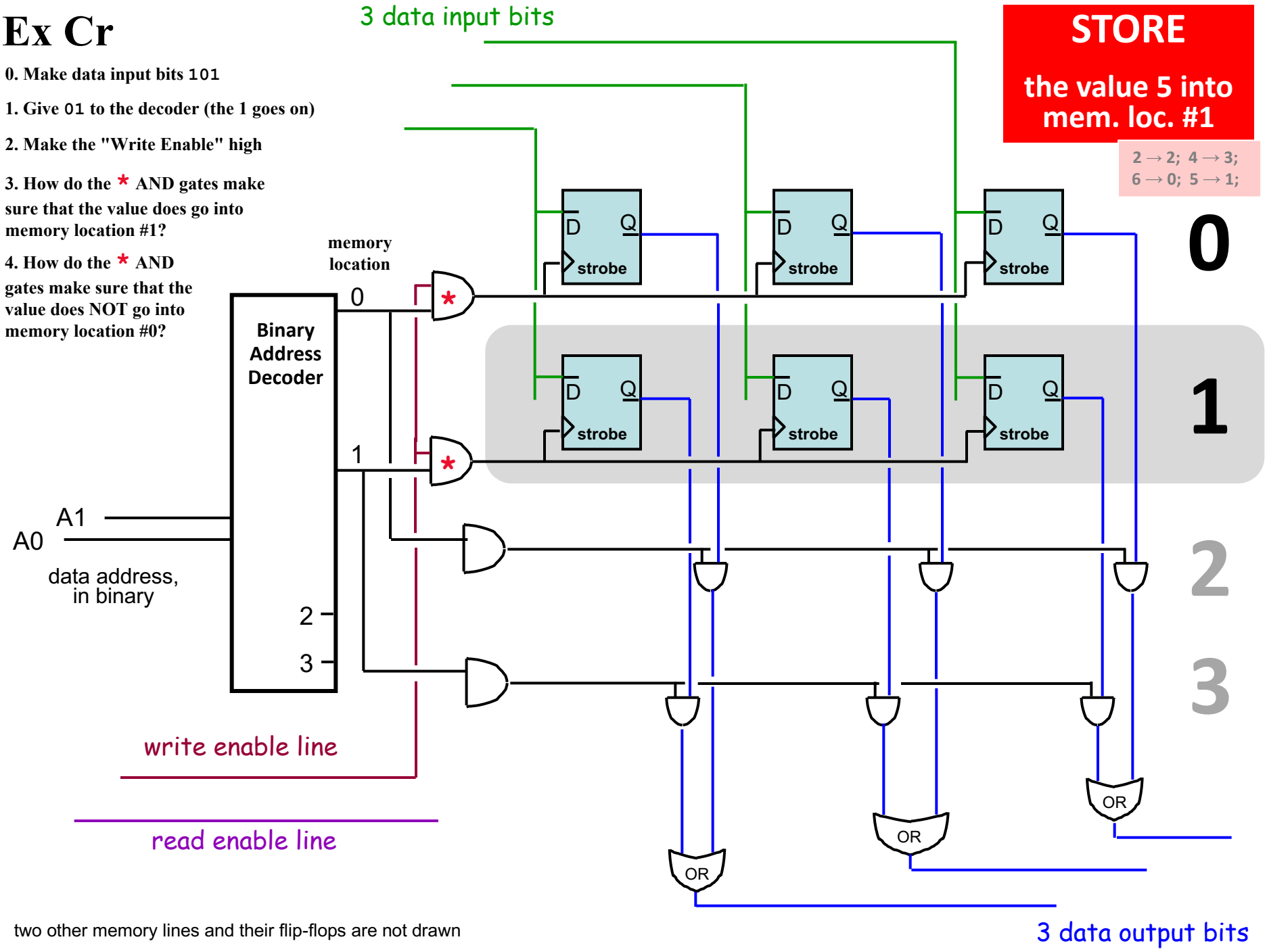
0. Make data input bits 101

1. Give 01 to the decoder (the 1 goes on)

2. Make the "Write Enable" high

3. How do the * AND gates make sure that the value does go into memory location #1?

4. How do the * AND gates make sure that the value does NOT go into memory location #0?



0. Suppose 101 is in Location #1

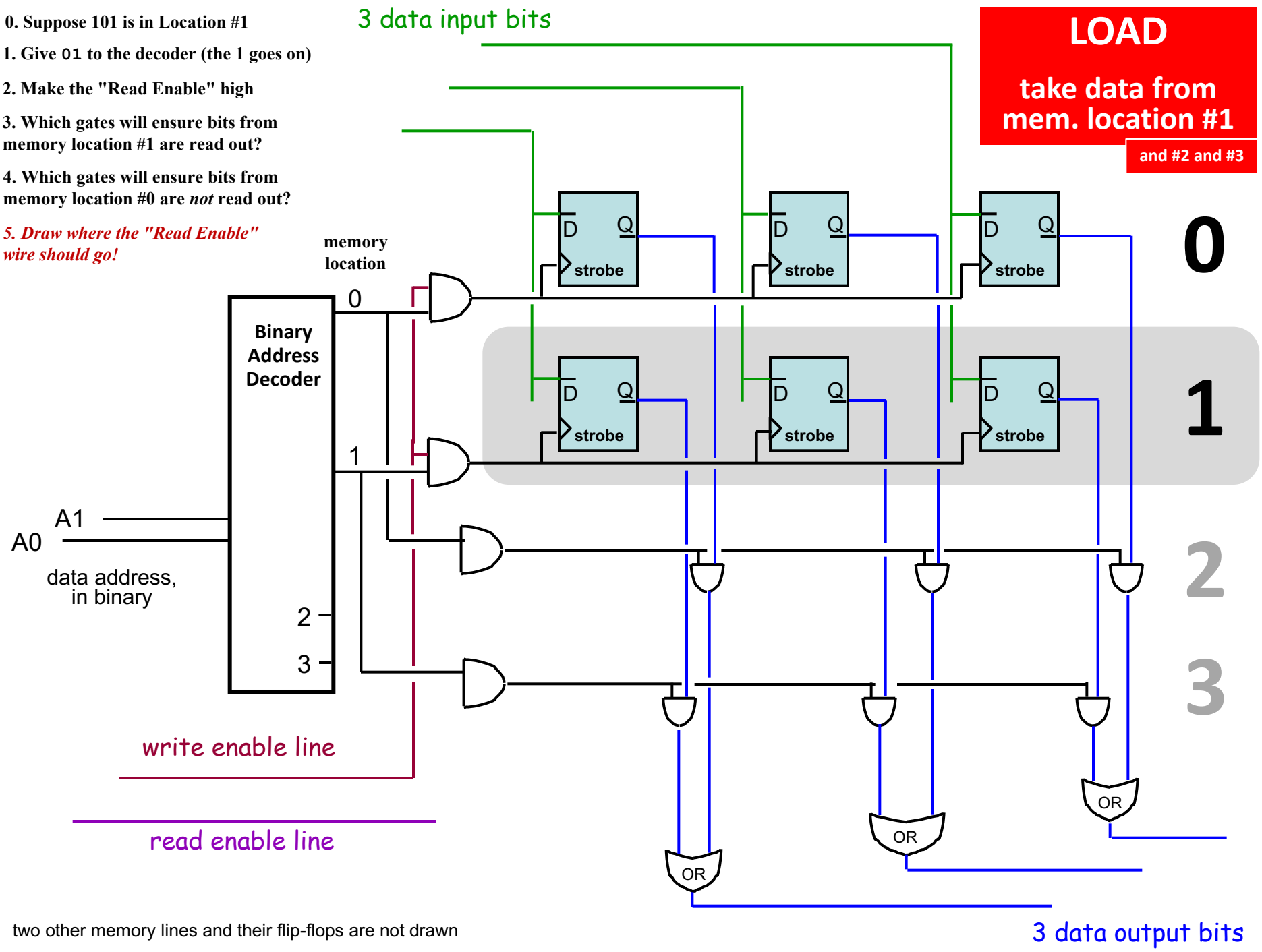
1. Give `01` to the decoder (the 1 goes on)

2. Make the "Read Enable" high

3. Which gates will ensure bits from memory location #1 are read out?

4. Which gates will ensure bits from memory location #0 are *not* read out?

5. Draw where the "Read Enable" wire should go!



two other memory lines and their flip-flops are not drawn

3 data output bits