6.S188

Build a Digital Clock from the Eighties

Lecture 1B:
Designing Digitally

Ok we actually want to design stuff

- Claude Shannon showed how we could actually design complicated digital circuits using math.

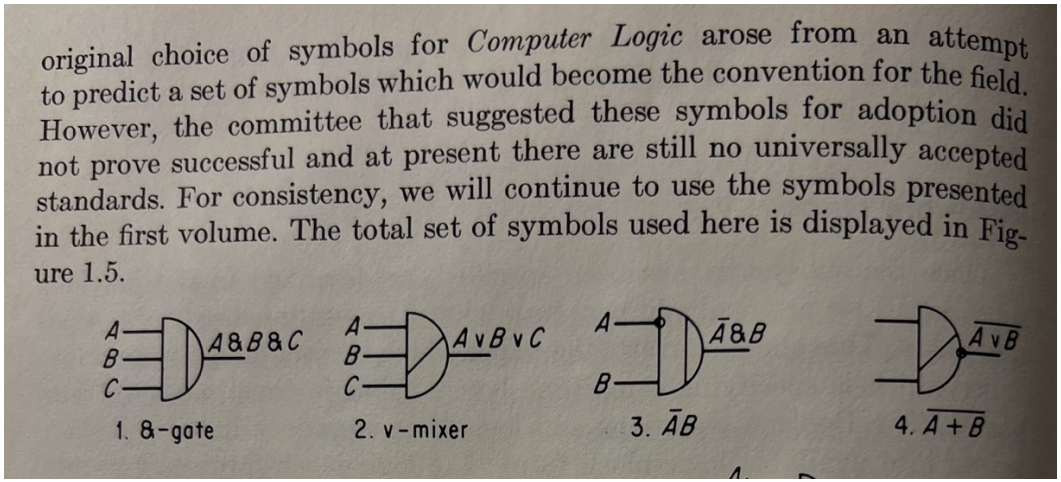
- We don't use regular, normal human math*

- We have to use Boolean Algebra/representation an algebra where things take on only one of two values (0 or 1)

**though what is "normal human math" really anyways, Joe?*

Standardizing

- Prior to late 1950's *how* exactly you'd do digital design wasn't super streamlined.
- A lot of the logic gates weren't standardized..

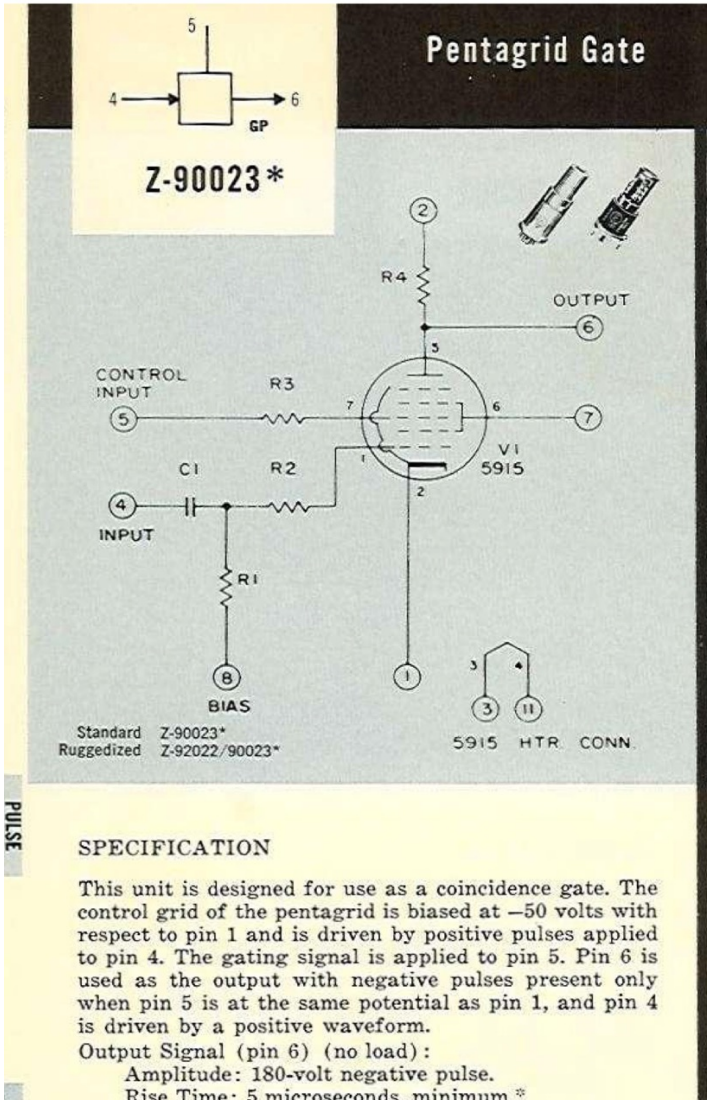


original choice of symbols for *Computer Logic* arose from an attempt to predict a set of symbols which would become the convention for the field. However, the committee that suggested these symbols for adoption did not prove successful and at present there are still no universally accepted standards. For consistency, we will continue to use the symbols presented in the first volume. The total set of symbols used here is displayed in Figure 1.5.

The Logic of Computer Arithmetic Ivan Flores 1963

Buying Logic

- Even prior to the late 1950's you couldn't really just happily design "logic" without intimate knowledge of the underlying circuits.
- Eventually startups started releasing logic you could buy and stick together.



506 EAST FIRST STREET
SANTA ANA, CALIFORNIA

TELEPHONE
Kimberley 7-5651

ENGINEERED ELECTRONICS Company

RUGGEDIZED SERIES PRICE LIST
Effective Date: February 1, 1960

Terms, FOB point, and footnotes, e.g., (1), (23), etc., will be found at the end of the price lists.

PART NUMBERS	DESCRIPTION (3)	1-9	10-24	25-49	50-99	100-199	200-499
Z-92000/8336 or Z-92001 *	Flip-Flop	14.55 Same	14.10	13.25	12.35	10.95	10.70
Z-92002/8339 or Z-92003 *	Flip-Flop	14.75 Same	14.30	13.40	12.50	11.10	10.80
Z-92004/8342 or Z-92005 *	Flip-Flop	14.80 Same	14.35	13.45	12.55	11.15	10.85
Z-92007/90052*	Flip-flop	14.05	13.50	13.05	12.55	12.00	11.50
Z-92008/90059 or Z-92009 *	Flip-Flop	14.75 Same	14.30	13.40	12.50	11.10	10.80
Z-92010/90166 *	Flip-Flop	14.95	14.45	13.65	12.65	11.20	10.90
Z-92011/90015 *	Blocking Oscillator	21.70	21.00	20.45	19.75	18.90	18.10
Z-92012/8318 *	One Shot	12.90	12.50	11.75	10.95	9.80	9.50
Z-92013/8889 *	One Shot	10.90	10.60	10.00	9.35	8.35	8.15
Z-92018/90002 or Z-92019 *	Gate Circuit	13.60 Same	13.20	12.40	11.55	10.30	10.05
Z-92022/90023 *	Pentagrid Gate	11.25	10.80	10.45	10.10	9.60	9.25
Z-92023/8327 *	Pulse Gate	10.90	10.45	10.15	9.80	9.35	8.95
Z-92024/90001 or Z-92025 *	Squaring Circuit	13.15 Same	12.75	12.00	11.20	9.95	9.70
Z-92026/90031 *	Squaring Circuit	13.65	13.15	11.75	11.35	10.95	10.40

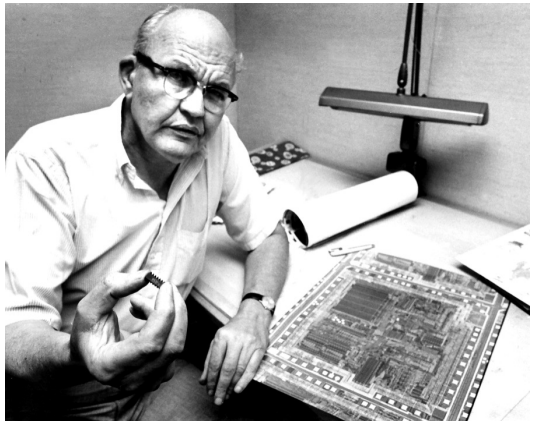
$11.25 for one AND gate in 1960...about $130.00 in 2026 dollars

Transistors and Integrated Circuits

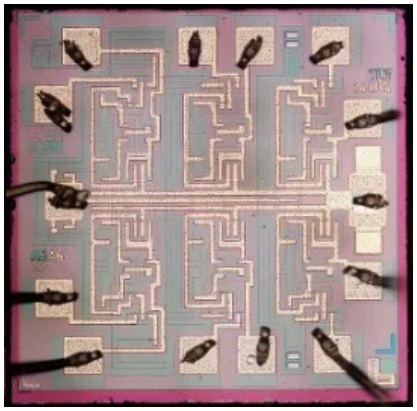
- Transistors on their own didn’t scale stuff down too much.
- It was the ability to merge them into small complicated circuits using lithographic techniques that led to integrated circuits.

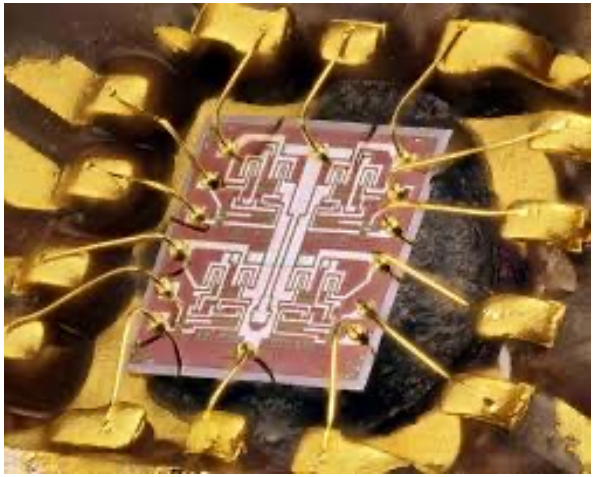
Mid 1960s

- Texas Instruments got integrated circuits fabrication to a strong enough point that they could release devices that had multiple logic gates all in one package!



Jack Kilby
TI engineer…one of inventors of Integrated Circuits





7400-series

AND logic: 7408
4 AND logic gate inside

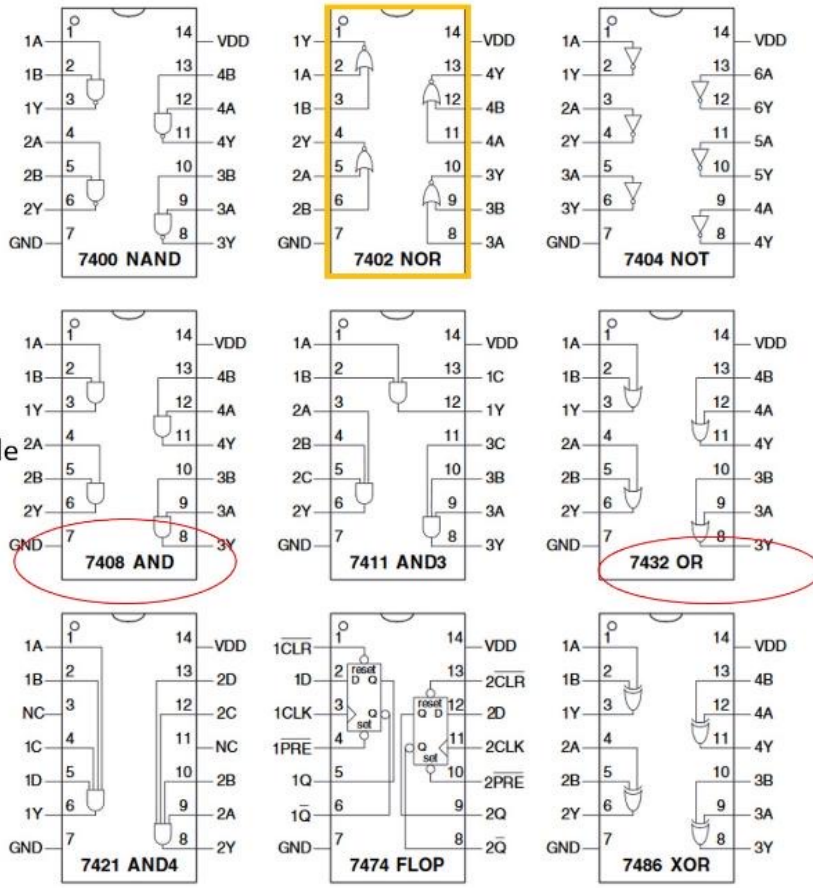
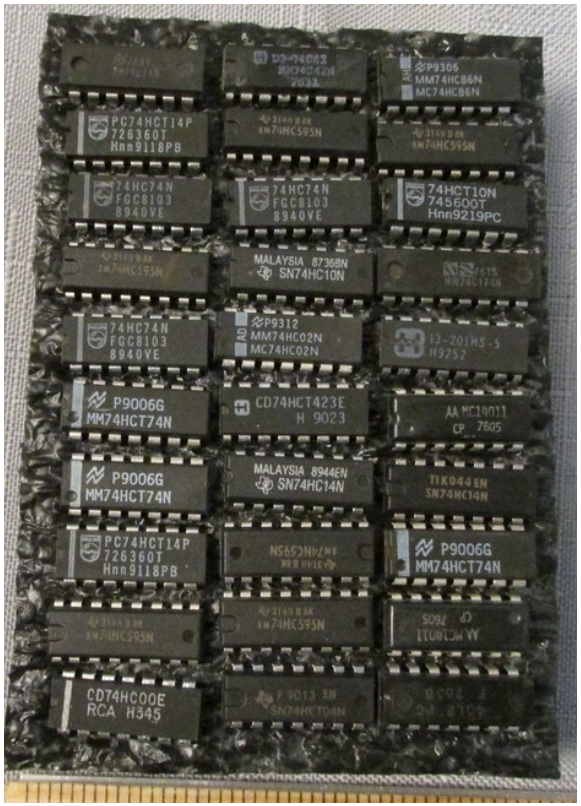


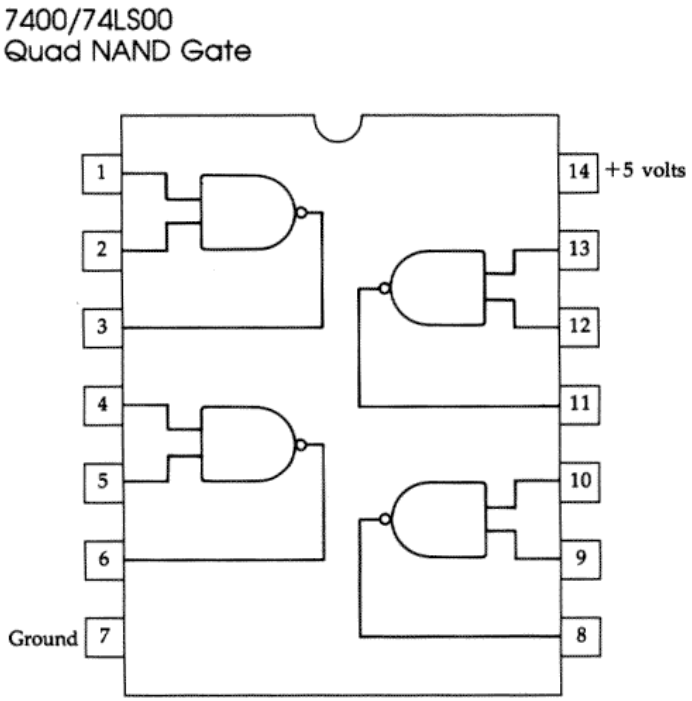
Figure eA.1 Common 74xx-series logic gates



Big Cost Savings (relatively)

- The 7400 cost $6.65 in 1966 FOR FOUR NAND GATES…that's about $66.50 in 2026 dollars…soooo cheap (actually! This was a big deal)

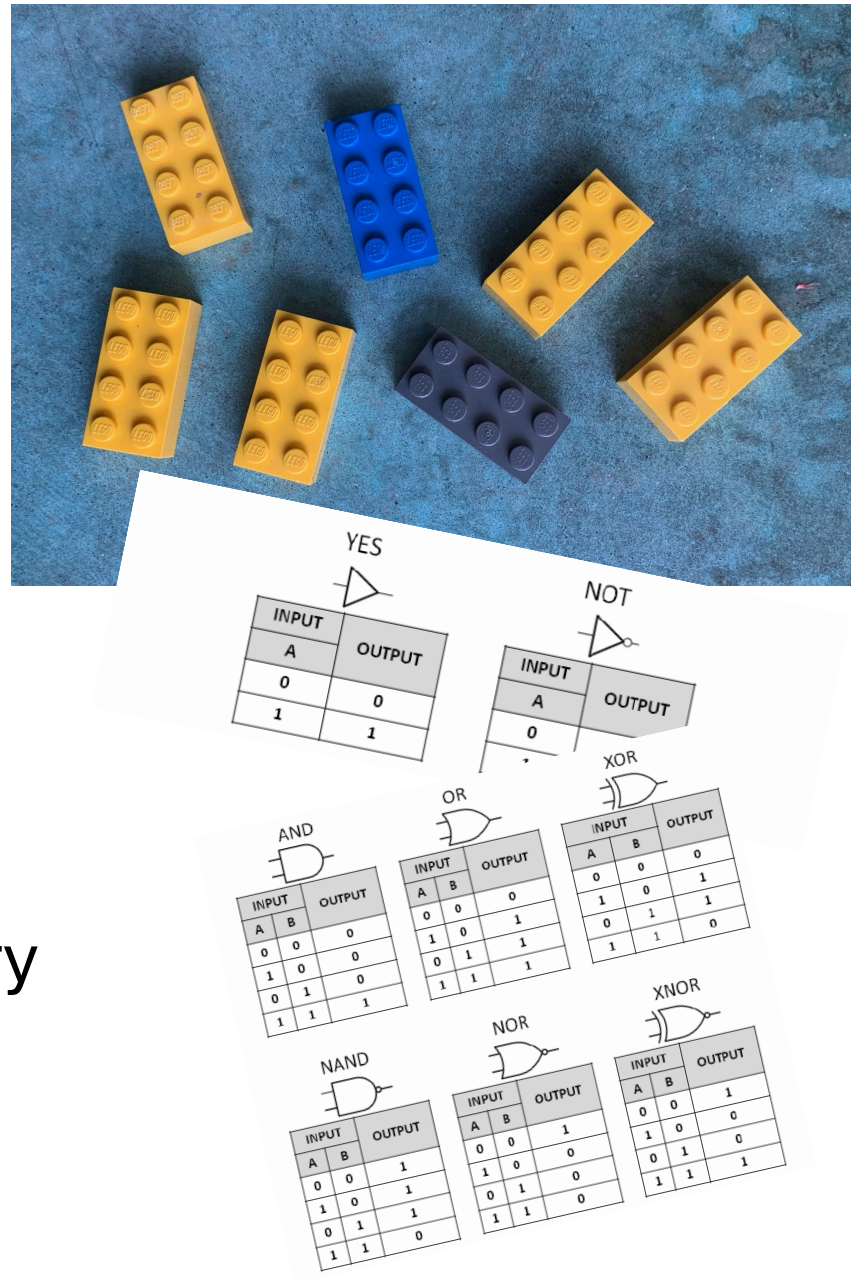
In 2026, an AMD Ryzen 5 5600G has the equivalent of about 2.5 billion NAND gates in it…and I saw them for like 80 bucks in December



So in 1966…you'd get 0.06 NAND gates per dollar
In 2026, you get 31.25 million NAND gates per dollar

So what did this mean?

- Design was much more accessible.
- Of course there was tons of gatekeeping.
- Older engineers would make younger engineers feel like sh*t because they weren't having to design logic gates from scratch and had to worry less about voltage.
- Tale as old as time. Bunch of boomers



Anyways….Let’s design

- Let’s design circuits as you would have in the mid 1960s and on…

Boolean Starters

- Values are either 0 or 1.
- Can have variables like a or b (keeping in mind they can only represent 0's and 1's)
- Three *core* operations…
- **NOT** (logical inversion):
  - "NOT a" is $\bar{a}$
- **OR** (sometimes called Boolean sum):
  - "a OR b" is $a + b$
- **AND** (sometimes called Boolean product):
  - "a AND b" is $a \cdot b$ but can also implicitly: ab or $a(b)$

NOT

INPUT	OUTPUT
A	
0	1
1	0

OR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

AND

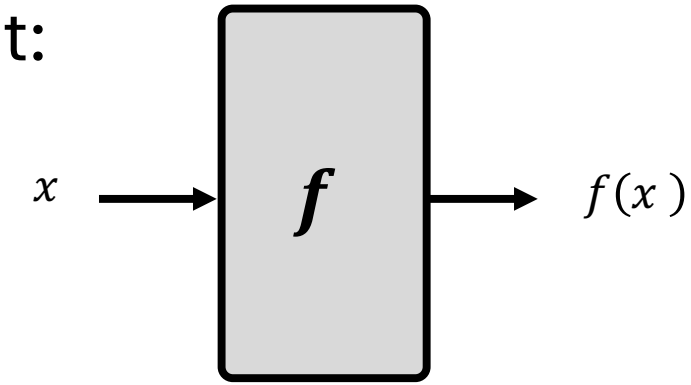
INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

Boolean Identities, Rules, Laws, Etc…

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

The Simplest Digital Function Class

- One Bit Input:



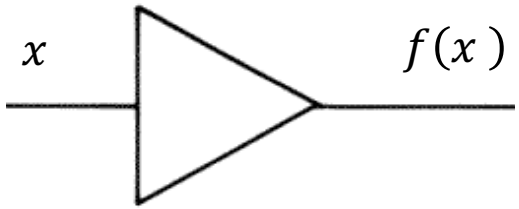
- How many possible 1-bit functions exist?

1-bit functions (input is a single value):

- How many possible 1-bit functions exist?
- Two (actually 4)…

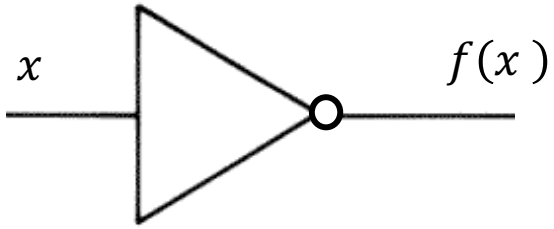
Buffer (Yes) gate:

x	$f(x)$
0	0
1	1



Inverter (Not) gate:

x	$f(x)$
0	1
1	0



Always On gate:

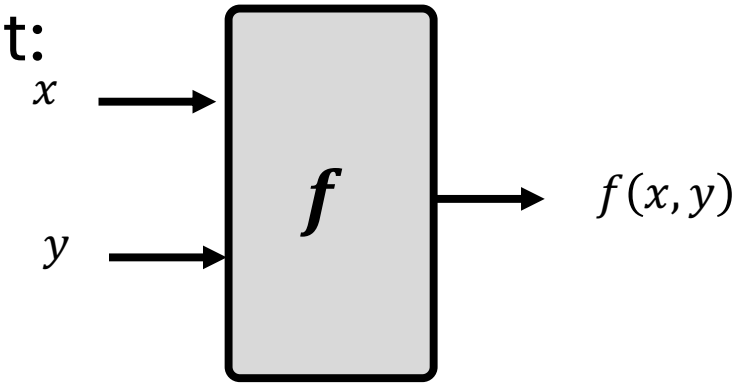
x	$f(x)$
0	1
1	1

Always Off gate:

x	$f(x)$
0	0
1	0

What About Two bits input?

- Two Bit Input:



- How many possible 2-bit functions exist?

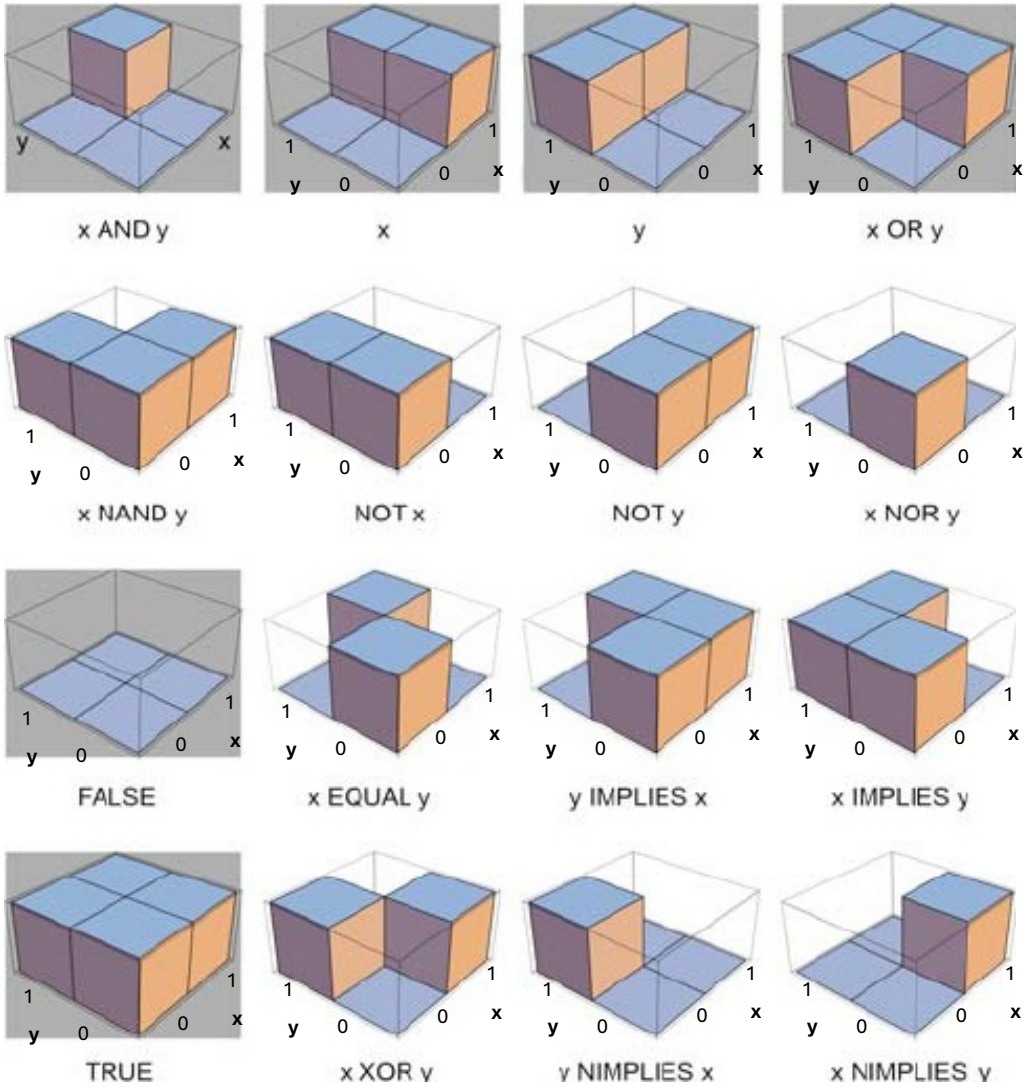
2-bit functions:

$f(x, y)$

x	y	$f(x, y)$
0	0	$f(0,0)$
0	1	$f(0,1)$
1	0	$f(1,0)$
1	1	$f(1,1)$

$2^4 = 16$ possible functions exist

Stated another way: there are 16 unique 1-0 combinations for:
$f(0,0)$, $f(0,1)$, $f(1,0)$, and $f(1,1)$



Mayo, Avi & Setty, Yaki & Shavit, Seagull & Zaslaver, Alon & Alon, Uri. (2006). Plasticity of the cis-Regulatory Input Function of a Gene. PLoS biology. 4. e45. 10.1371/journal.pbio.0040045.

Simple Truth Tables

- For a single-input system, there are four possible mappings (two non-negligible)
- For a two input system, you have 4 input combinations and 16 possible truth tables
- There is a lot of complexity that these give us

YES

INPUT A	OUTPUT
0	0
1	1

NOT

INPUT A	OUTPUT
0	1
1	0

AND

INPUT A	INPUT B	OUTPUT
0	0	0
1	0	0
0	1	0
1	1	1

OR

INPUT A	INPUT B	OUTPUT
0	0	0
1	0	1
0	1	1
1	1	1

XOR

INPUT A	INPUT B	OUTPUT
0	0	0
1	0	1
0	1	1
1	1	0

NAND

INPUT A	INPUT B	OUTPUT
0	0	1
1	0	1
0	1	1
1	1	0

NOR

INPUT A	INPUT B	OUTPUT
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

INPUT A	INPUT B	OUTPUT
0	0	1
1	0	0
0	1	0
1	1	1

Abels and Khisamutdinov, 2015,
https://www.researchgate.net/publication/291418819_Nucleic_Acid_Computing_and_its_Potential_to_Transform_Silicon-Based_Technology

Logical Reduction

- All high level operations we may want can be reduced down to combinations of these simpler logical operations
- We just need to start to see how.
- Don’t just think of the “AND” gate as ”AND” in the quasi-grammar sense of the term. A lot of things we’d want to do when writing high-level logic/programs rely on it, even if we don’t name it that explicitly.
- Same with “OR” or “XOR”

Consider just one of these truth tables “XOR”

- If 0 and 1 are numbers, XOR performs base 2 addition:
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=0 (carry 1)

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

- Or, if 0 means positive and 1 means negative, XOR performs sign determination of multiplication:
  - 0×0 = 0 (positive×positive = positive)
  - 0×1 = 1 (positive×negative = negative)
  - 1×0 = 1 (negative×positive = negative)
  - 1×1 = 0 (negative×negative = positive)

Or still thinking about ways of using XOR

- XOR expresses the if/else check:

```
if(A==1):
    output = !B
else:
    output = B
```

- XOR it does the check: A!=B
- XOR does others
- All high-level algorithmic needs find their basic implementation in these fundamental functions

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

But this is backwards…

- We usually have a thing we want to build and we need to figure out how to make it.

- We do not generally start with some random logic circuit and assign meaning to it like a piece of literature.

Truth Tables and Sum of Product Expressions

- The most purest, truest, guaranteed way to represent digital functions is by either writing out their Truth Table or their Sum of Products (SOP)

INPUT		Y
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

$$Y = A \cdot B$$

SOP is a OR-ing (”sum”) of every non-zero row (product) in the truth table

Truth Tables and Sum of Product Expressions

- The most purest, truest, guaranteed way to represent digital functions is by either writing out their Truth Table or their Sum of Products (SOP)

INPUT		Y
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

$$Y = A \cdot \bar{B} + \bar{A} \cdot B$$

SOP is a OR-ing (”sum”) of every non-zero row (product) in the truth table

As an engineer you'd generally start with…

- Some sort of user-specified truth table.
- You could always use the resulting SOP as a recipe of what to build…
- BUT the SOP is very often NOT in the most simplified form.
- So your job would be to use the Boolean laws to reduce your equations (and therefore designs) down to the minimal number of gates to save costs (financial, time, emotional, etc…)

So here's a truth table given to you by your boss

Build this:

INPUT		Y
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

Start with SOP and let's use Boolean Laws to get to the purest circuit form!!!

More Complicated Circuit

Build a circuit with three inputs a, b, c and one output y.

y should be high if a is on or if b is on with c off or if c is on or if b is off with a on.

Boolean Algebra is Tricky

- **Karnaugh maps** can help us out here!
- A higher-dimensional representation of truth tables which can be used to graphically simplify Boolean Expressions

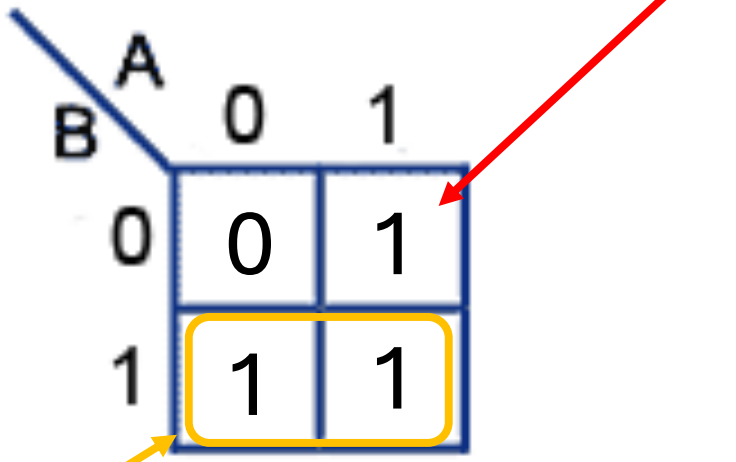
Karnaugh Map

- Instead of doing truth table like this:

INPUT		Y
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

- Do like this:

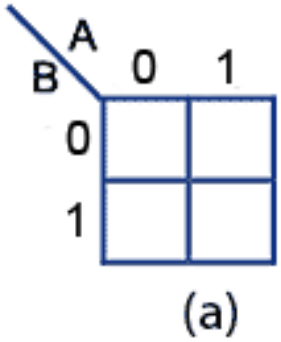
Single box is full-term product $a\bar{b}$



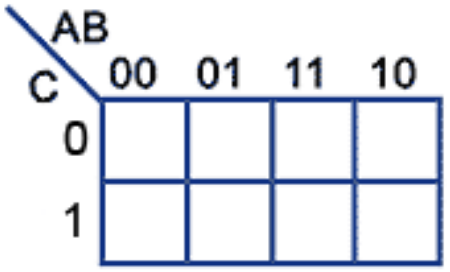
Larger continuous power-of-two rectangles are simplified terms… In this case this is b

For more inputs, can have larger K-maps

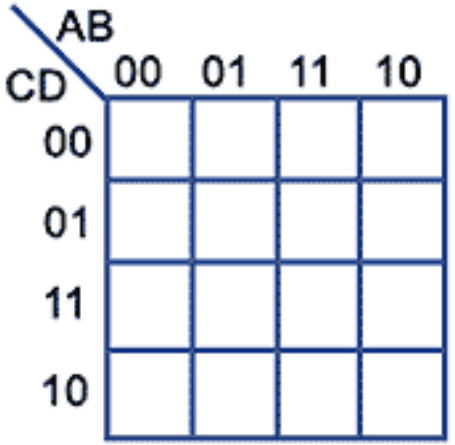
- Input sequences are broken up and listed out in Grey-code count
- Same idea. Circle the largest continuous power-of-two-rectangles until all 1's are covered on the board…that's your final SOP



(a)



(b)



(c)

OK…Complicating the More Complicated Circuit

Good job on designing the previous circuit. You get a bonus of 1600 dollars.*

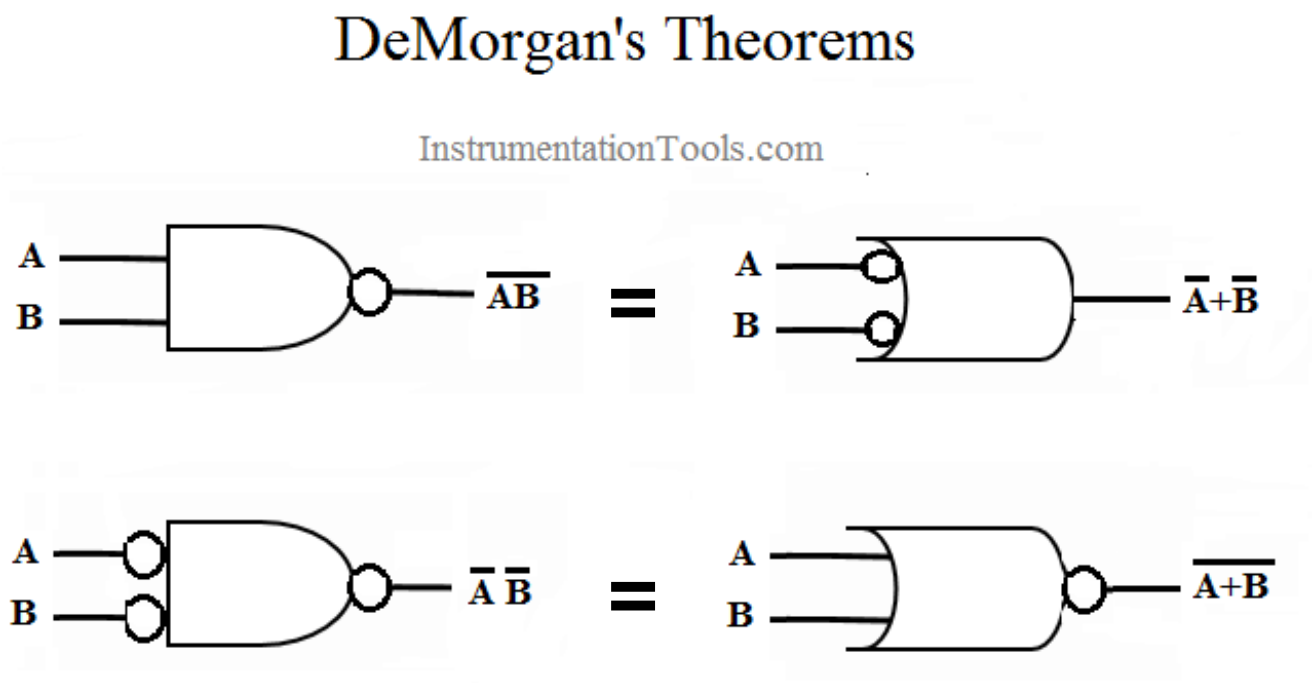
Unfortunately we are not splurging on three-input OR gates. All we have are 7400 chips which we buy in bulk. Build the circuit using just those.

You do that, we'll make your bonus 2400 dollars.**

*enough to buy a brand new Volkswagen Beetle in 1966

**enough to buy a new base-model Ford Mustang in 1966.

Remember De Morgan!!



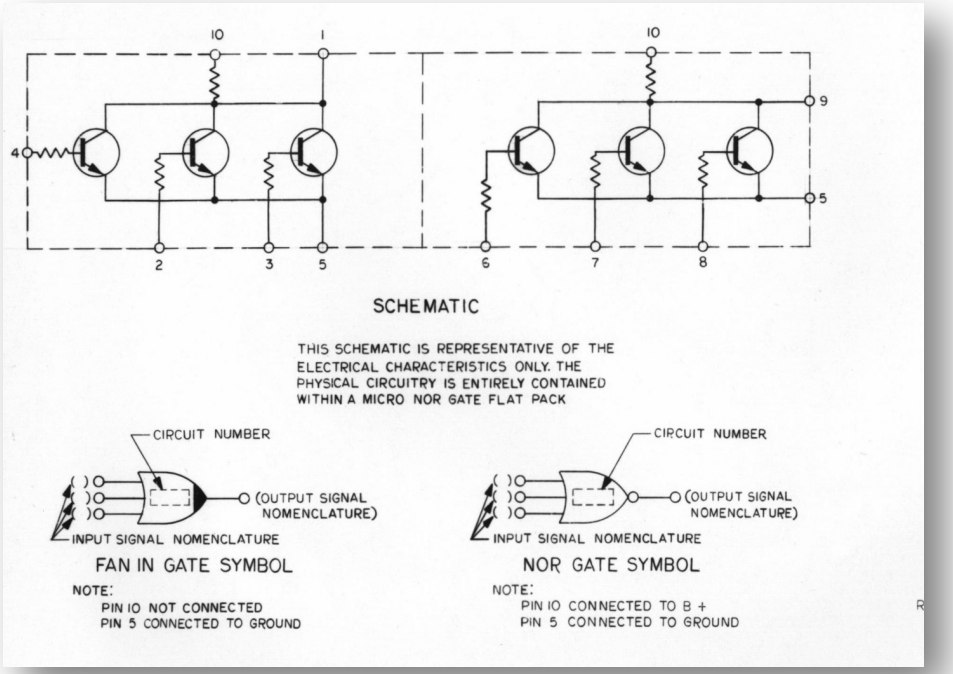
Gee that seems silly

- Yeah, but early on,
- In very very mission-critical things it was hard to rigorously quality-control lots of different specialized chips.
- The most mission-critical thing of all in the 1960s was the Apollo program

AGC

- The Apollo Guidance Computer was comprised only of three-input NOR gates

The tri-NOR-gate “definition”:

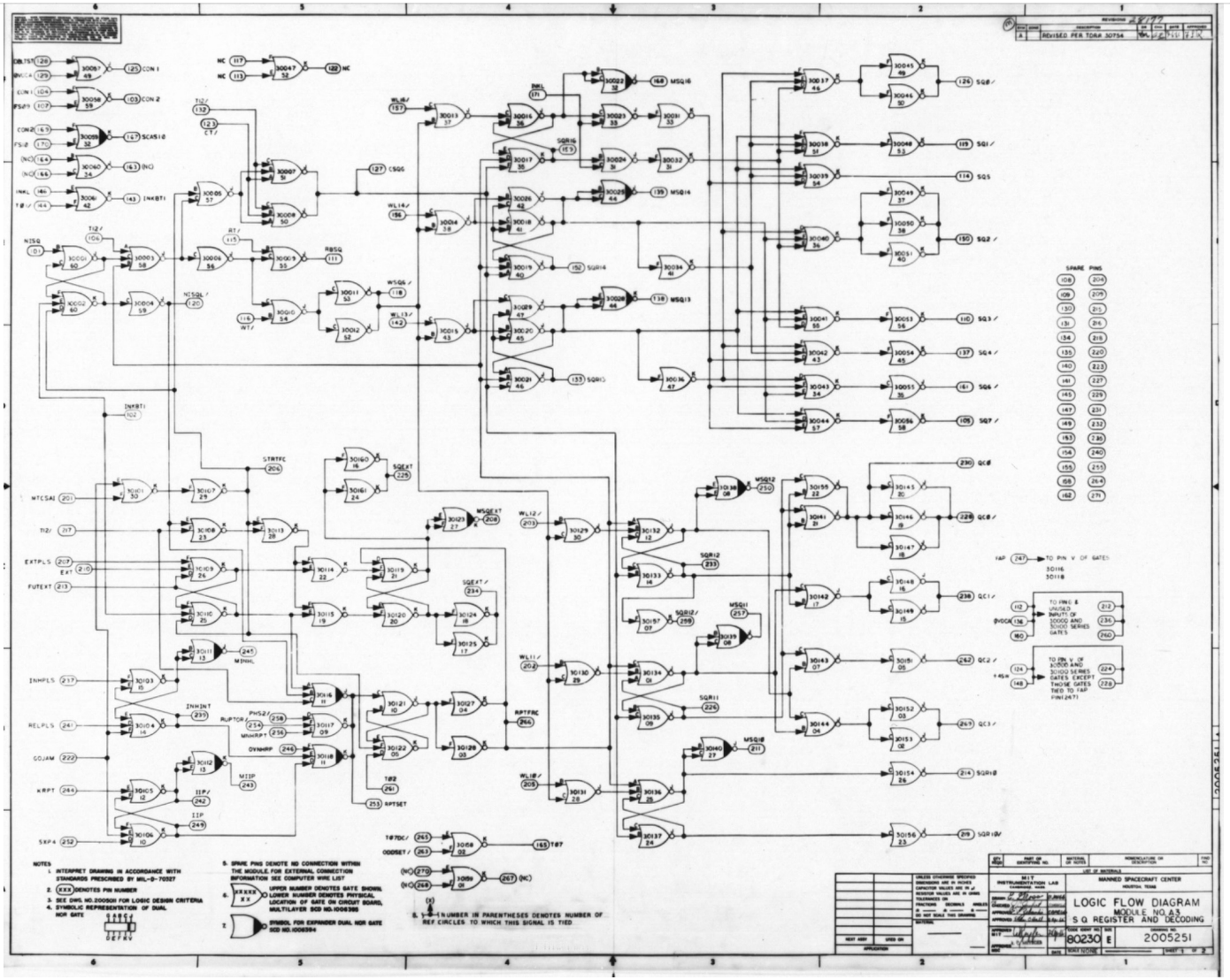


https://klabs.org/history/ech/agc_schematics

AGC

Everything else is design ONLY using those NOR gates

Many pages of schematics



https://klabs.org/history/ech/agc_schematics

Binary Numbers

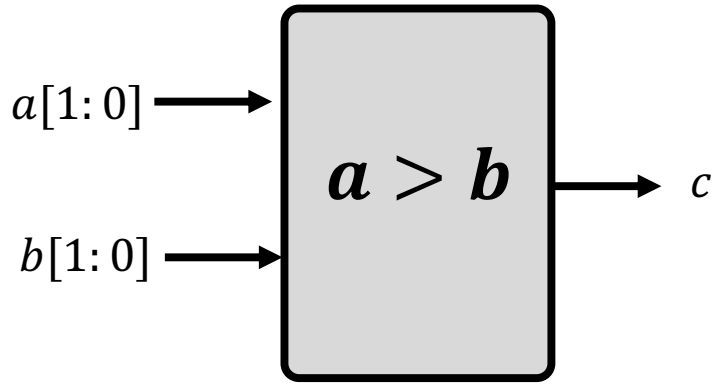
- Each signal is 1 or 0…we can have multiple signals in parallel to represent combined higher level concepts.

- One of those can be numbers.

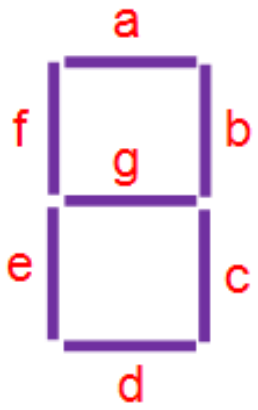
Let's do the following

- "a" is a two bit number and "b" is a two bit number

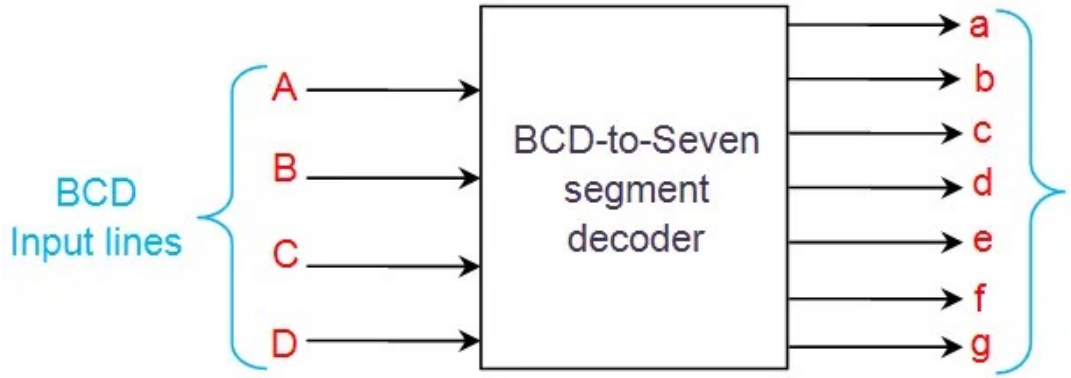
- I want a circuit like the following:



Binary to Seven-Segment



- Four bits of binary…
- To sixteen symbols…(hexadecimal)
- I need to build this…



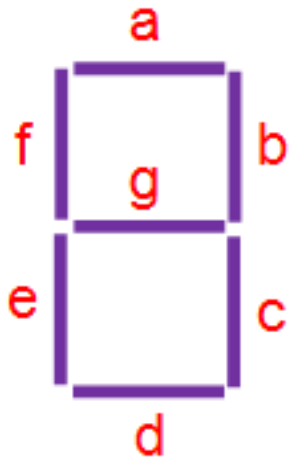
https://blog.tindie.com/2022/03/chainable-seven-segment-display-module/

https://electronics.stackexchange.com/questions/351606/7-segment-binary-to-hex

https://www.electrical4u.com/bcd-to-seven-segment-decoder/

Step 1…

- Derive Truth Table



Inputs				Segments							
A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	For display 0
0	0	0	1	0	1	1	0	0	0	0	For display 1
0	0	1	0	1	1	0	1	1	0	1	For display 2
0	0	1	1	1	1	1	1	0	0	1	For display 3
0	1	0	0	0	1	1	0	0	1	1	For display 4
0	1	0	1	1	0	1	1	0	1	1	For display 5
0	1	1	0	1	0	1	1	1	1	1	For display 6
0	1	1	1	1	1	1	0	0	0	0	For display 7
1	0	0	0	1	1	1	1	1	1	1	For display 8
1	0	0	1	1	1	1	1	0	1	1	For display 9
1	0	1	0	1	1	1	0	1	1	1	For display A
1	0	1	1	0	0	1	1	1	1	1	For display b
1	1	0	0	1	0	0	1	1	1	0	For display C
1	1	0	1	0	1	1	1	1	0	1	For display d
1	1	1	0	1	0	0	1	1	1	1	For display E
1	1	1	1	1	0	0	0	1	1	1	For display F

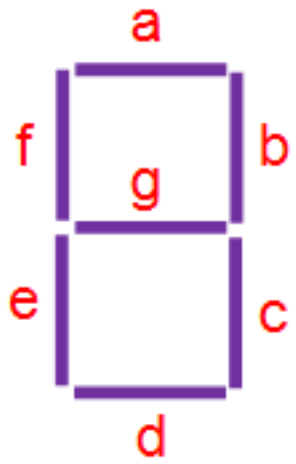
https://www.electrical4u.com/bcd-to-seven-segment-decoder/

https://www.quora.com/Can-we-show-hexa-decimal-using-seven-segment-display

Let's do one of those segments…

- Segment a

Inputs				Segments							
A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	For display 0
0	0	0	1	0	1	1	0	0	0	0	For display 1
0	0	1	0	1	1	0	1	1	0	1	For display 2
0	0	1	1	1	1	1	1	0	0	1	For display 3
0	1	0	0	0	1	1	0	0	1	1	For display 4
0	1	0	1	1	0	1	1	0	1	1	For display 5
0	1	1	0	1	0	1	1	1	1	1	For display 6
0	1	1	1	1	1	1	0	0	0	0	For display 7
1	0	0	0	1	1	1	1	1	1	1	For display 8
1	0	0	1	1	1	1	1	0	1	1	For display 9
1	0	1	0	1	1	1	0	1	1	1	For display A
1	0	1	1	0	0	1	1	1	1	1	For display b
1	1	0	0	1	0	0	1	1	1	0	For display C
1	1	0	1	0	1	1	1	1	0	1	For display d
1	1	1	0	1	0	0	1	1	1	1	For display E
1	1	1	1	1	0	0	0	1	1	1	For display F

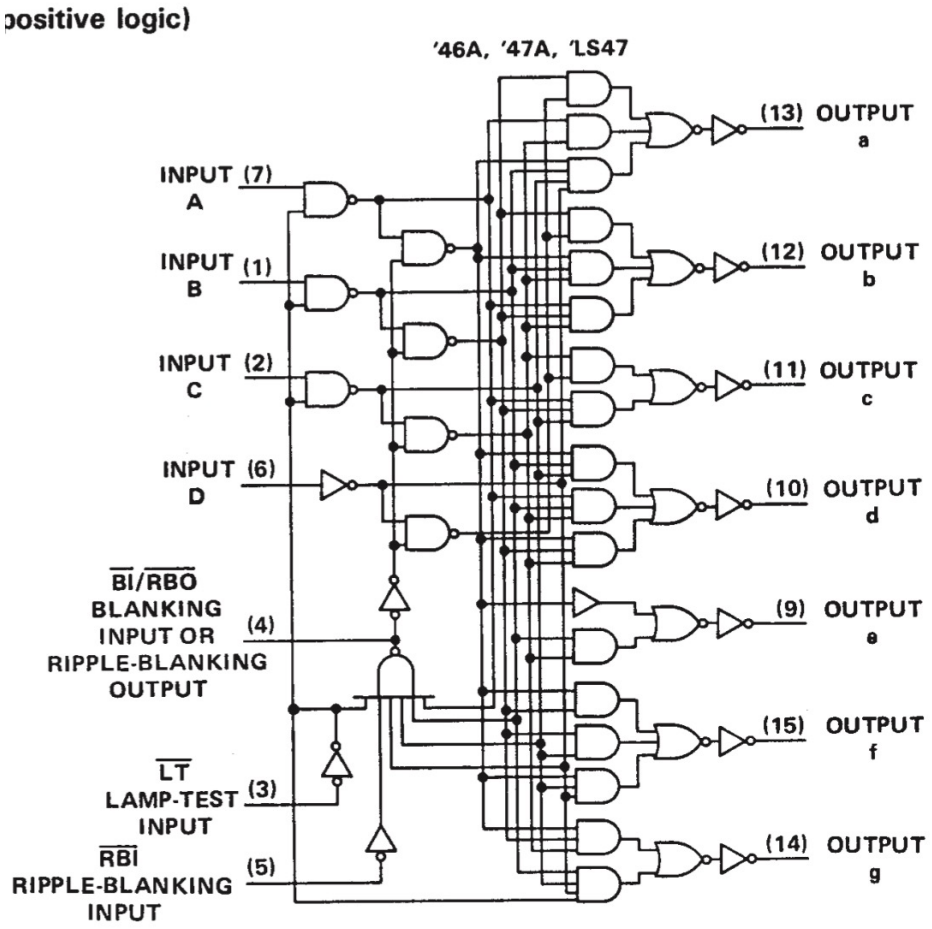


The 7447

- Only does 0-9
- NOT 0-F

- Pack all this in a chip and make money

- TI was selling these nasty things for about 12 bucks a piece by late 1960s.



Another Example (Multiplexer)