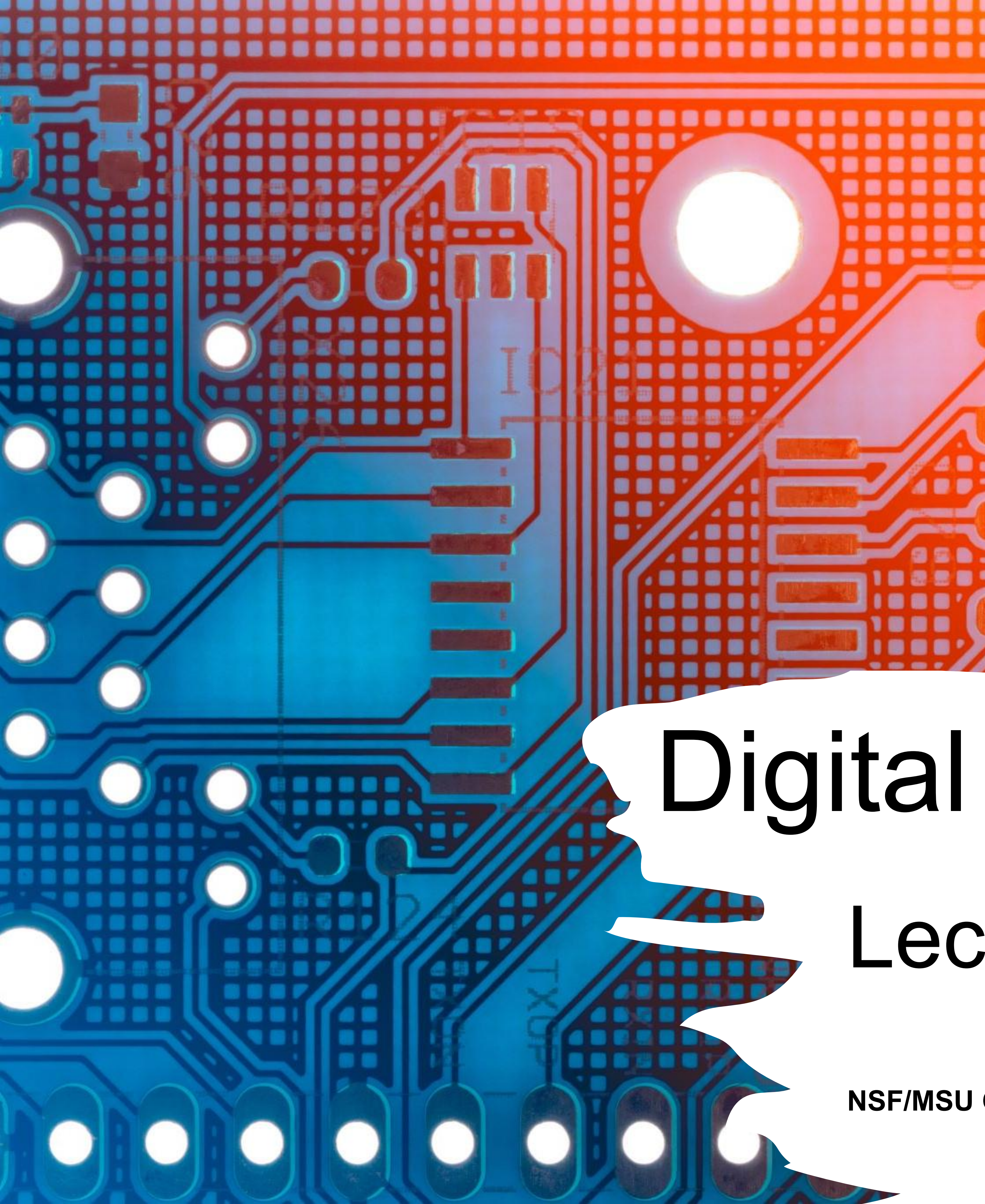


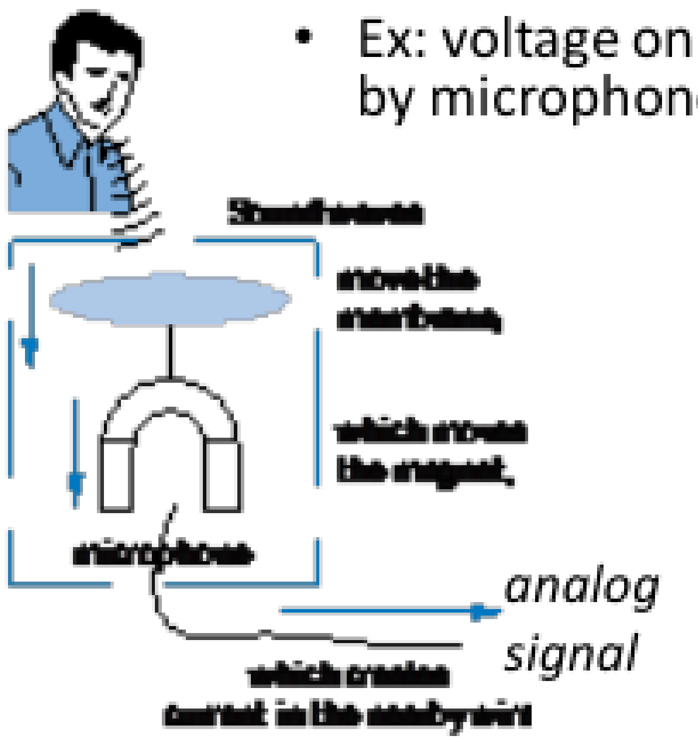
Digital Electronics

Lecture 1

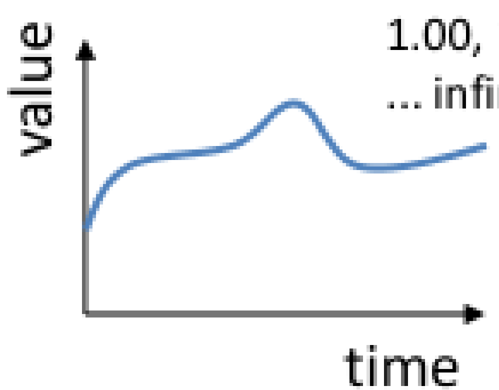
NSF/MSU ConstructionCI Cyber Training Program

What Does “Digital” Mean?

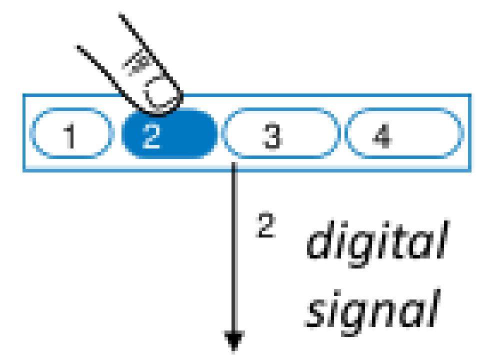
- Analog signal
  - Infinite possible values
    - Ex: voltage on a wire created by microphone



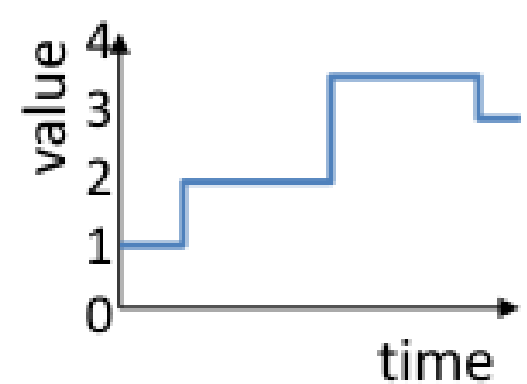
Possible values:
1.00, 1.01, 2.0000009,
... infinite possibilities



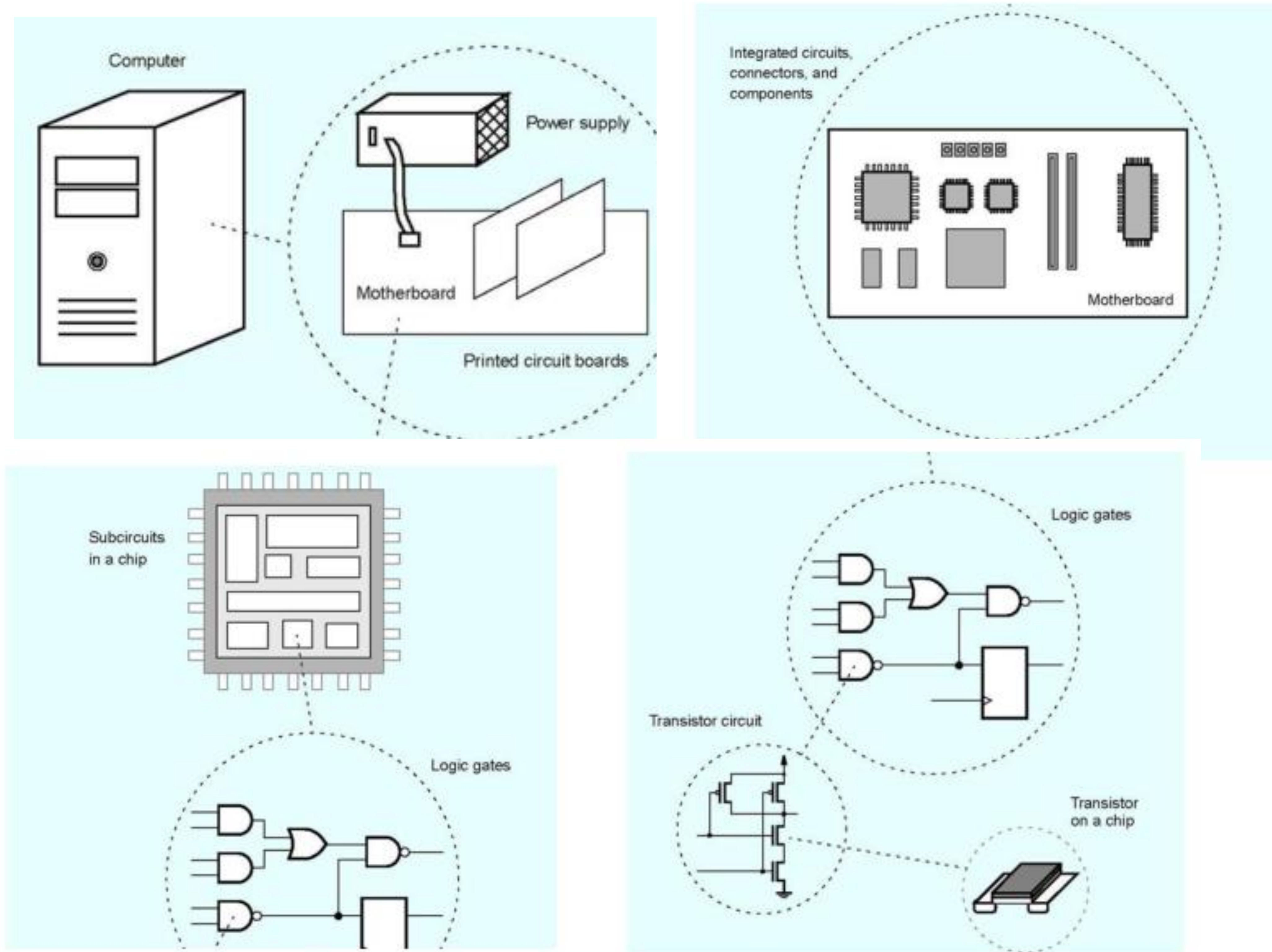
- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad



Possible values:
0, 1, 2, 3, or 4.
That’s it.



What’s Inside a Computer?



Binary Representation

- The basis of all digital data is **binary** representation.
- Binary means “**two**”. Bit = **binary digit**.
  A. 0 or 1
  B. True or False
  C. On or Off
- A parameter in the real world may need **more than** two values to represent.
  A. π (3.1415926….)
  B. Natural logarithm: e (2.71828…)
  C. (1.414…)
  D. Hot, Cold, Warm ….
  E. Red, Blue, Green, Yellow ….
- How to represent analog data? —> Number system

Number Systems

- **Decimal number:** A digit ranges from 0 to 9.

- **Binary value:** A digit ranges from 0 to 1. A digit in binary is also called '**bit**'. Use **0b** to indicate a value is binary: 0b100101

- **Hexadecimal:** A digital ranges from 0 to 16.

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

Two digitals for a value: e.g. 1 and 2 for 12

Use **letters A to F** to represent 10 to 15

Hex: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, **A, B, C, D, E, F**

- Use **0x** to indicate a value is Hex: 0x7AF35

Significant Digit

$$53 = 0b\ \boxed{1}\ 1\ 0\ 1\ 0\ \boxed{1}$$

Most Significant Digit

For binary, also called Most Significant Bit (MSB)

Least Significant Digit

For binary, also called Least Significant Bit (LSB)

Why call a digit as MSB or LSB ?

- MSB has the most impact on a value. For example: change MSB of 0b110101 from 1 to 0, the result is 0b010101 = 21 —> Completely different from original value (53)

- LSB has the least impact on a value. For example: change LSB of 0b110101 from 1 to 0, the result is 0b110100 = 52 —> Close to original value (53)

Hex to Binary Conversion

0x0 = 0b 0000
0x1 = 0b 0001
0x2 = 0b 0010
0x3 = 0b 0011
0x4 = 0b 0100
0x5 = 0b 0101
0x6 = 0b 0110

0x7 = 0b 0111
0x8 = 0b 1000
0x9 = 0b 1001
0xA = 0b 1010
0xB = 0b 1011
0xC = 0b 1100
0xD = 0b 1101

0xE = 0b 1110
0xF = 0b 1111

- To convert a Hex number to Binary, simply convert each Hex digit to its four bit value.

0x A 2 F

1010 0010 1111

0b 1010 0010 1111

Binary to Hex Conversion

0x0 = 0b 0000
0x1 = 0b 0001
0x2 = 0b 0010
0x3 = 0b 0011
0x4 = 0b 0100
0x5 = 0b 0101
0x6 = 0b 0110
0x7 = 0b 0111
0x8 = 0b 1000
0x9 = 0b 1001
0xA = 0b 1010
0xB = 0b 1011
0xC = 0b 1100
0xD = 0b 1101
0xE = 0b 1110
0xF = 0b 1111

- Create groups of 4 bits starting from **right to left.**
- If last group does not have 4 bits, then **pad with zeros** for unsigned numbers.

0b 101 0001

↓

0b **0**101 0001

0x 5 1

Range of Binary Values

- For 1-bit binary value, have 2 different combinations (0 and 1)
- For 2-bits binary value, there are 4 different combinations (00, 01, 10, and 11).
- For 3-bits binary value, there are $2^3 = 8$ different combinations (000, 001, 010, 011, 100, 101, 110, 111)

For N-bits binary value, how many combinations do we have?

↓

2^N

↓

What is the range an N-bits binary value can represent for unsigned integers?

↓

0 to $2^N - 1$

Binary Arithmetic and Subtraction

Addition	Subtraction
0 + 0 = 0, carry = 0	0 - 0 = 0, borrow = 0
1 + 0 = 1, carry = 0	1 - 0 = 1, borrow = 0
0 + 1 = 1, carry = 0	0 - 1 = 1, borrow = **1**
1 + 1 = 0, carry = 1	1 - 1 = 0, borrow = 0

Unsigned Overflow

- Overflow: when adding or subtracting two numbers, and the correct result is a number that is **outside** the range of allowable numbers.

- N-bits binary value can represent an unsigned integer from 0 to 2^N-1.

  A. 8 bits: unsigned 0 to $2^8 - 1$ —> **0 to 255**
  B. 16 bits: unsigned 0 to $2^{16} - 1$ —> **0 to 65535**
  C. 32 bits: unsigned 0 to $2^{32} - 1$ —> **0 to 4,294,967,295**

Unsigned Overflow Example

Assume we use 8 bits binary value to represent each number

Let's calculate 255 + 1

255	=		0b 11111111
+ 1	=	+	0b 00000001
256			0b 00000000

Correct answer

Wrong answer (overflow)

Maximum unsigned integer that a 8-bits binary can represent is $2^8 - 1 = 255$

Codes for Characters

- N bits binary can represent **2^N different values**
- We can represent Characters as digital data
- The **ASCII** code (American Standard Code for Information Interchange) is a 7-bit code for Character data.
- 7 bits can represent 128 different values. This enough to represent Latin alphabet (A-Z, a-z, 0-9, and some symbols)
- In **ASCII** code, typically 8 bits (1 Byte) are actually used with the 8th bit being zero or used for error detection (parity checking).

A
S
C
II

ASCII characters 0 to 127

Code 0 to 31 (and # 127) are non-printing, mostly obsolete control characters that affect how text is processed. There are 95 printable characters.

To print one, press the ALT key (hold it down) and type the decimal number.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Signed Numbers – Signed Magnitude

Decimal	Sign Magnitude
+8	
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111
-8	

1. MSB is sign bit
2. Problem:

$$\begin{array}{r} 0110 = +6 \\ \underline{1110} = \underline{-6} \\ 10100 \neq 0 \end{array}$$

3. Problem:

$$\begin{array}{r} 0000 = +0 \\ 1000 = -0 \end{array}$$

Signed Numbers – 1s Complement

Decimal		1s Complement	
+8			
+7		0111	
+6		0110	
+5		0101	
+4		0100	
+3		0011	
+2		0010	
+1		0001	
+0		0000	
-0		1111	
-1		1110	
-2		1101	
-3		1100	
-4		1011	
-5		1010	
-6		1001	
-7		1000	
-8			

1. MSB is sign bit
2. Negative numbers are inverse of positive.
3. Problem: still have 2 zeros

Signed Numbers – 2s Complement

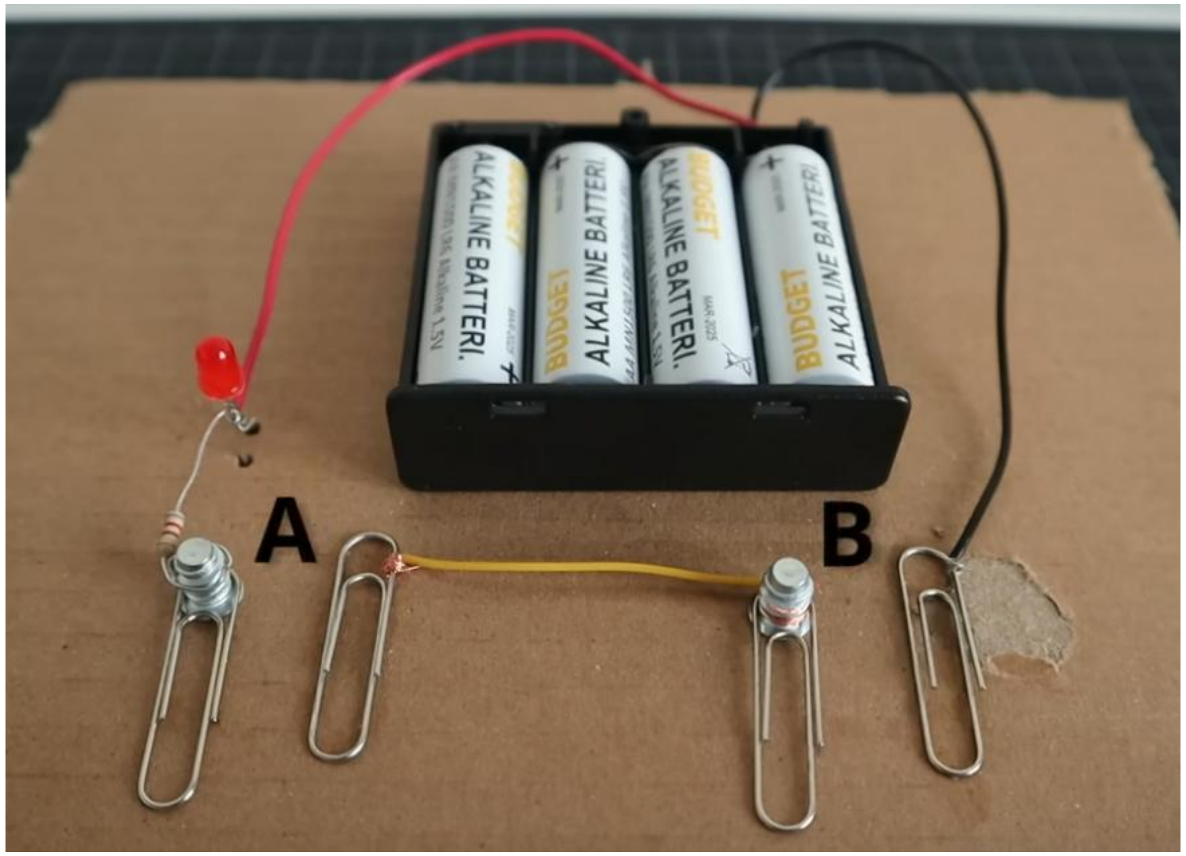
Decimal			2s Complement
+8			
+7			0111
+6			0110
+5			0101
+4			0100
+3			0011
+2			0010
+1			0001
+0			0000
-0			0000
-1			1111
-2			1110
-3			1101
-4			1100
-5			1011
-6			1010
-7			1001
-8			1000

1. Positive numbers look the same.
2. To flip sign:
   a. Flip all bits
   b. Add 1

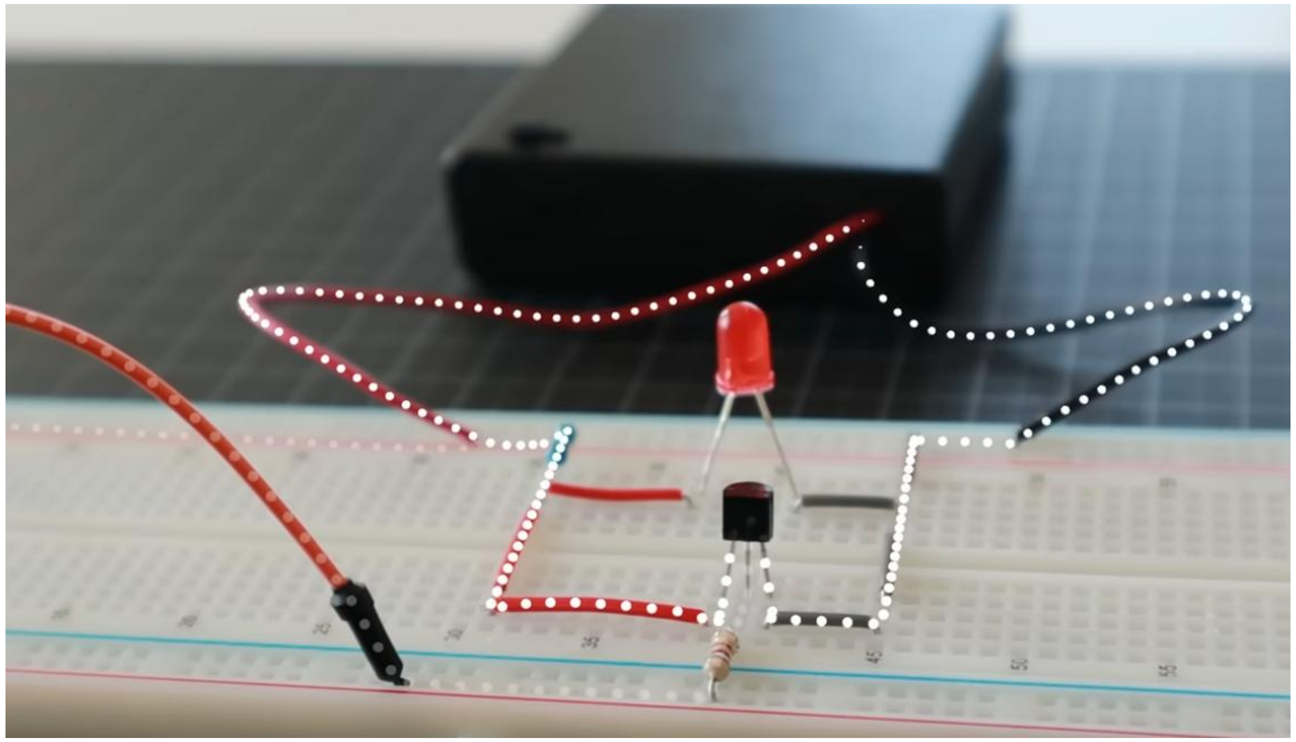
+5 = 0101
flipped: 1010
add 1: 1011 = -5

The most common method

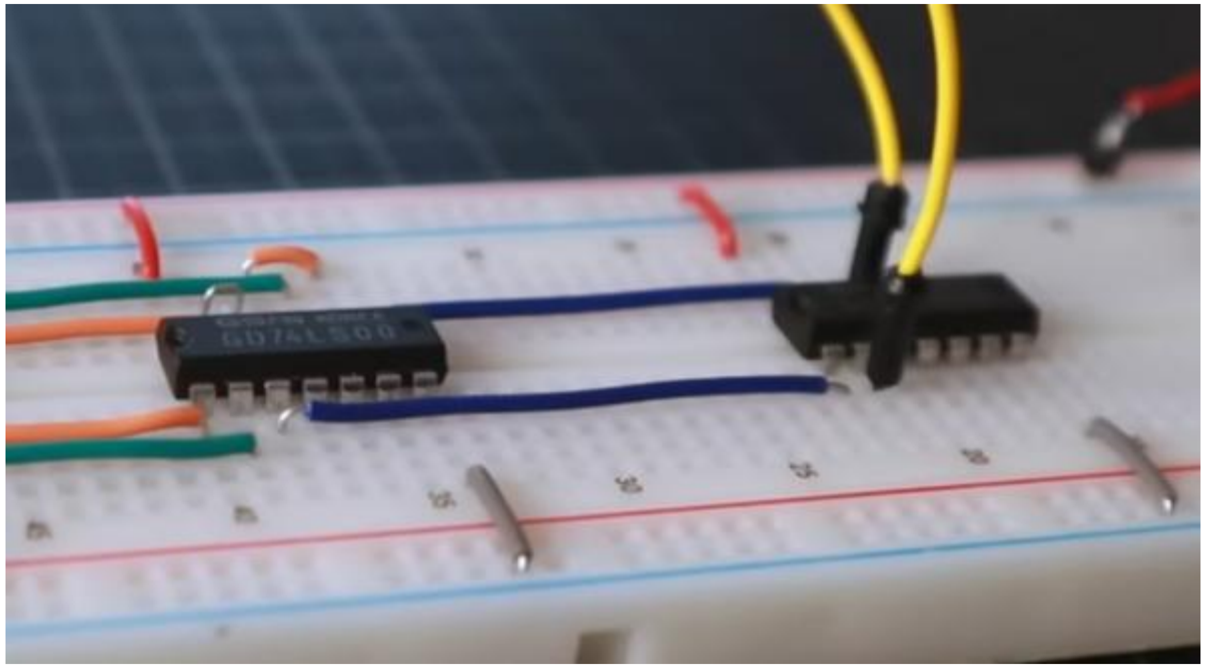
Logic Gate Examples



AND

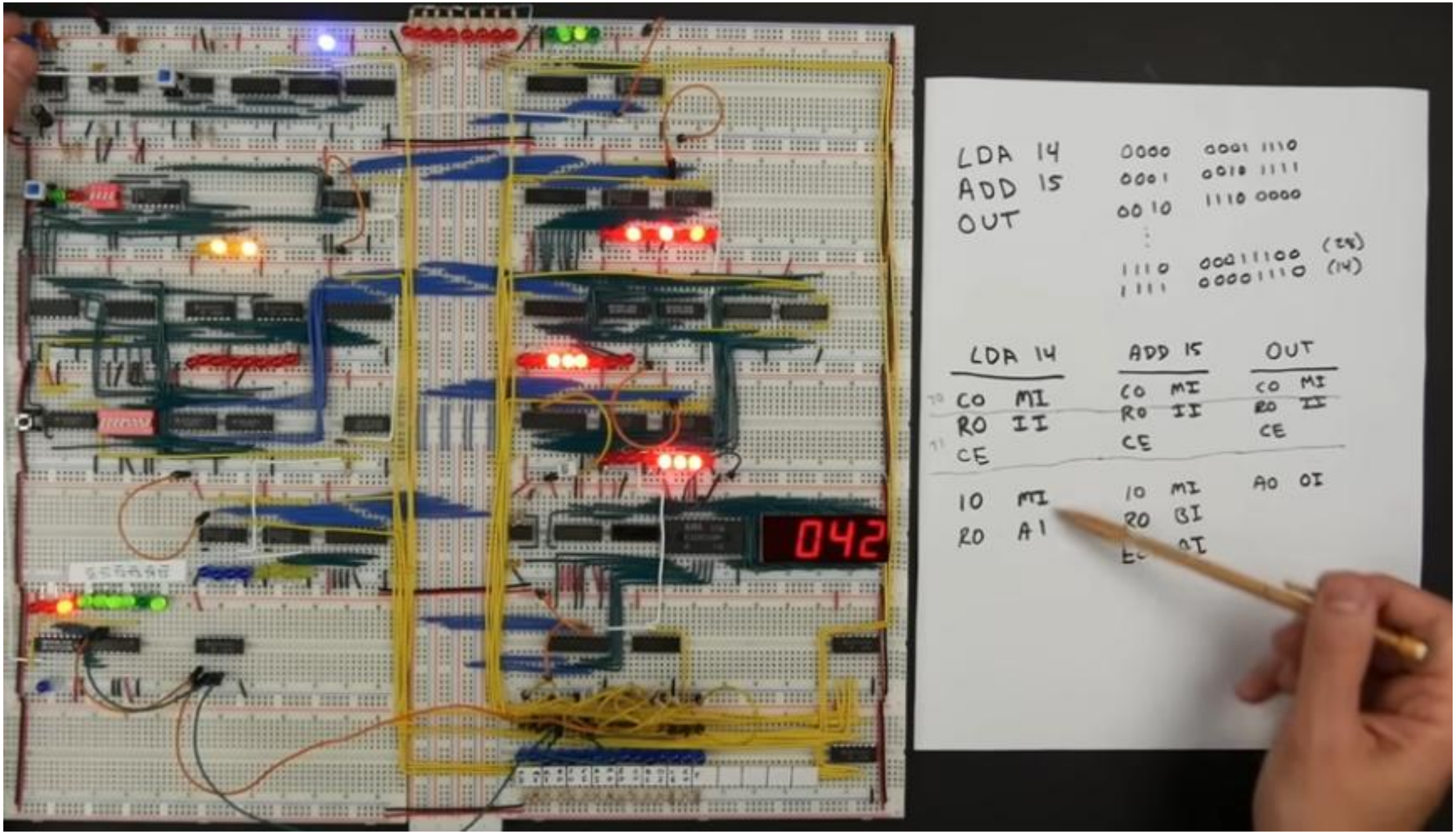


OR



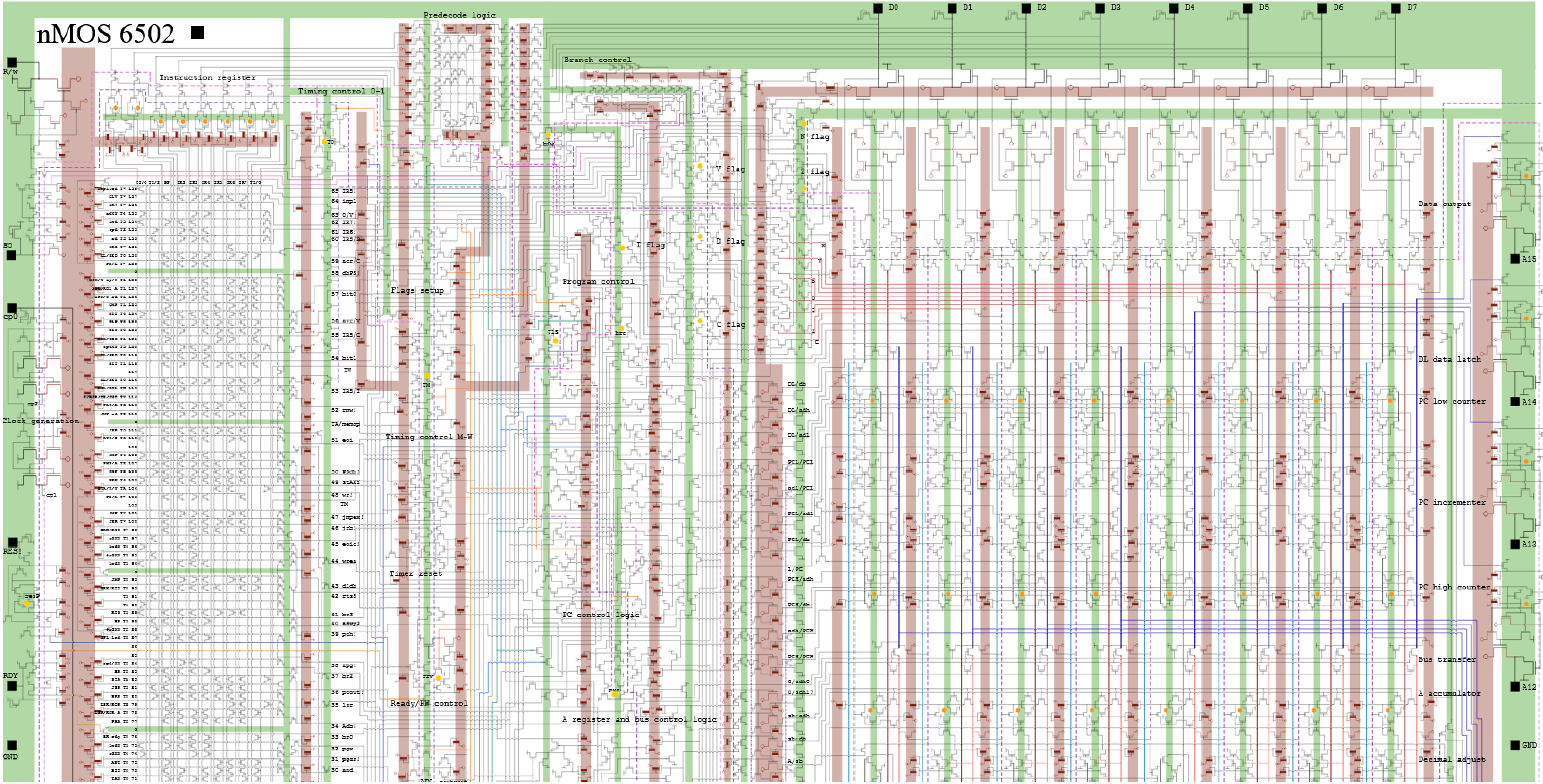
NANDs

“Simple” computer



Ben Eater’s 8-bit CPU

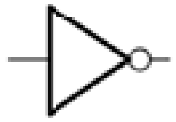
1975 computer



This is approximately half the circuitry for the 6502, the CPU in the Apple II (and many other late 70s computers).

Basic Logic Gates

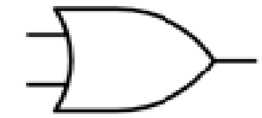
NOT



INPUT	OUTPUT
A	
0	1
1	0

Output is the reverse of input

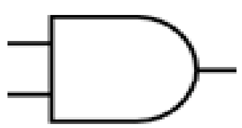
OR



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

Output = 1 if at least one of input = 1

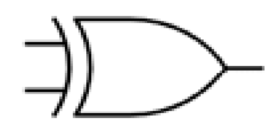
AND



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

Output = 1 if and only if two inputs are 1;
otherwise output = 0

XOR



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

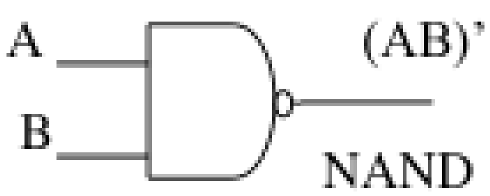
Output = 1 if two inputs are different;
otherwise output = 0

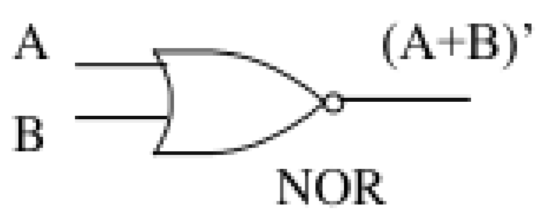
NAND and NOR

There are two other gate types that produce the complement of a boolean function!

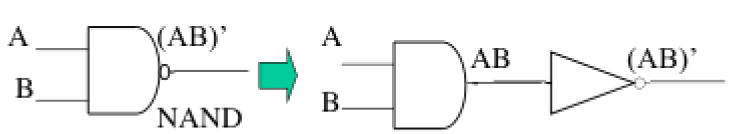
A B	Y
0 0	1
0 1	1
1 0	1
1 1	0

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

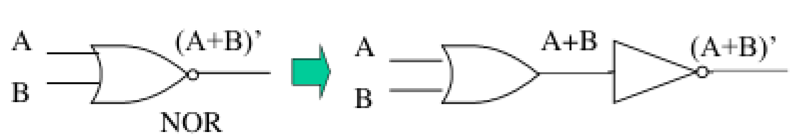




NAND (NOT AND) - can be thought of as an AND gate followed by an inverter.



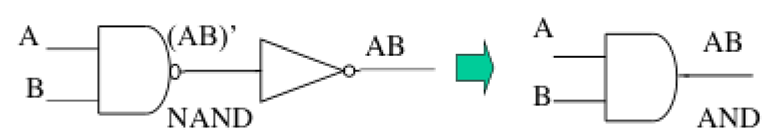
NOR (NOT OR) - can be thought as an OR gate followed by an inverter.

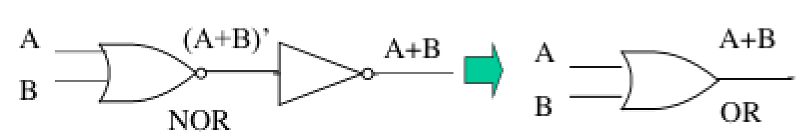


Actually….

In the real world, an AND gate is made from an NAND gate followed by an inverter!!!

An OR gate is made from a NOR gate followed by an inverter!!!

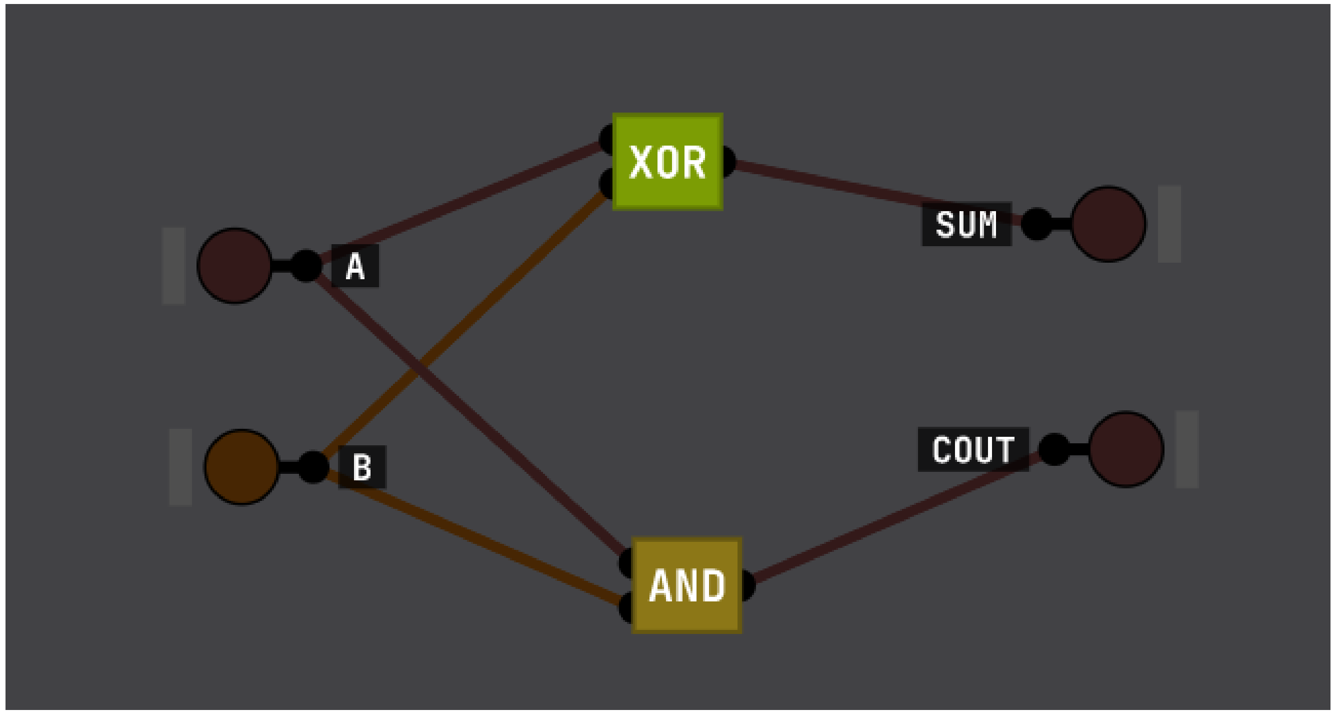




Building a Half Adder

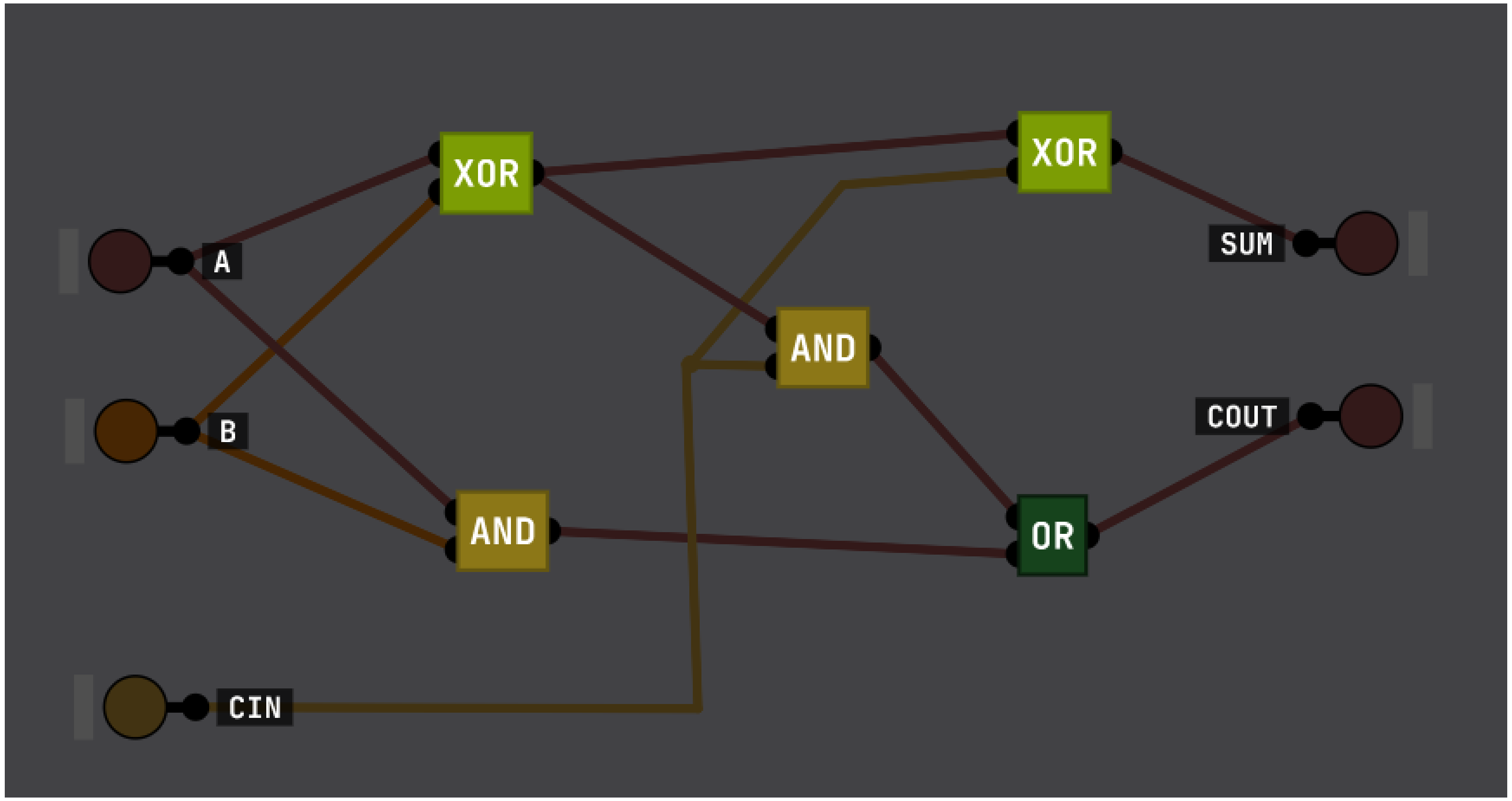
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

AND XOR



Full Adder

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Memory

- So far, all our circuits have been **combinational** circuits, where the output is only dependent on the present inputs.

- Often we need a circuit to remember.

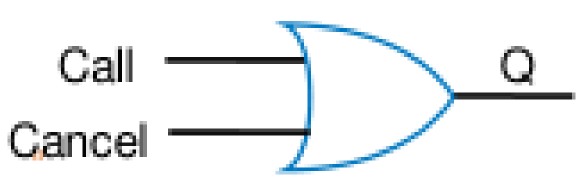
- Sequential circuit ⇒
  - Output depends not just on present inputs (as in combinational circuit), but on past sequence of inputs
    - Stores bits, also known as having “state”

Call Button

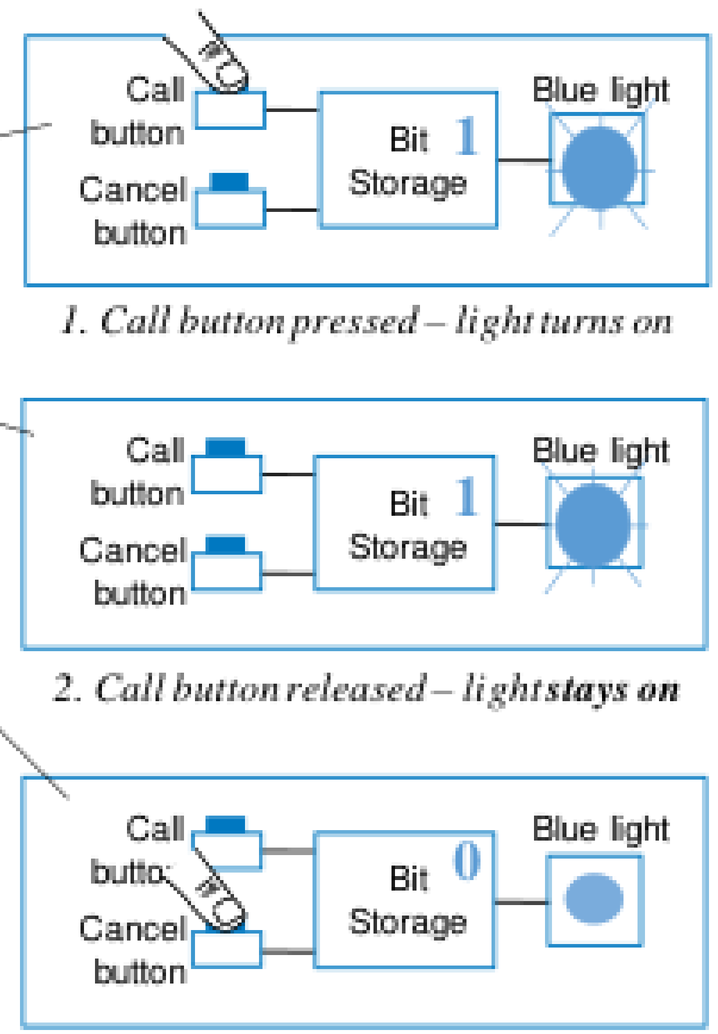
Storing One Bit – Flip-Flops

Example Requiring Bit Storage

- Flight attendant call button
  - Press call: light turns on
    - ***Stays on*** after button released
  - Press cancel: light turns off
    - Stays off after button released
  - Logic gate circuit to implement this?



Doesn't work. Q=1 when Call=1, but doesn't stay 1 when Call returns to 0



1. Call button pressed – light turns on

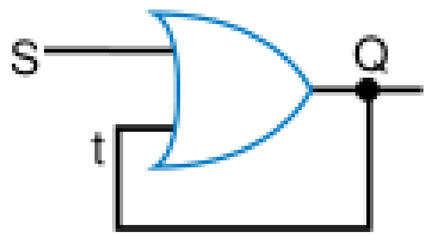
*2. Call button released – light **stays on***

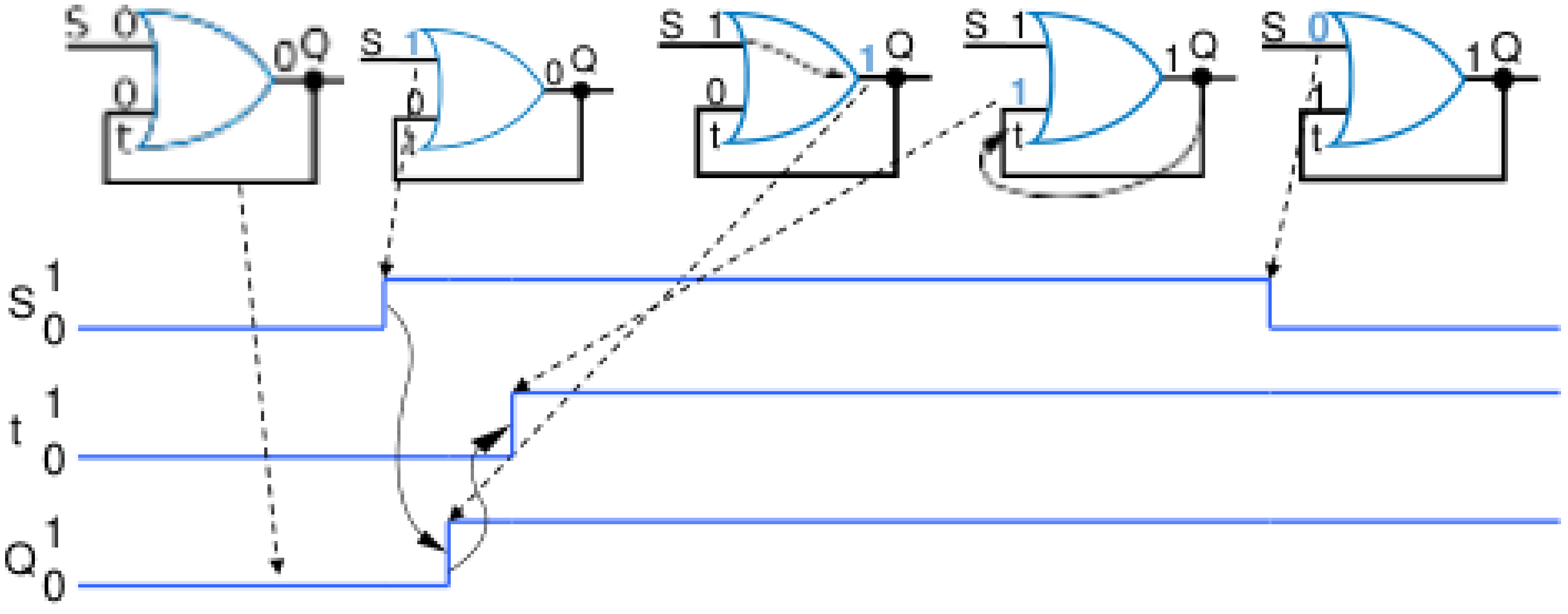
3. Cancel button pressed – light turns off

First Attempt

Need some sort of feedback

- Does circuit on the right do what we want?



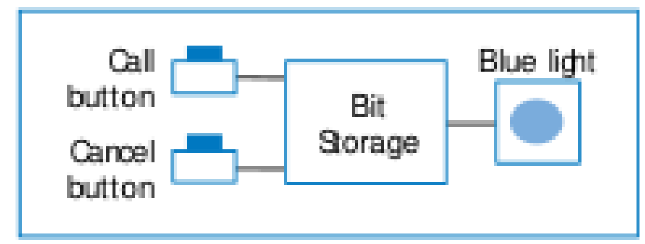


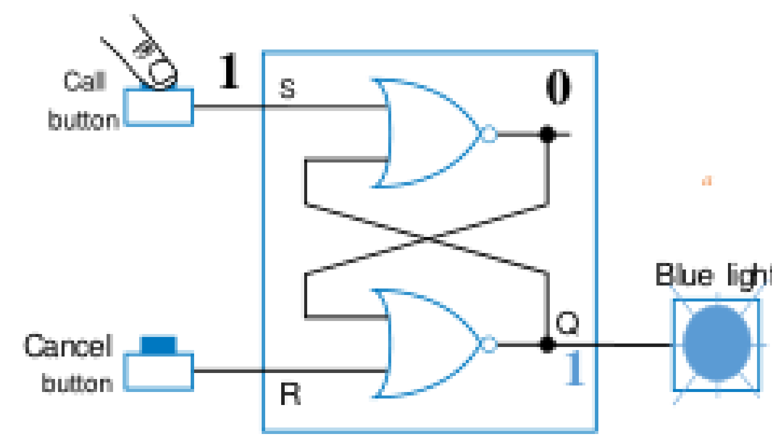
No: Once Q becomes 1 (when S=1), Q stays 1 forever – no value of S can bring Q back to 0

SR Latch

Example Using SR Latch for Bit Storage

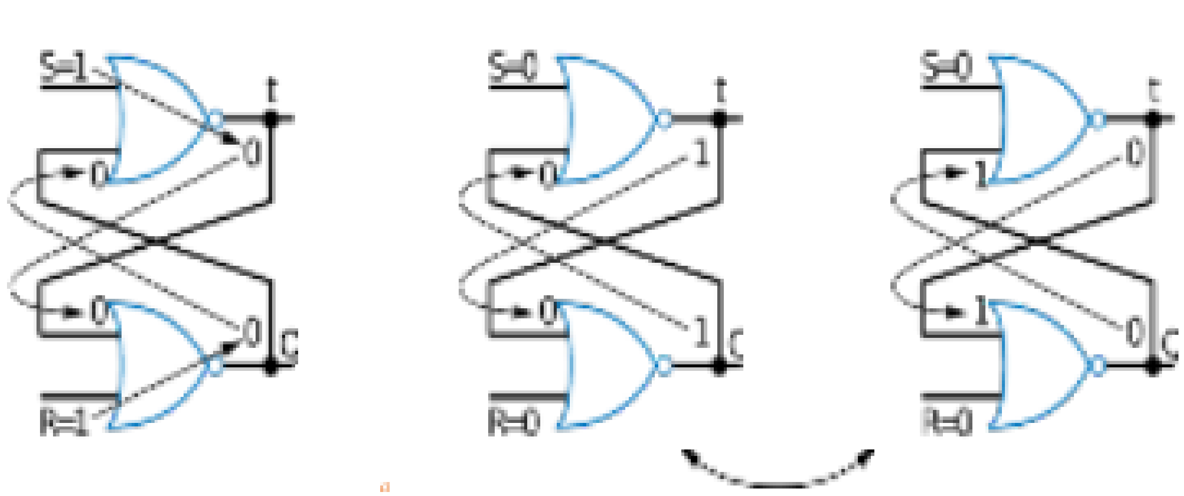
- SR latch can serve as bit storage in previous example of flight-attendant call button
  - Call=1 : sets Q to 1
    - Q stays 1 even after Call=0
  - Cancel=1 : resets Q to 0
- But, there’s a problem...

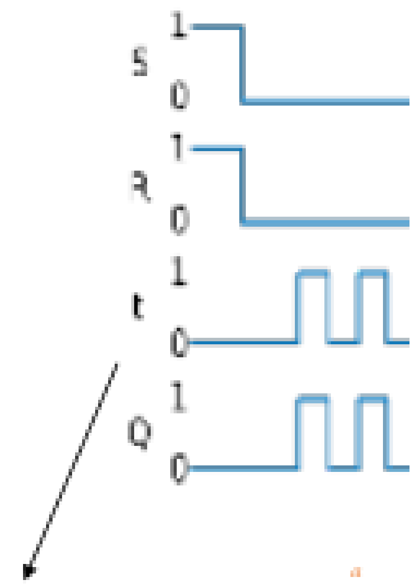




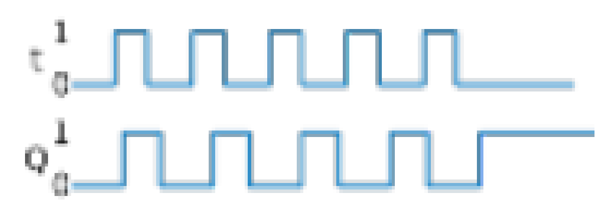
Problem with SR Latch

- Problem
  - If S=1 and R=1 simultaneously, we don't know what value Q will take

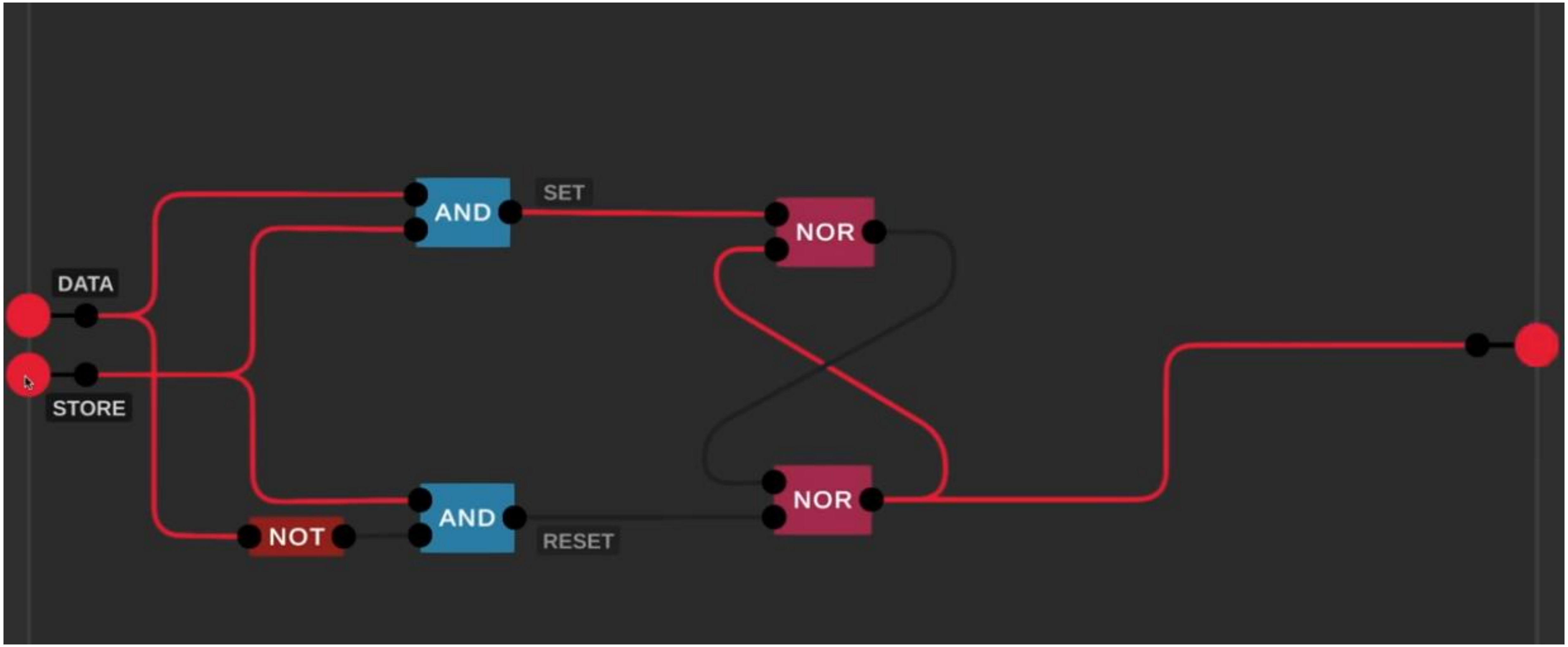




Q may oscillate. Then, because one path will be slightly longer than the other, Q will eventually settle to 1 or 0 – but we don't know which. Known as a *race*

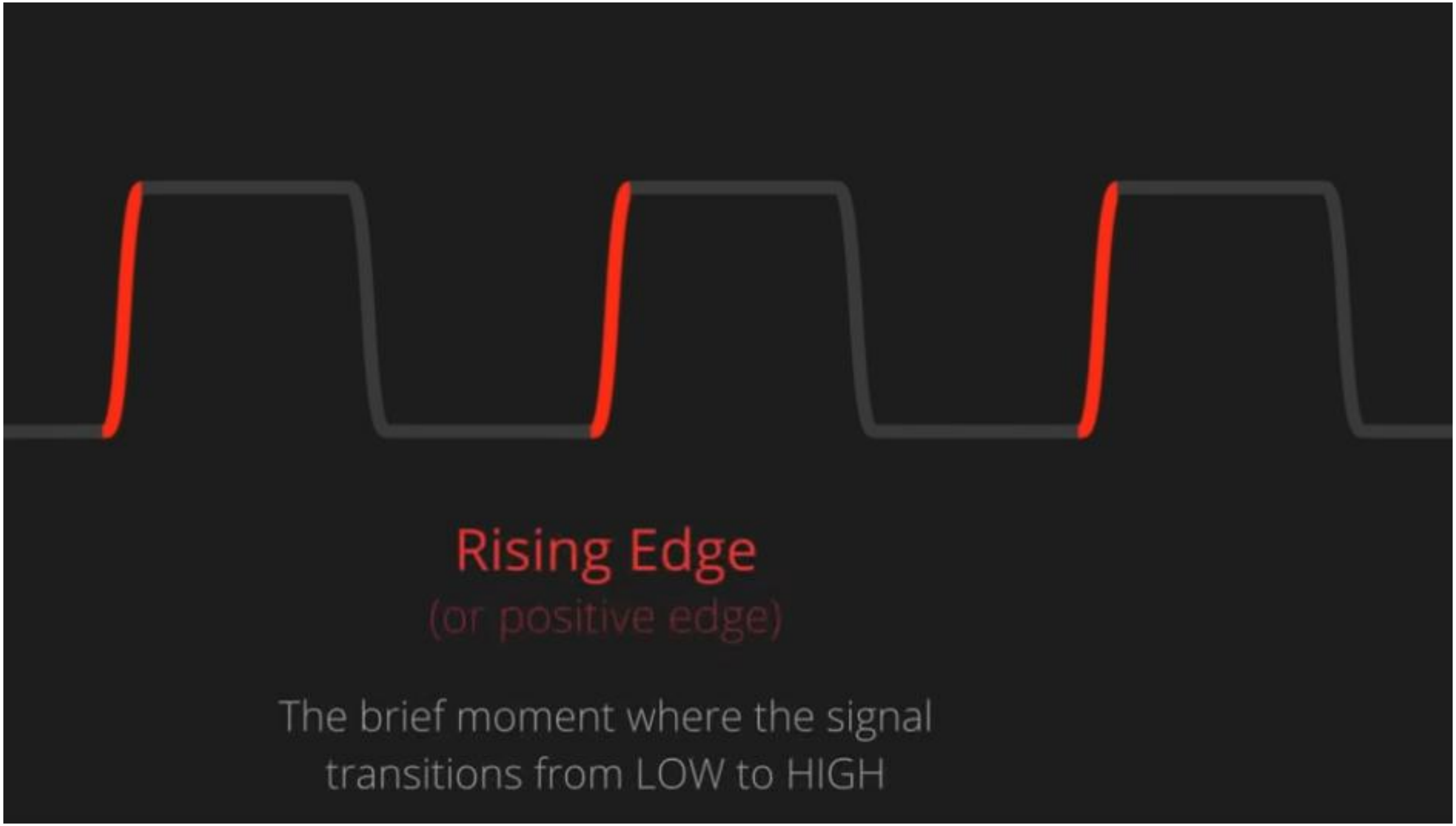


D Latch

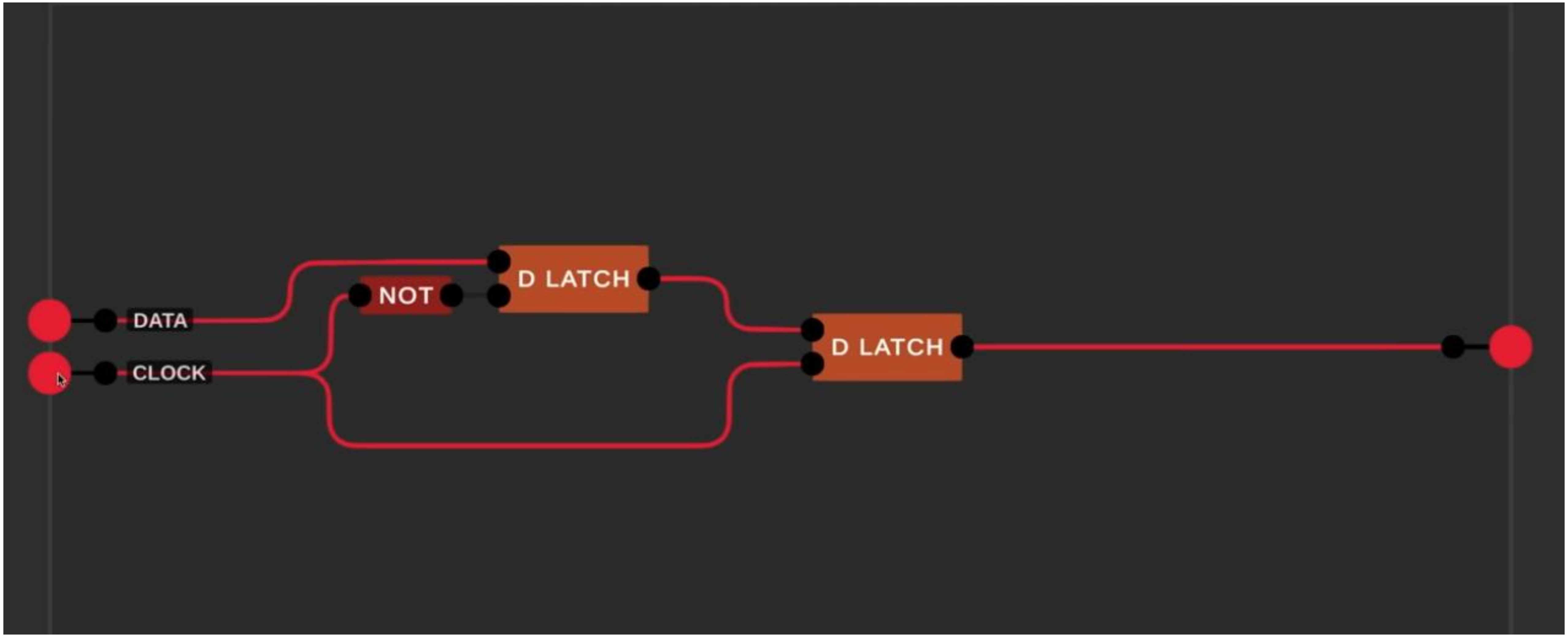


- When STORE = 1, output = DATA
- Still has issue with race condition
- Issue of synchronizing with other latches

Solution: Clock

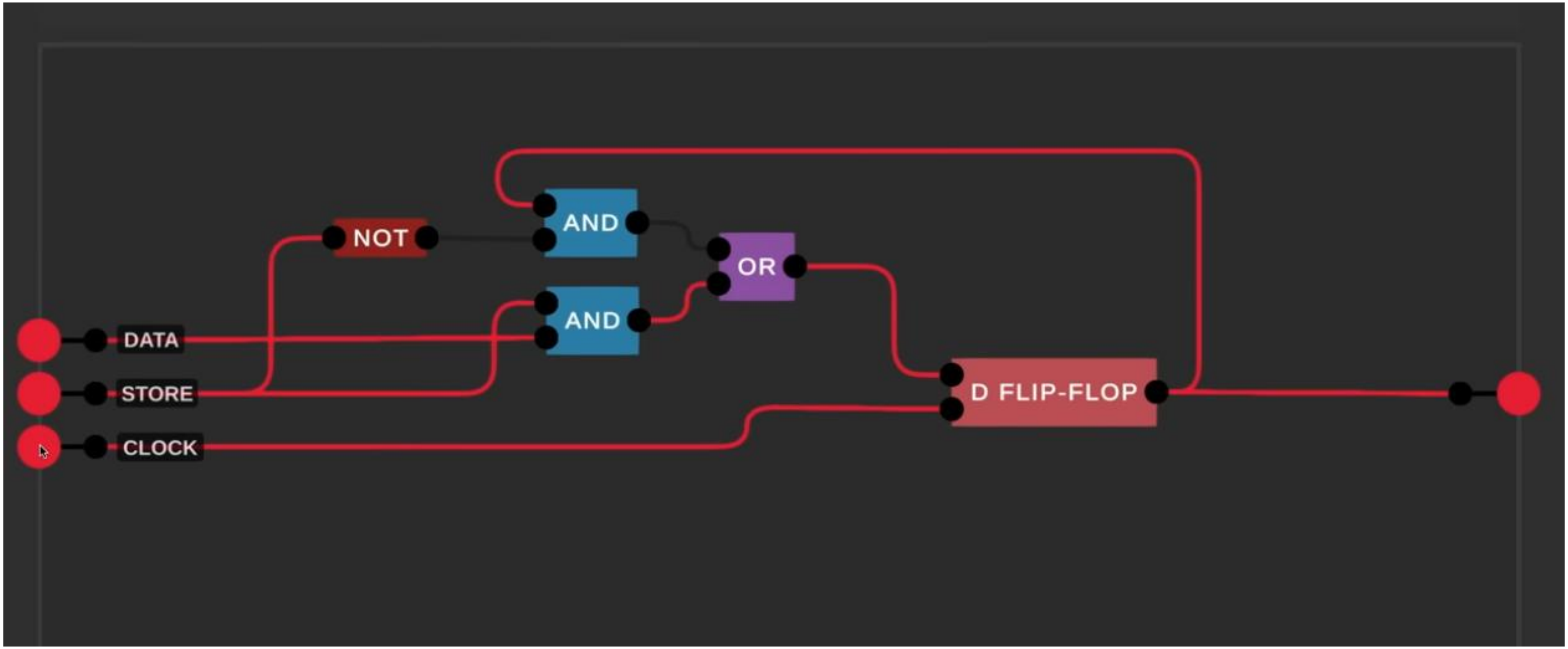


D Flip-Flop



Triggered on rising edge of clock signal

Register



One bit of memory