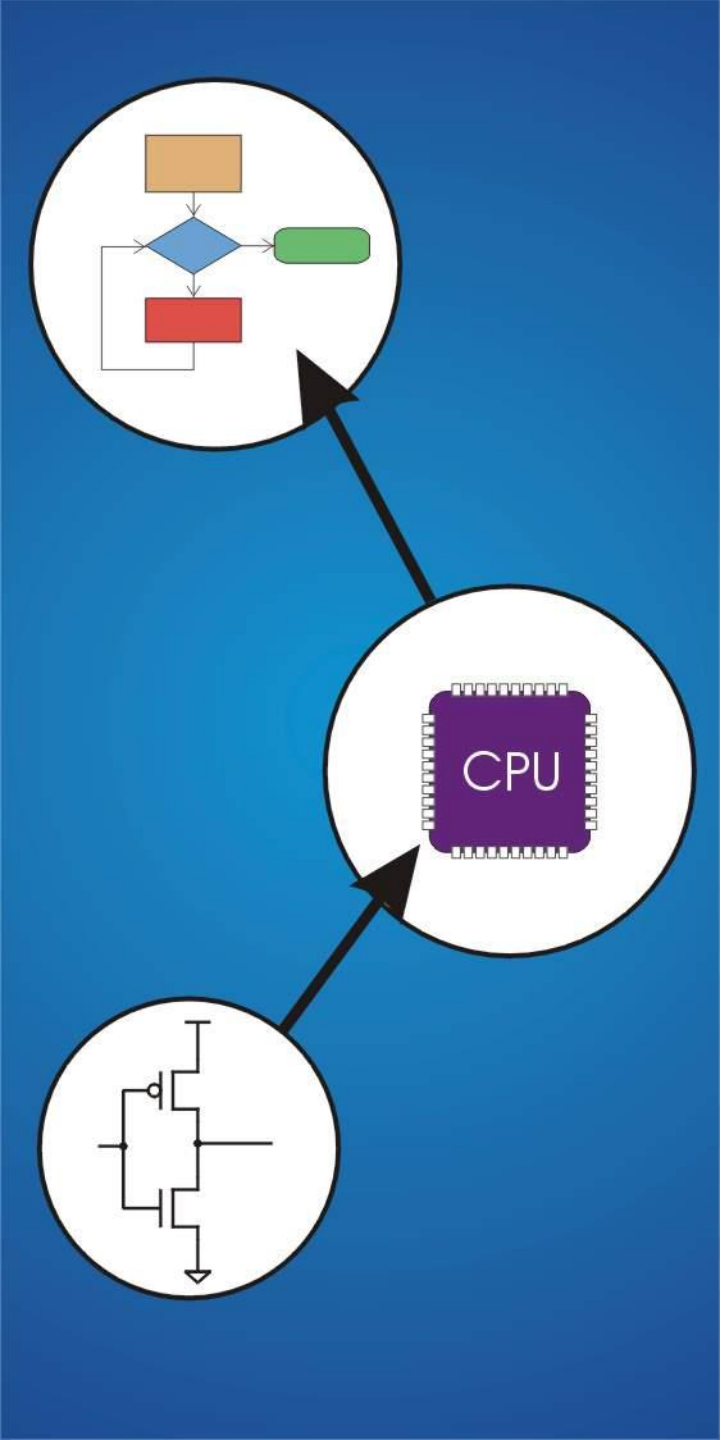
Introduction to Computer Engineering

CS/ECE 252, Spring 2017

Rahul Nayar

Computer Sciences Department

University of Wisconsin – Madison



Chapter 3
Digital Logic Structures

Slides based on set prepared by
Gregory T. Byrd, North Carolina State University

In Class Exercise

1. Convert “Transistor” to hex

2. Convert the results above to binary

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(	38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29	)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

In Class Exercise

- **0x 54 72 61 6e 73 69 73 74 6f 72**
- **0101 0100 0111 0010 0110 0001 0110 1110 0111 0011 0110 1001 0111 0011 0111 0100 0110 0110 1111 0111 0010**

In Class Exercise

- **Using 8 bits for each number, write the 1's complement, 2's complement, and signed magnitude binary number of the decimal numbers in the table below:**

Decimal	1's Complement	2's Complement	Signed Magnitude
19			
-10			

In Class Exercise

- **Using 8 bits for each number, write the 1's complement, 2's complement, and signed magnitude binary number of the decimal numbers in the table below:**

Decimal	1's Complement	2's Complement	Signed Magnitude
19	00010011	00010011	00010011
-10	11110101	11110110	10001010

In Class Exercise

- **Convert the following 32-bit single-precision IEEE floating point number into decimal value**
- **1 10000010 01001100000000000000000**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the following 32-bit single-precision IEEE floating point number into decimal value**
- **1 10000010 01001100000000000000000**
- **-10.375**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the following 32-bit single-precision IEEE floating point number into decimal value**
- **1 10000010 01001100000000000000000**
- **-10.375**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Step 1:**
- **Write -10.25 as binary "-1010.01"**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Step 2:**
- **Normalize "-1.01001 x (2^3)"**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Step 3:**
- **Sign bit "1"**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Step 4:**
- **Exponent must satisfy x-127=3, thus x=130**
- **x= 128 +2**
- **x= 2^7+2^1**
- **x=10000010**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Step 5**
- **The fraction field consists of numbers on the right side of the decimal point in -1.01001 with 23 bit precision**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126}, \ \text{exponent} = 0$$

In Class Exercise

- **Convert the decimal value -10.25 into its 32 bit IEEE single-precision floating point representation.**
- **Answer**
- **1 10000010 01001000000000000000000**

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \leq \text{exponent} \leq 254$$

$$N = -1^{S} \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

Transistor: Building Block of Computers

Microprocessors contain millions of transistors

- Intel Pentium II: **7 million**
- Compaq Alpha 21264: **15 million**
- Intel Pentium III: **28 million**

Logically, each transistor acts as a switch

Combined to implement logic functions

- **AND, OR, NOT**

Combined to build higher-level structures

- **Adder, multiplexer, decoder, register, …**

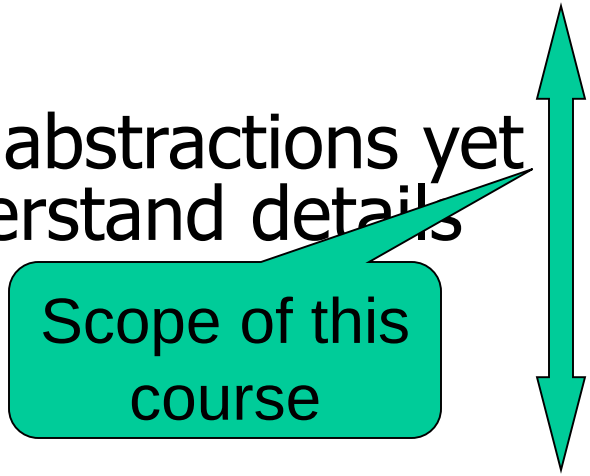
Combined to build processor

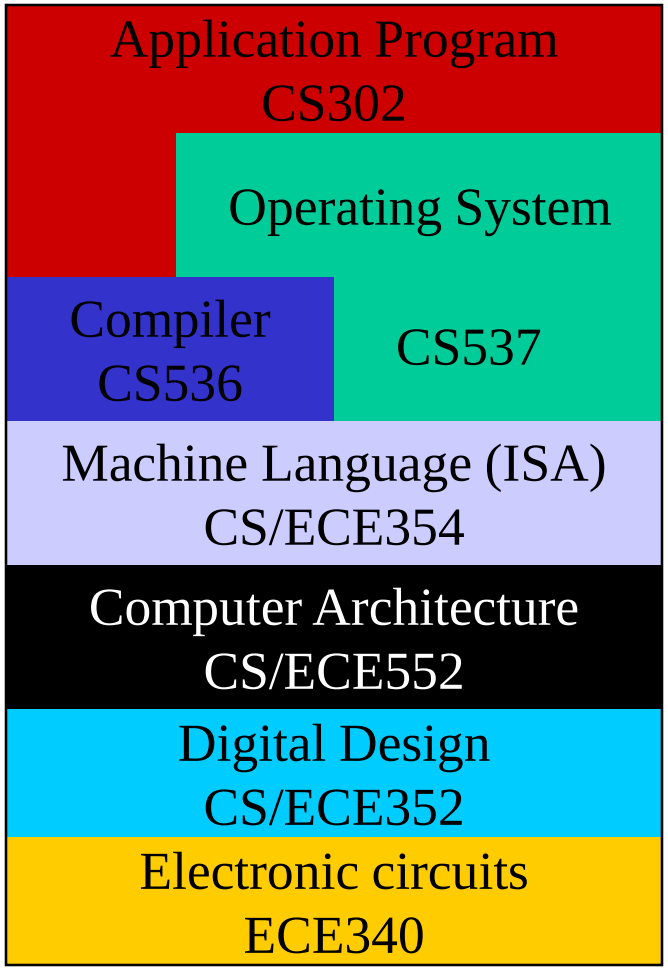
- **LC-3**

Abstraction and Complexity

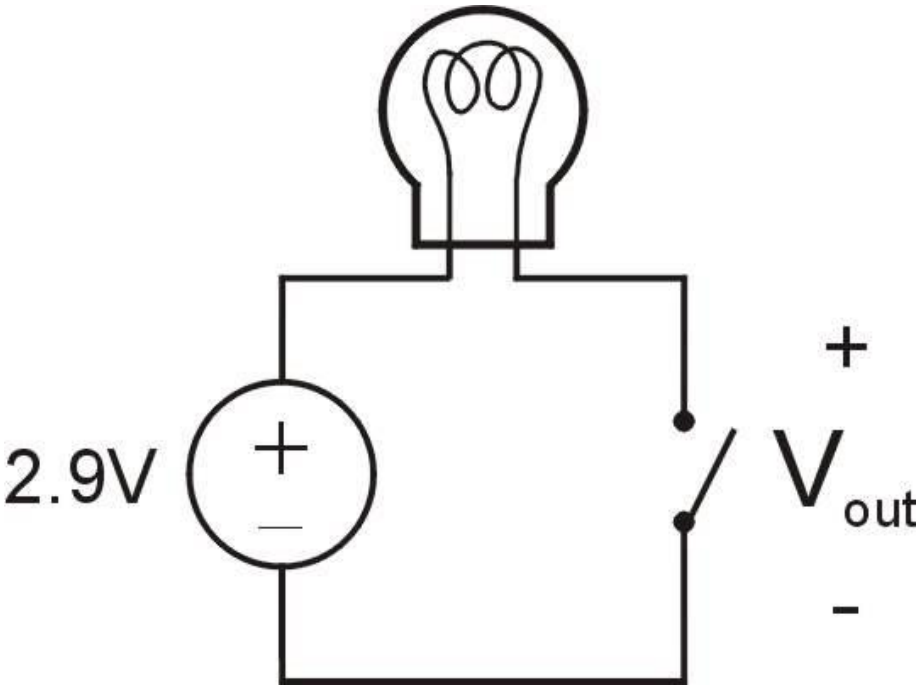
- **Abstraction helps us manage complexity**
- **Complex interfaces**
  - **Specify what to do**
  - **Hide details of how**

- Goal: Use abstractions yet still understand details





Simple Switch Circuit



Switch open:

- **No current through circuit**
- **Light is off**
- **V_{out} is +2.9V**

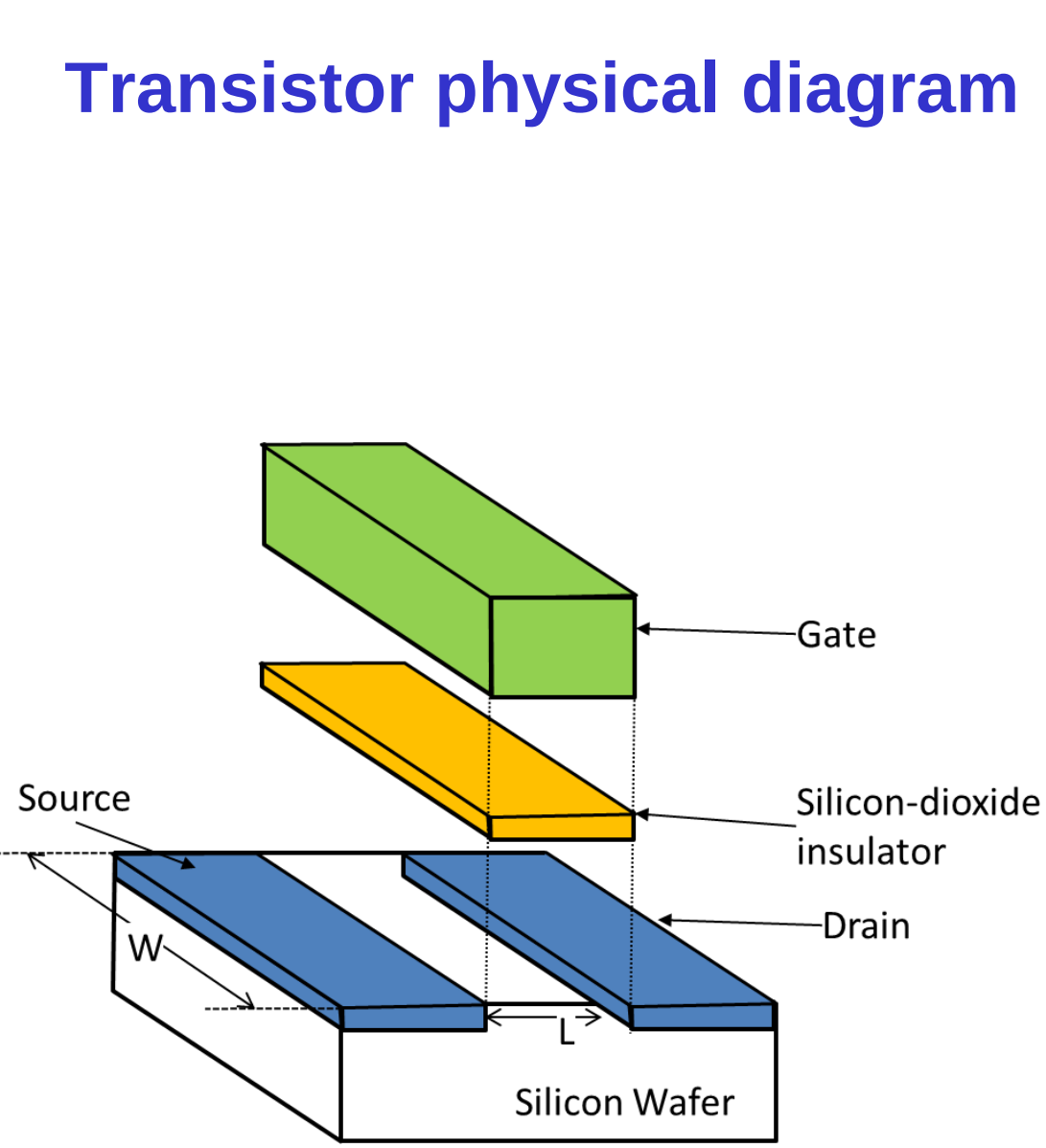
Switch closed:

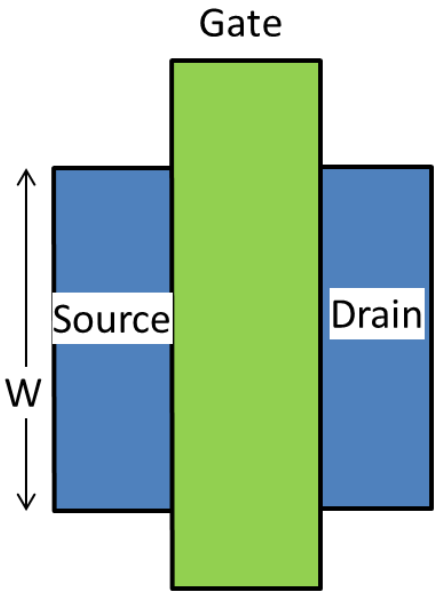
- **Short circuit across switch**
- **Current flows**
- **Light is on**
- **V_{out} is 0V**

Switch-based circuits can easily represent two states: on/off, open/closed, voltage/no voltage.

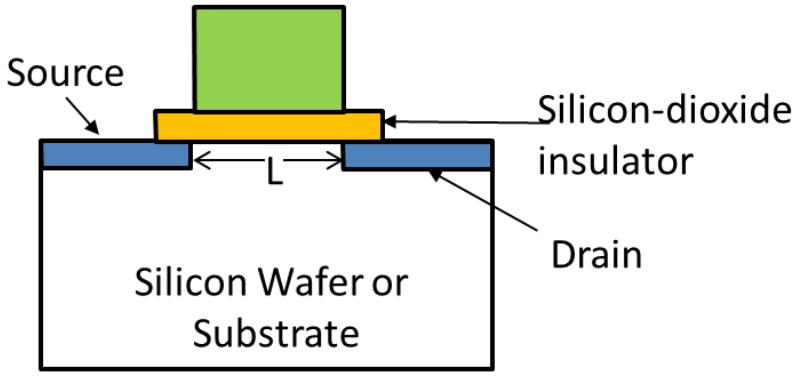
"Miracle Month" of November 17 to December 23, 1947

Transistor physical diagram



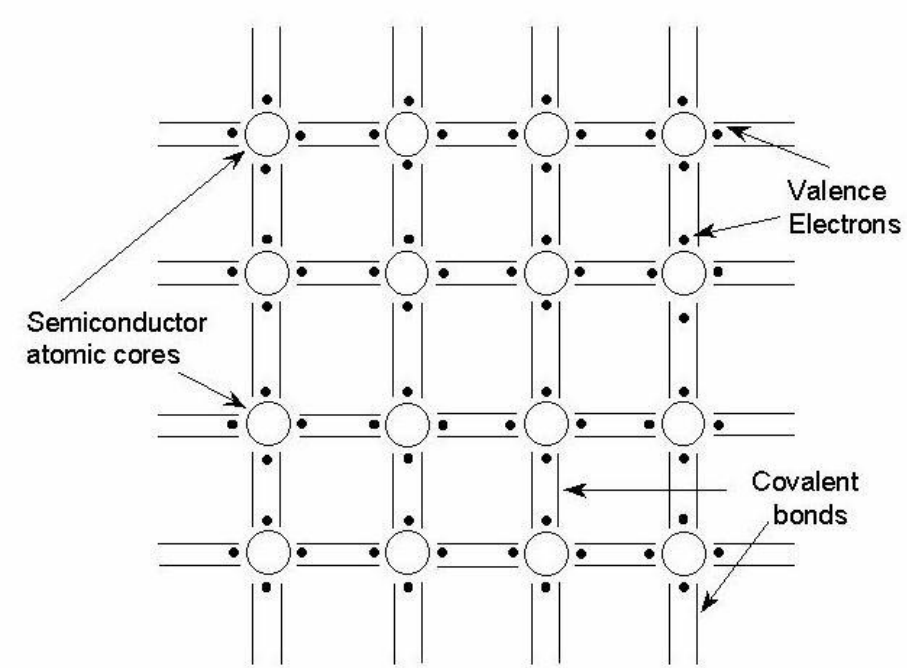


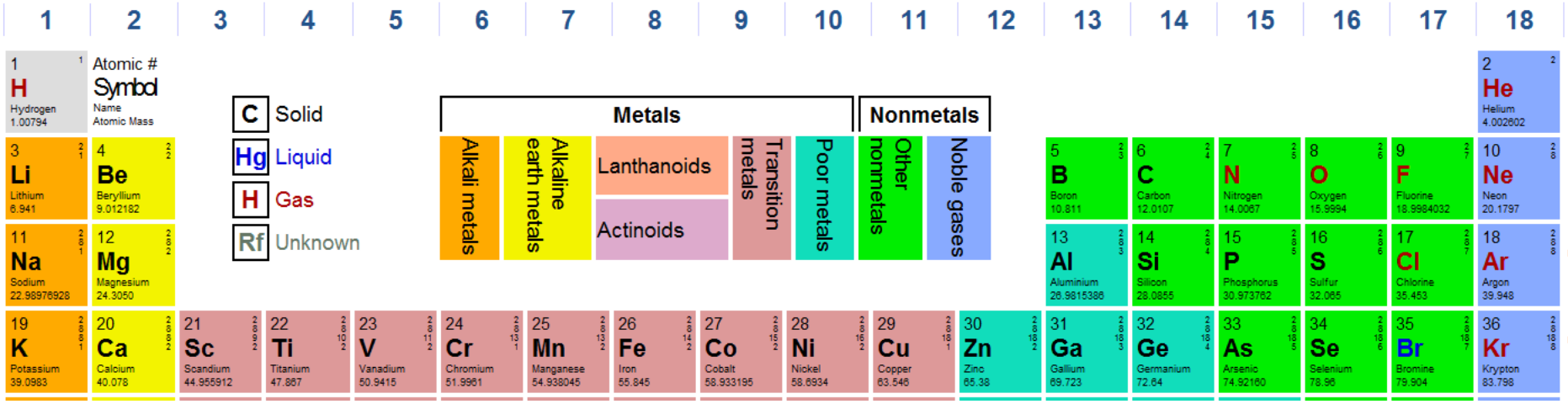
Top view



Side view

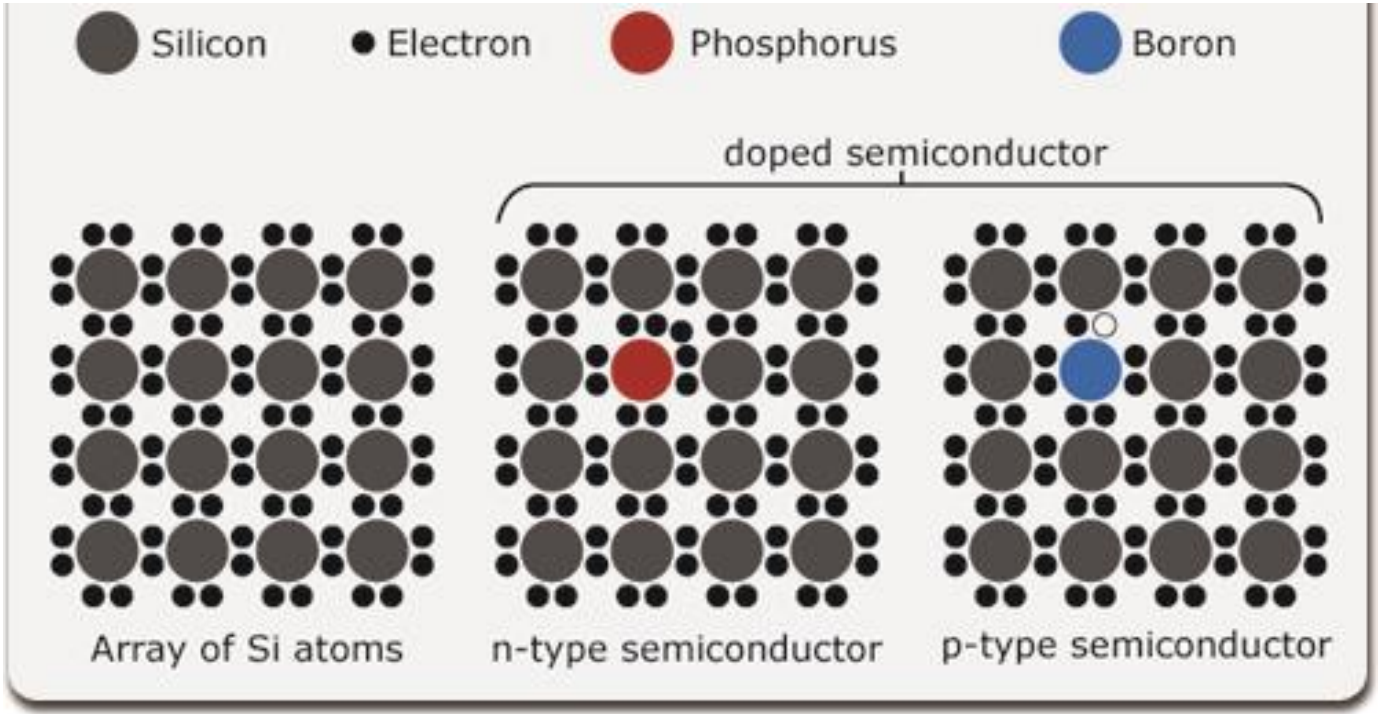
- **Silicon by itself is semiconductor**
  - **Has 4 valence electrons and forms a co-valent bond with 4 neighbors**
  - **Hard to break and does not conduct ☹**

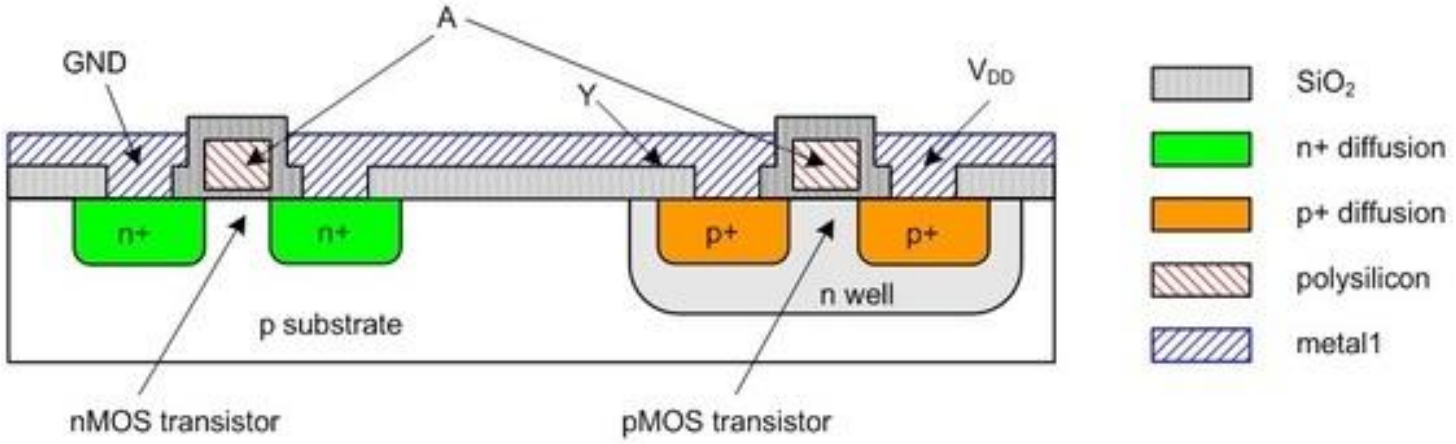




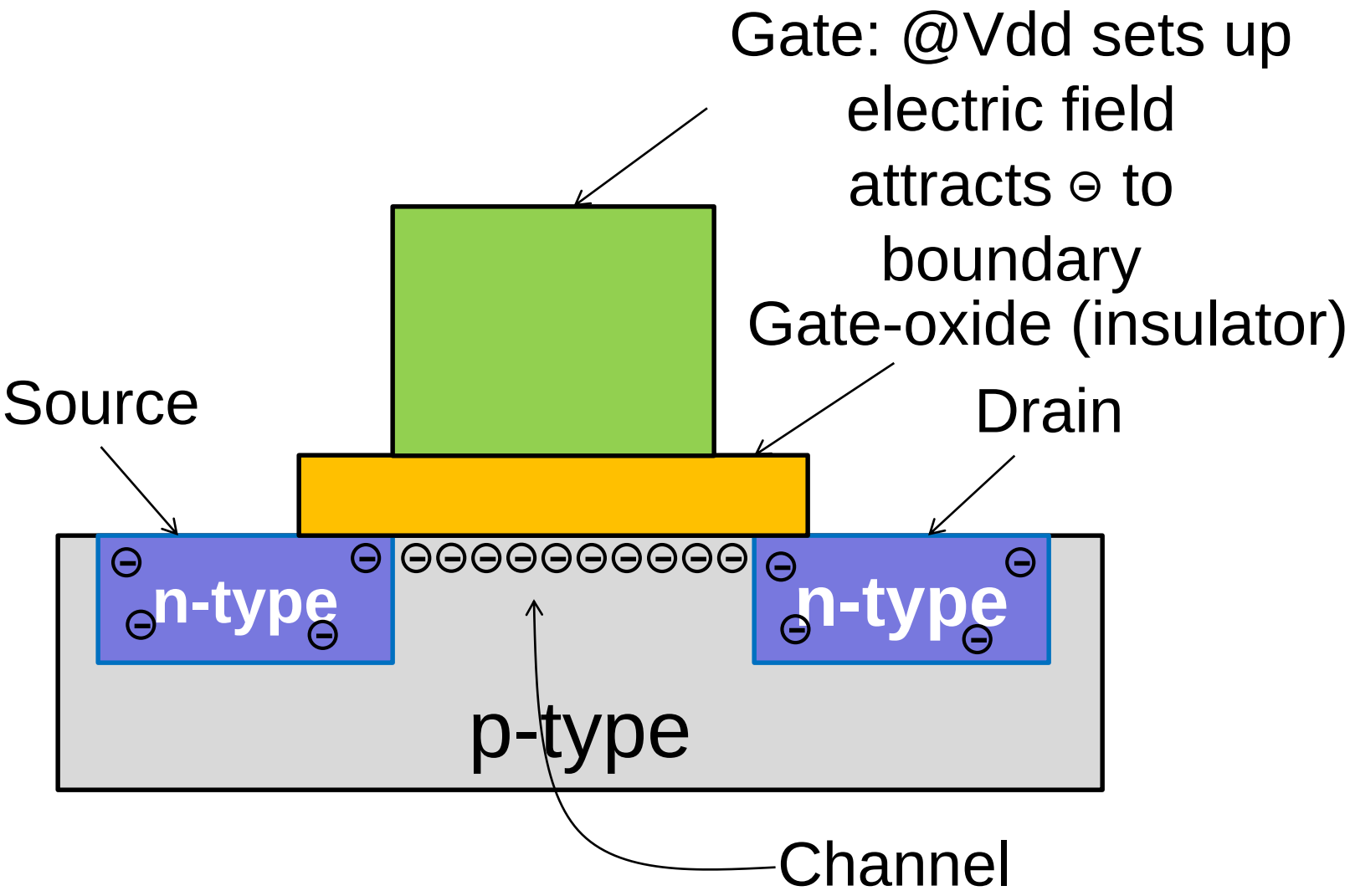
- **Silicon by itself is insulator**
  - **Has 4 valence electrons and forms a co-valent bond with 4 neighbors**
  - **Had to break and does not conduct ☹**
- **Silicon + Arsenic makes it filled with electrons and conducting**
- **Silicon + Boron makes it filled with “holes” and conducting**

nmos and pmos Transistors





Transistor Operation (nmos)



If the figure on the left is a (musical) conductor, then what is the figure on the right?

semiconductor

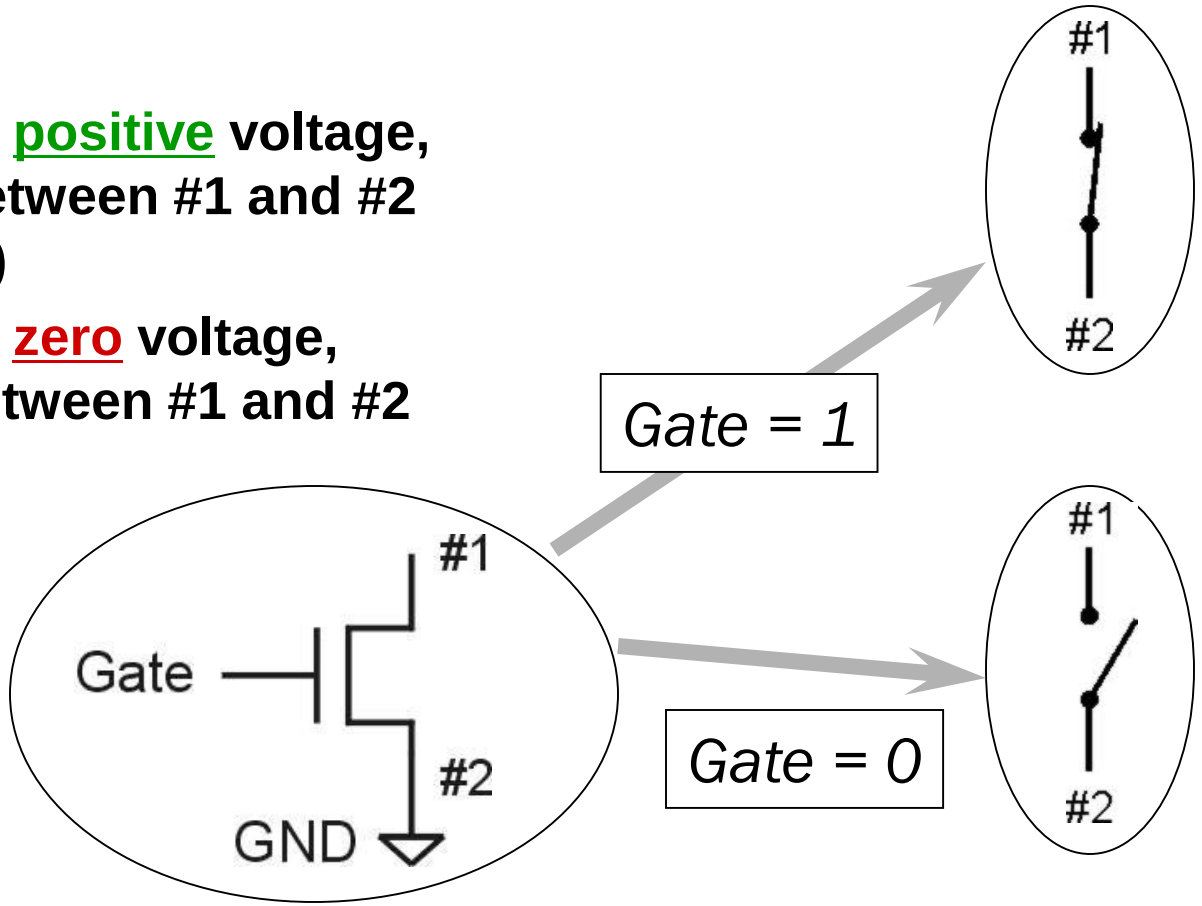
N-type MOS Transistor

MOS = Metal Oxide Semiconductor

- two types: N-type and P-type

N-type

- when Gate has positive voltage, short circuit between #1 and #2 (switch closed)
- when Gate has zero voltage, open circuit between #1 and #2 (switch open)

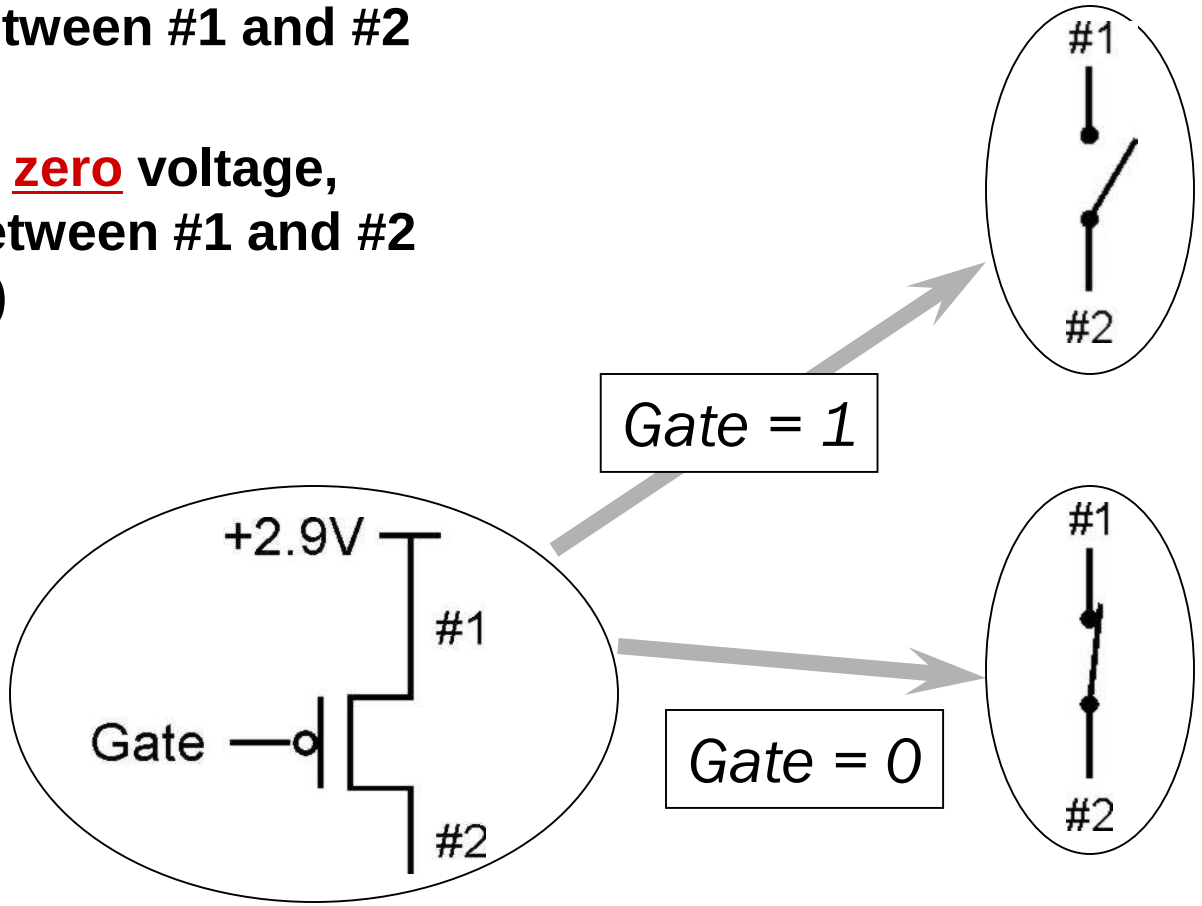


Terminal #2 must be connected to GND (0V).

P-type MOS Transistor

P-type is *complementary* to N-type

- **when Gate has positive voltage, open circuit between #1 and #2 (switch open)**
- **when Gate has zero voltage, short circuit between #1 and #2 (switch closed)**



Terminal #1 must be connected to +2.9V.

Logic Gates

Use switch behavior of MOS transistors to implement logical functions: AND, OR, NOT.

Digital symbols:

- **recall that we assign a range of analog voltages to each digital (logic) symbol**

- **assignment of voltage ranges depends on electrical properties of transistors being used**
  - **typical values for "1": +5V, +3.3V, +2.9V, +1.1V**
  - **for purposes of illustration, we'll use +2.9V**

CMOS Circuit

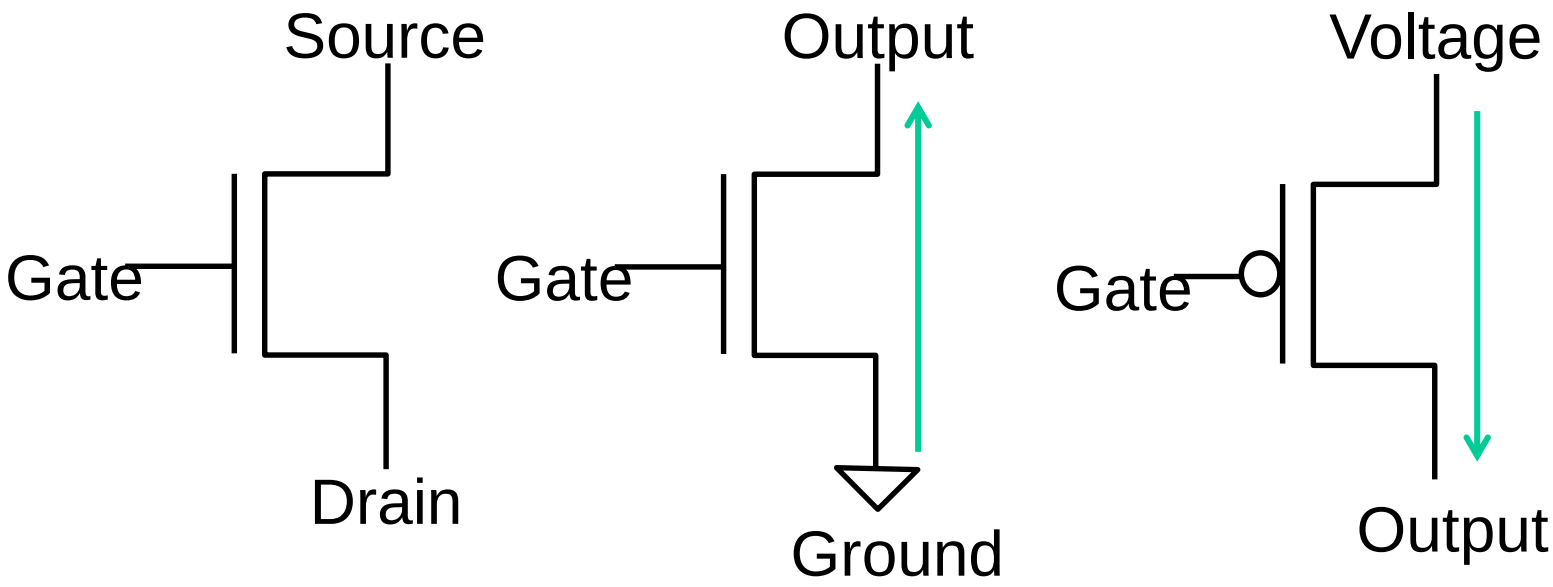
Complementary MOS

Uses both N-type and P-type MOS transistors

- **P-type**
  - **Attached to + voltage**
  - **Pulls output voltage UP when input is zero**
- **N-type**
  - **Attached to GND**
  - **Pulls output voltage DOWN when input is one**

For all inputs, make sure that output is either connected to GND or to +, but not both!

Transistor



Transistor terminals

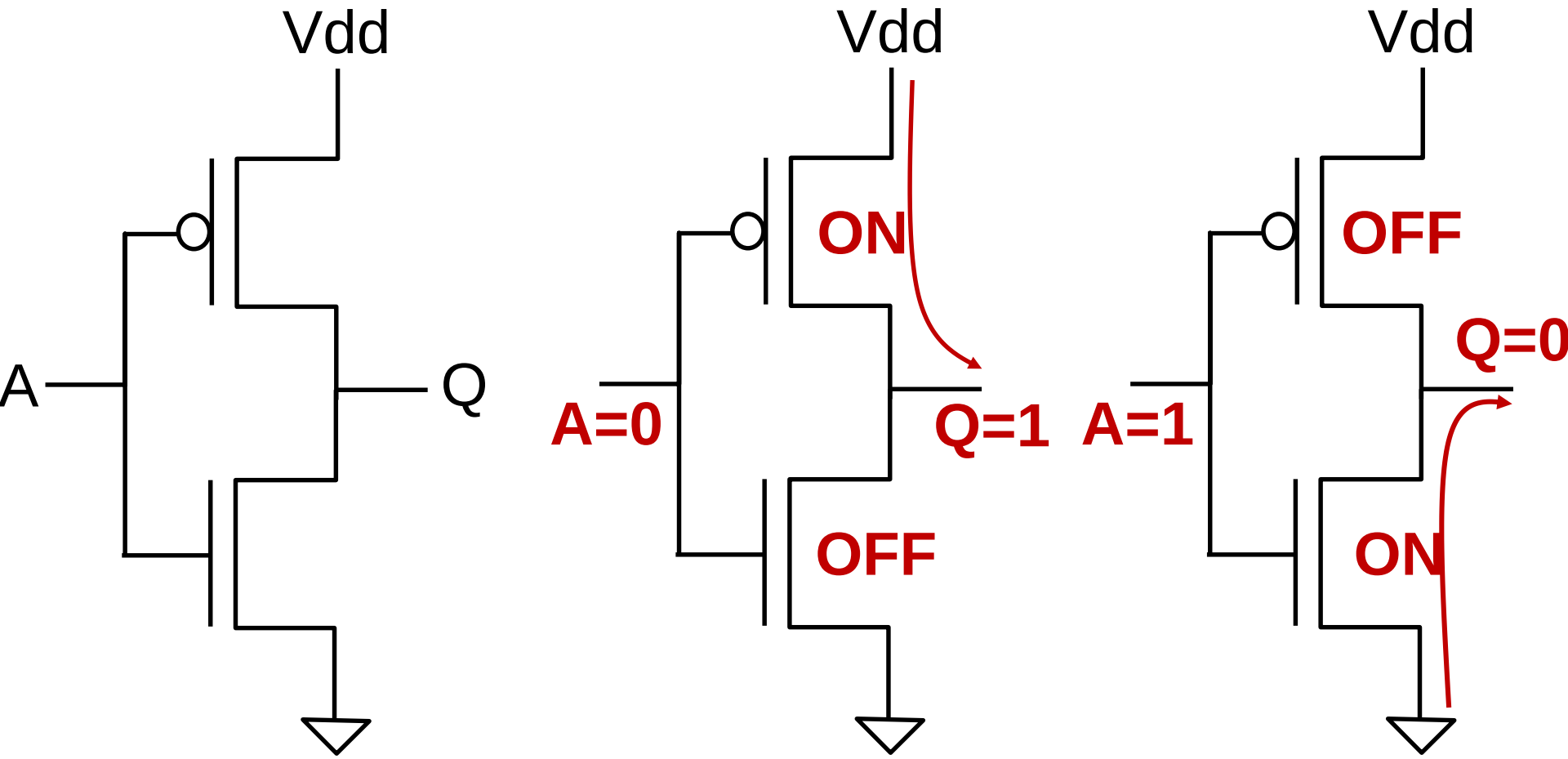
N-type transistor

Gate	Behavior
1	Closed Output=0
0	Open Output=Z

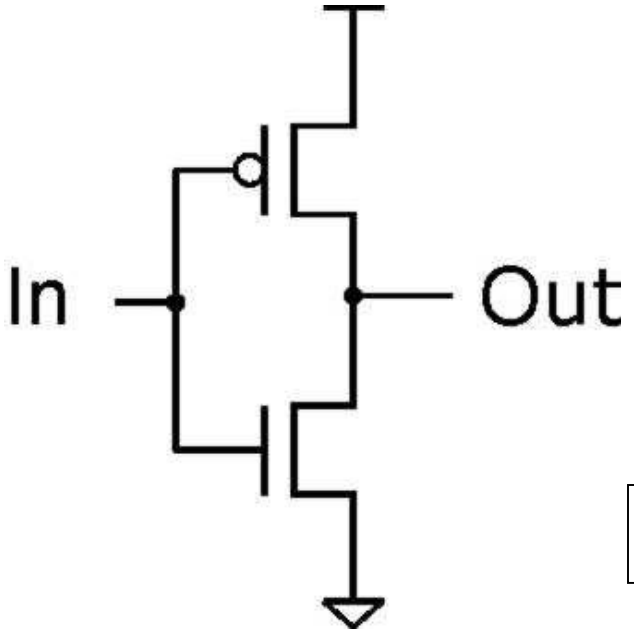
P-type transistor

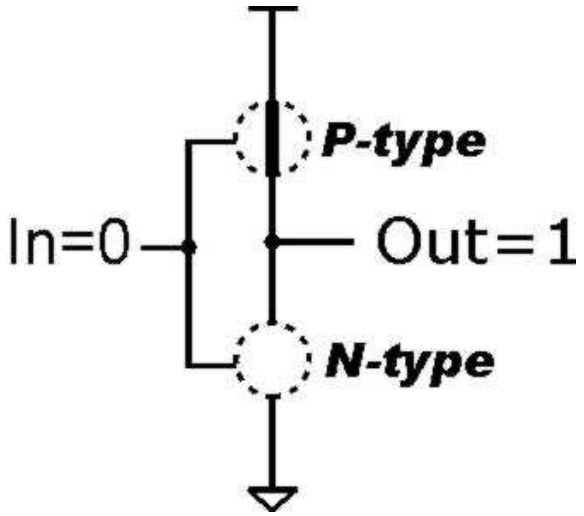
Gate	Behavior
0	Closed Output=1
1	Open Output=Z

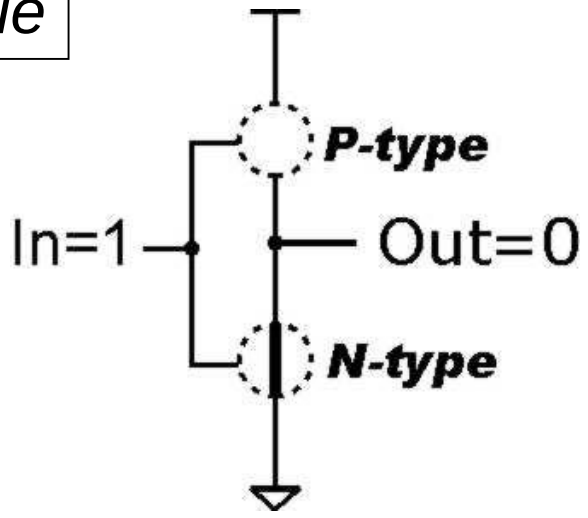
NOT Gate with Transistors



Inverter (NOT Gate)





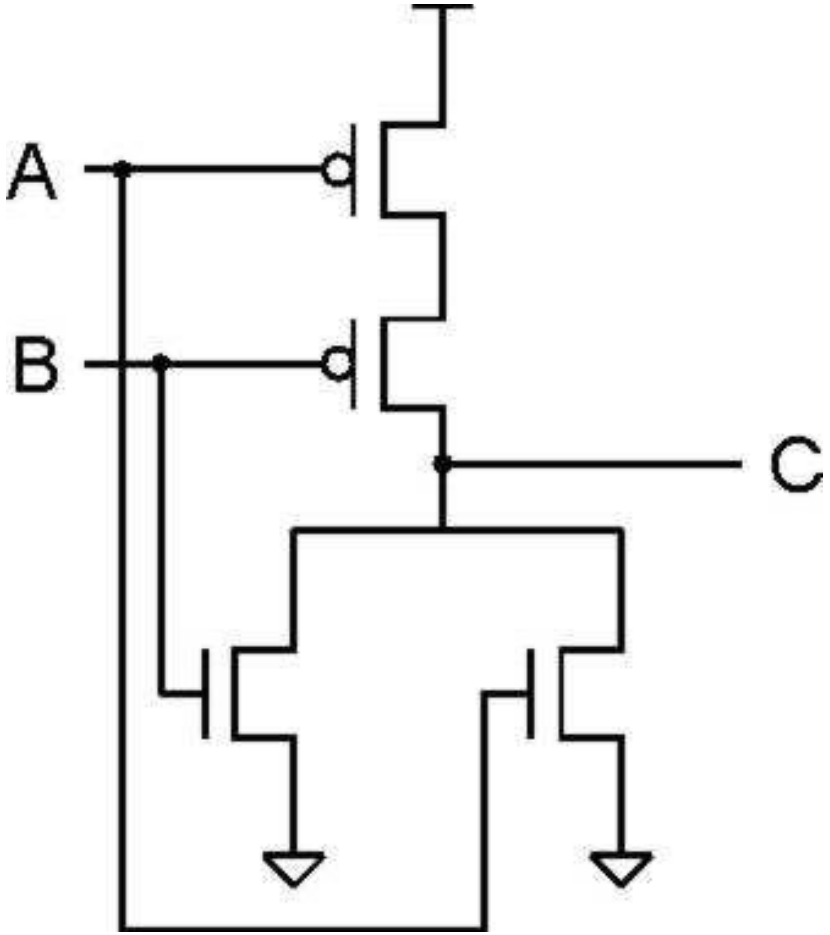


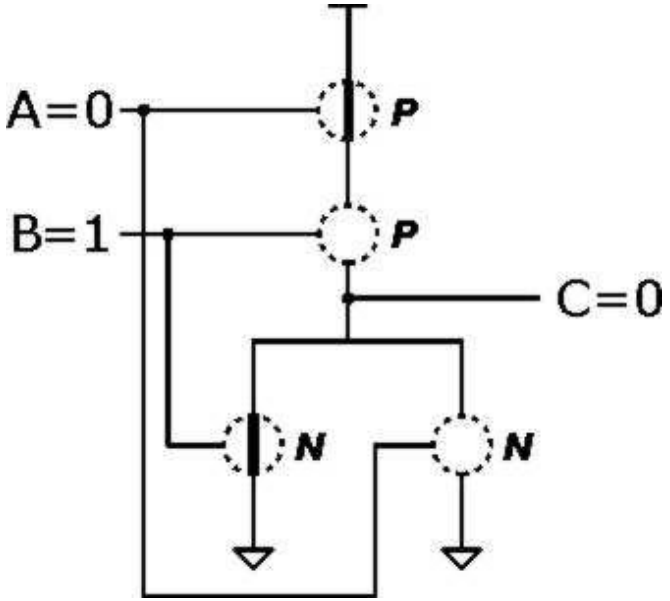
Truth table

In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0

NOR Gate

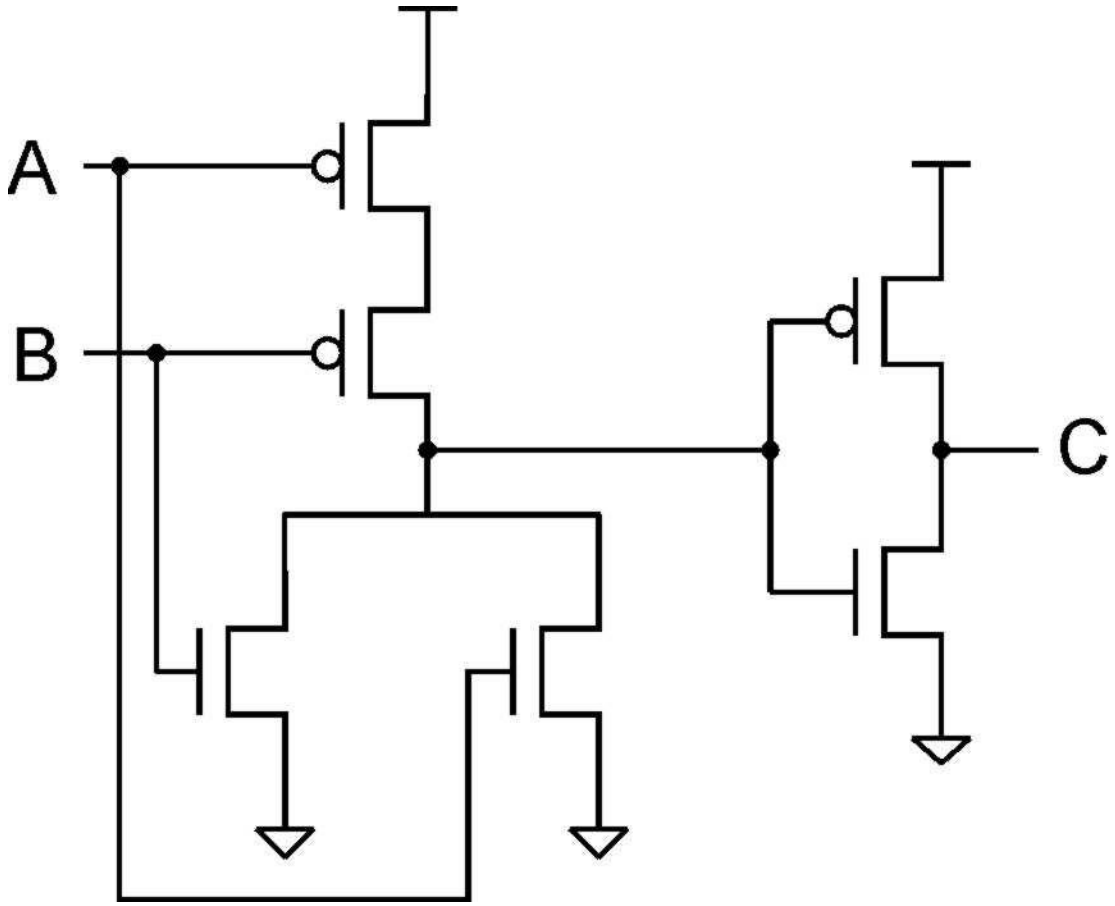




A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure on top, parallel on bottom.

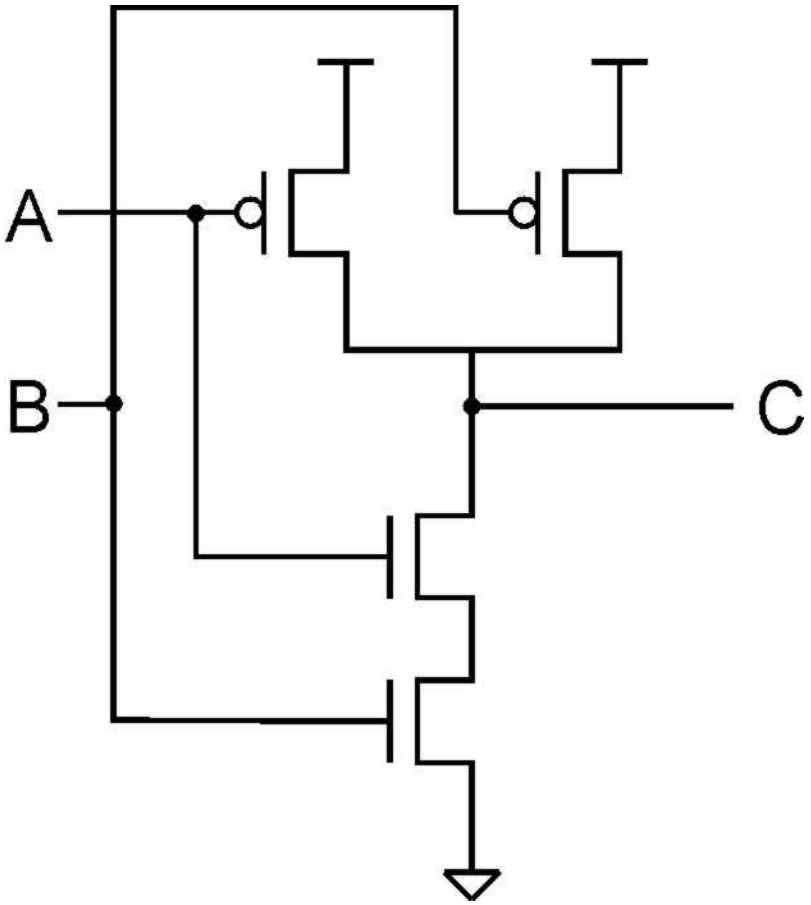
OR Gate

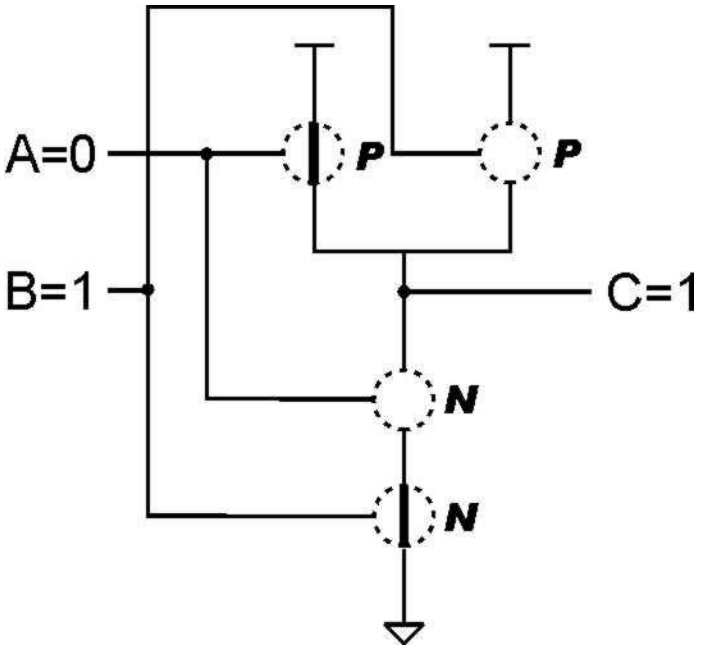


A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Add inverter to NOR.

NAND Gate (AND-NOT)

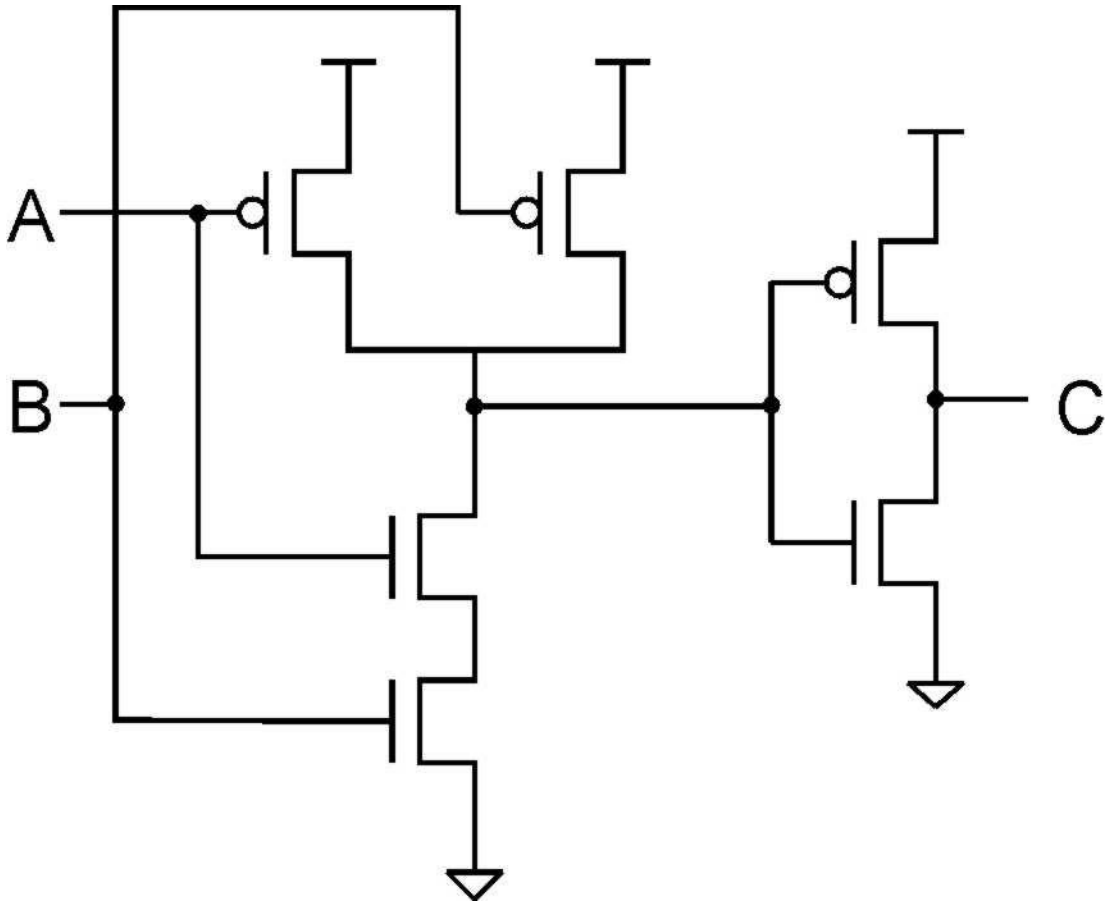




A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Note: Parallel structure on top, serial on bottom.

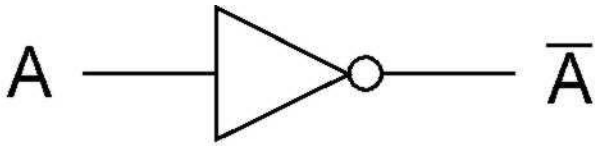
AND Gate



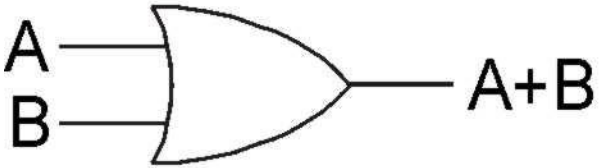
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Add inverter to NAND.

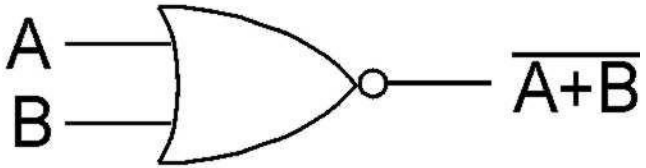
Basic Logic Gates



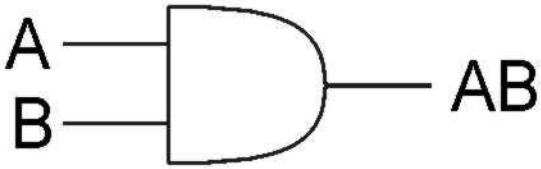
NOT



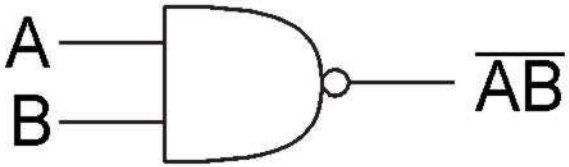
OR



NOR



AND



NAND

In-Class Exercise

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

Convert 777 to IEEE 32-bit floating point

Convert the results back to decimal

In-Class Exercise

$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

Convert 777 to IEEE 32-bit floating point

- **777 = 1100001001 = 1.100001001 * 2^9**
- **9 = exp – 127**
- **exp = 136 = 10001000**
- **sign = 0 since positive**
- **0 10001000 10000100100000000000000**

Convert the results back to decimal

In-Class Exercise

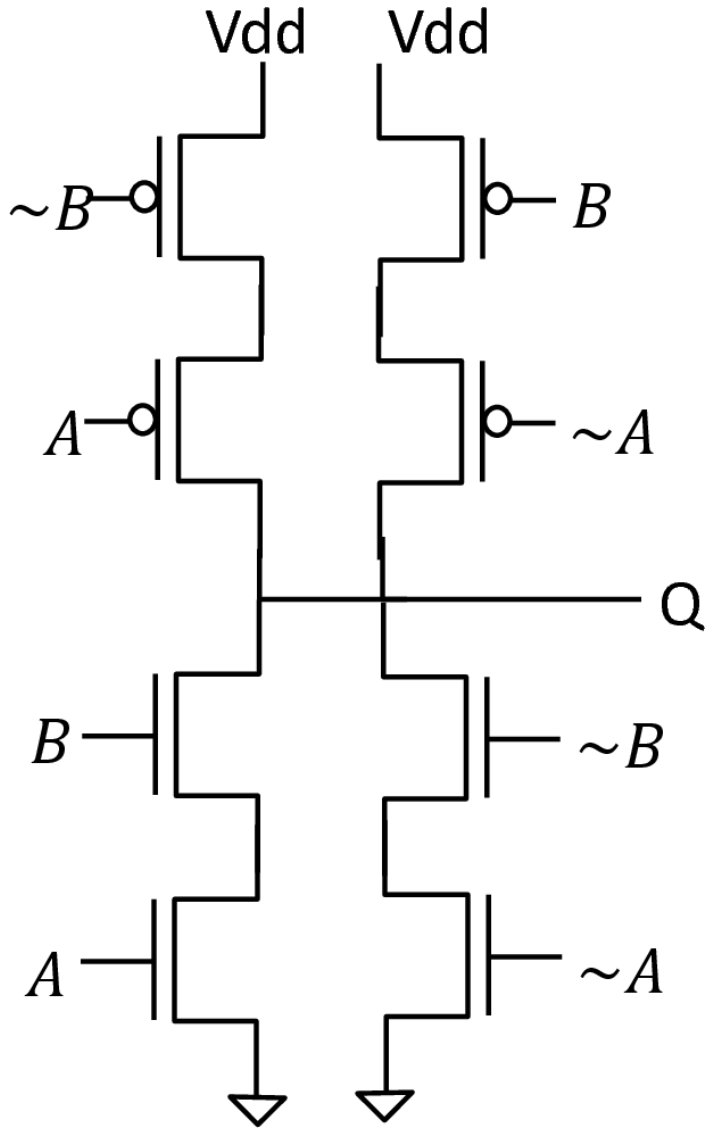
$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

Convert 777 to IEEE 32-bit floating point

Convert the results back to decimal

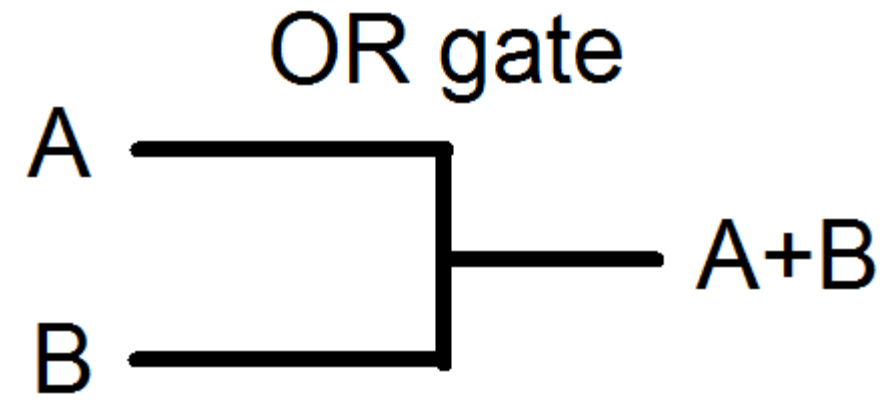
- **0 10001000 10000100100000000000000**
- **0b10001000 = 0d136**
- **N = (-1)^S * 1.fraction * 2^(exp-127)**
- **N = (-1)^0 * 1.100001001 * 2^(136-127)**
- **N = 1* 1.100001001 * 2^9**
- **N = 1100001001.0**
- **N = 777**

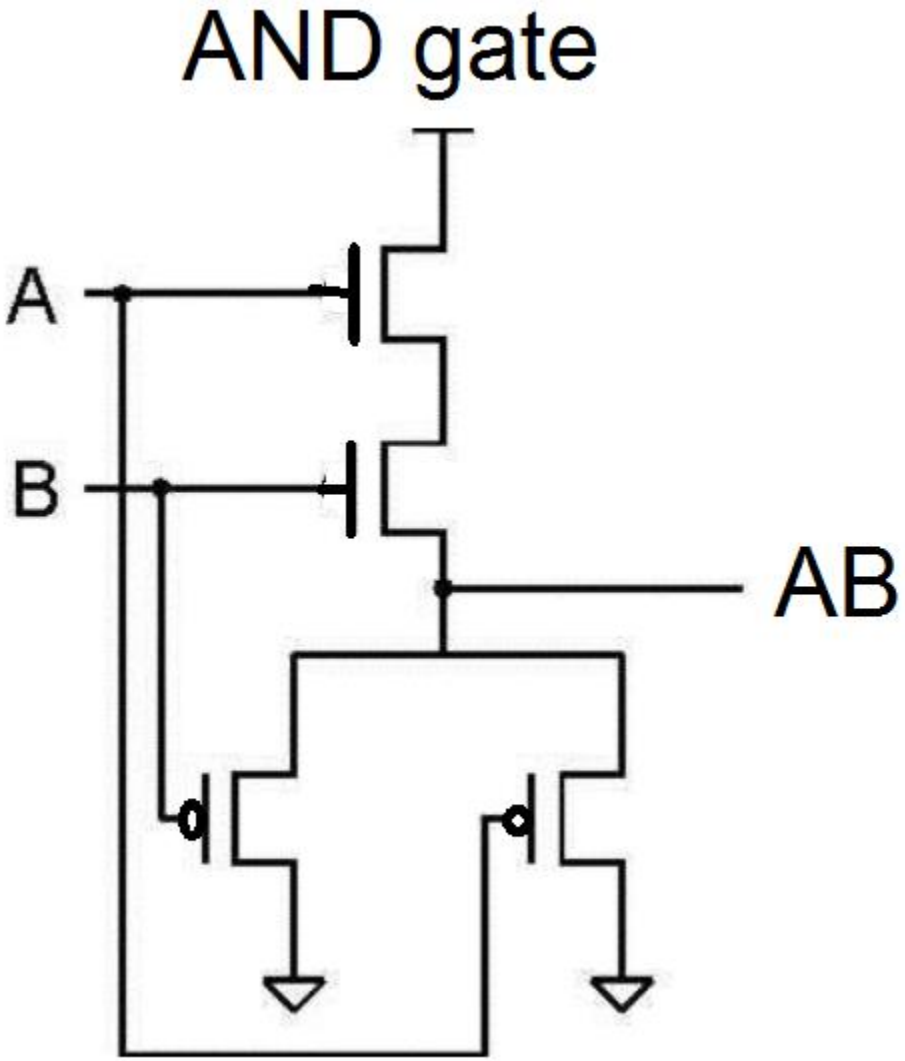
Take-home exercise
Fill out the truth table for the following transistor circuit

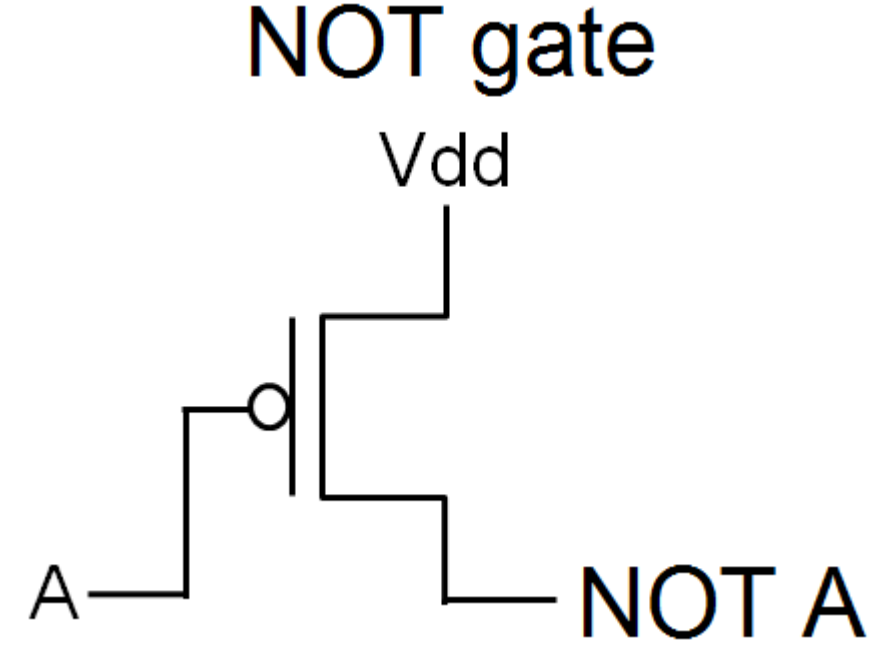


A	B	Q
0	0	
0	1	
1	0	
1	1	

Common Misconceptions





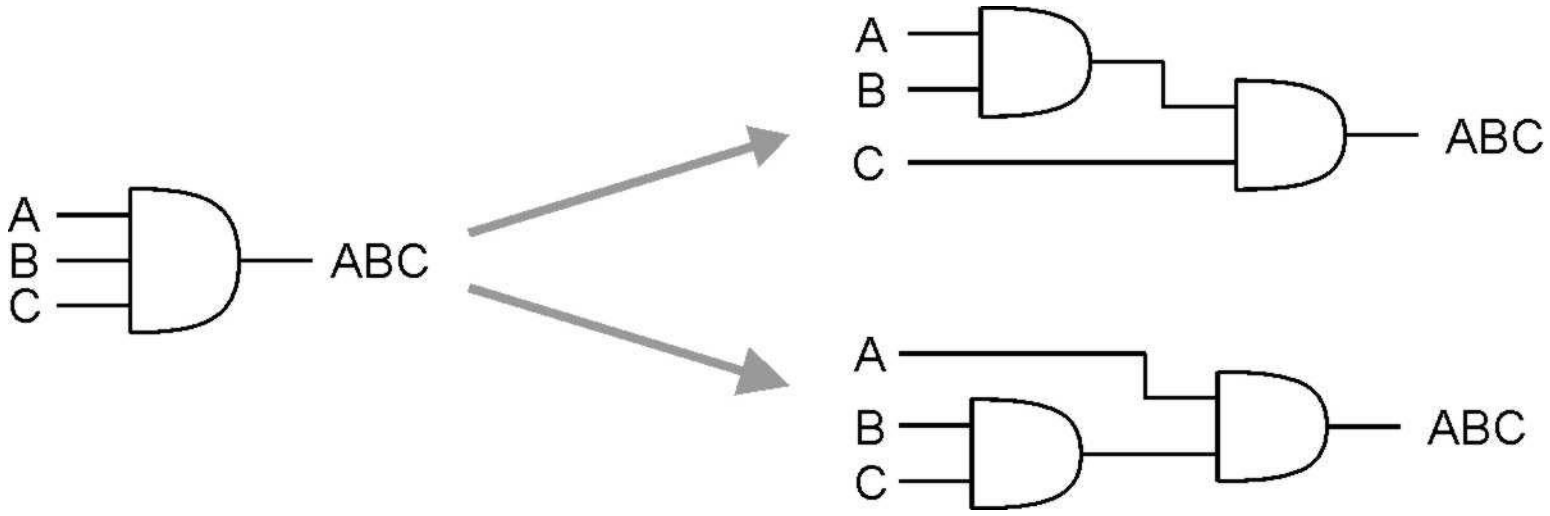


More than 2 Inputs?

AND/OR can take any number of inputs.

- **AND = 1 if all inputs are 1.**
- **OR = 1 if any input is 1.**
- **Similar for NAND/NOR.**

Can implement with multiple two-input gates, or with single CMOS circuit.



Note for all

- **HW2 is due today (1st Feb)**
- **HW3 is out**
- **Solutions for HW2 will be posted after HW2 is graded**

Note for all

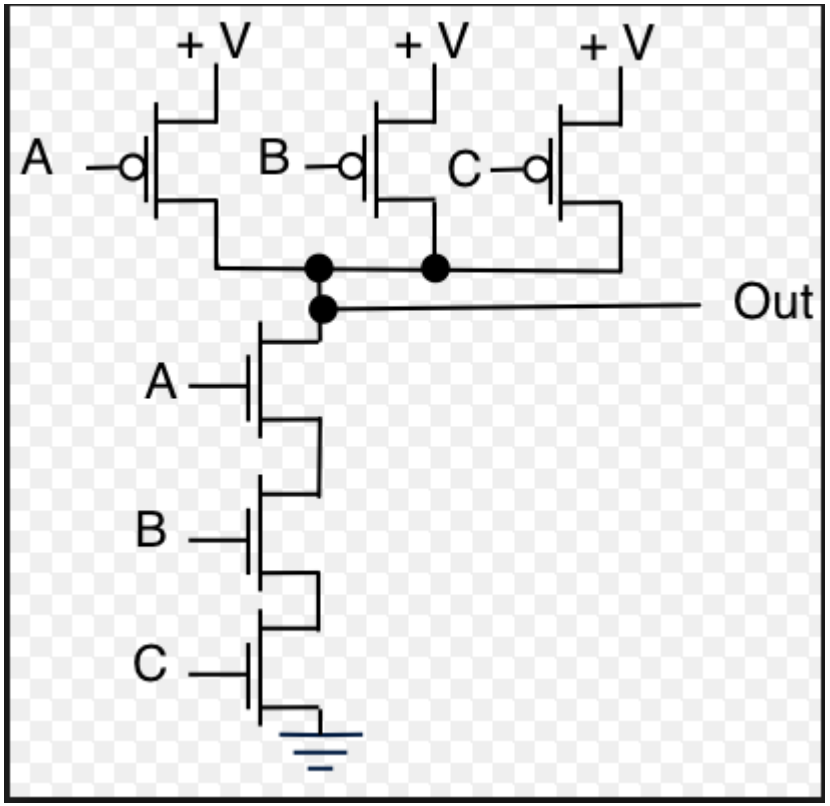
- **Exam 1 will be held on 10th Feb (Education Sciences 204)**
- **50 min exam**
- **NO**
  - **Books, notes, calculator or electronic devices of any sort**

Points Covered so far..

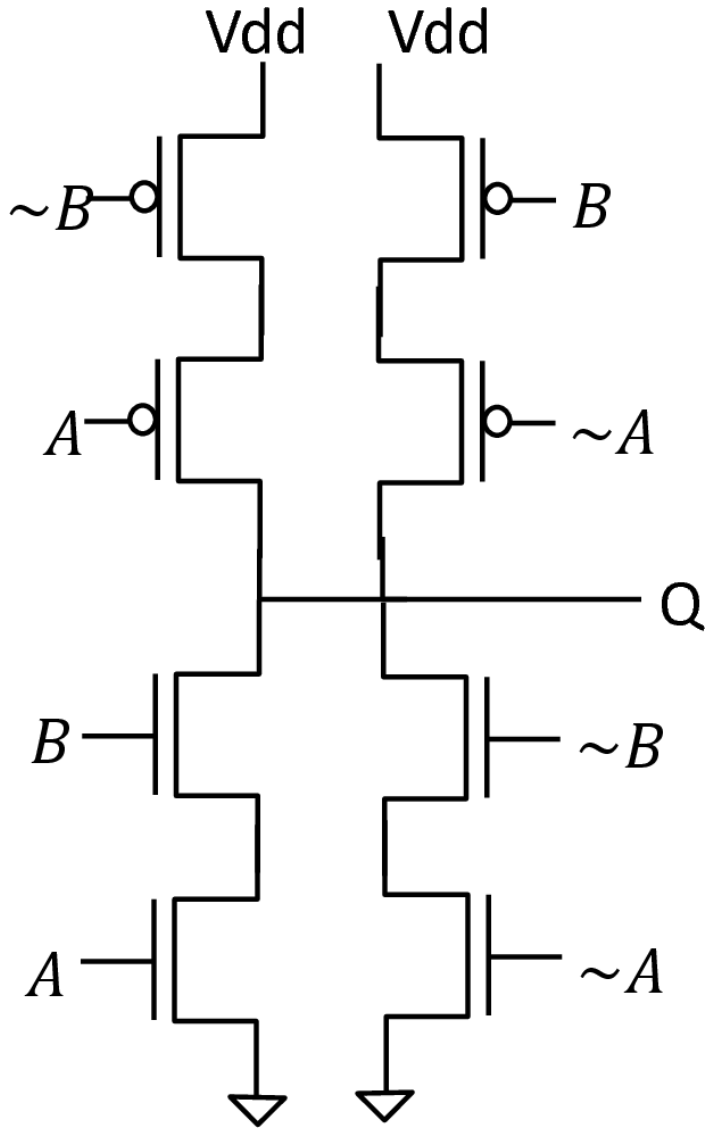
- Semiconductor properties of silicon
- Using silicon to build transistor
  - p type transistor
  - n type transistor
- Using transistors to build logic gates
  - NAND
  - NOR
  - NOT

Multiple input logic gates

3-input NAND gate with CMOS.



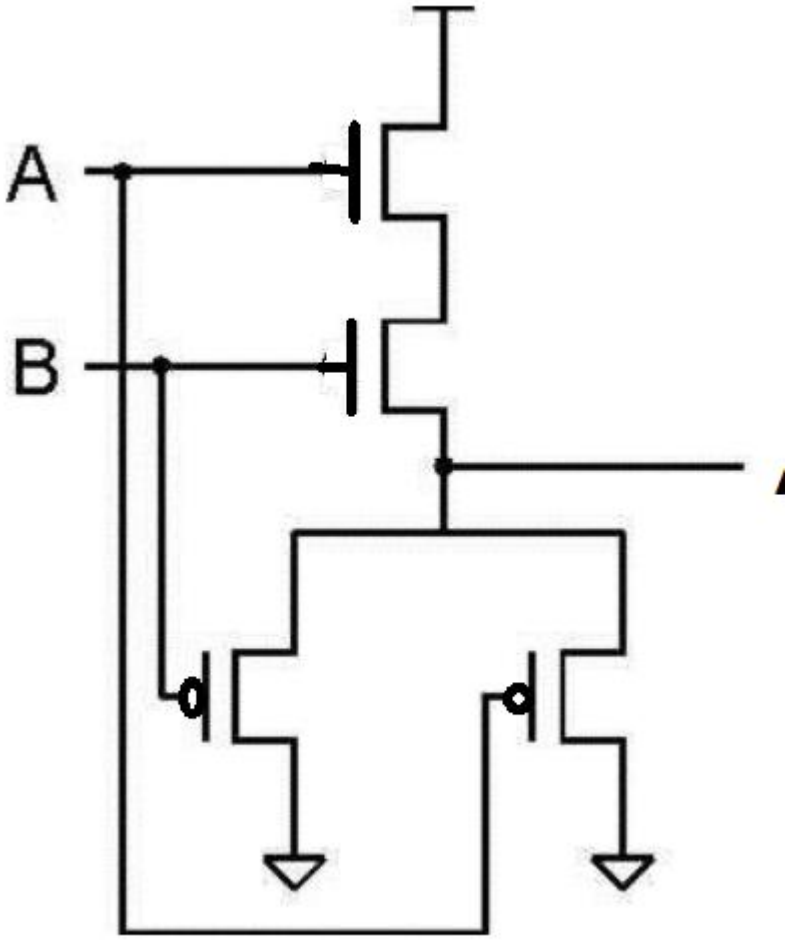
Take-home exercise
Fill out the truth table for the following transistor circuit



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

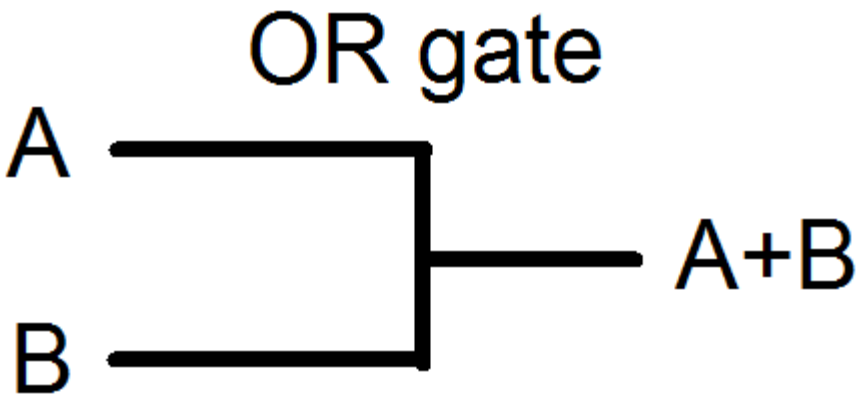
Q= A (xor) B

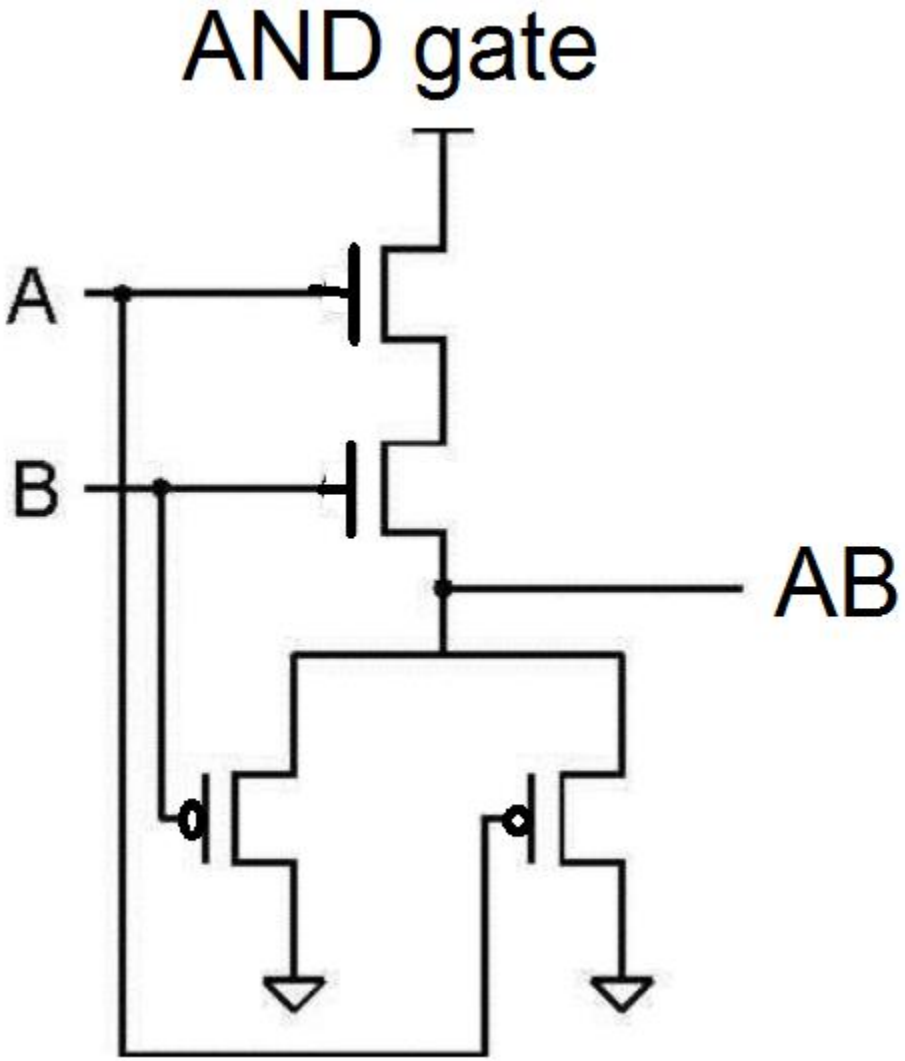
Fill out the truth table for the following transistor circuit

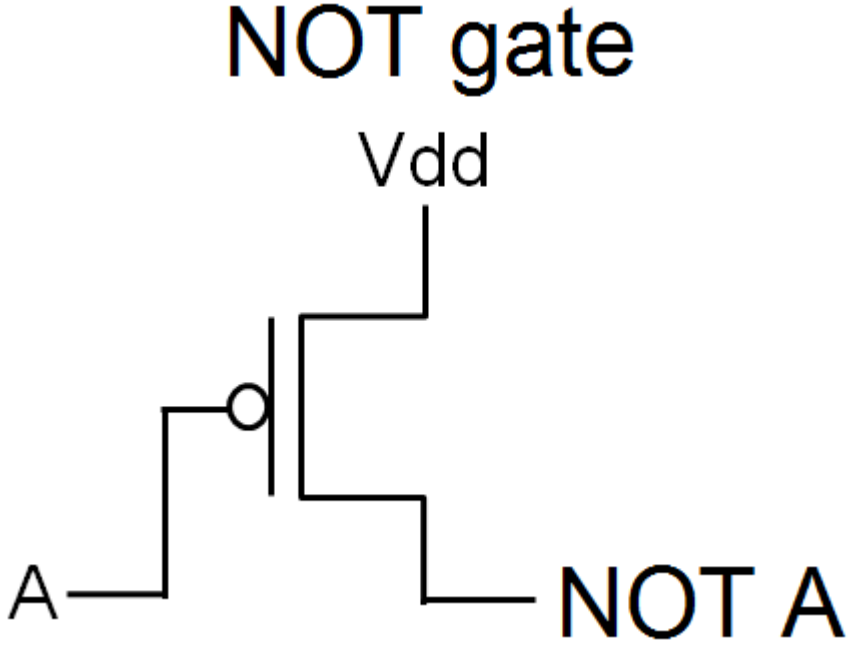


A	B	Q
0	0	
0	1	
1	0	
1	1	

Common Misconceptions



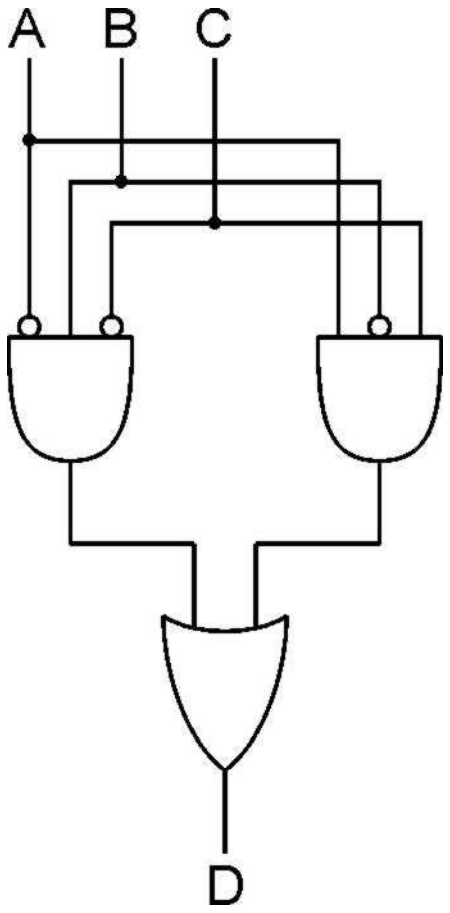




Logical Completeness

Can implement ANY truth table with AND, OR, NOT.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

Practice

Implement the following truth table.

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Practice

Implement the following truth table.

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

C= A`B` + A`B + AB`

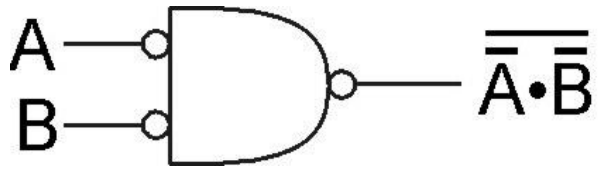
C= A`(B`+B) +AB`

C= A` + AB`

DeMorgan's Law

Converting AND to OR (with some help from NOT)

Consider the following gate:



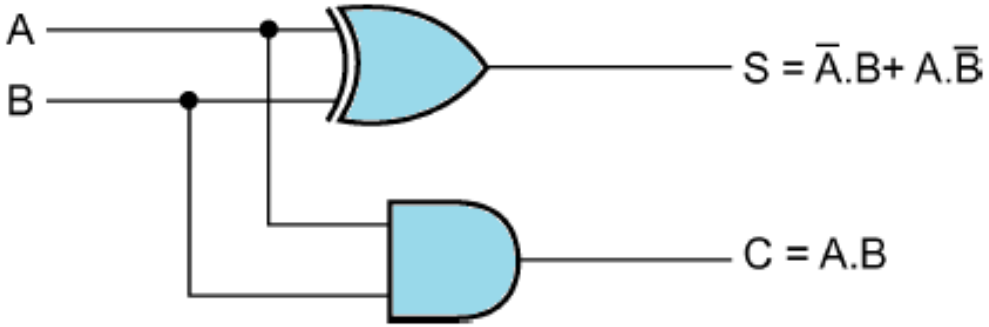
A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Same as A+B

To convert AND to OR (or vice versa), invert inputs and output.

Half Adder

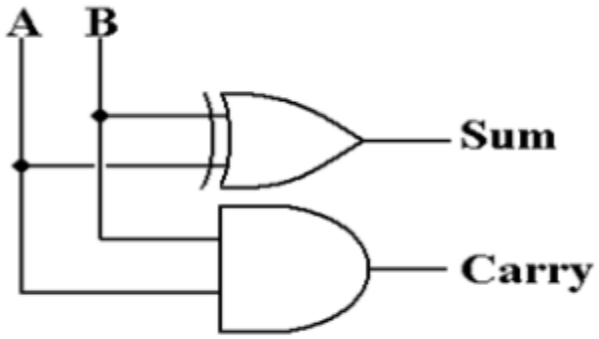
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

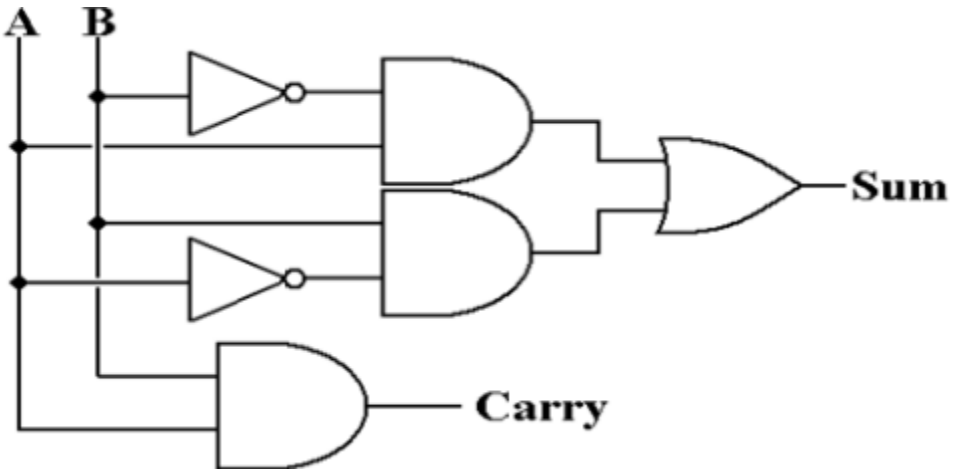


S= A (XOR) B

C= AB

Half Adder

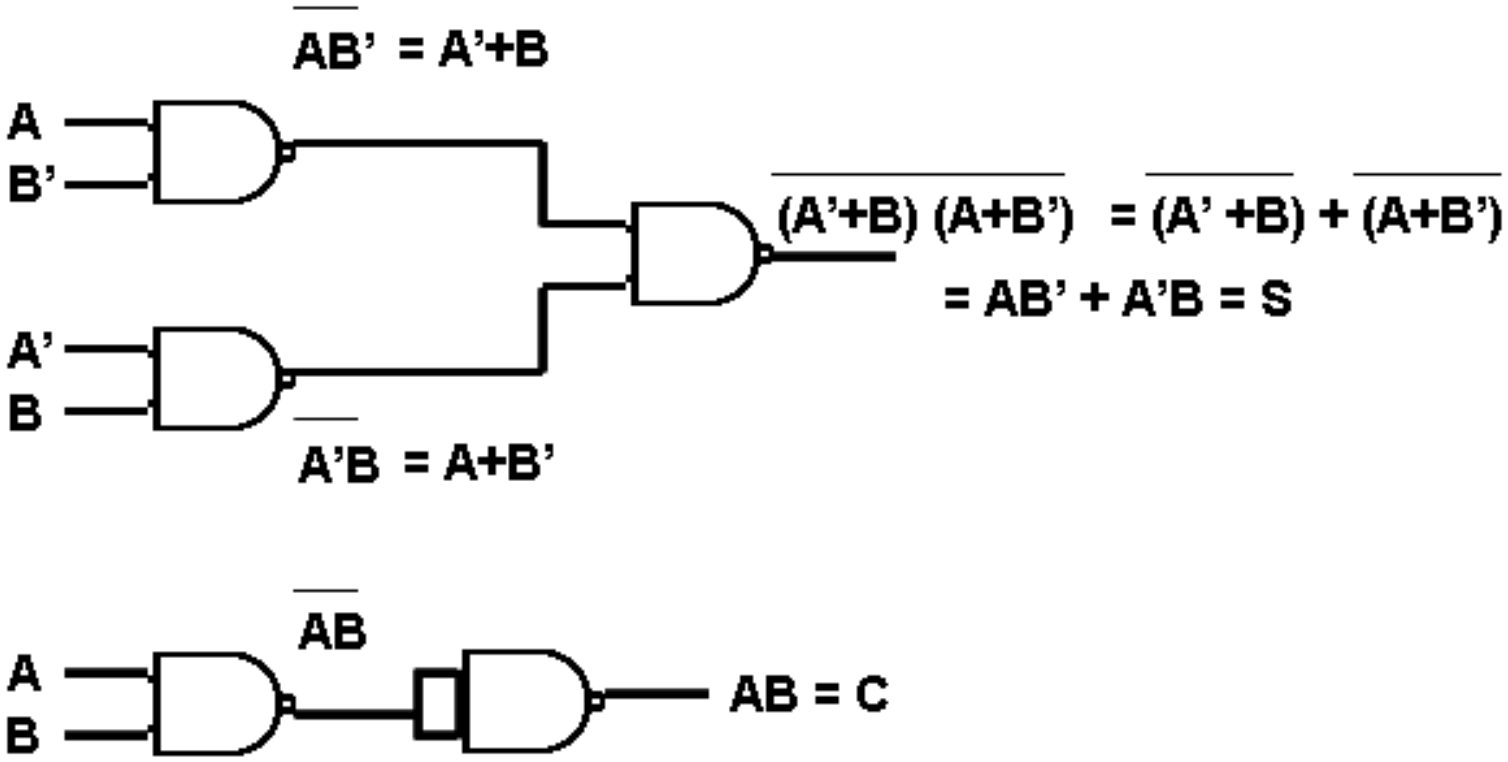




S= A (XOR) B

C= AB

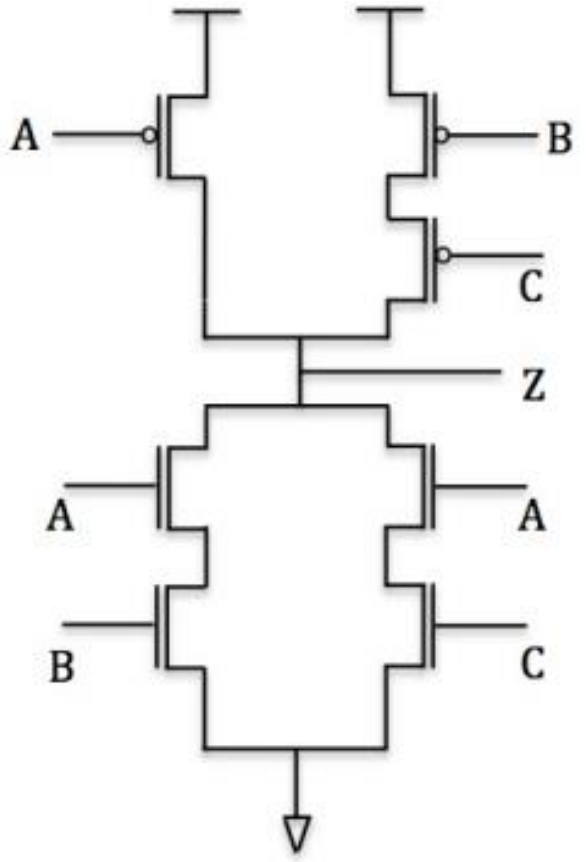
Half Adder



S= A (XOR) B

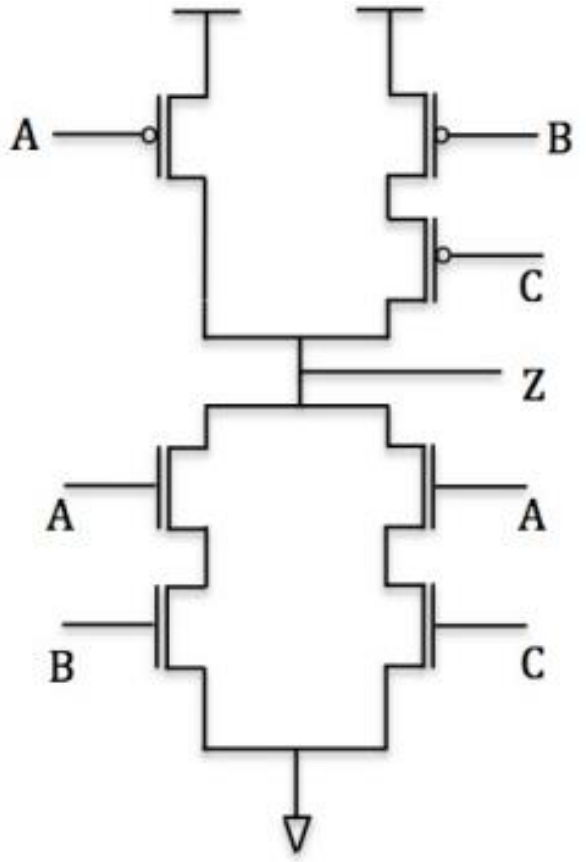
C= AB

Fill out the following truth table



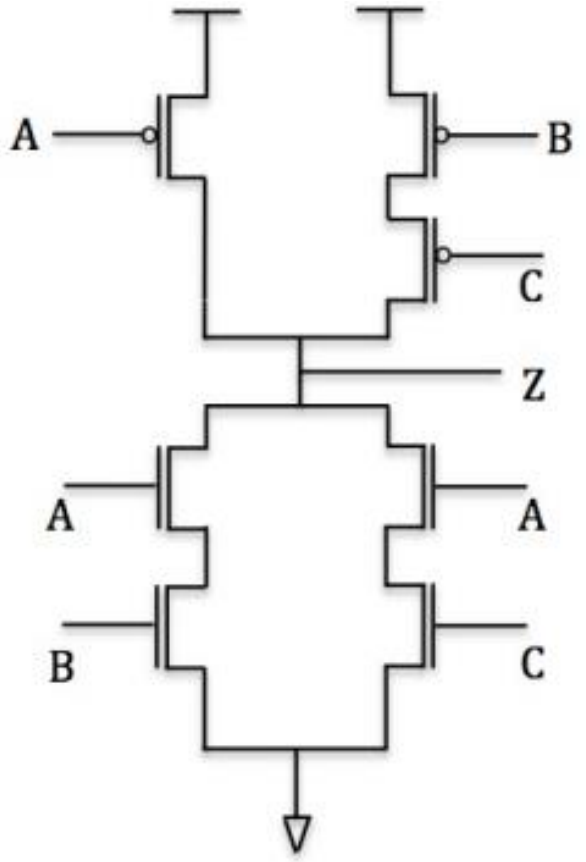
A	B	C	Z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Fill out the following truth table



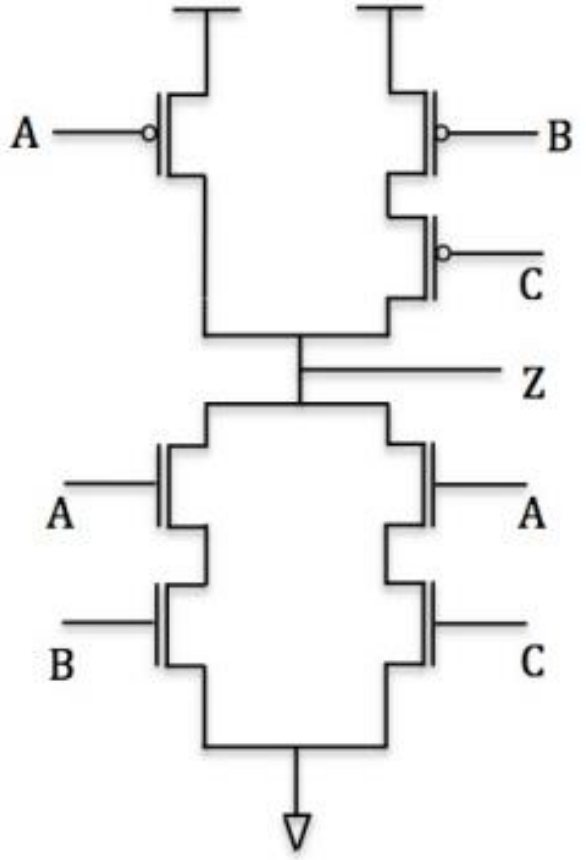
A	B	C	Z
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

Write the function of Z



A	B	C	Z
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

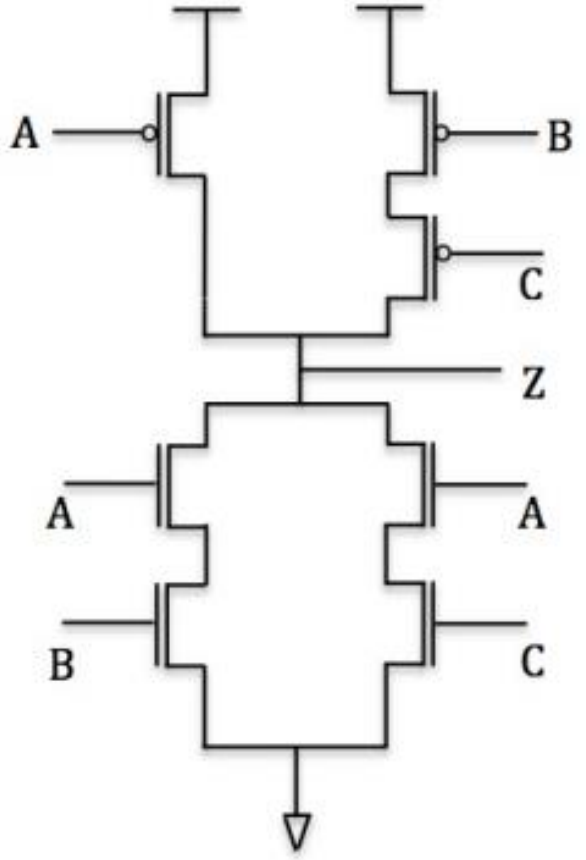
Write the function of Z



A	B	C	Z
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

Z= A`B`C` + A`B`C + A`BC` + A`BC + AB`C`

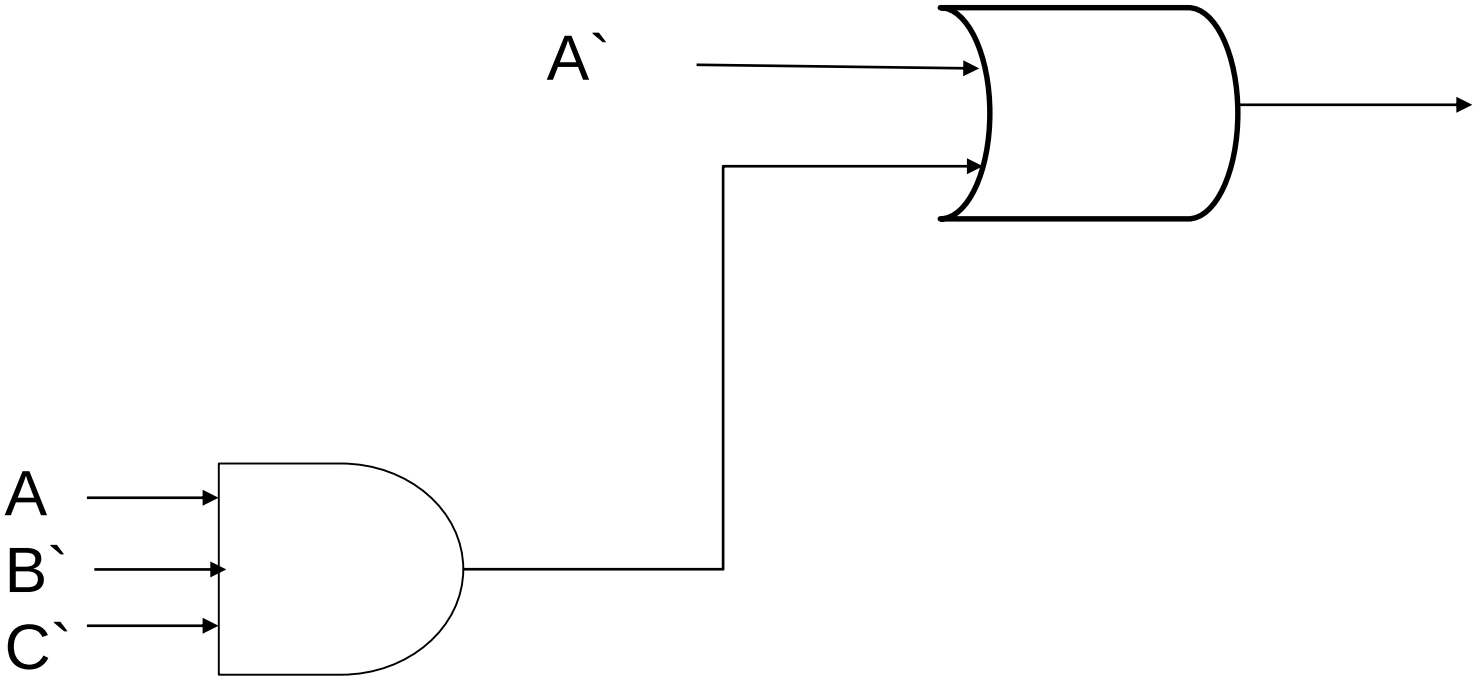
Draw the gate implementation of Z



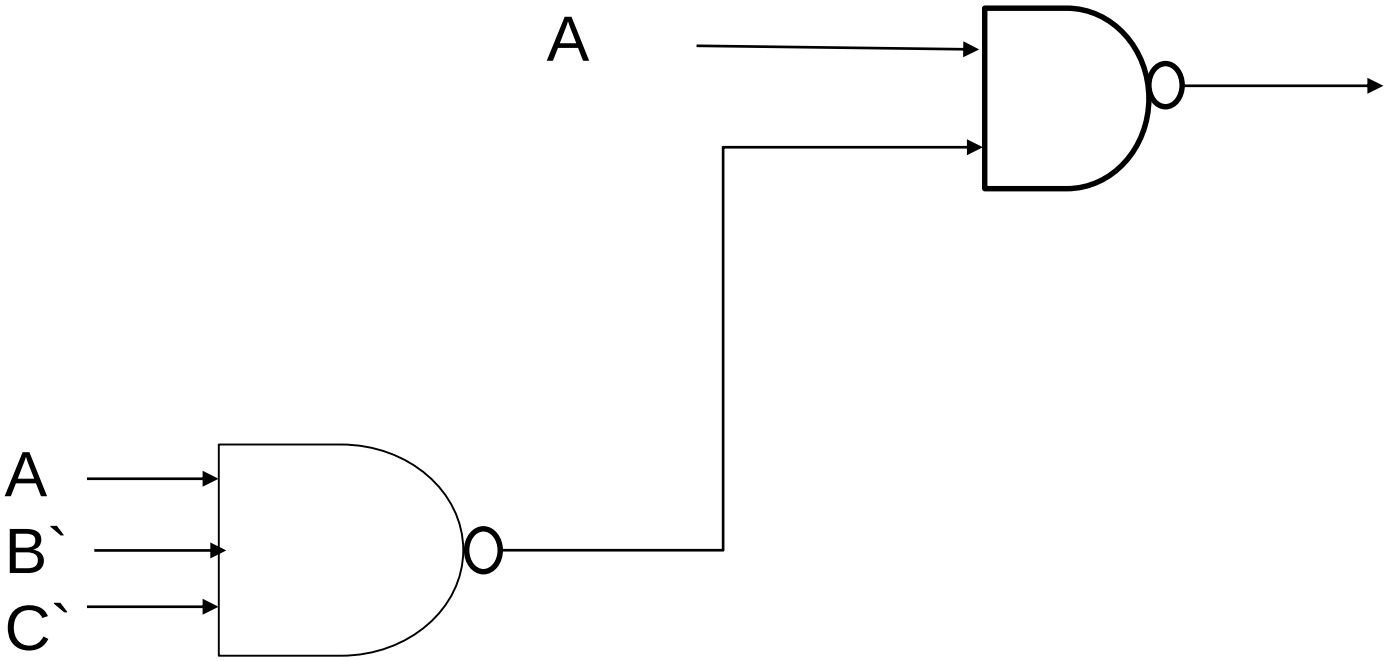
A	B	C	Z
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

Z= A`+ AB`C`

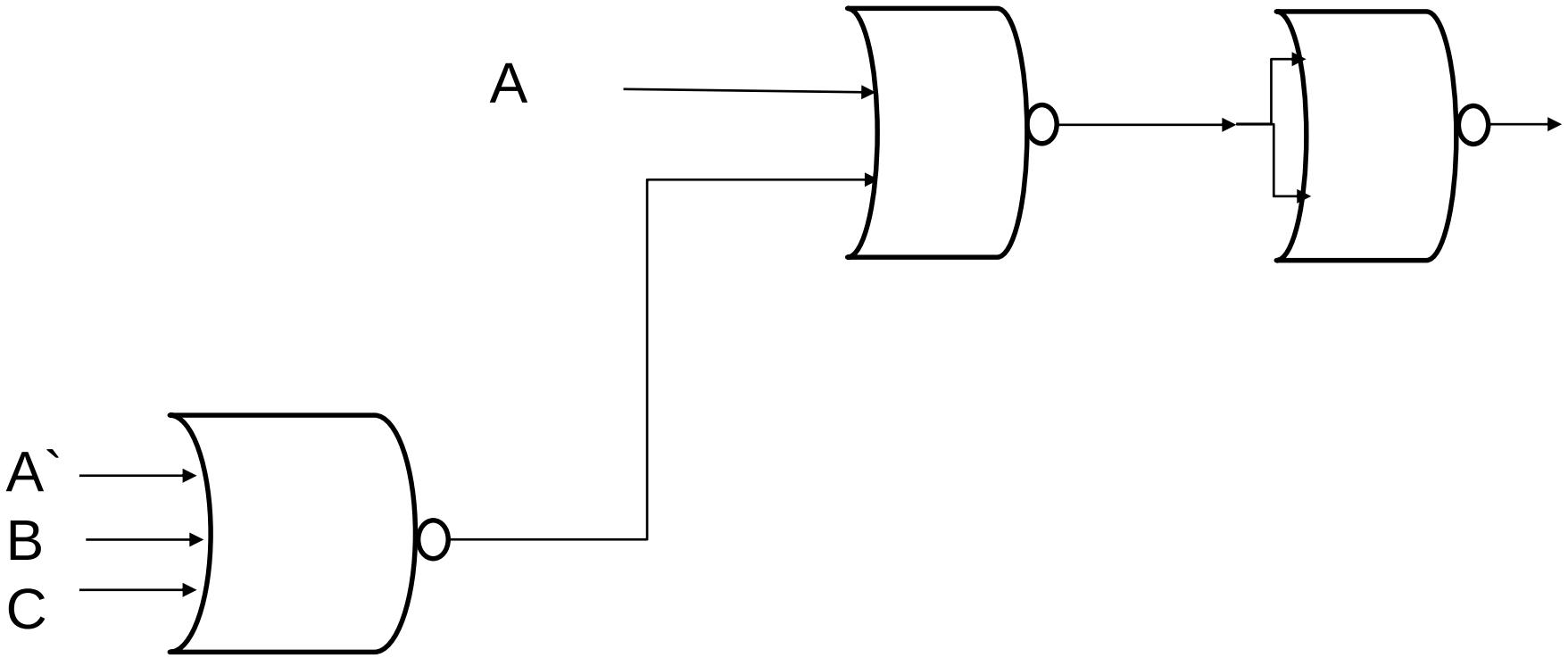
Gate implementation



Gate implementation (NAND)



Gate implementation (NOR)



Summary

MOS transistors are used as switches to implement logic functions.

- **N-type: connect to GND, turn on (with 1) to pull down to 0**
- **P-type: connect to +2.9V, turn on (with 0) to pull up to 1**

Basic gates: NOT, NOR, NAND

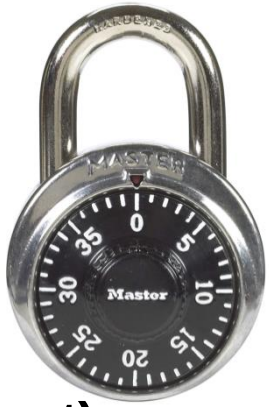
- **Logic functions are usually expressed with AND, OR, and NOT**

Properties of logic gates

- **Completeness**
  - **can implement any truth table with AND, OR, NOT**
- **DeMorgan's Law**
  - **convert AND to OR by inverting inputs and output**

Building Functions from Logic Gates

We've already seen how to implement truth tables using AND, OR, and NOT -- an example of *combinational logic*.



Combinational Logic Circuit

- **output depends only on the current inputs**
- **stateless**

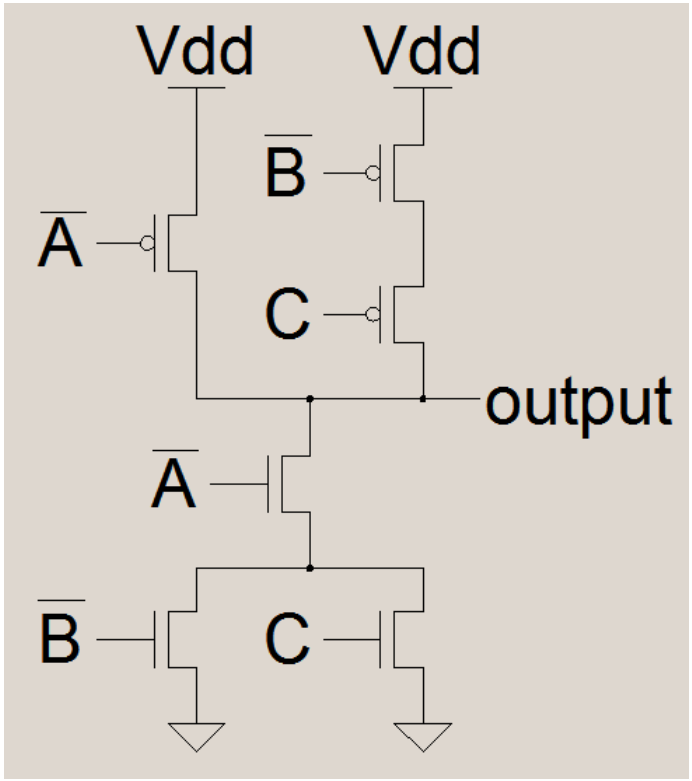
Sequential Logic Circuit

- **output depends on the sequence of inputs (past and present)**
- **stores information (state) from past inputs**

We'll first look at some useful combinational circuits, then show how to use sequential circuits to store information.

In-class exercise

Fill out the truth table for the following transistor circuit



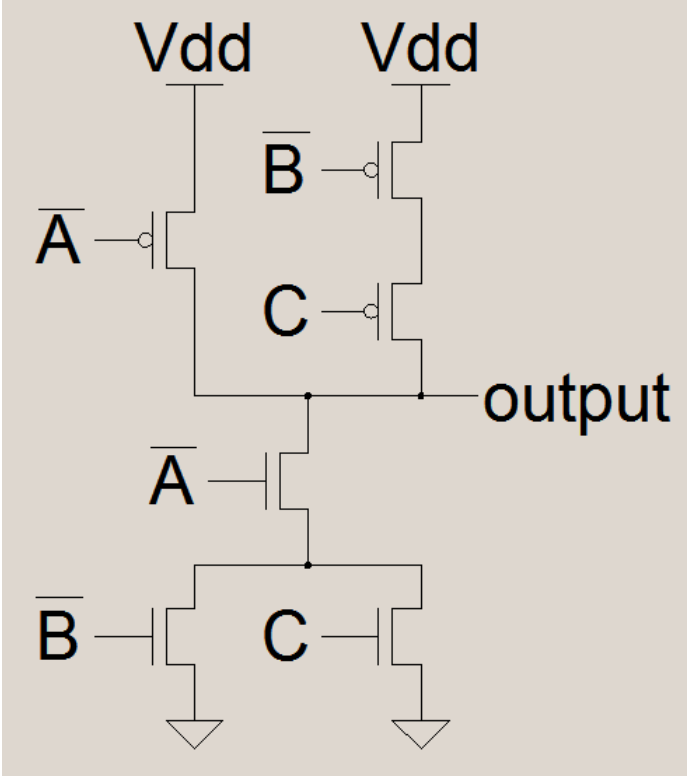
A	B	C	output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Use DeMorgan's Law to simplify

NOT(NOT(A) AND (NOT(B) OR C))

In-class exercise

Fill out the truth table for the following transistor circuit



A	B	C	output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Use DeMorgan's Law to simplify

NOT(NOT(A) AND (NOT(B) OR C))

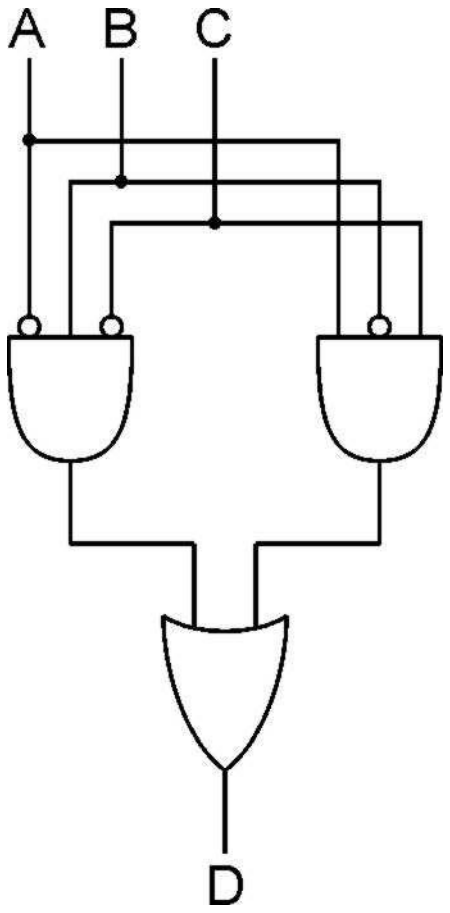
NOT(NOT(A)) OR NOT(NOT(B) OR C)

A OR (B AND NOT(C))

Logical Completeness

Can implement ANY truth table with AND, OR, NOT.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



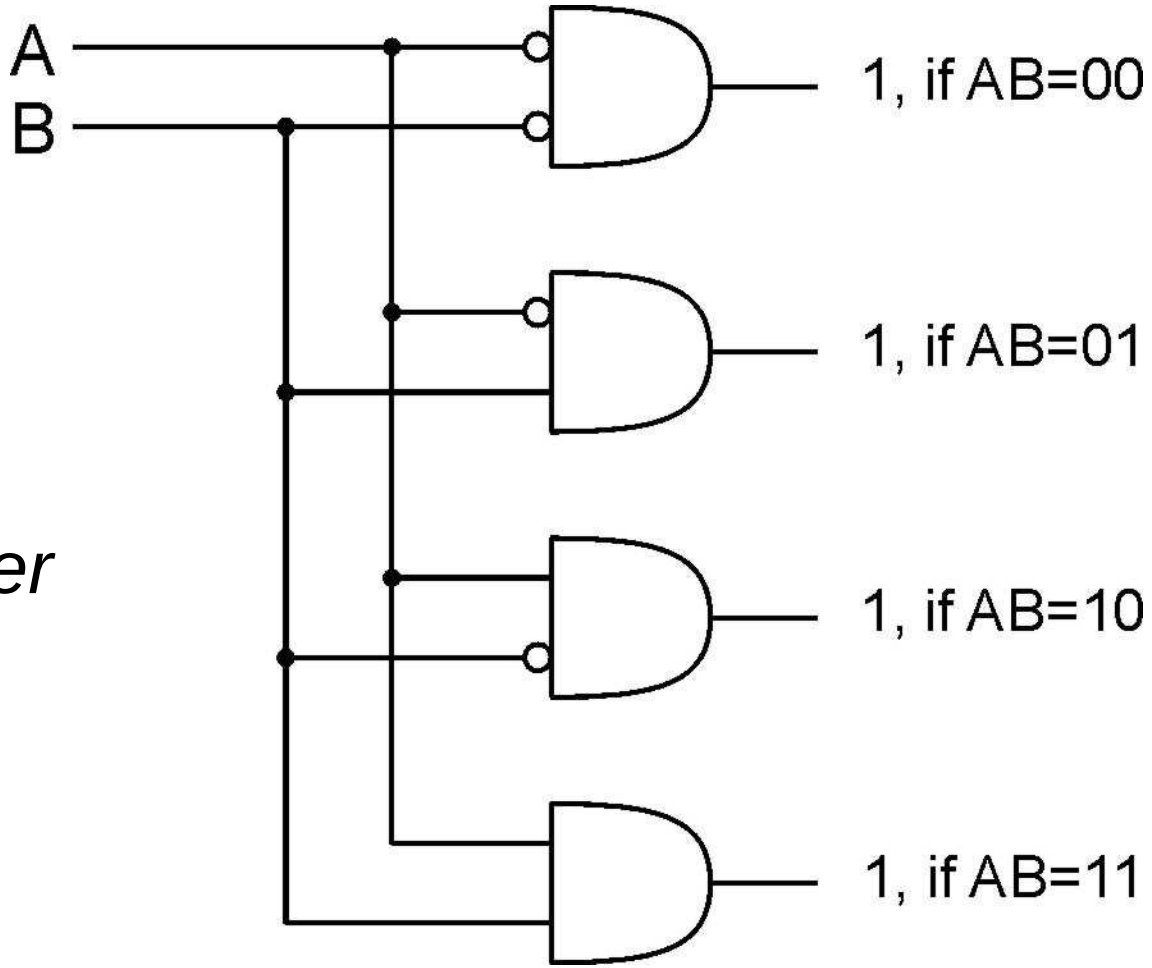
1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

Decoder

n inputs, 2^n outputs

- **exactly one output is 1 for each possible input pattern**

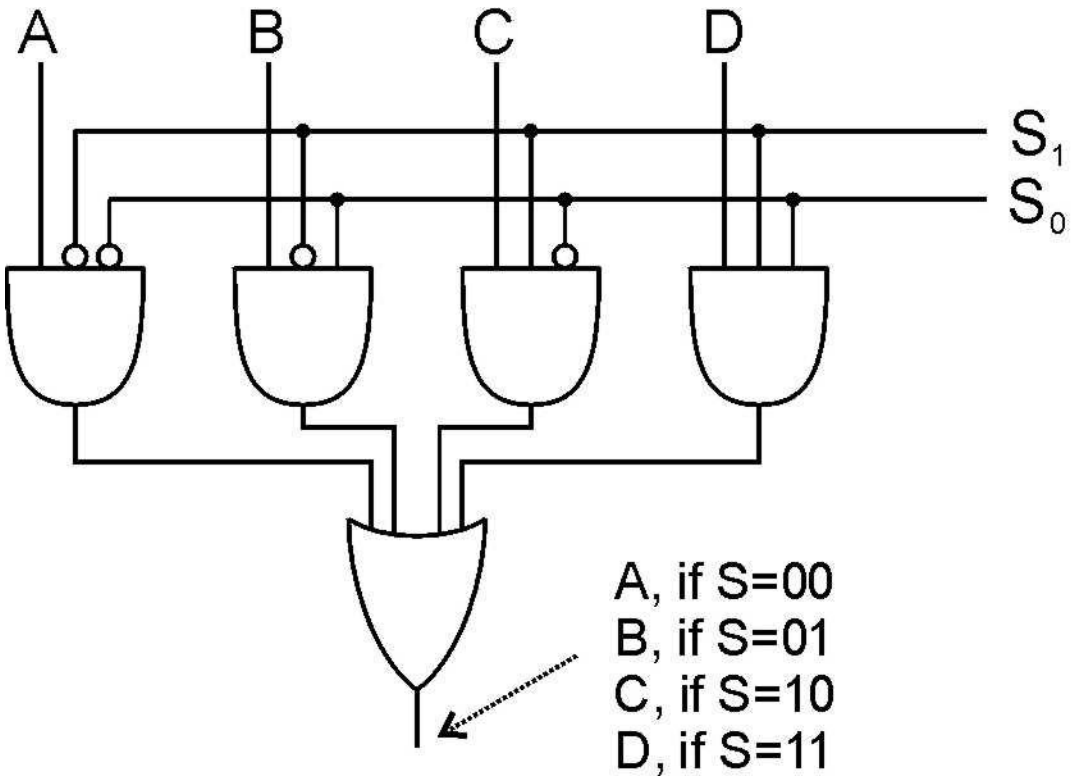


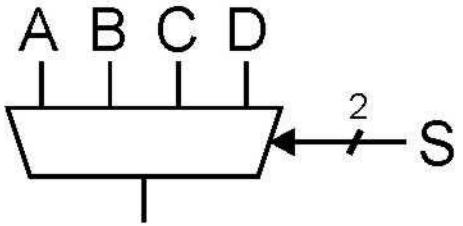
2-bit decoder

Multiplexer (MUX)

n-bit selector and 2^n inputs, one output

- output equals one of the inputs, depending on selector

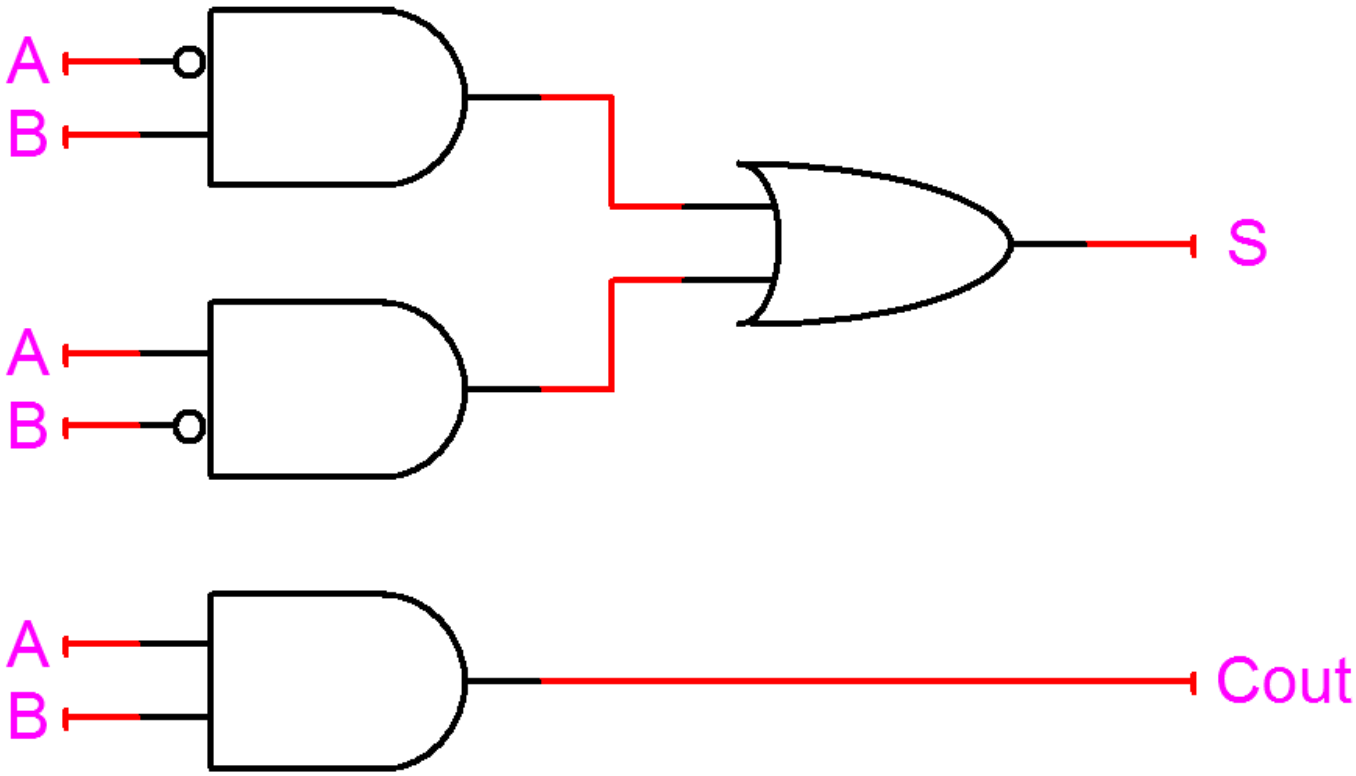




4-to-1 MUX

Half Adder

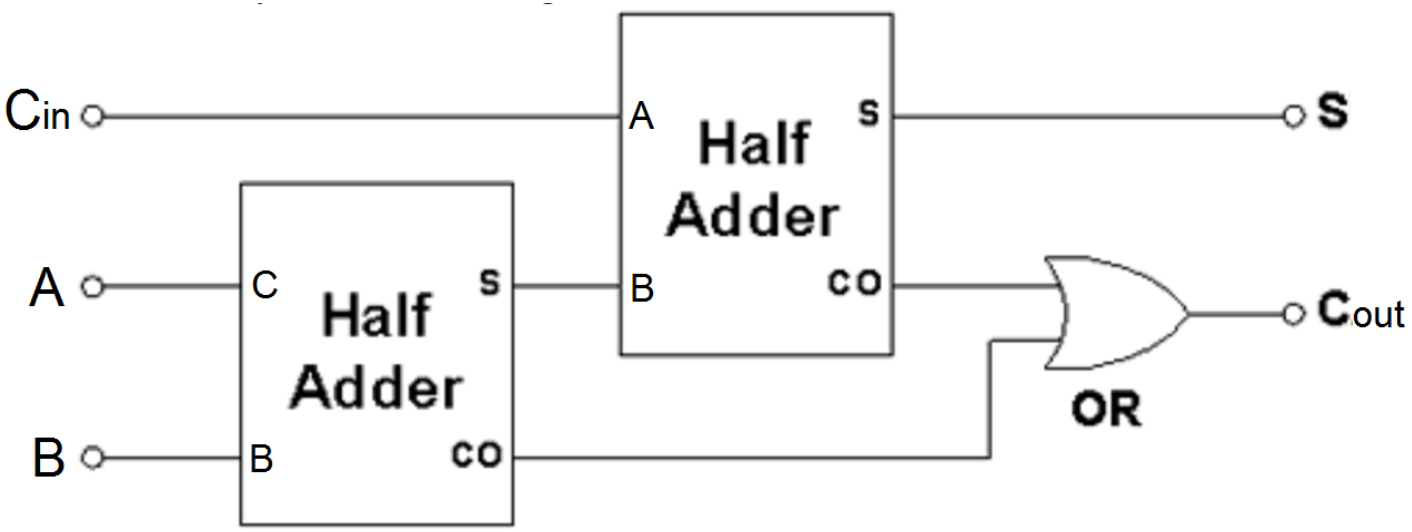
Add two bits,
produce one-bit sum and carry-out.



A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder from Half Adder

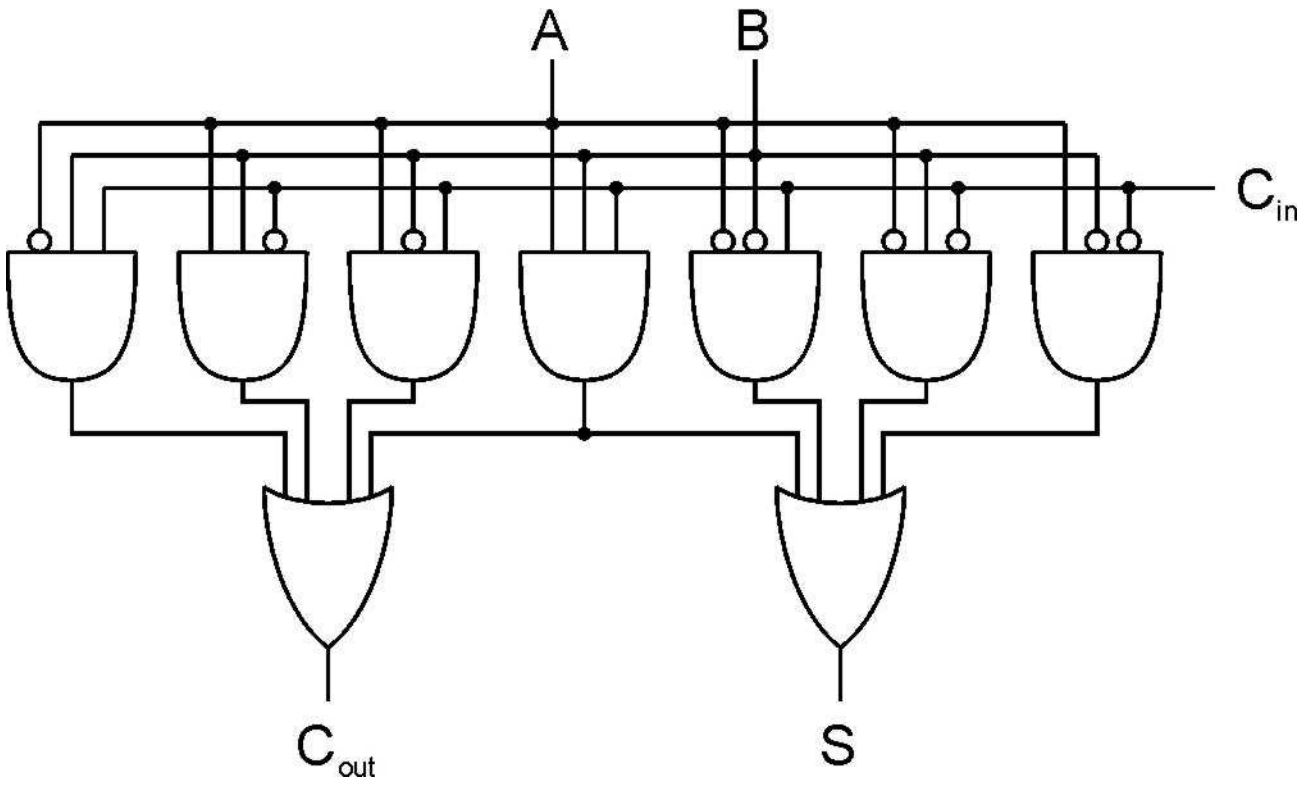
Add two bits and carry-in,
produce one-bit sum and carry-out.



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

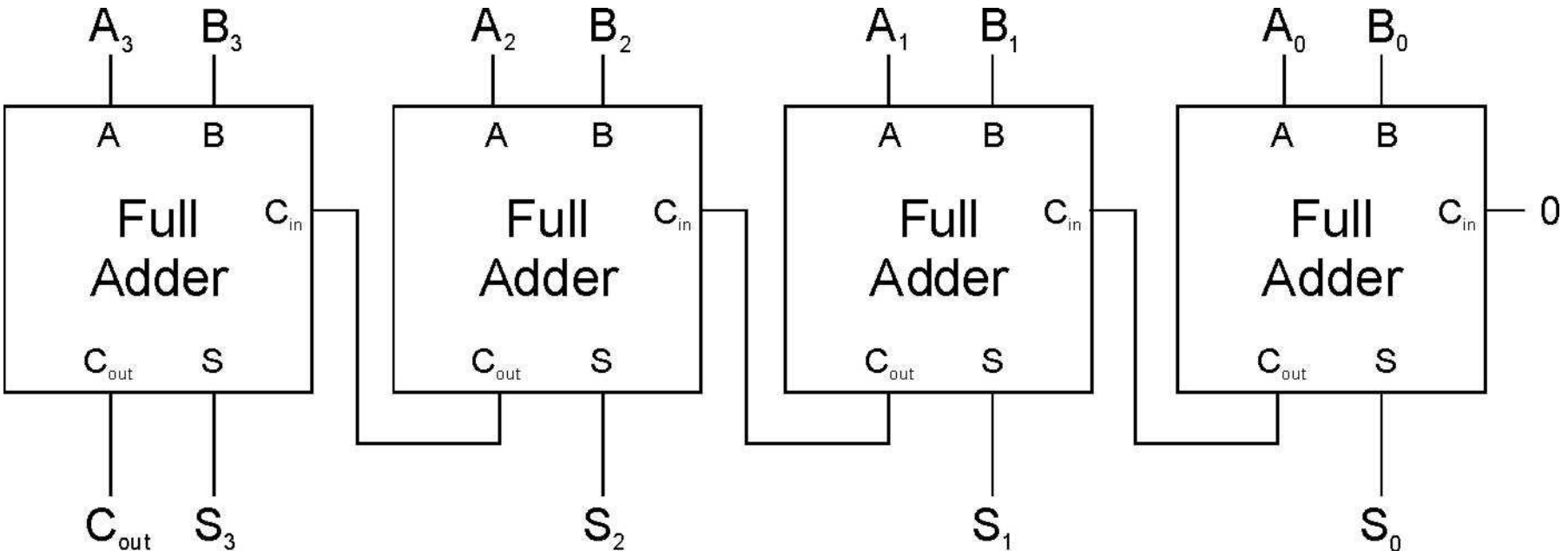
Full Adder

Add two bits and carry-in,
produce one-bit sum and carry-out.



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

Four-bit Adder



Combinational vs. Sequential

Combinational Circuit

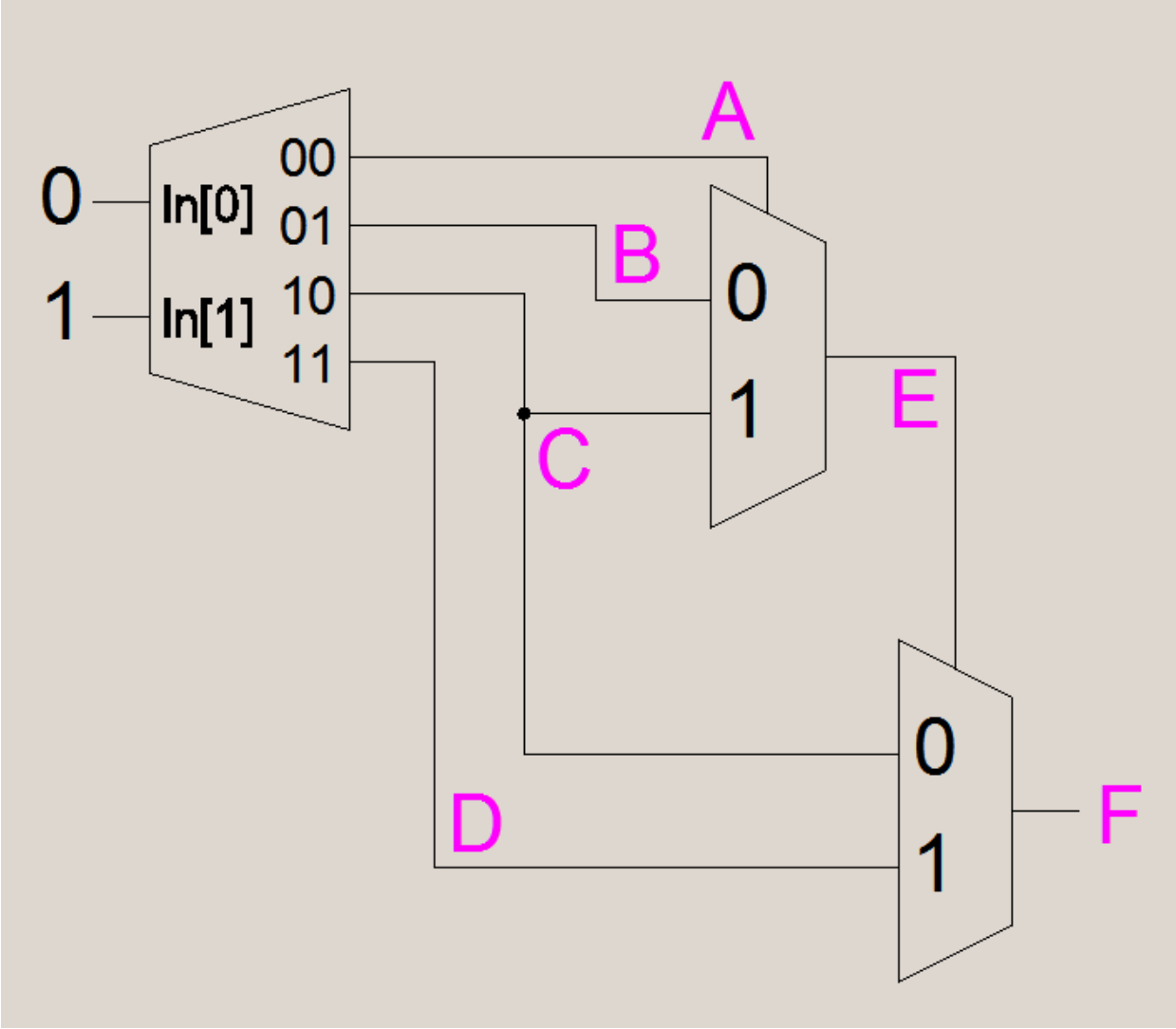
- always gives the same output for a given set of inputs
  - ex: adder always generates sum and carry, regardless of previous inputs

Sequential Circuit

- stores information
- output depends on stored information (state) plus input
  - so a given input might produce different outputs, depending on the stored information
- *example:* ticket counter
  - advances when you push the button
  - output depends on previous state
- useful for building “memory” elements and “state machines”

In-Class Exercise

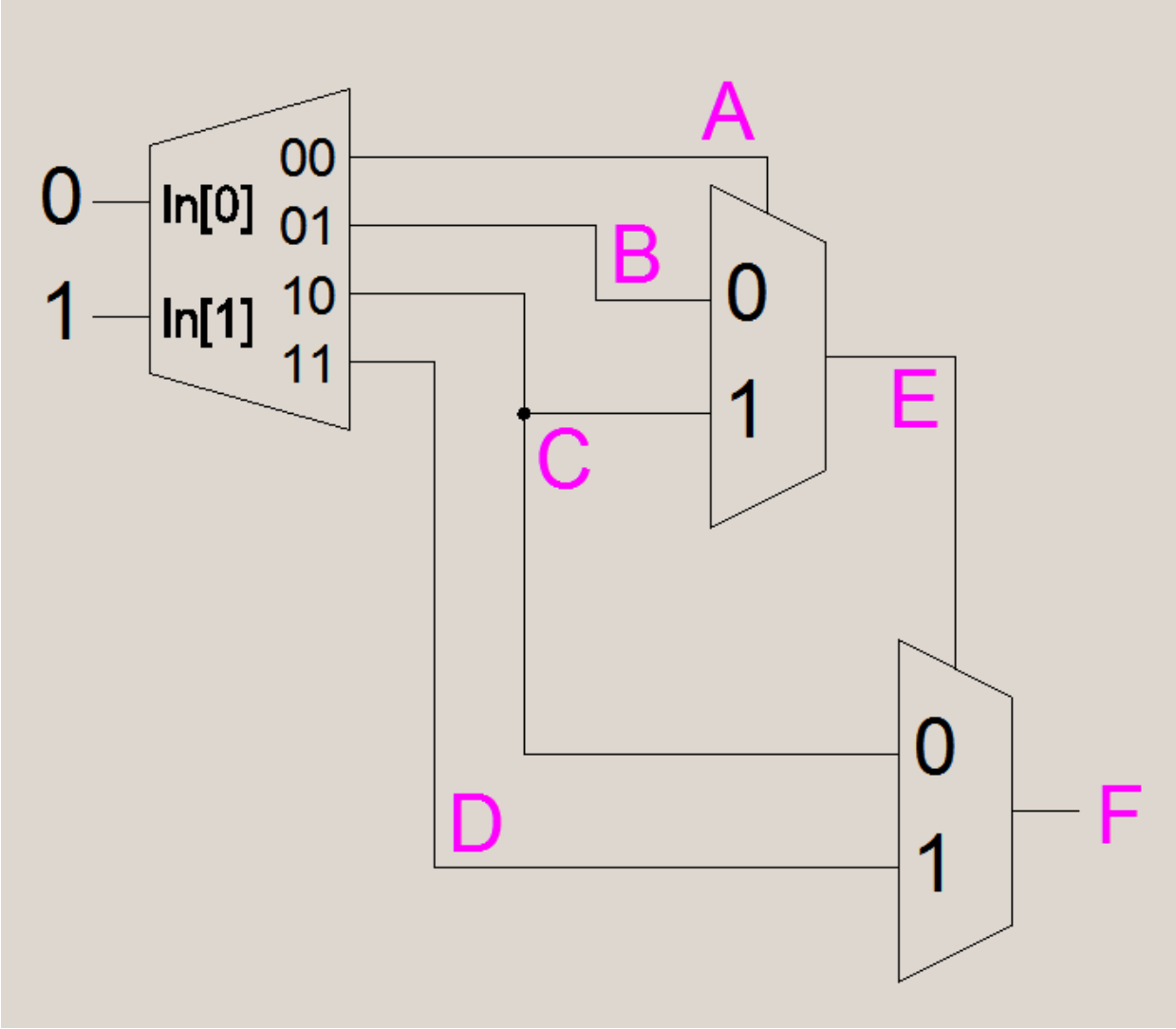
- **Find A, B, C, D, E, and F**



Note: Bits are numbered starting from 0 from right to left.
So if X = 10110, then
X[4] = 1,
X[3] = 0,
X[2] = 1,
X[1] = 1, and
X[0] = 0.

In-Class Exercise Solutions

- **Find A, B, C, D, E, and F**



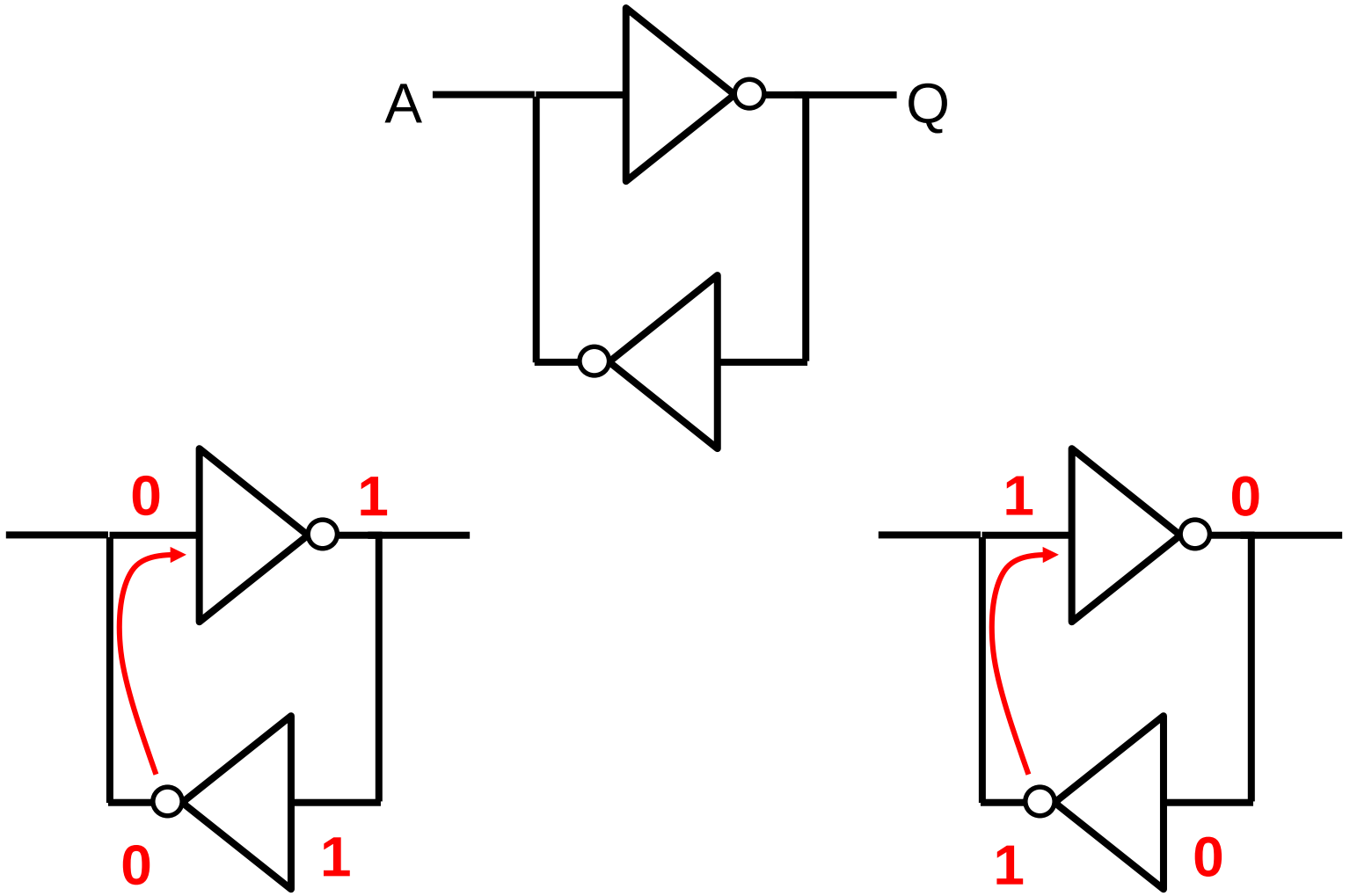
Note: Bits are numbered starting from 0 from right to left.
So if X = 10110, then
X[4] = 1,
X[3] = 0,
X[2] = 1,
X[1] = 1, and
X[0] = 0.

A=0
B=0
C=1
D=0
E=0
F=1

Cross-coupled invertors

Good news: holds on to value

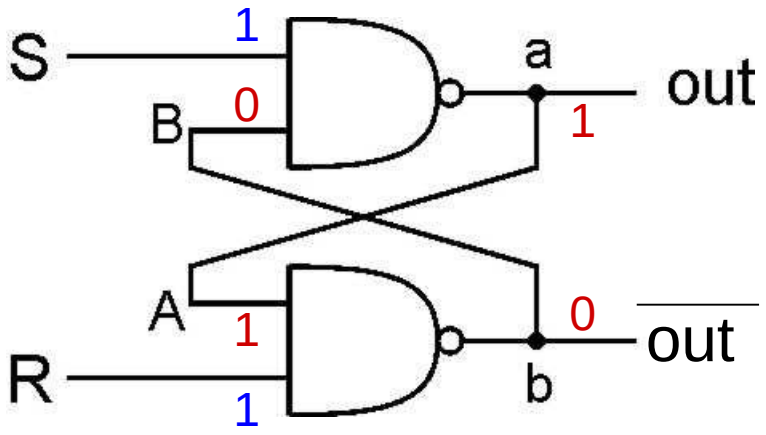
Bad news: How do we write to it?

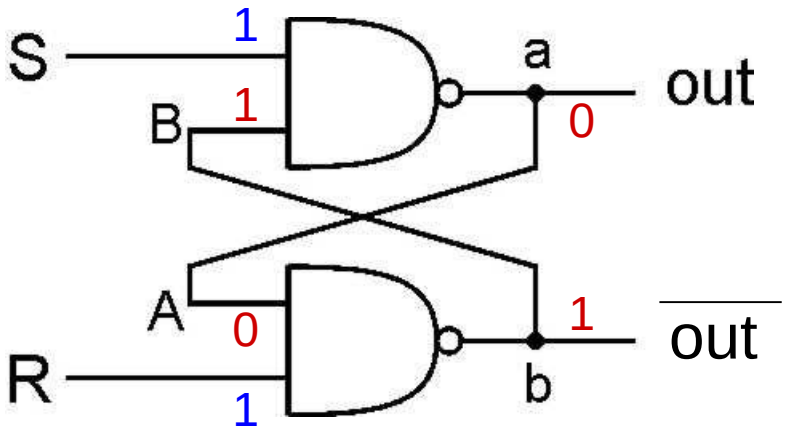


R-S Latch: Simple Storage Element

R is used to "reset" or "clear" the element – set it to zero.

S is used to "set" the element – set it to one.



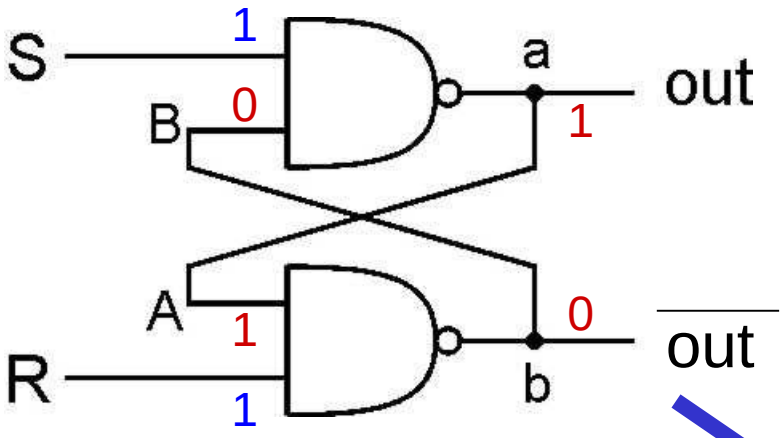


If both R and S are one, out could be <u>either</u> zero or one.

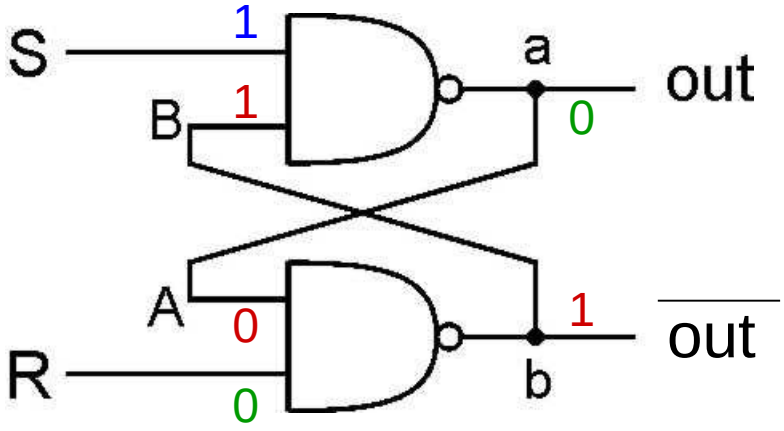
- **"quiescent" state -- holds its previous value**
- **note: if a is 1, b is 0, and vice versa**

Clearing the R-S latch

Suppose we start with output = 1, then change R to zero.



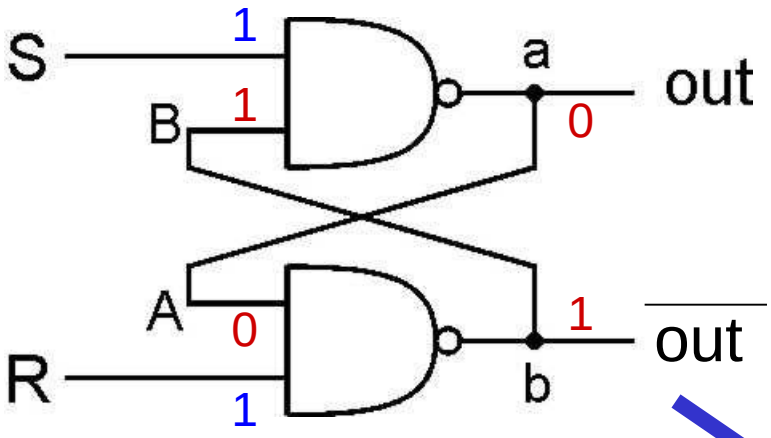
Output changes to zero.



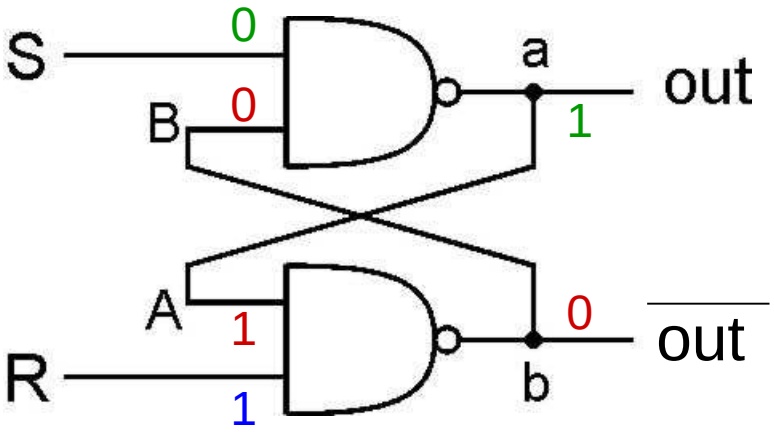
Then set R=1 to "store" value in quiescent state.

Setting the R-S Latch

Suppose we start with output = 0, then change S to zero.



Output changes to one.



Then set S=1 to "store" value in quiescent state.

R-S Latch Summary

R = S = 1

- **hold current value in latch**

S = 0, R=1

- **set value to 1**

R = 0, S = 1

- **set value to 0**

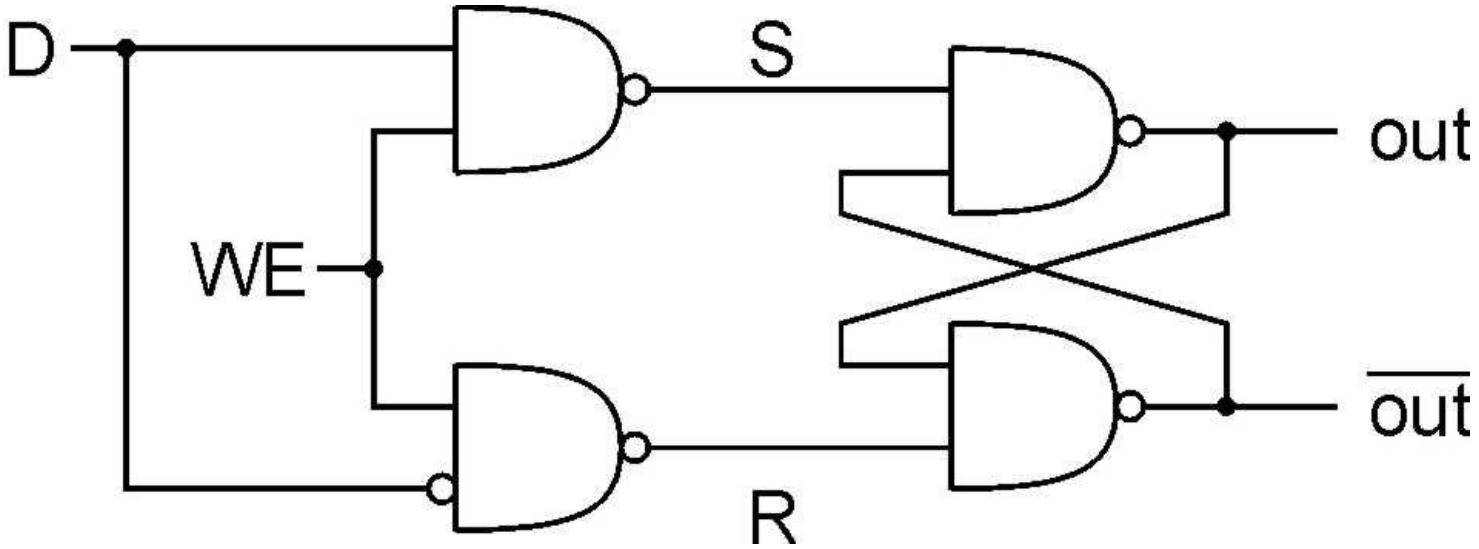
R = S = 0

- **both outputs equal one**
- *Don't do it!*

Gated D-Latch

Two inputs: D (data) and WE (write enable)

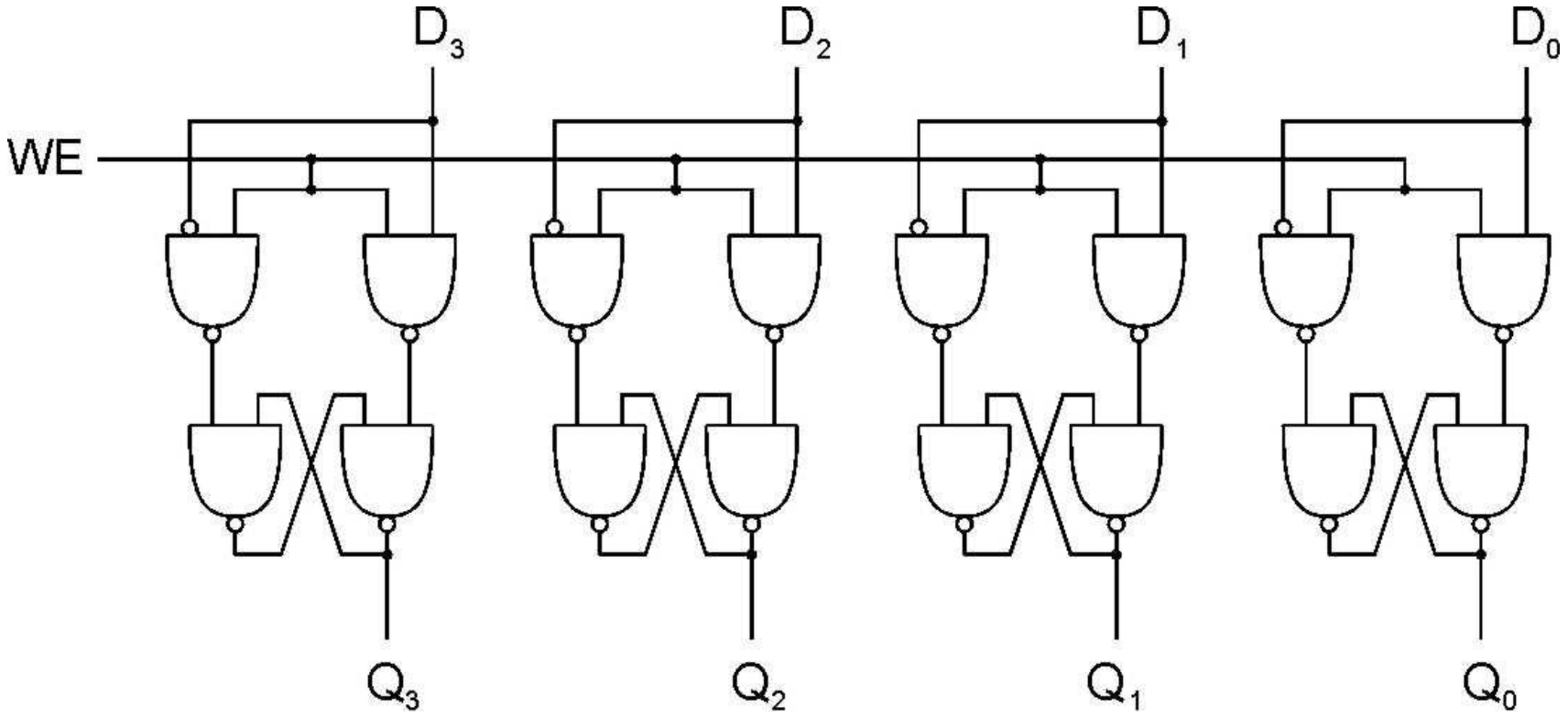
- when WE = 1, latch is set to value of D
  - S = NOT(D), R = D
- when WE = 0, latch holds previous value
  - S = R = 1



Register

A register stores a multi-bit value.

- **We use a collection of D-latches, all controlled by a common WE.**
- **When WE=1, n-bit value D is written to register.**

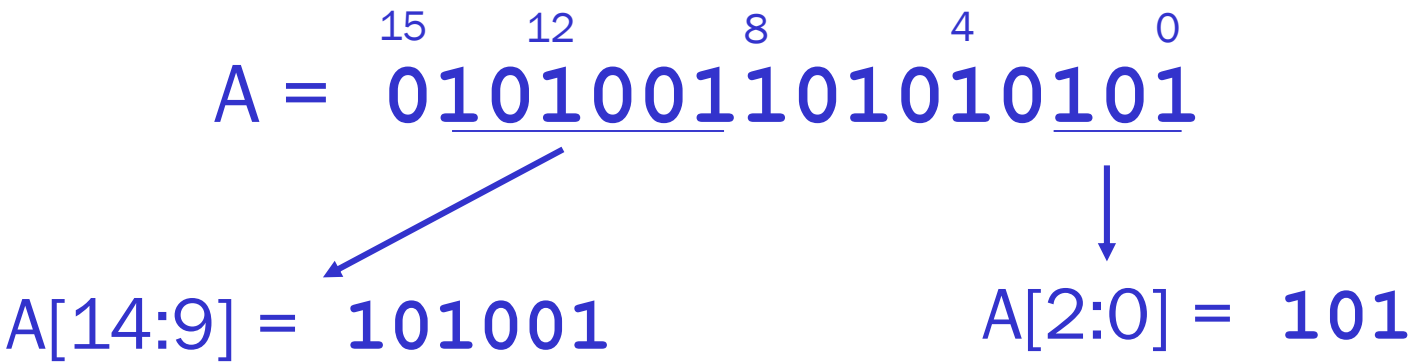


Representing Multi-bit Values

Number bits from right (0) to left (n-1)

- **just a convention -- could be left to right, but must be *consistent***

Use brackets to denote range:
D[l:r] **denotes bit** l **to bit** r**, from *left* to *right***

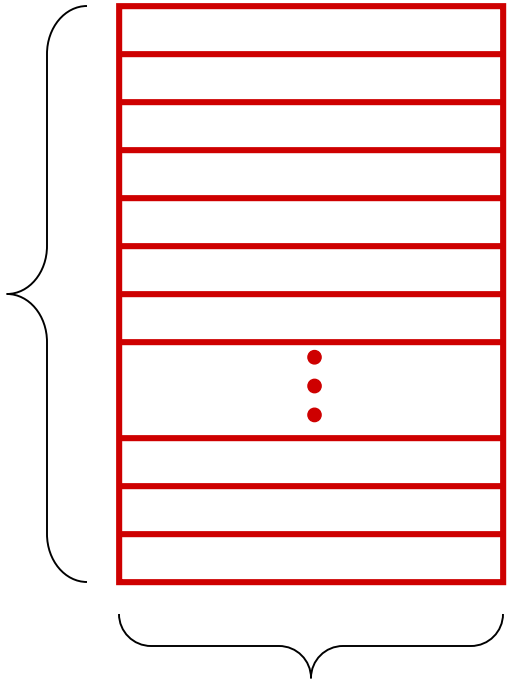


May also see A<14:9>**,**
especially in hardware block diagrams.

Memory

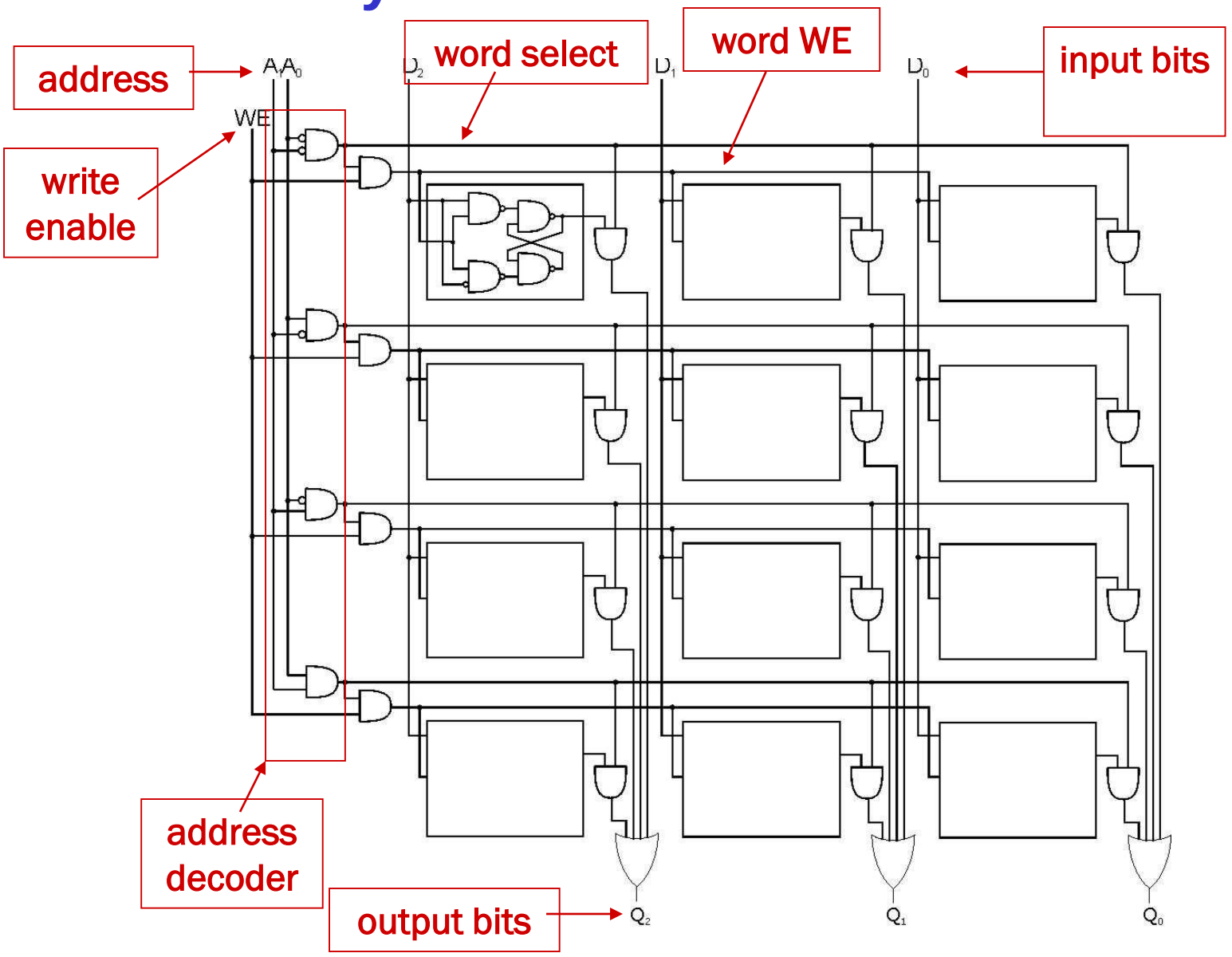
Now that we know how to store bits, we can build a memory – a logical *k* × *m* array of stored bits.

Address Space:
number of locations
(usually a power of 2)



Addressability:
number of bits per location
(e.g., byte-addressable)

2^2 x 3 Memory



More Memory Details

This is a not the way actual memory is implemented.

- **fewer transistors, much more dense, relies on electrical properties**

But the logical structure is very similar.

- **address decoder**
- **word select line**
- **word write enable**

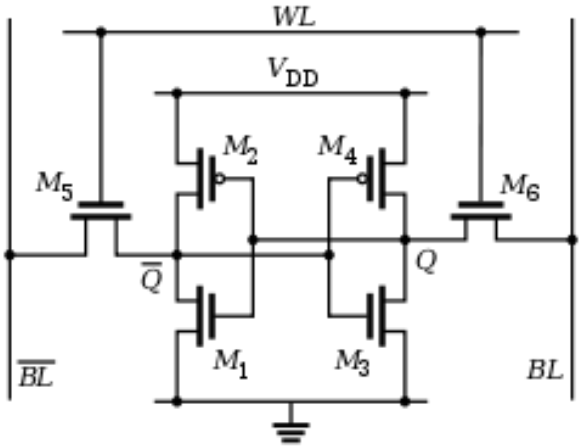
Also, non-volatile memories: ROM, PROM, flash, …

More Memory Details

Two basic kinds of RAM (Random Access Memory)

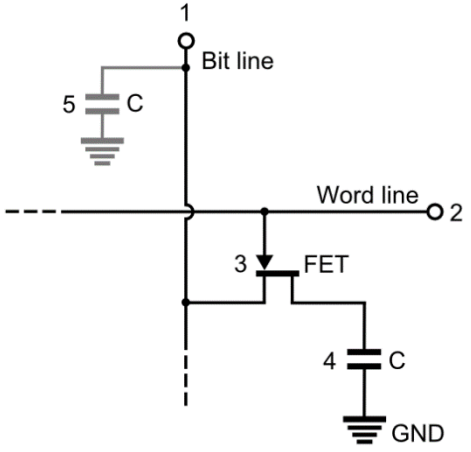
Static RAM (SRAM)

- fast, maintains data without power



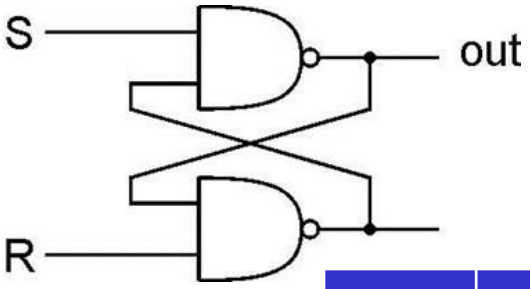
Dynamic RAM (DRAM)

- slower but denser, bit storage must be periodically refreshed



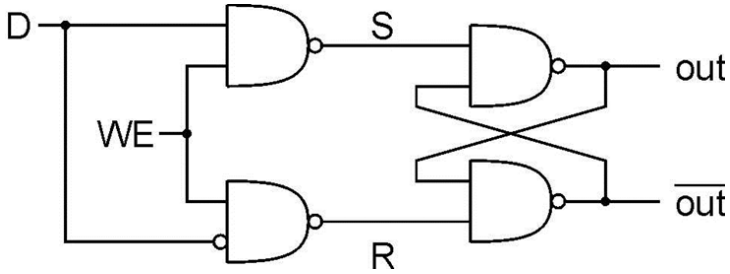
Also, non-volatile memories: ROM, PROM, flash, …

In-class Exercise



Specify the value of "out" signal for each set of inputs for the RS latch.

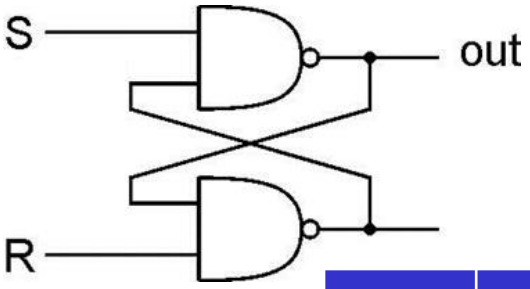
Time	0	1	2	3	4	5	6	7	8
S	1	1	0	0	1	0	1	1	0
R	0	1	1	1	1	1	0	1	1
out	0	0							



Specify the value of "out" signal for each set of inputs for the gated d-latch.

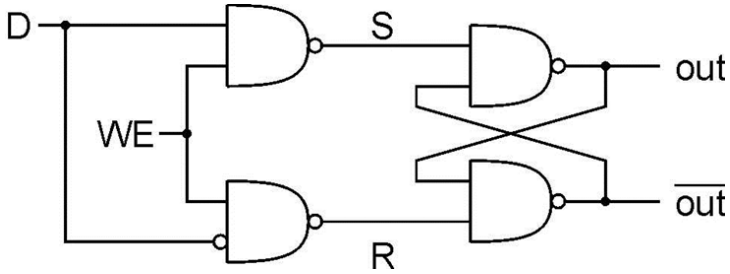
Time	0	1	2	3	4	5	6	7	8
D	1	0	0	0	1	0	1	0	0
WE	1	0	1	0	1	1	0	0	1
out	1	1							

In-class Exercise Solution



Specify the value of "out" signal for each set of inputs for the RS latch.

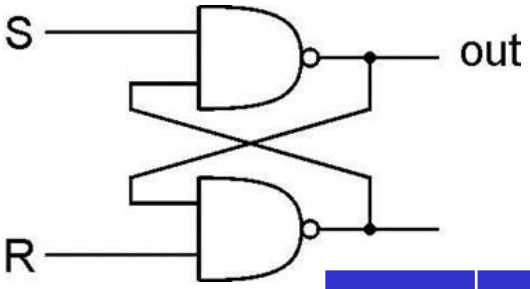
Time	0	1	2	3	4	5	6	7	8
S	1	1	0	0	1	0	1	1	0
R	0	1	1	1	1	1	0	1	1
out	0	0	1	1	1	1	0	0	1



Specify the value of "out" signal for each set of inputs for the gated d-latch.

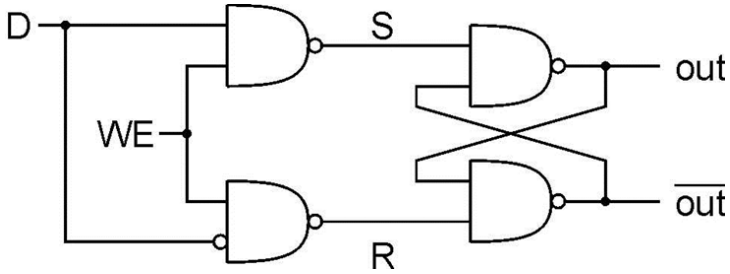
Time	0	1	2	3	4	5	6	7	8
D	1	0	0	0	1	0	1	0	0
WE	1	0	1	0	1	1	0	0	1
out	1	1							

In-class Exercise Solutions



Specify the value of "out" signal for each set of inputs for the RS latch.

Time	0	1	2	3	4	5	6	7	8
S	1	1	0	0	1	0	1	1	0
R	0	1	1	1	1	1	0	1	1
out	0	0	1	1	1	1	0	0	1

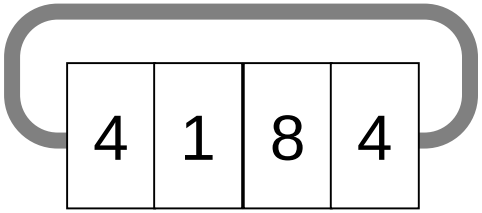


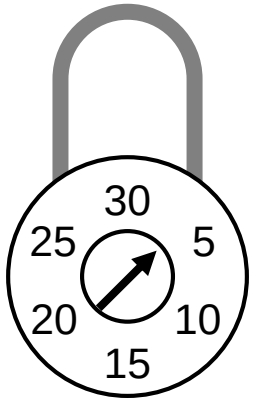
Specify the value of "out" signal for each set of inputs for the gated d-latch.

Time	0	1	2	3	4	5	6	7	8
D	1	0	0	0	1	0	1	0	0
WE	1	0	1	0	1	1	0	0	1
out	1	1	0	0	1	0	0	0	0

Combinational vs. Sequential

Two types of locks



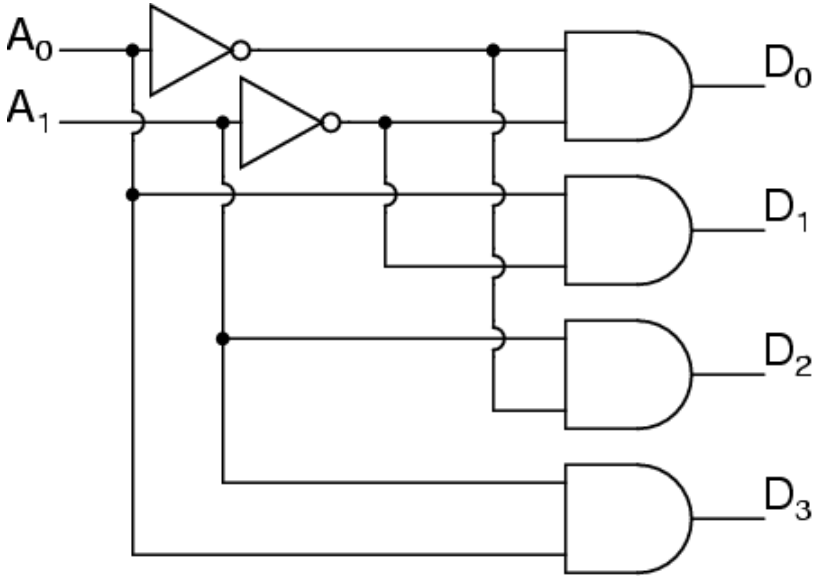


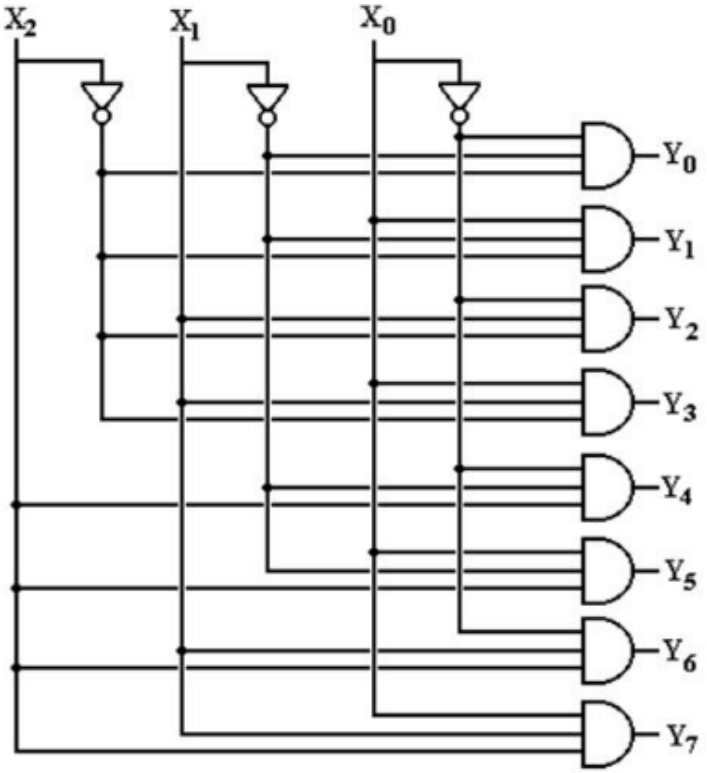
Combinational
Success depends only on the values, not the order in which they are set.

Sequential
Success depends on the sequence of values (e.g, R-13, L-22, R-3).

Points Covered

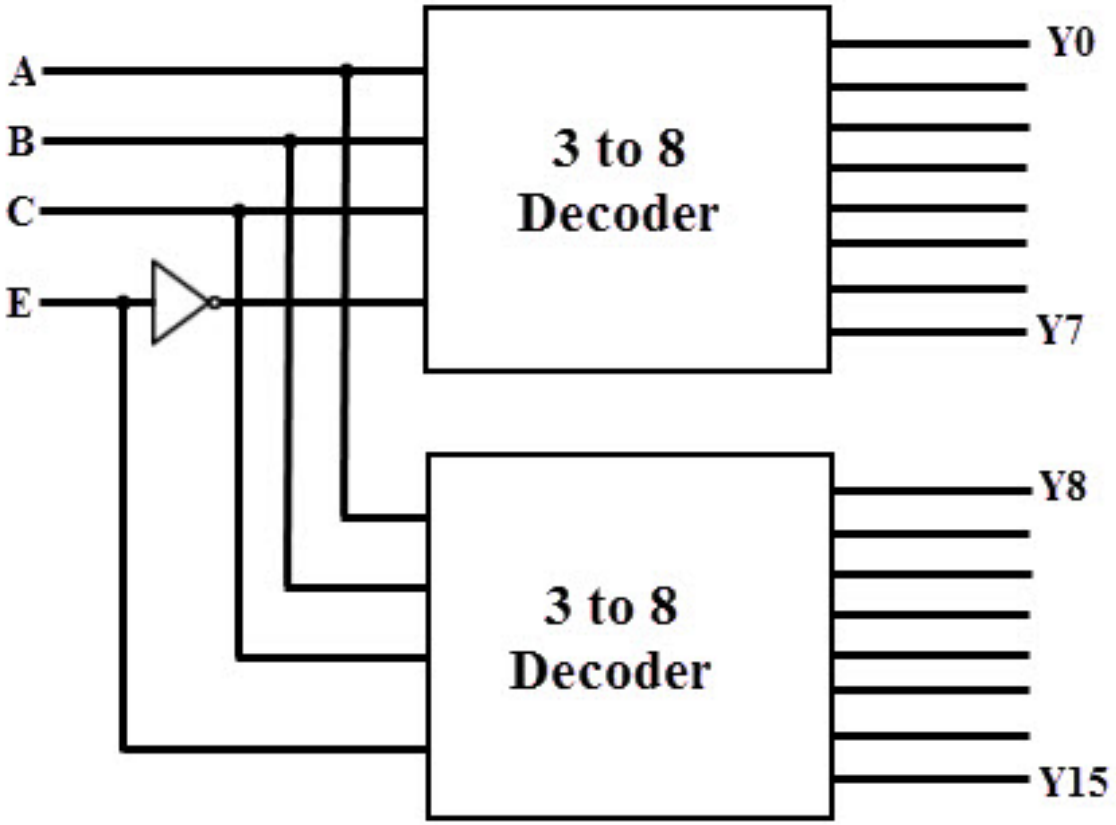
- **Decoder n input (2^n) output**
  - ➢ **decoder asserts exactly one of its output bits**





Points Covered

Using two 3-to-8 decoder to get 4-to-16 decoder



Points Covered

- **Multiplexer**
- **Combinational circuits and Sequential circuits**
- **Using logic gates to make storage elements**
  - **Feedback path**
  - **RS latch**
  - **D latch**

R-S Latch Summary

R = S = 1

- **hold current value in latch**

S = 0, R=1

- **set value to 1**

R = 0, S = 1

- **set value to 0**

R = S = 0

- **both outputs equal one**
- *Don't do it!*

D Latch Summary

D = X, WE=0 (X-> Don’t care)

- **hold current value in latch**

D = 1, WE=1

- **set value to 1**

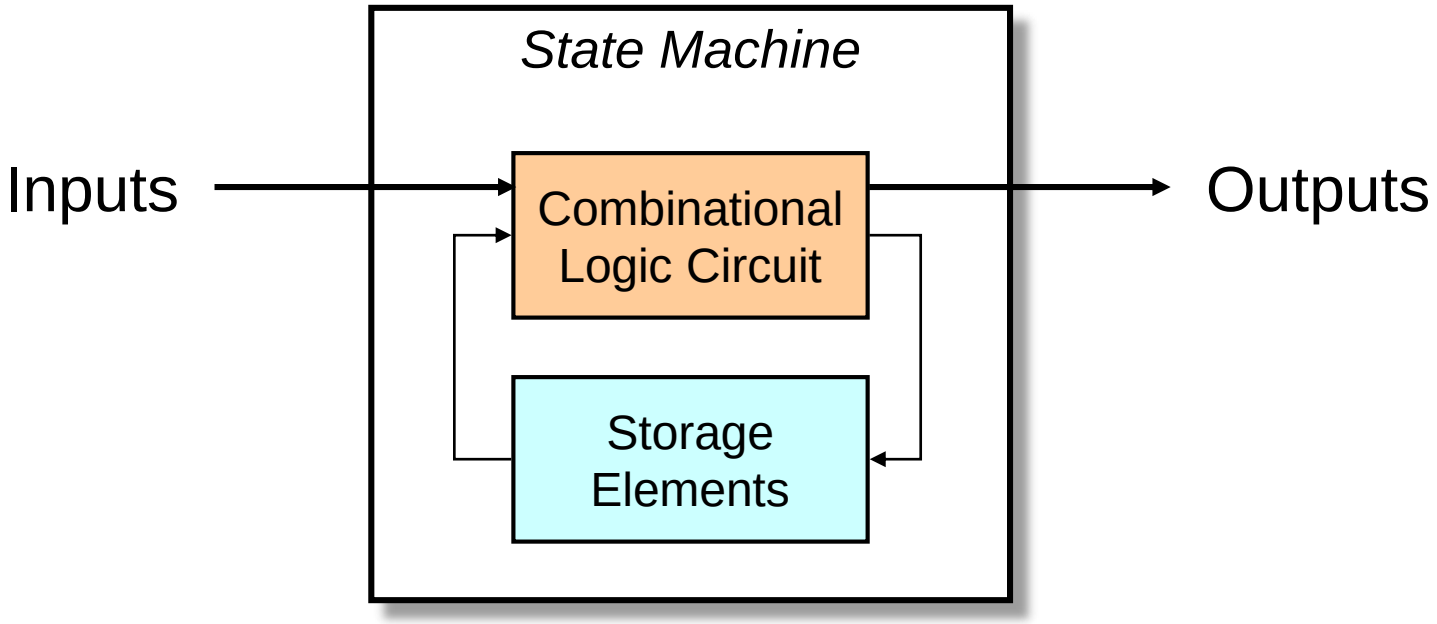
D = 0, WE=1

- **set value to 0**

State Machine

Another type of sequential circuit

- **Combines combinational logic with storage**
- **"Remembers" state, and changes output (and state) based on inputs and current state**



State

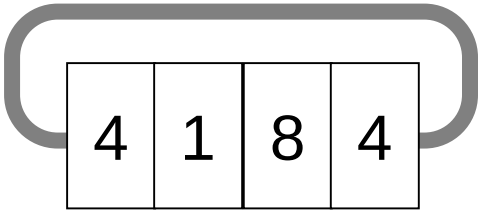
The state of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken.

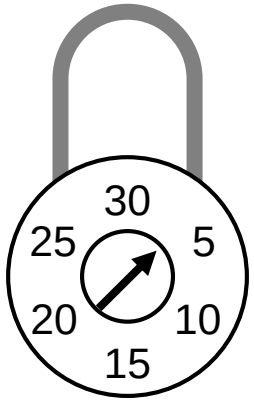
Examples:

- **The state of a basketball game can be represented by the scoreboard.**
  - **Number of points, time remaining, possession, etc.**
- **The state of a tic-tac-toe game can be represented by the placement of X's and O's on the board, and who's turn it is.**

Sequential Lock

Two types of locks





Combinational
Success depends only on the values, not the order in which they are set.

Sequential
Success depends on the sequence of values (e.g, R-13, L-22, R-3).

State of Sequential Lock

Our lock example has four different states, labelled A-D:

A: The lock is not open,
and no relevant operations have been performed.

B: The lock is not open,
and the user has completed the R-13 operation.

C: The lock is not open,
and the user has completed R-13, followed by L-22.

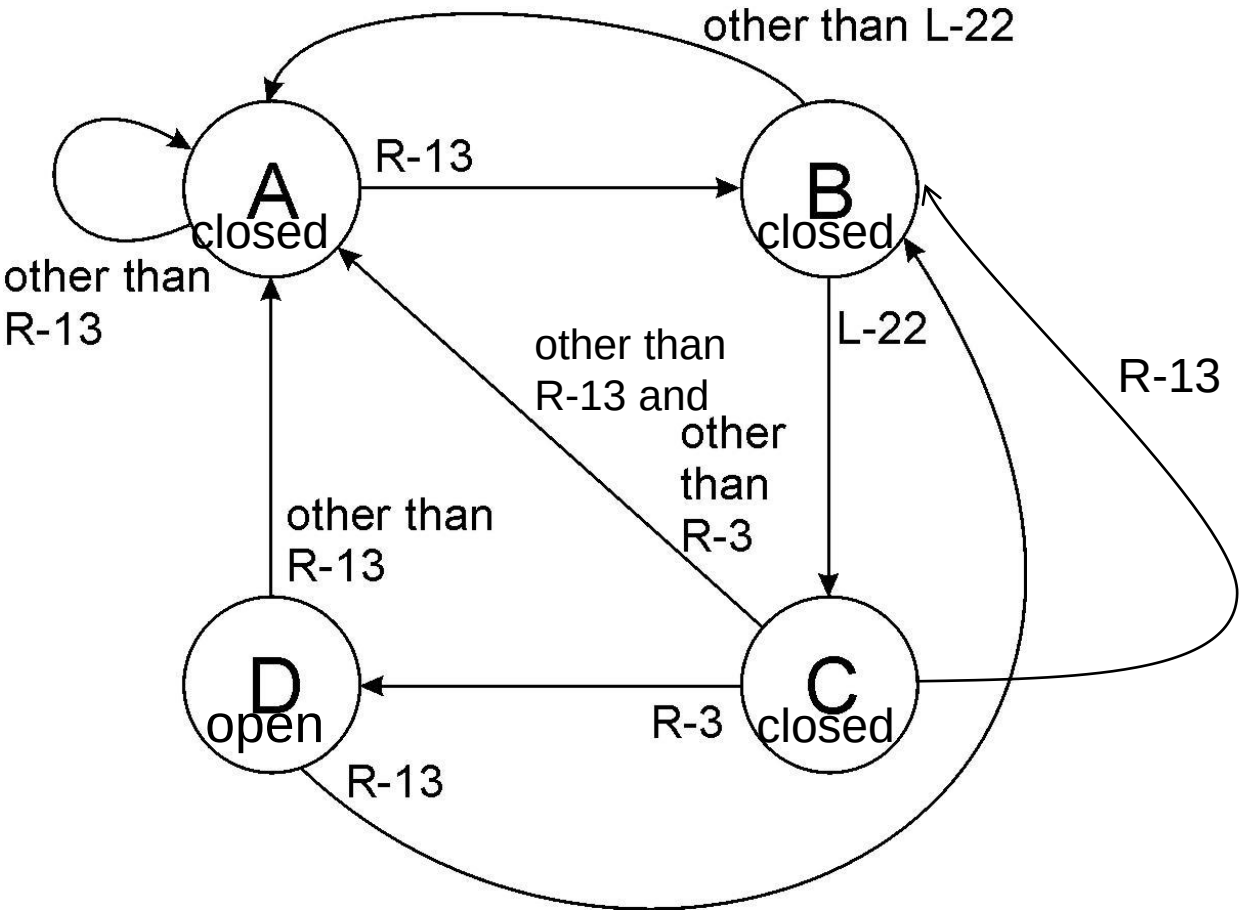
D: The lock is open,

and the user has completed R-13, followed by L-22,

followed by R-3

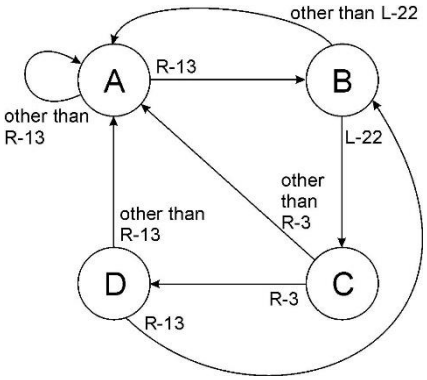
State Diagram

Shows states and actions that cause a transition between states.



Finite State Machine

A description of a system with the following components:



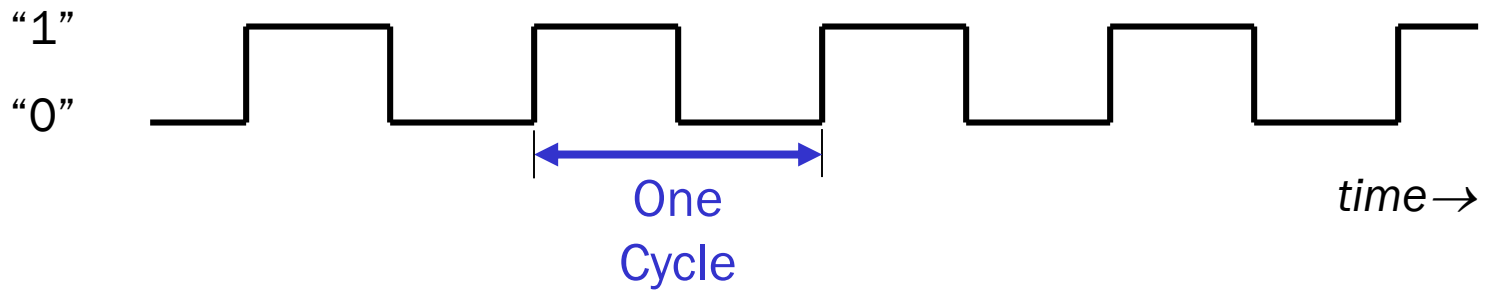
1. **A finite number of states**
2. **A finite number of external inputs**
3. **A finite number of external outputs**
4. **An explicit specification of all state transitions**
5. **An explicit specification of what causes each external output value.**

Often described by a state diagram.

- **Inputs may cause state transitions.**
- **Outputs are associated with each state (or with each transition).**

The Clock

Frequently, a clock circuit triggers transition from one state to the next.



**At the beginning of each clock cycle,
state machine makes a transition,
based on the current state and the external inputs.**

- Not always required. In lock example, the input itself triggers a transition.

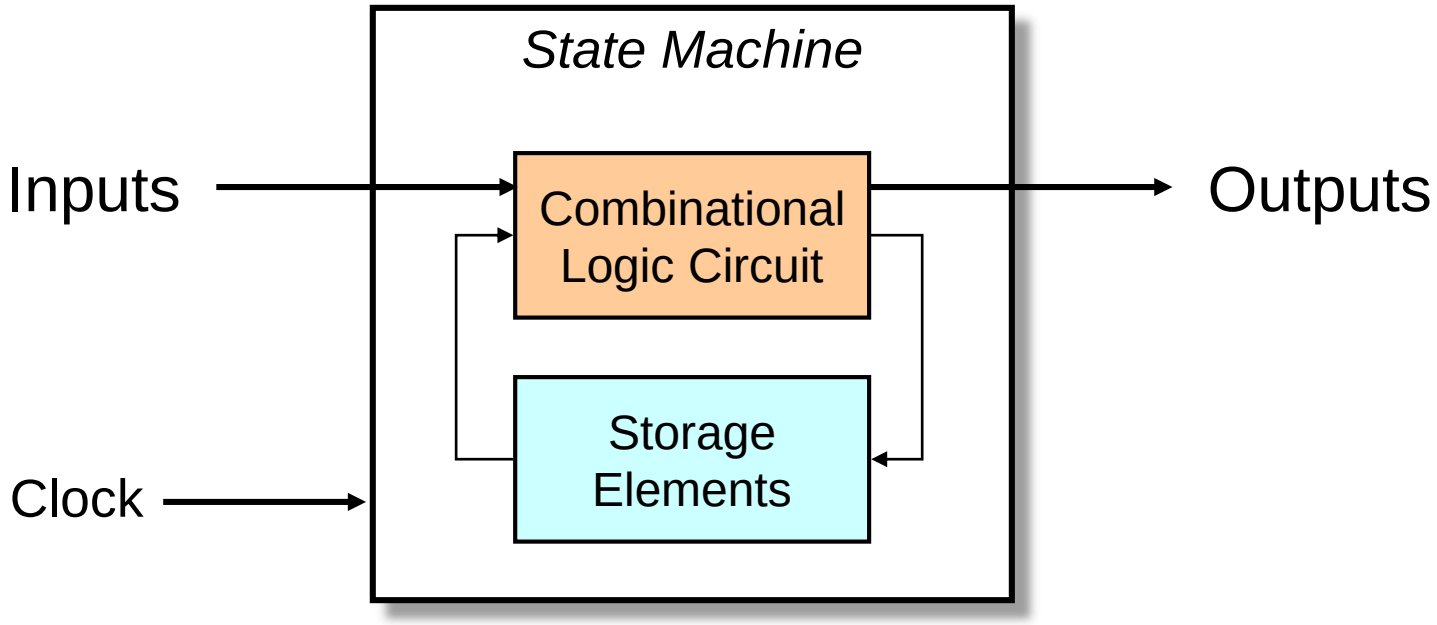
Implementing a Finite State Machine

Combinational logic

- **Determine outputs and next state.**

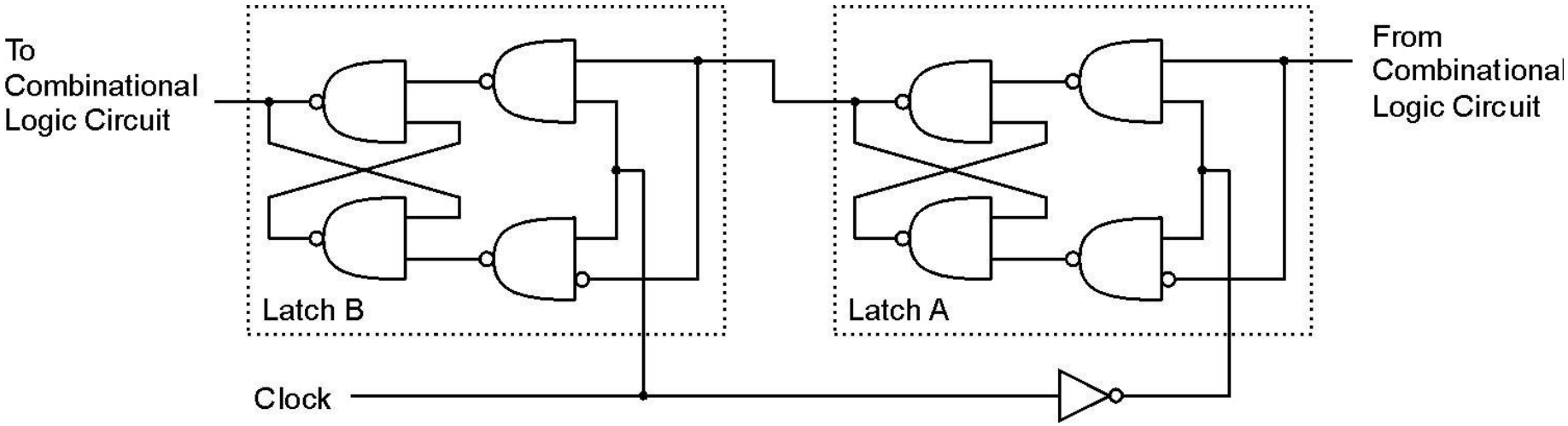
Storage elements

- **Maintain state representation.**



Storage: Master-Slave Flipflop

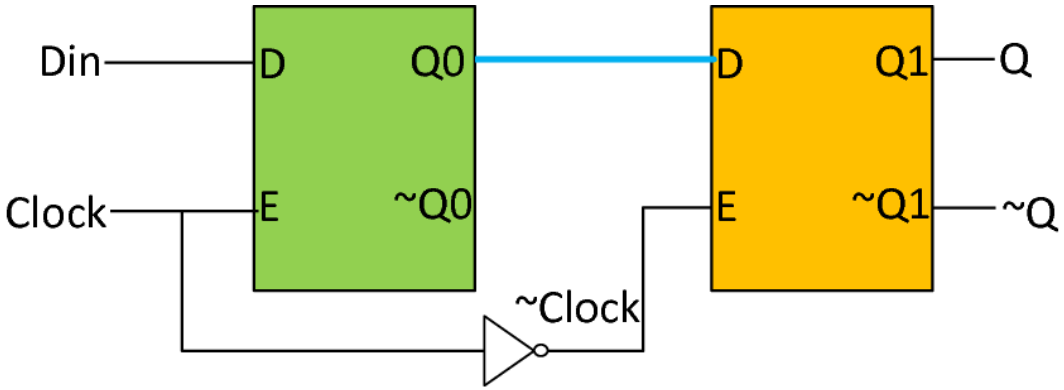
A pair of gated D-latches,
to isolate *next* state from *current* state.

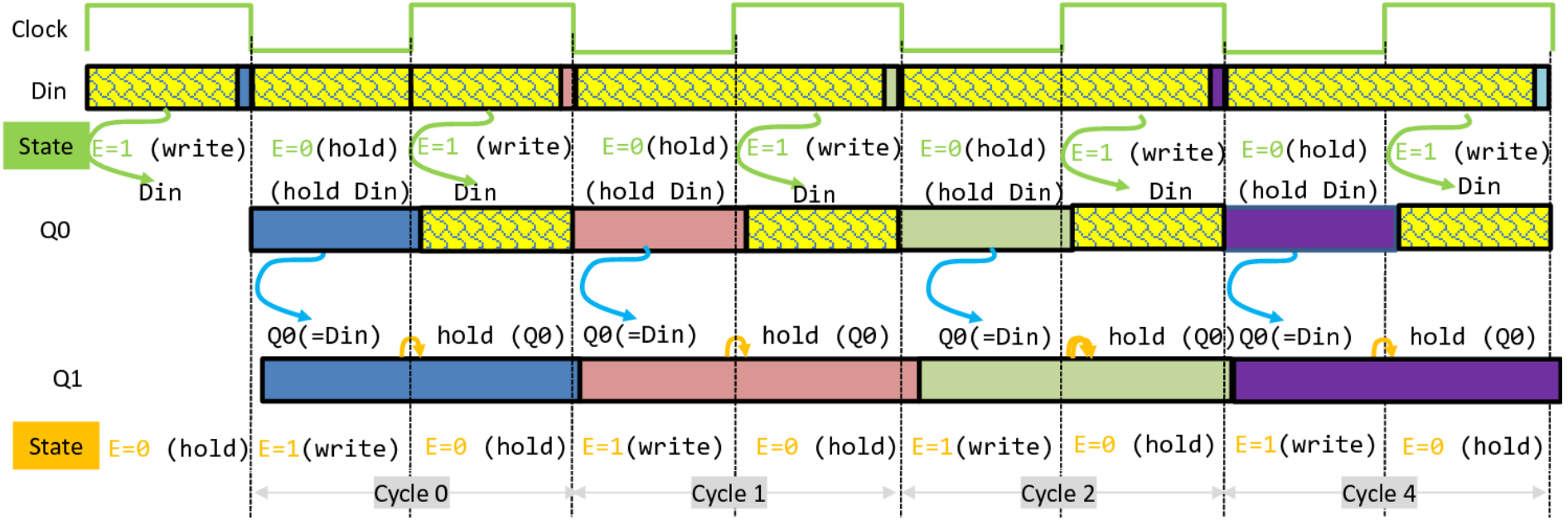


During 1st phase (clock=1), previously-computed state becomes *current* state and is sent to the logic circuit.

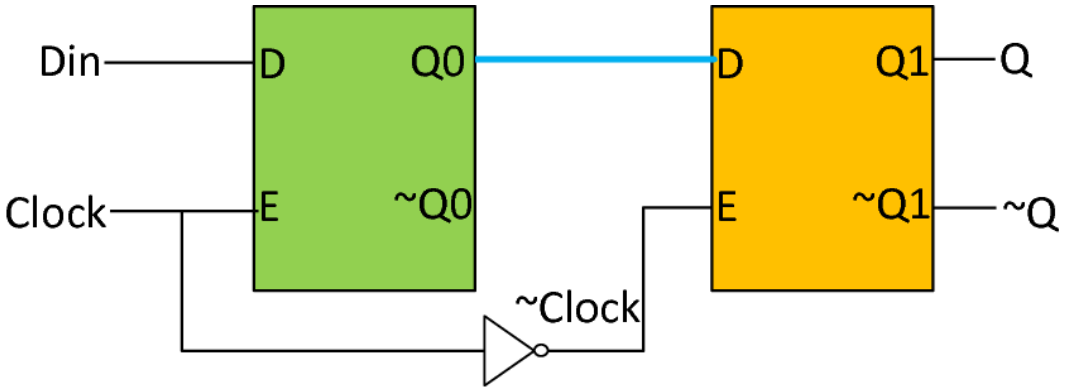
During 2nd phase (clock=0), *next* state, computed by logic circuit, is stored in Latch A.

Storage: Master-Slave Flipflop





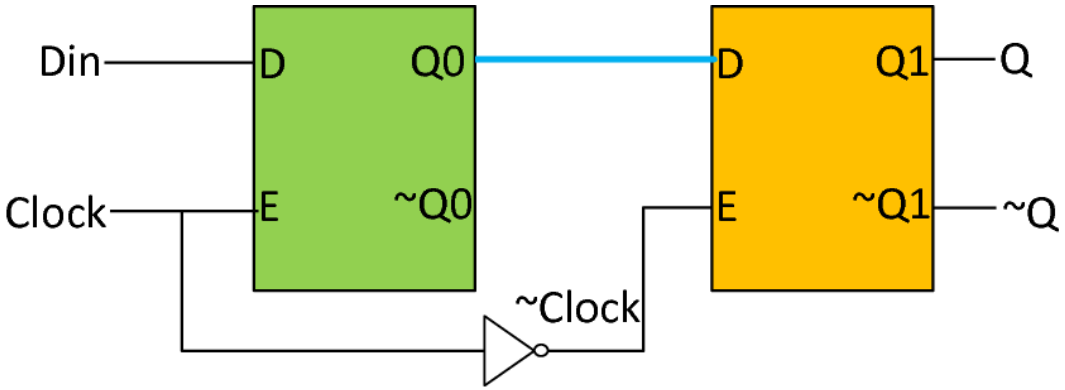
In-class Exercise



Specify the value of “Q0” and “Q1” signal for each set of inputs for the master-slave flipflop.

Time	0	0.3	0.4	0.5	.7	1	1.2	1.5	1.7	1.8	2	2.2	2.3	2.4	2.5
Clock	1	1	1	0	0	1	1	0	0	0	1	1	1	1	0
Din	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0
Q0	0	1													
Q1	1	1													

In-class Exercise Solutions

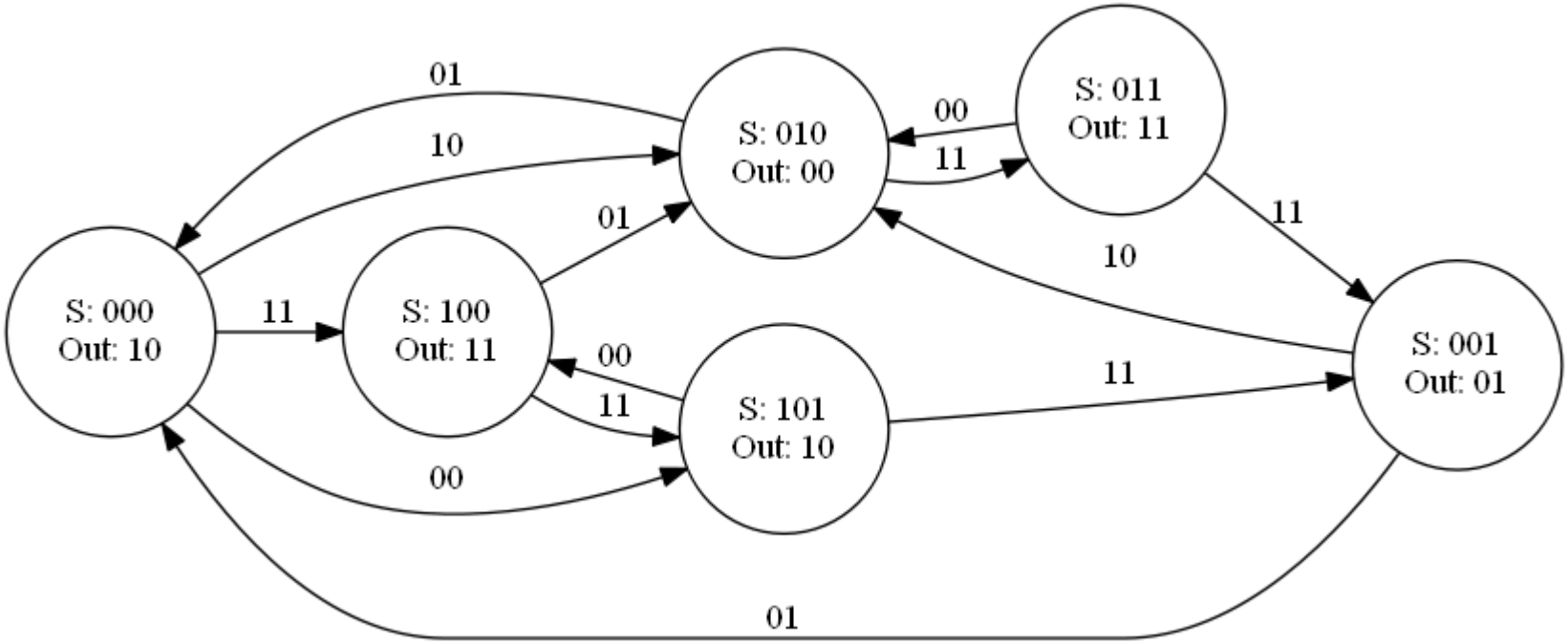


Specify the value of “Q0” and “Q1” signal for each set of inputs for the master-slave flipflop.

Time	0	0.3	0.4	0.5	.7	1	1.2	1.5	1.7	1.8	2	2.2	2.3	2.4	2.5
Clock	1	1	1	0	0	1	1	0	0	0	1	1	1	1	0
Din	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0
Q0	0	1	0	0	0	0	1	1	1	1	1	0	1	0	0
Q1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0

In-class Exercise 1

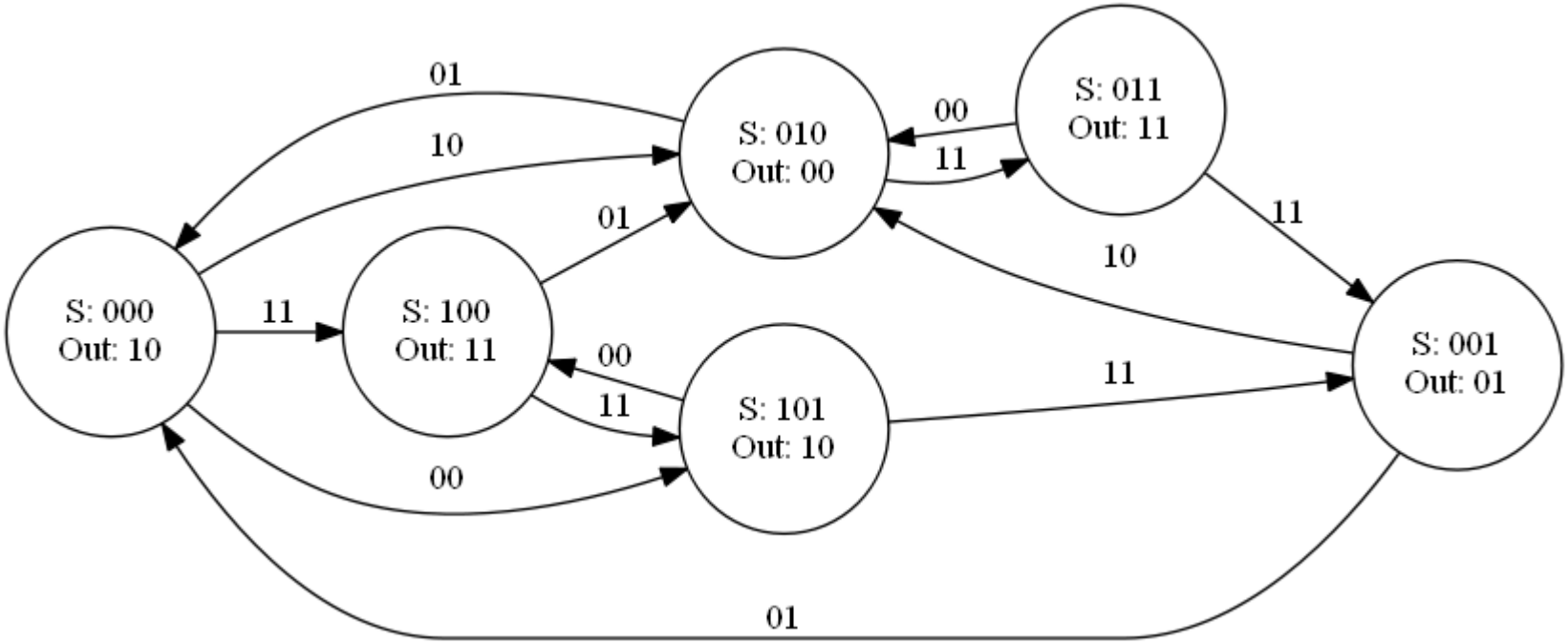
Specify the next State and Out for each set of inputs.



Time	0	1	2	3	4	5	6	7	8	9	10
State	000	100	010								
Input	11	01	01	00	11	10	11	11	01	10	11
Output	10	11	00								

In-class Exercise 1 Solutions

Specify the next State and Out for each set of inputs.



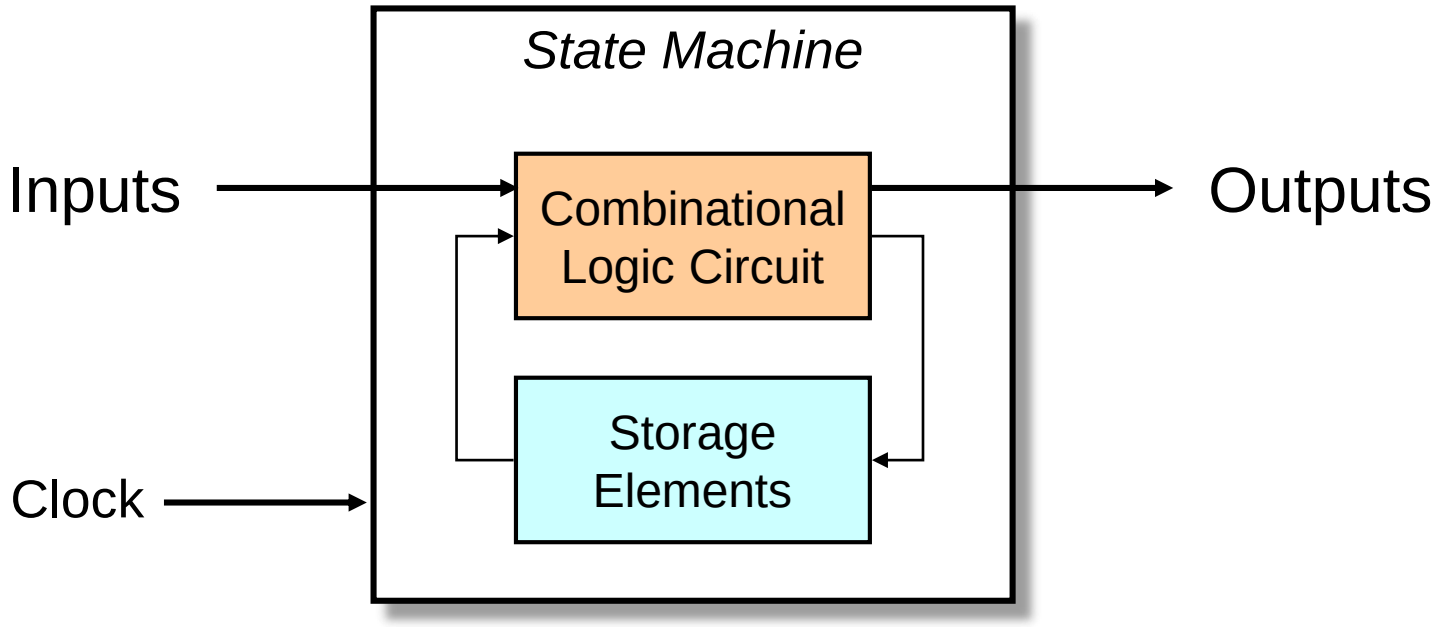
Time	0	1	2	3	4	5	6	7	8	9	10
State	000	100	010	000	101	001	010	011	001	000	010
Input	11	01	01	00	11	10	11	11	01	10	11
Output	10	11	00	10	10	01	00	11	01	10	00

Storage

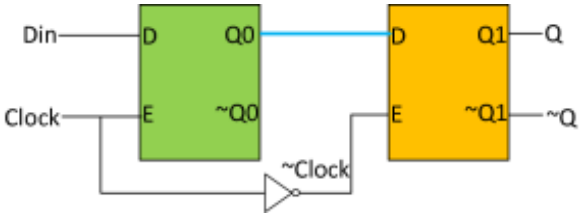
Each master-slave flipflop stores one state bit.

Examples:

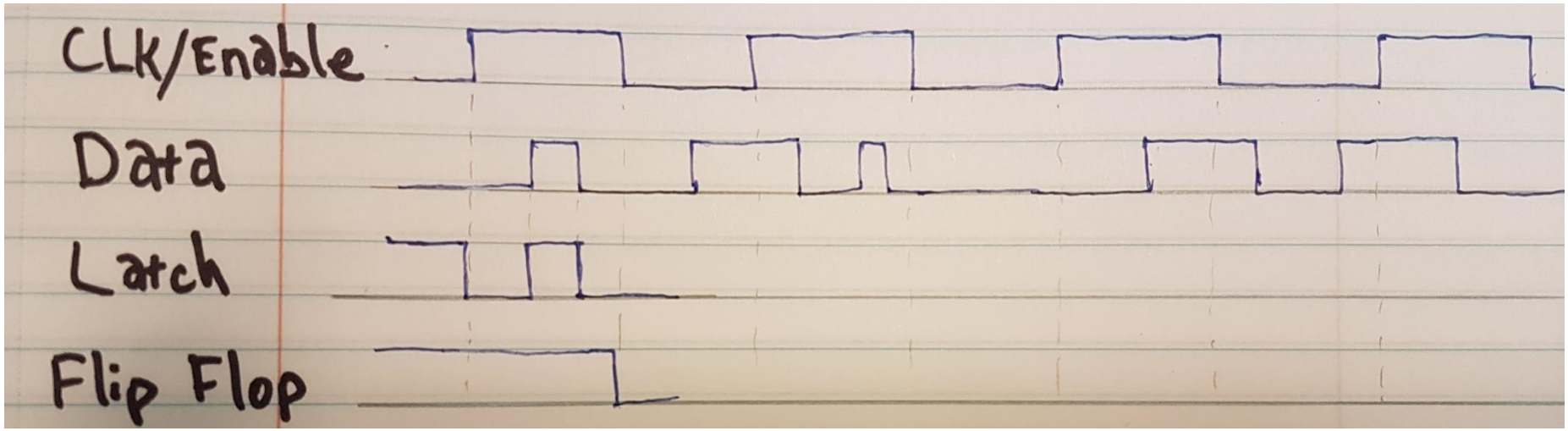
- **Sequential lock**
  - **Four states – two bits**



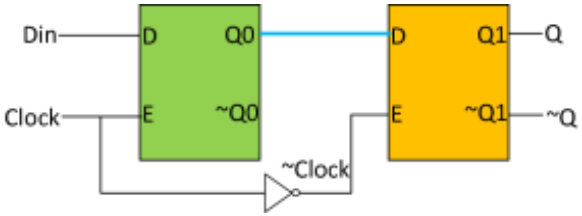
In-class Exercise 2



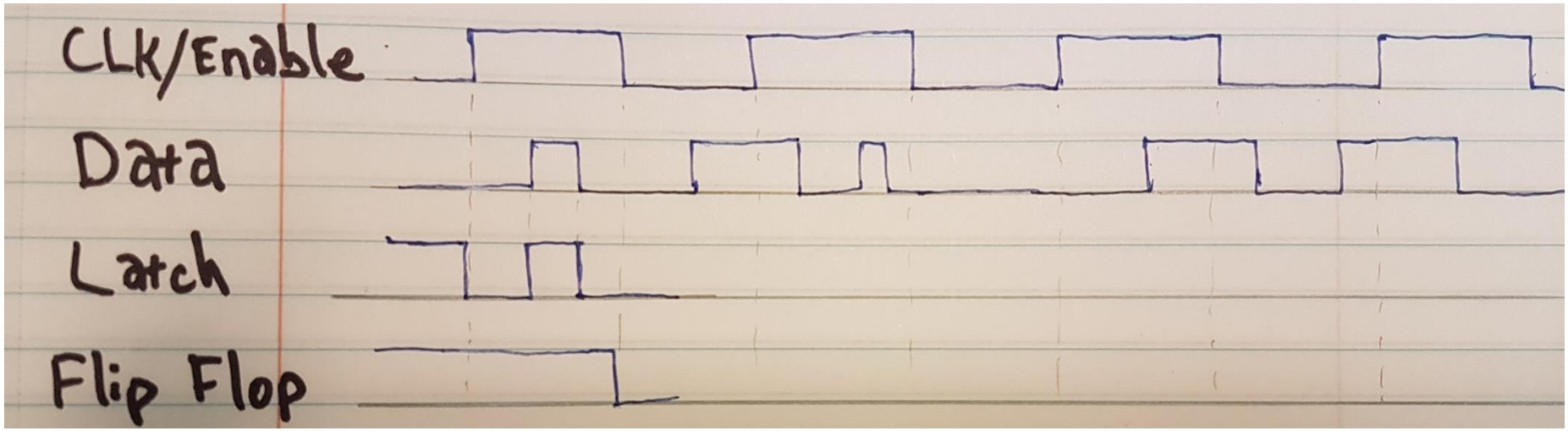
Given the clock/enable and data signal, specify the “Latch” and “Flip-Flop” signal.

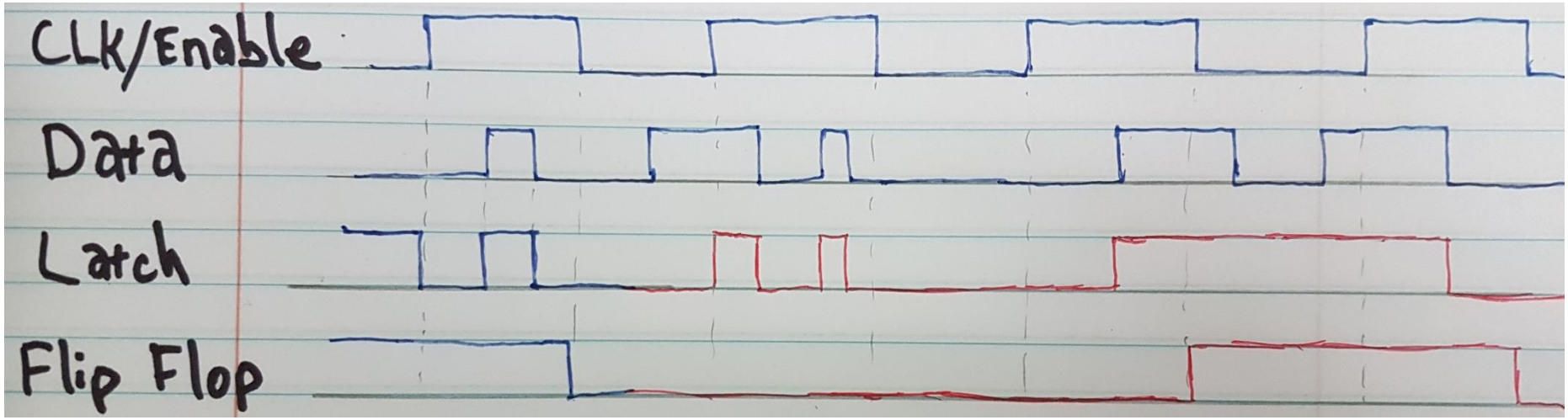


In-class Exercise 2 Solutions



Given the clock/enable and data signal, specify the “Latch” and “Flip-Flop” signal.

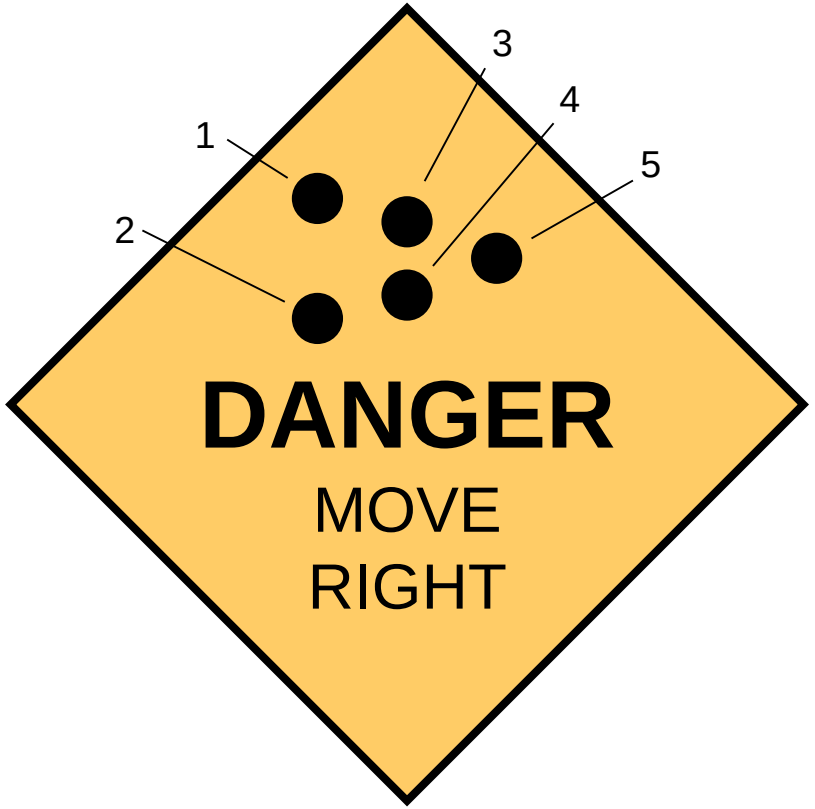




Complete Example

A blinking traffic sign

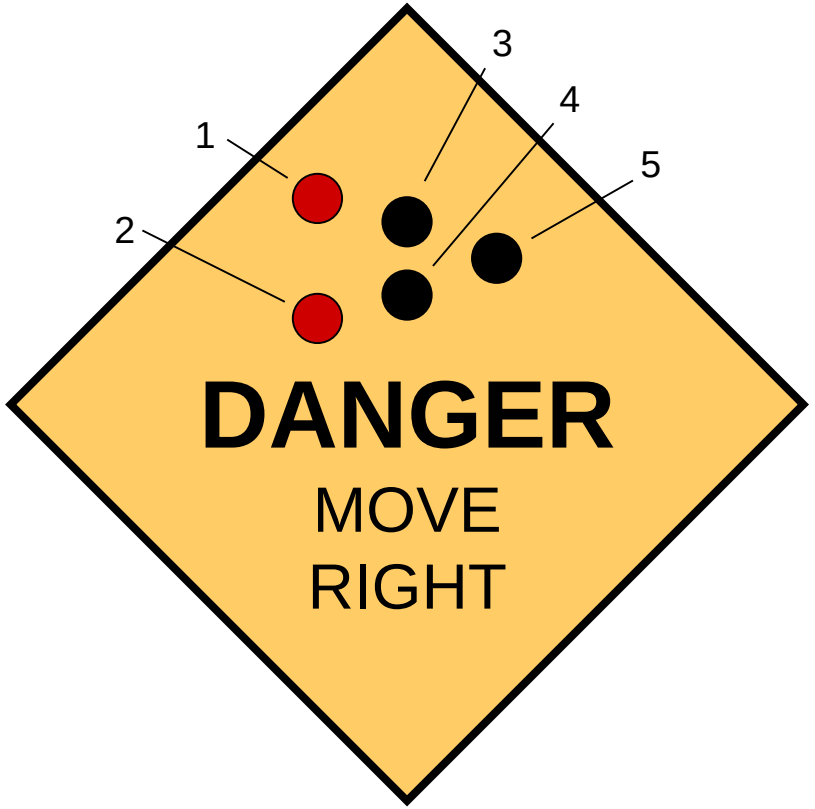
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

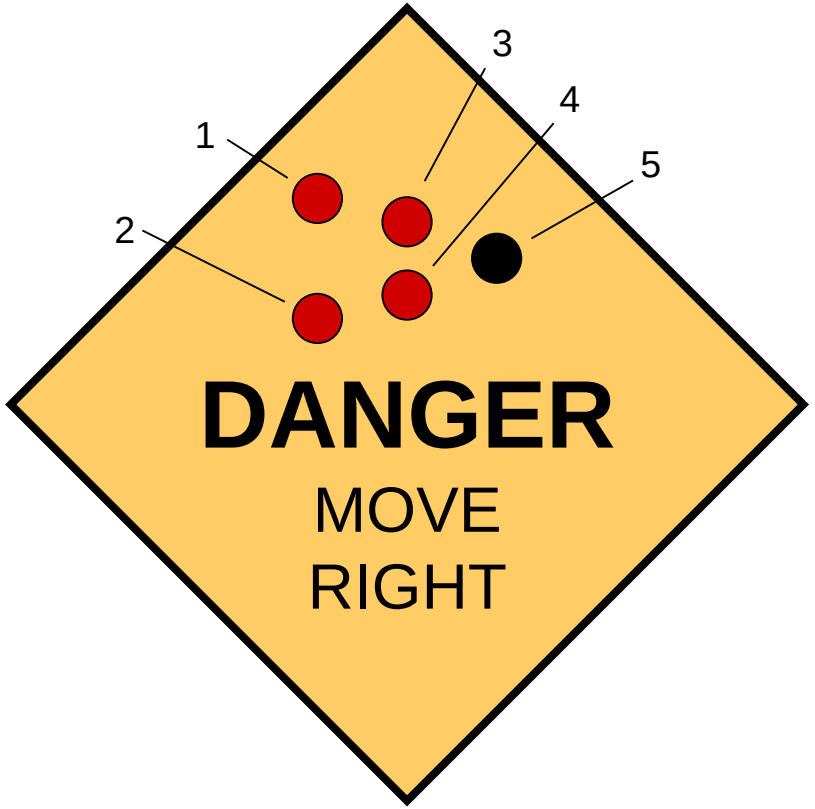
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

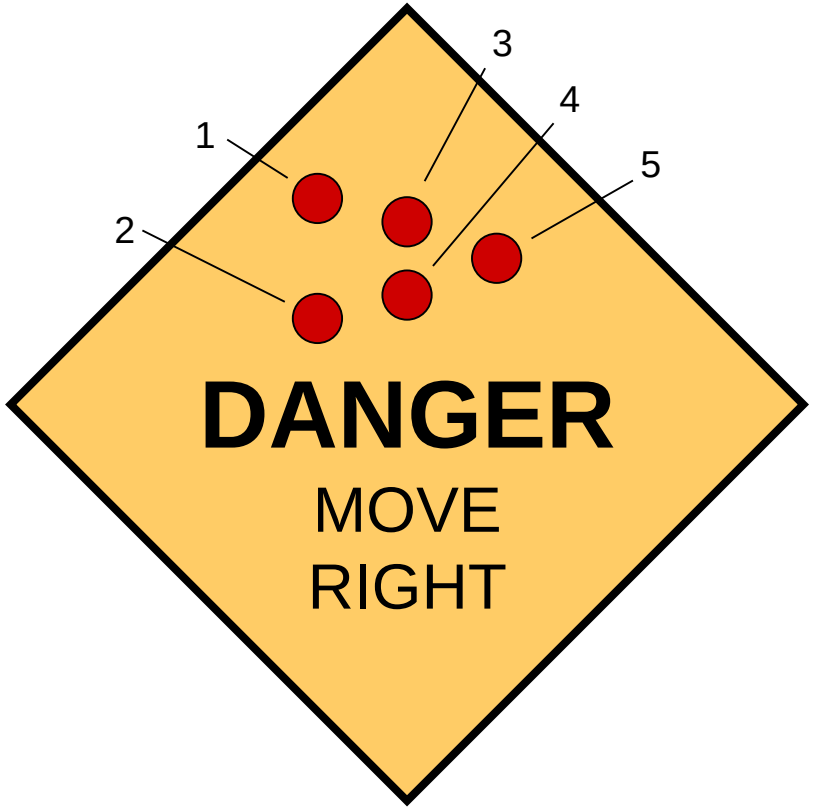
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

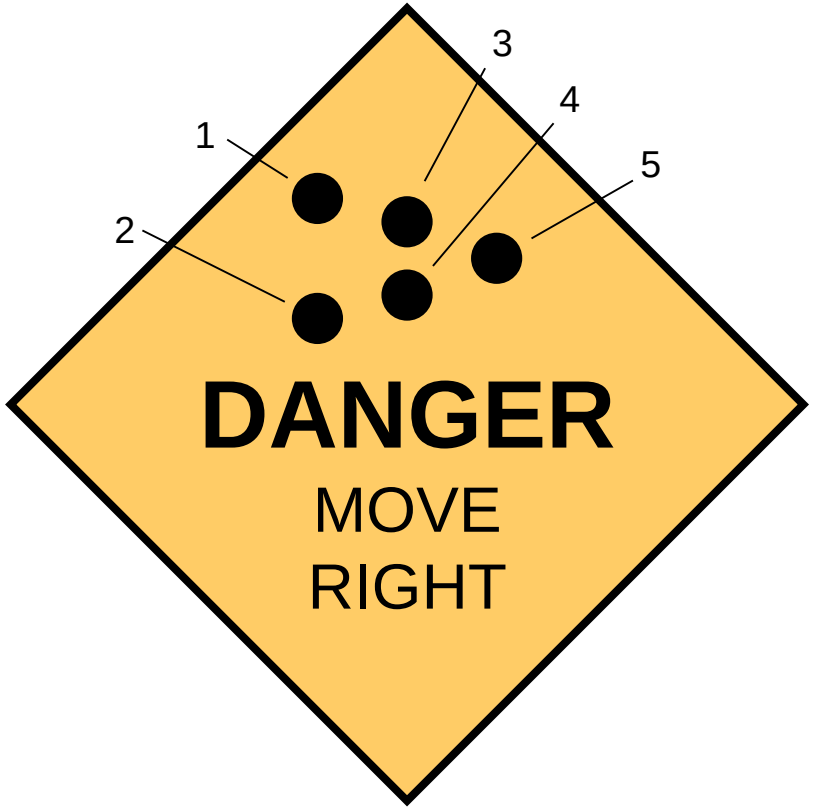
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

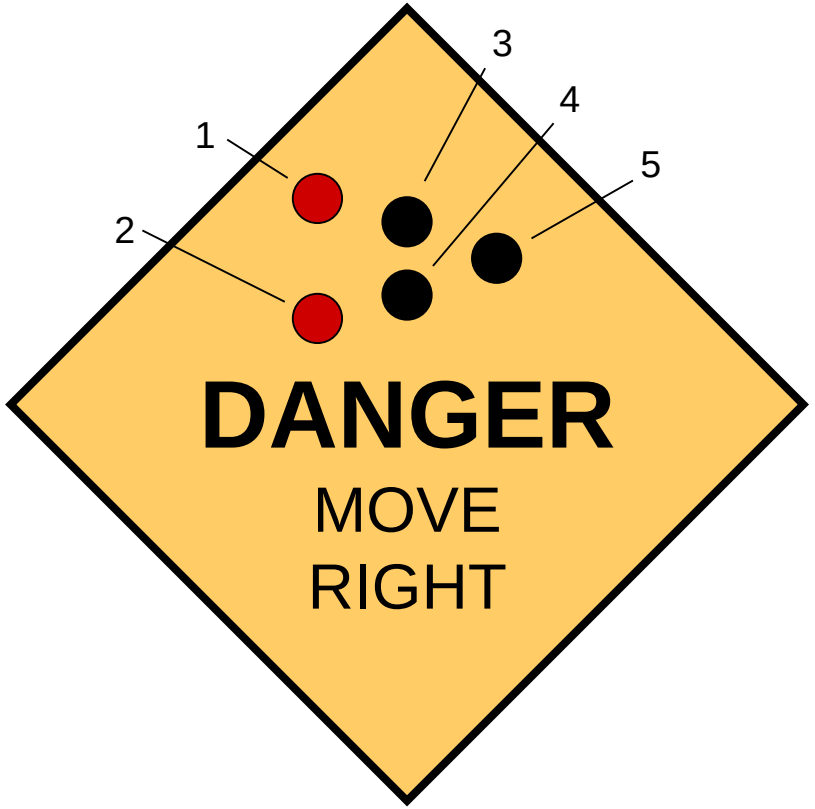
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

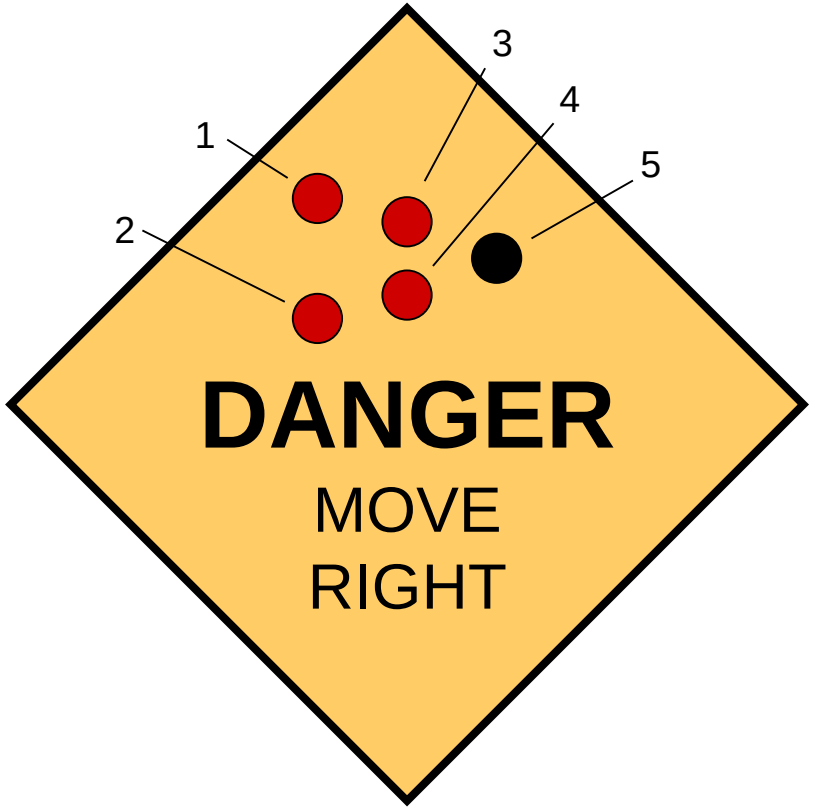
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

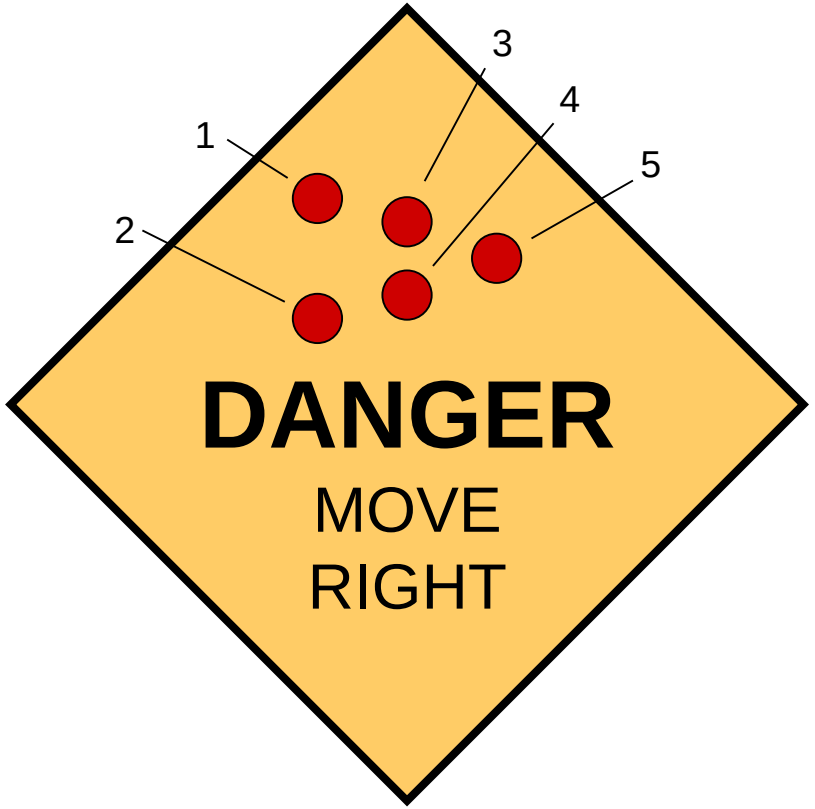
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

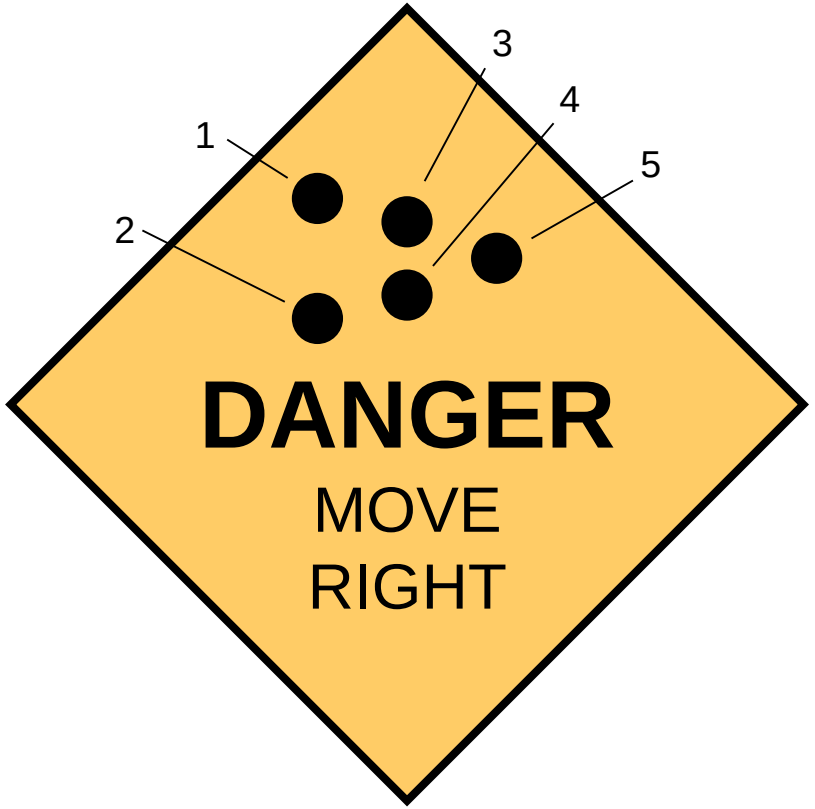
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

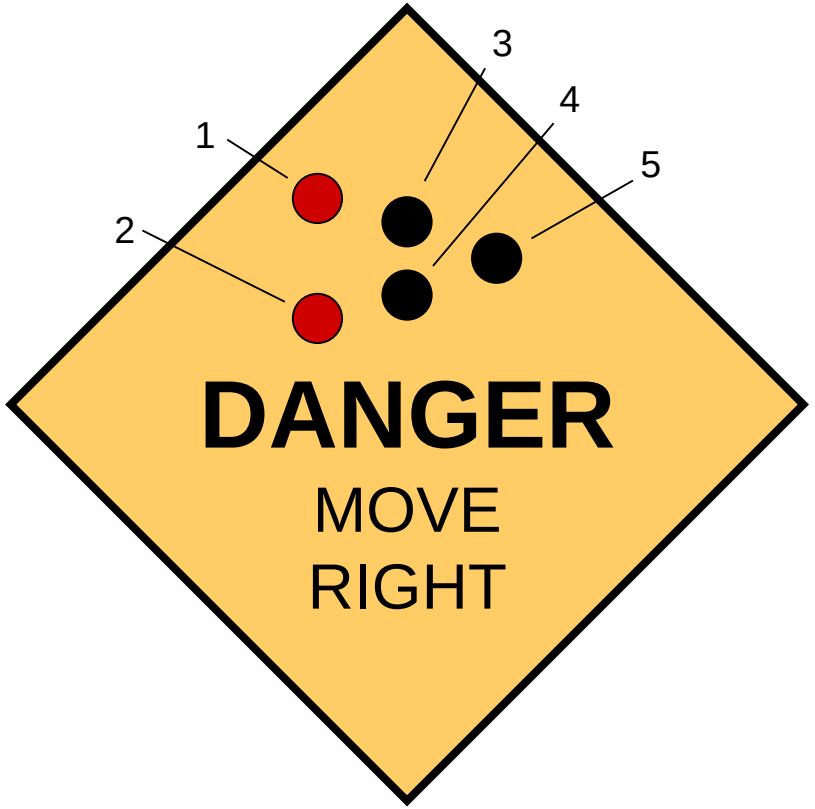
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

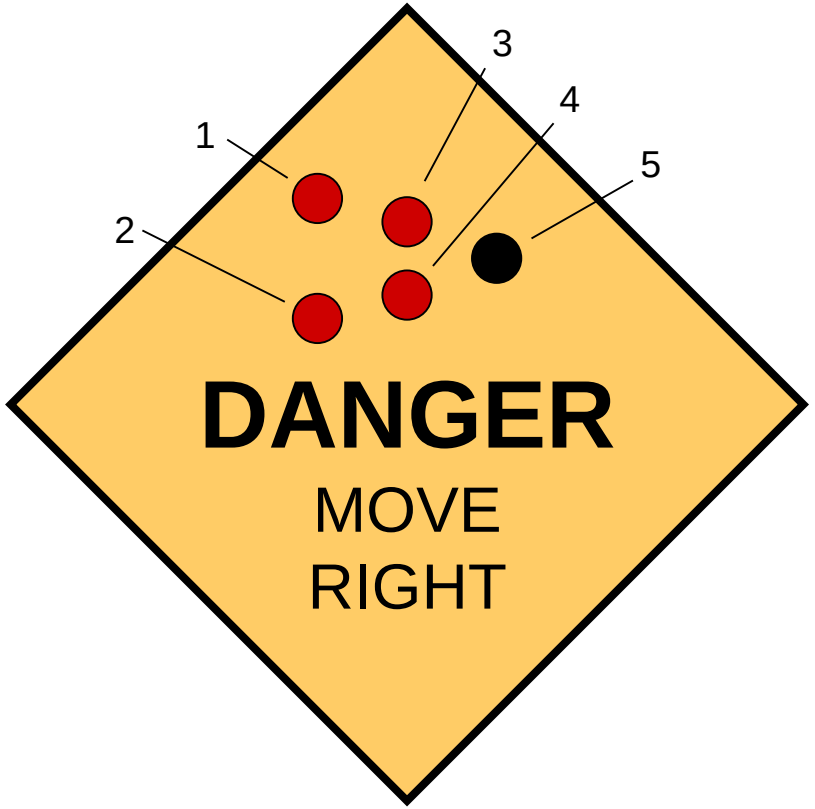
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Complete Example

A blinking traffic sign

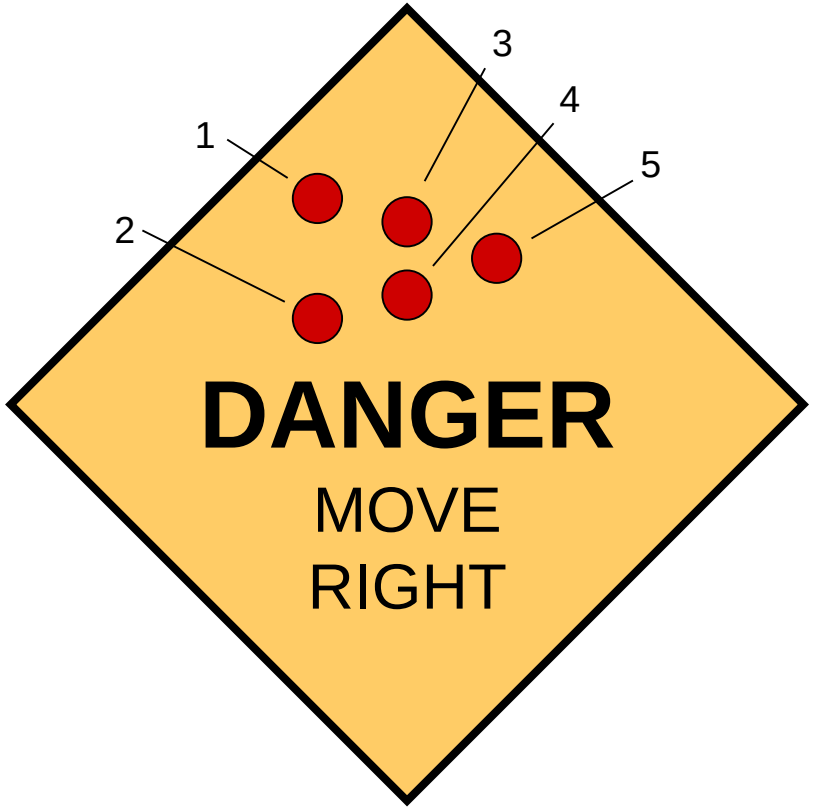
- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



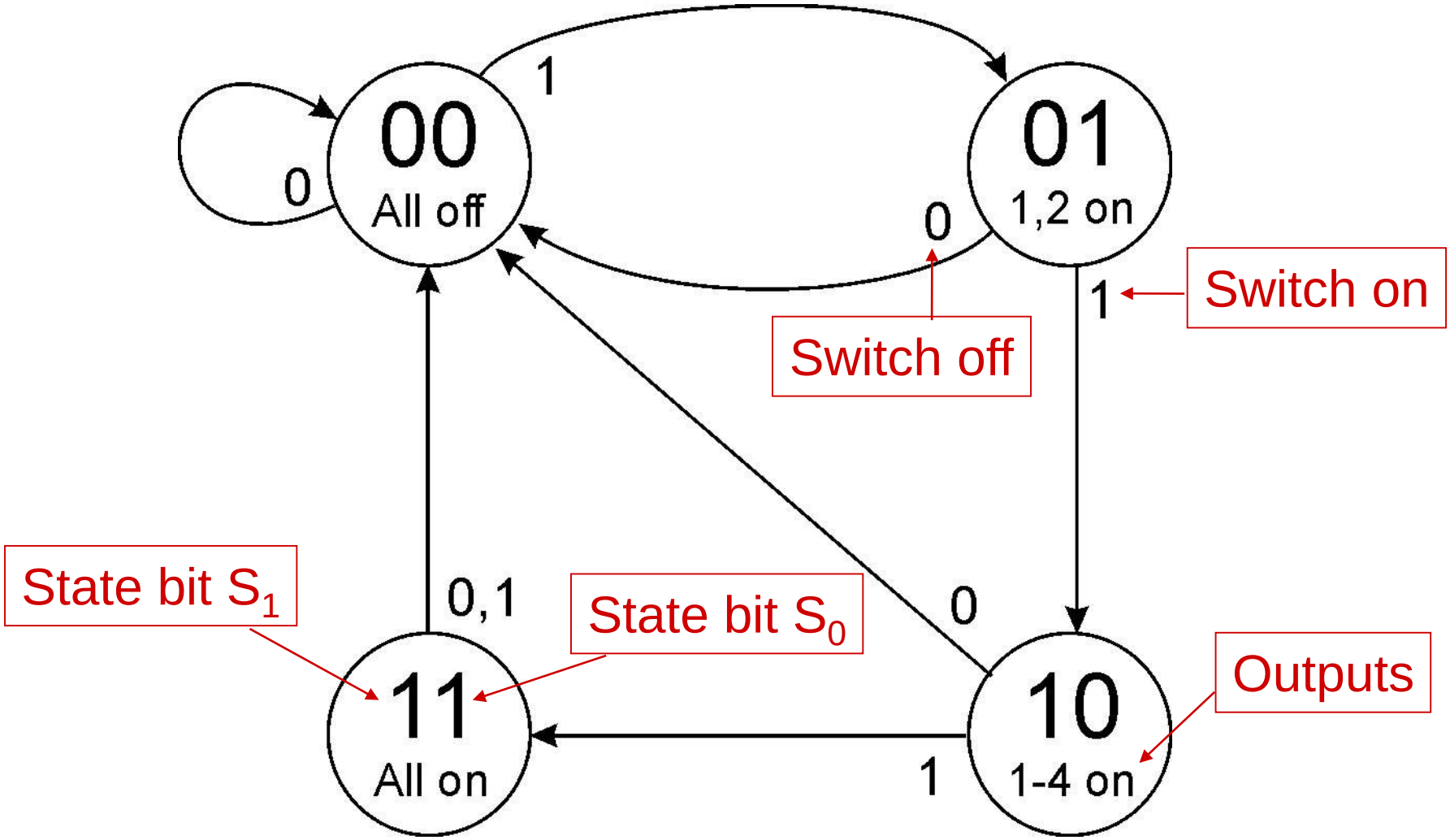
Complete Example

A blinking traffic sign

- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)



Traffic Sign State Diagram



Transition on each clock cycle.

Traffic Sign Truth Tables

Outputs (depend only on state: S_1S_0)

Lights 1 and 2
Lights 3 and 4
Light 5

S_1	S_0	Z	Y	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

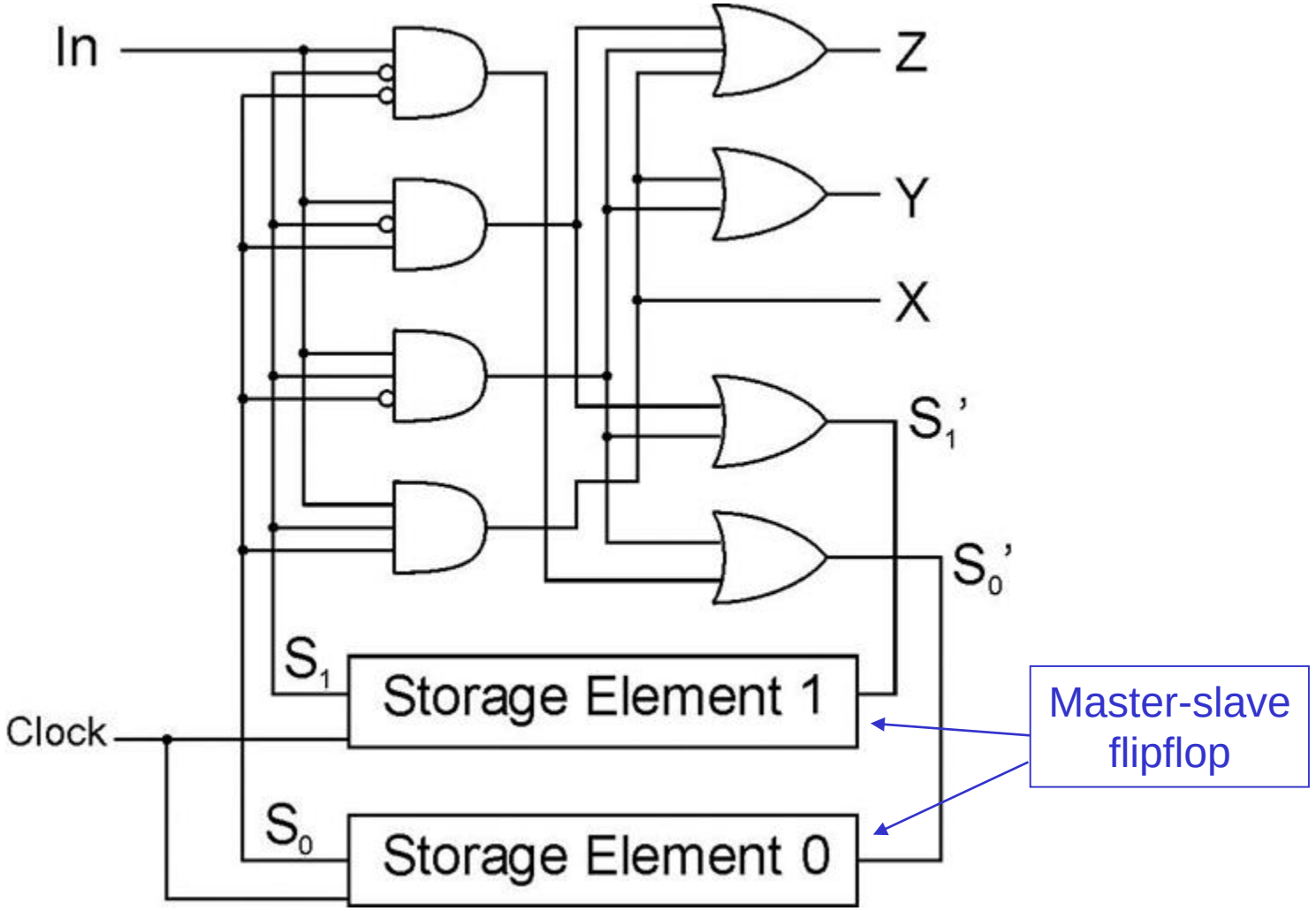
Next State: $S_1'S_0'$ (depend on state and input)

Switch

In	S_1	S_0	S_1'	S_0'
0	X	X	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Whenever In=0, next state is 00.

Traffic Sign Logic



From Logic to Data Path

The data path of a computer is all the logic used to process information.

- **See the data path of the LC-3 on next slide.**

Combinational Logic

- **Decoders -- convert instructions into control signals**
- **Multiplexers -- select inputs and outputs**
- **ALU (Arithmetic and Logic Unit) -- operations on data**

Sequential Logic

- **State machine -- coordinate control signals and data movement**
- **Registers and latches -- storage elements**

LC-3 Data Path