DIGITAL CIRCUITS

LECTURE 06-1

JIM FIX, REED COLLEGE CS2-F20

WHERE WE ARE AT IN THE SCHEDULE

- ***TODAY:*** Part II. Circuits and Processors.
  - Download Logisim-Evolution for simulating digital circuits.
  - Needs installation of Oracle's JDK (Java Development kit) or of OpenJDK

RANDOM FACT ABOUT MY PROGRAMMING HISTORY

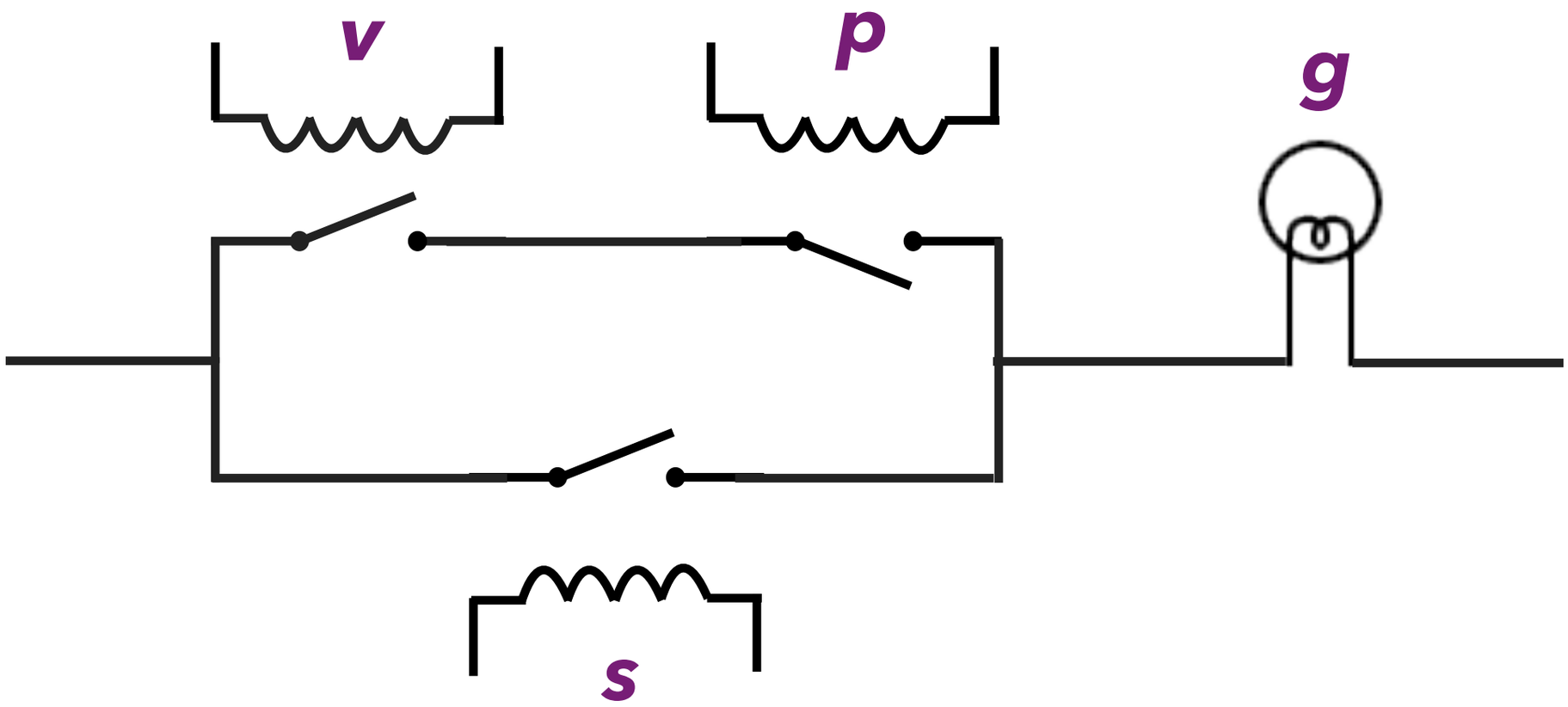
- I had a summer internship worked on a full-scale nuclear reactor simulator.
- I was asked to transcribe logic circuits into FORTRAN code.
- There were reactor control diagrams for control behavior, e.g.

  If that valve has *not* failed *and* that pipe's pressure is normal, *or* we are in shutdown mode, *then* a certain green light on the control panel should be lit up.

- That logic might lead to this code in C++:

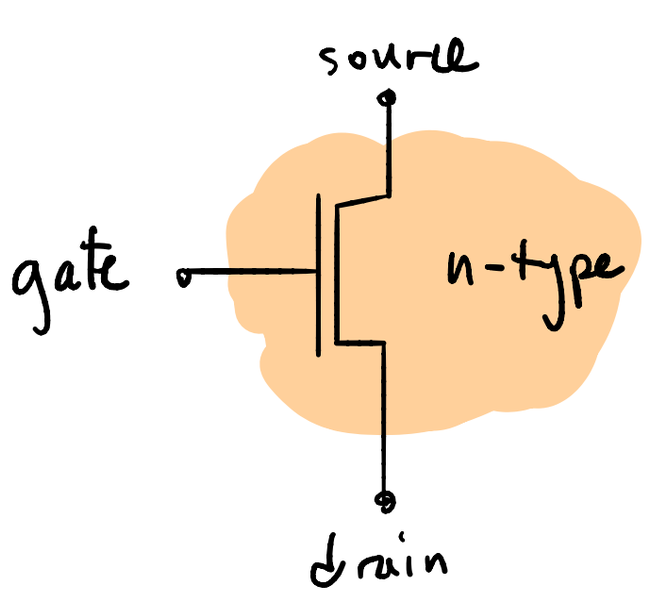
```
bool v = valve_failure(vlv120);
bool p = pressure_normal(ps456);
bool s = in_shutdown_mode(system);
g = (!v && p) || s
```

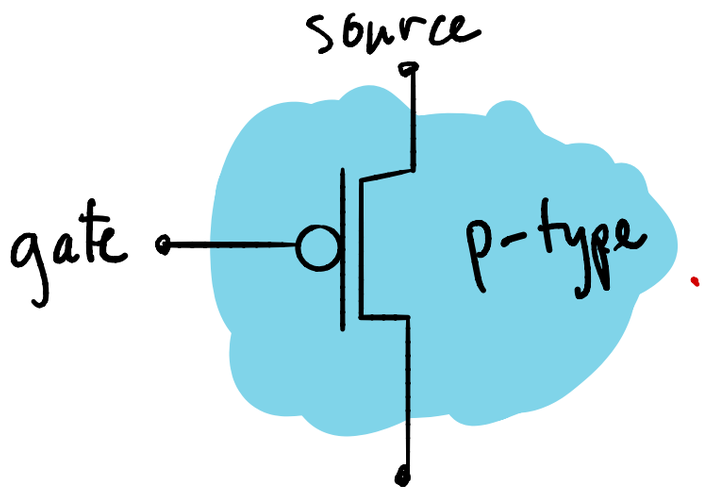
BEHAVIOR WAS ENCODED AS A RELAY CIRCUIT



TRANSISTOR LOGIC

There are two kinds of transistors: n-type and p-type

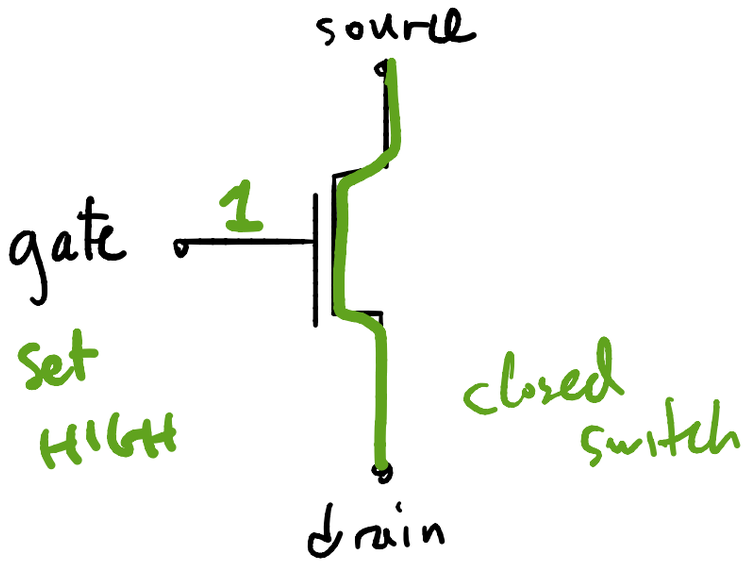




- gate/source/drain wired up possibly to HIGH/LOW voltage (e.g. +5V/+0V)
- pathway between source-drain "gated", acts like a switch controlled by gate signal

TRANSISTOR LOGIC

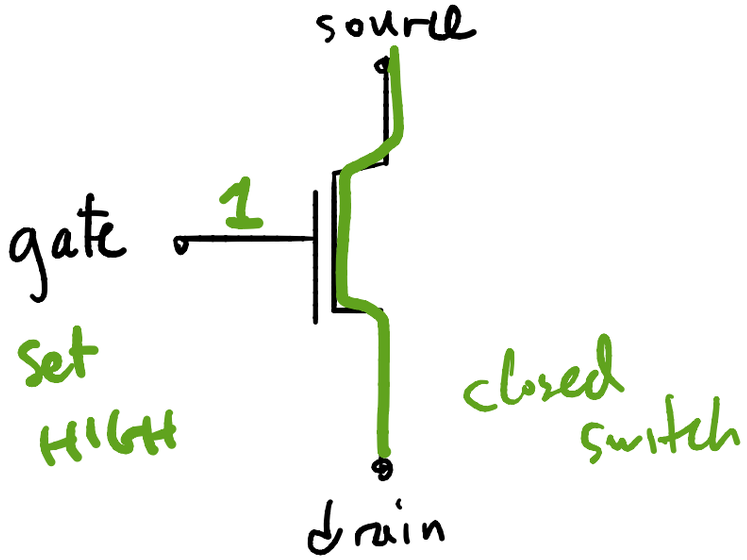
n-type transistor behavior



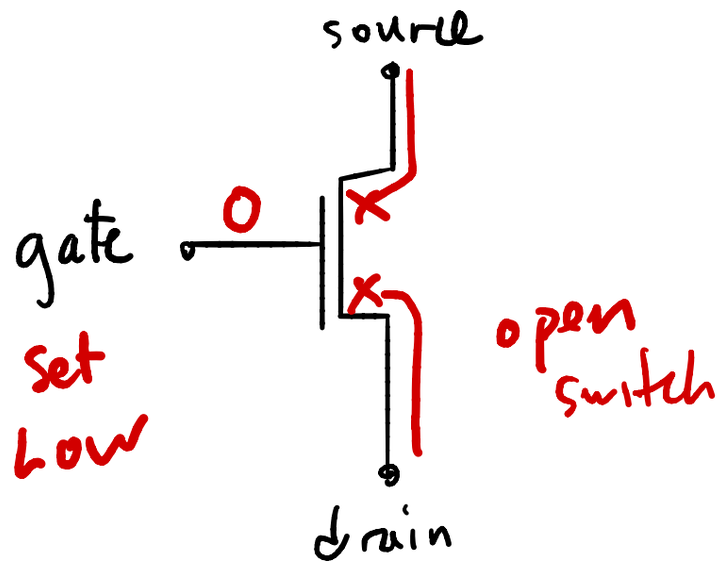
⇒ When an n-type transistor's gate is connected HIGH (signal of "1") it acts like a closed switch from the source to the drain.

TRANSISTOR LOGIC

n-type transistor behavior



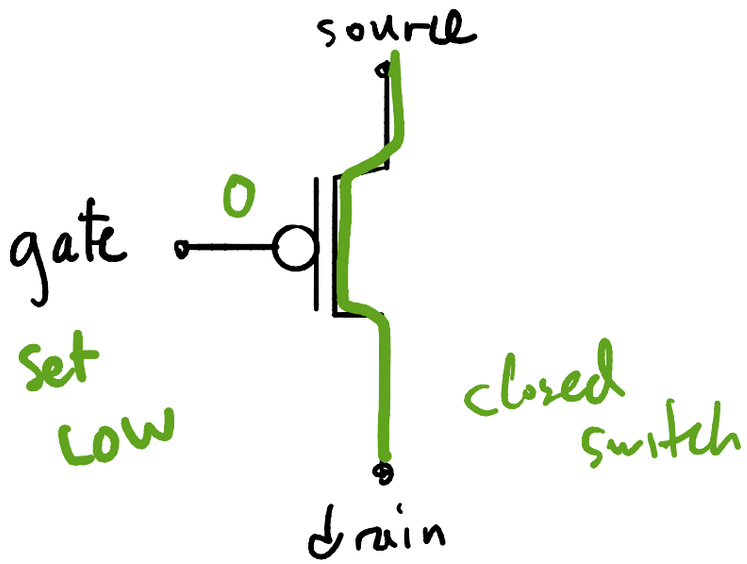
⇒ When an n-type transistor's gate is connected to HIGH (signal of "1") it acts like a closed switch from the source to the drain.



⇒ when its gate is connected to Low (set "0") it acts like an open switch. Does not conduct.

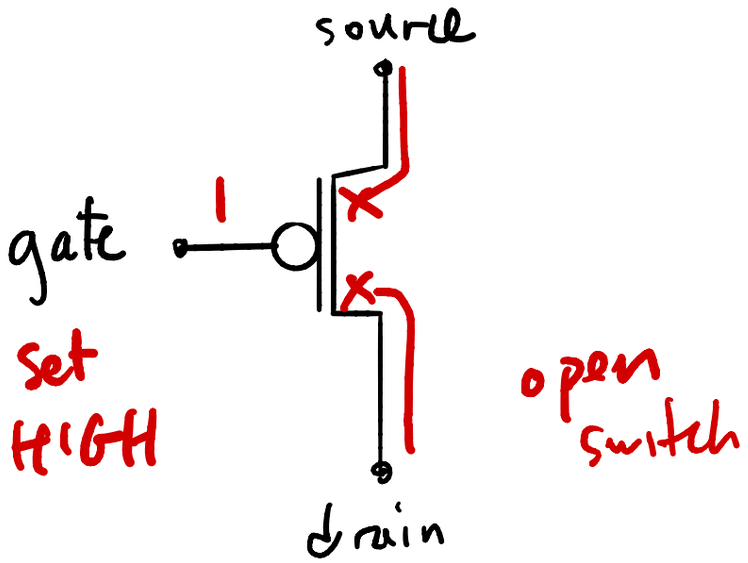
TRANSISTOR LOGIC

p-type transistor behaves oppositely...



Its logic is inverted...

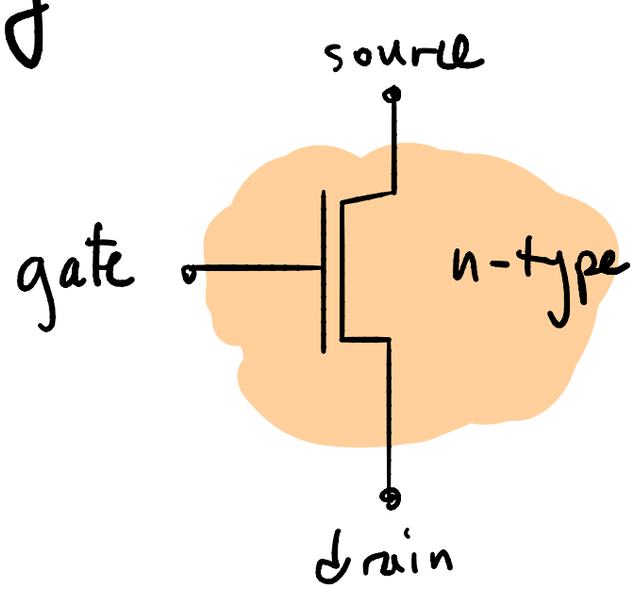
⇒ when gate is LOW (0) the p-type transistor conducts.

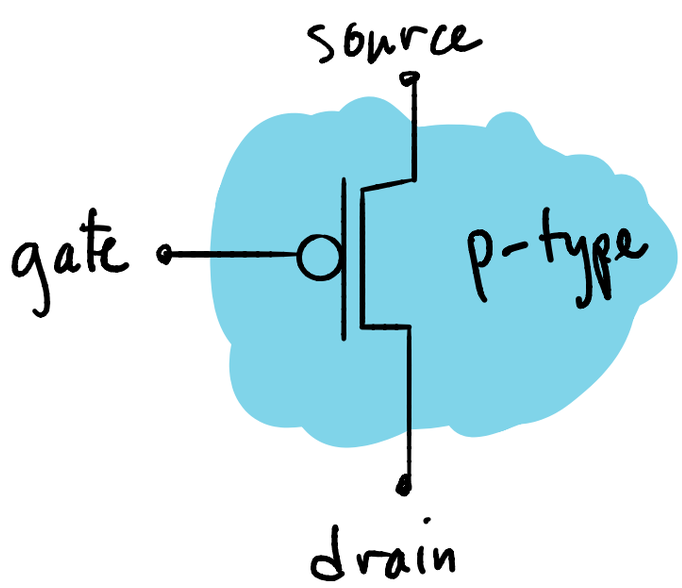


⇒ when gate is set HIGH (1) it does not conduct.

TRANSISTOR LOGIC

Summary:





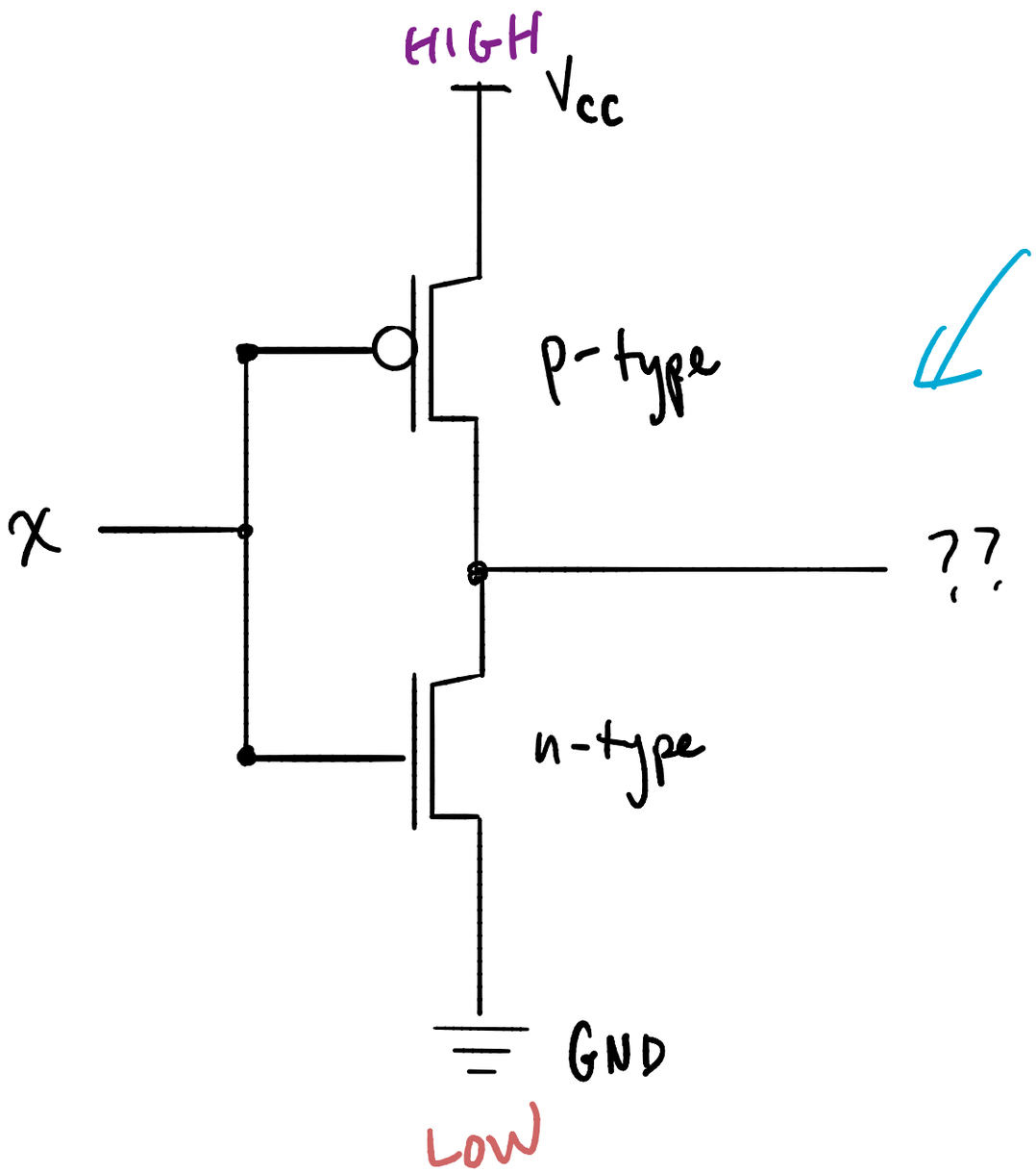
n-type

- when gate connected to HIGH, conduction between source and drain
- when connected to LOW, no conduction

p-type

- when gate connected to LOW, conduction between source and drain
- when connected to HIGH, no conduction

TRANSISTOR LOGIC: INVERTER

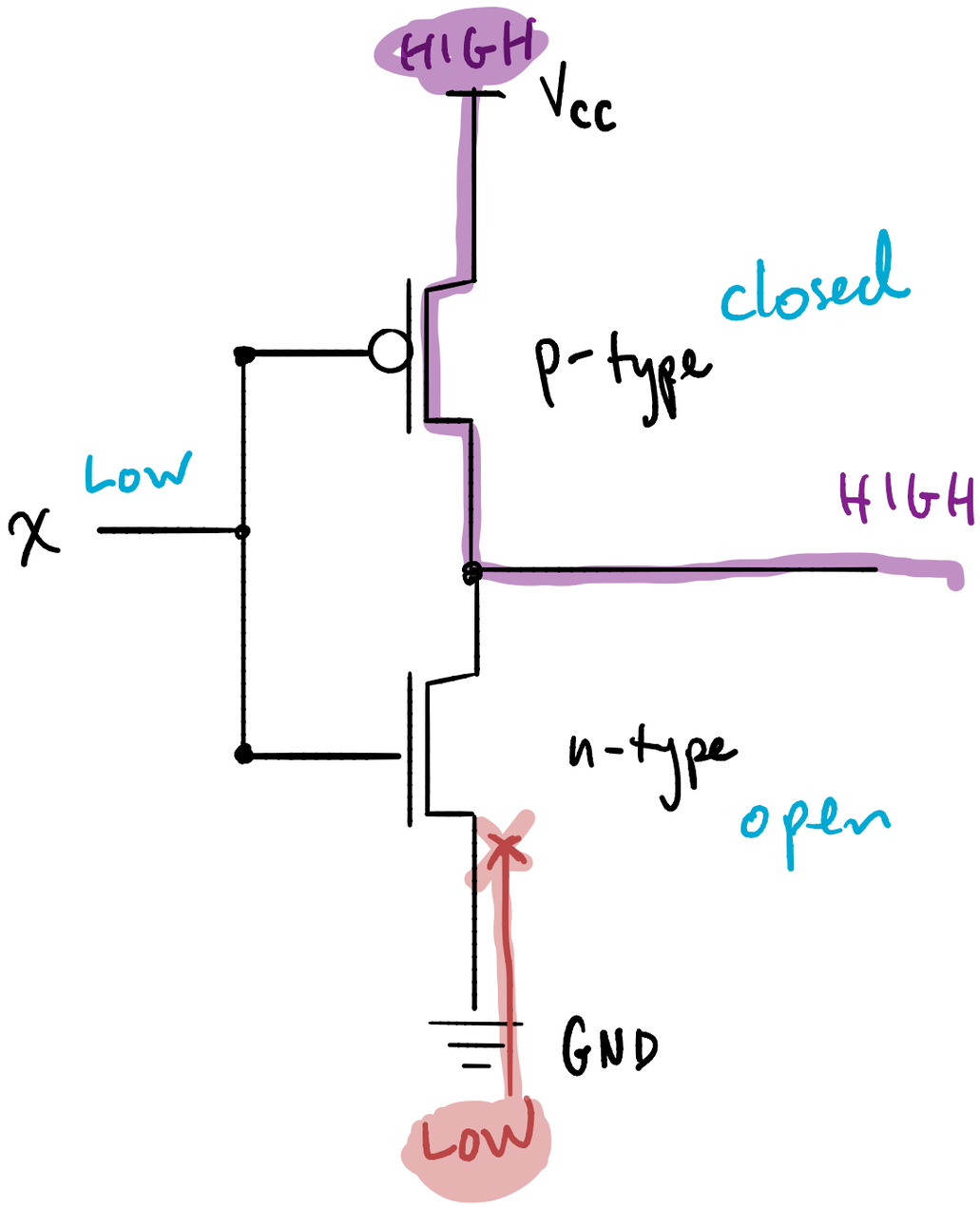


How does this transistor circuit behave?

truth table:

x	output
Low	??
HIGH	??

TRANSISTOR LOGIC: INVERTER

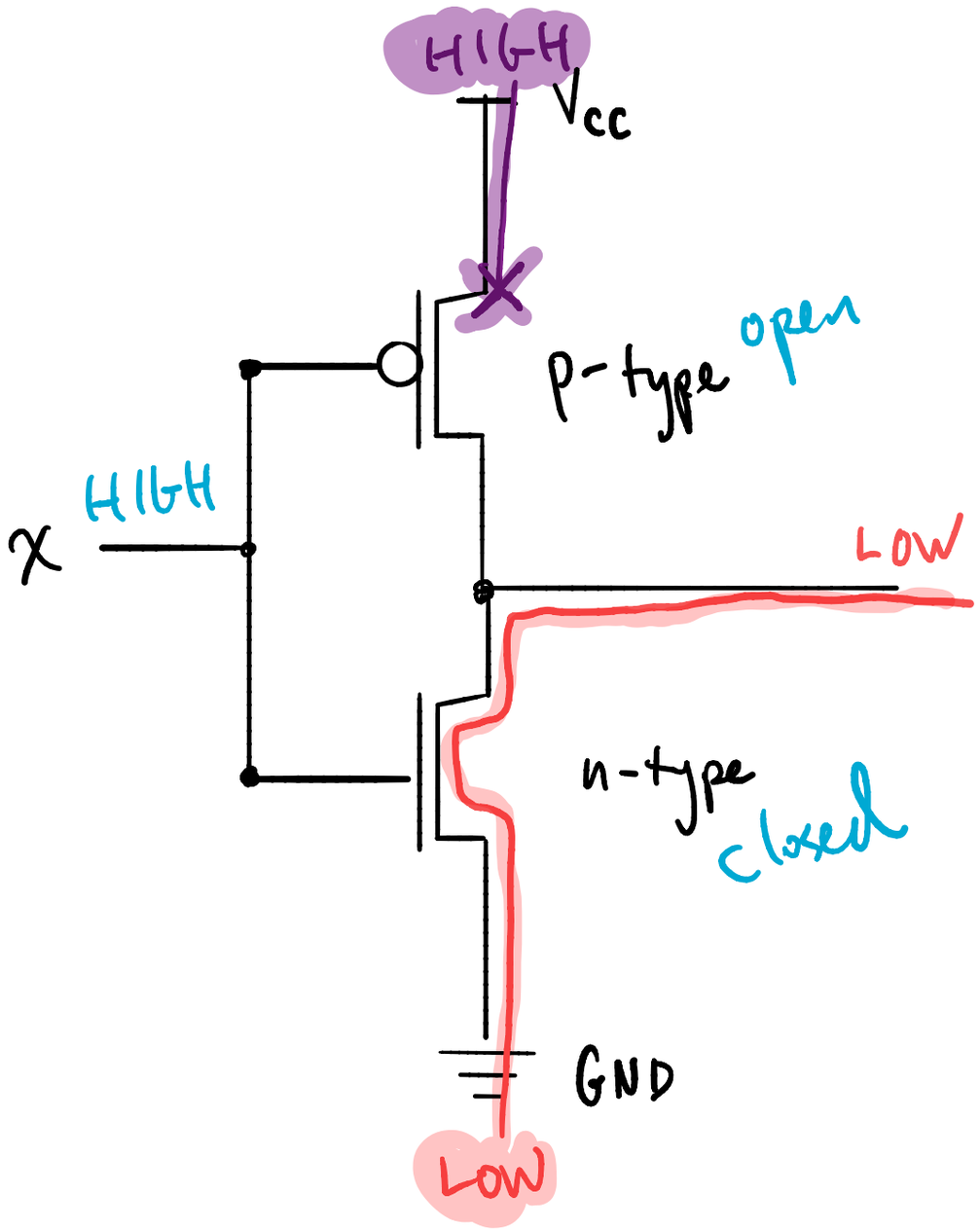


Let's set the input line x to Low...

truth table:

x	output
Low	HIGH
HIGH	

TRANSISTOR LOGIC: INVERTER

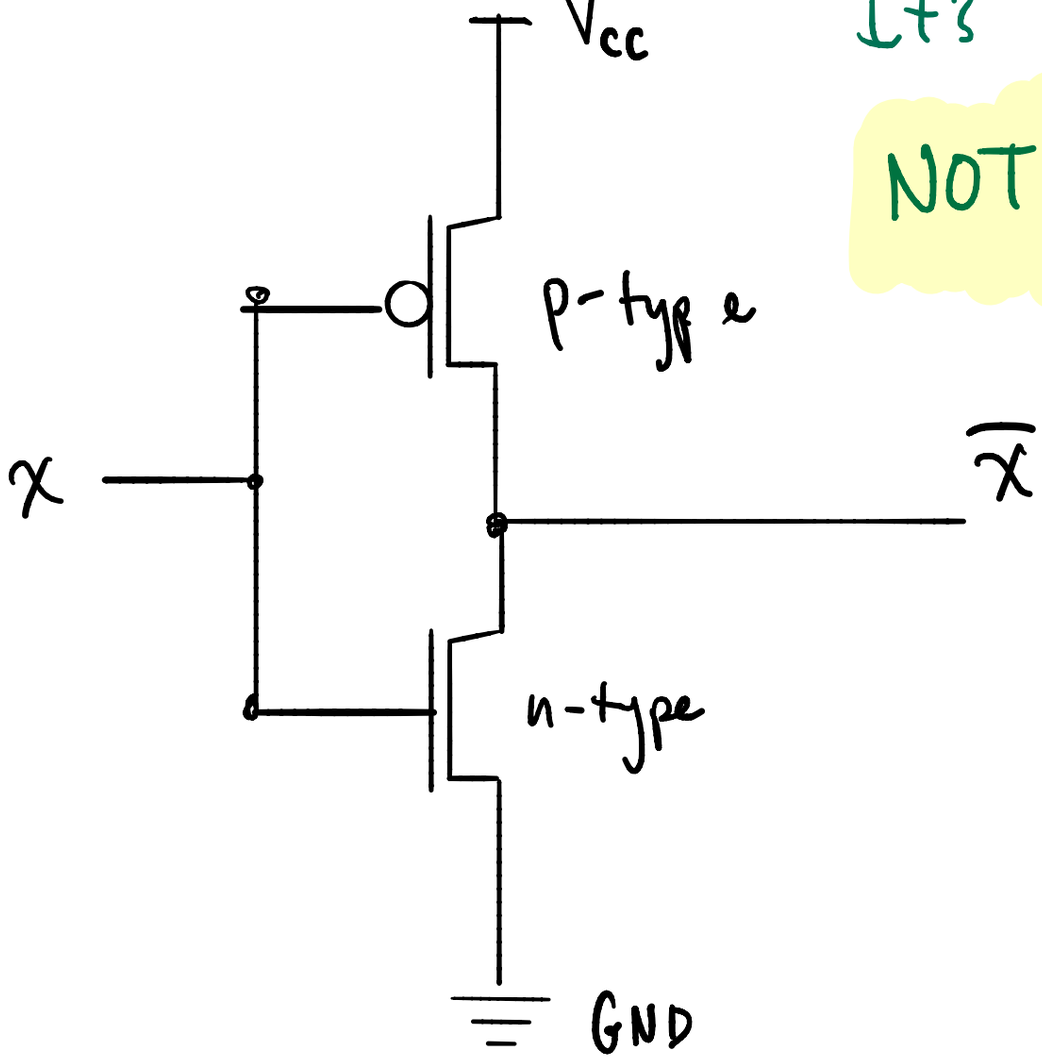


let's set the input line x to HIGH...

truth table:

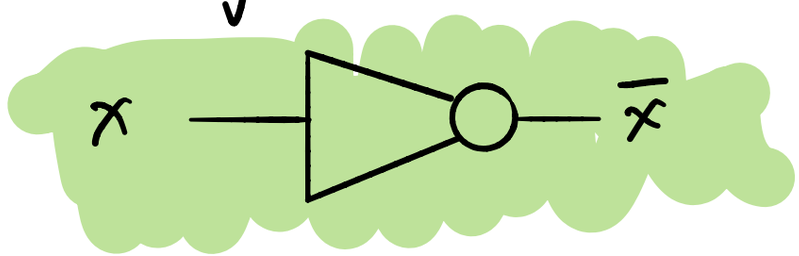
x	output
Low	HIGH
HIGH	Low

TRANSISTOR LOGIC: INVERTER



It's a NOT gate!

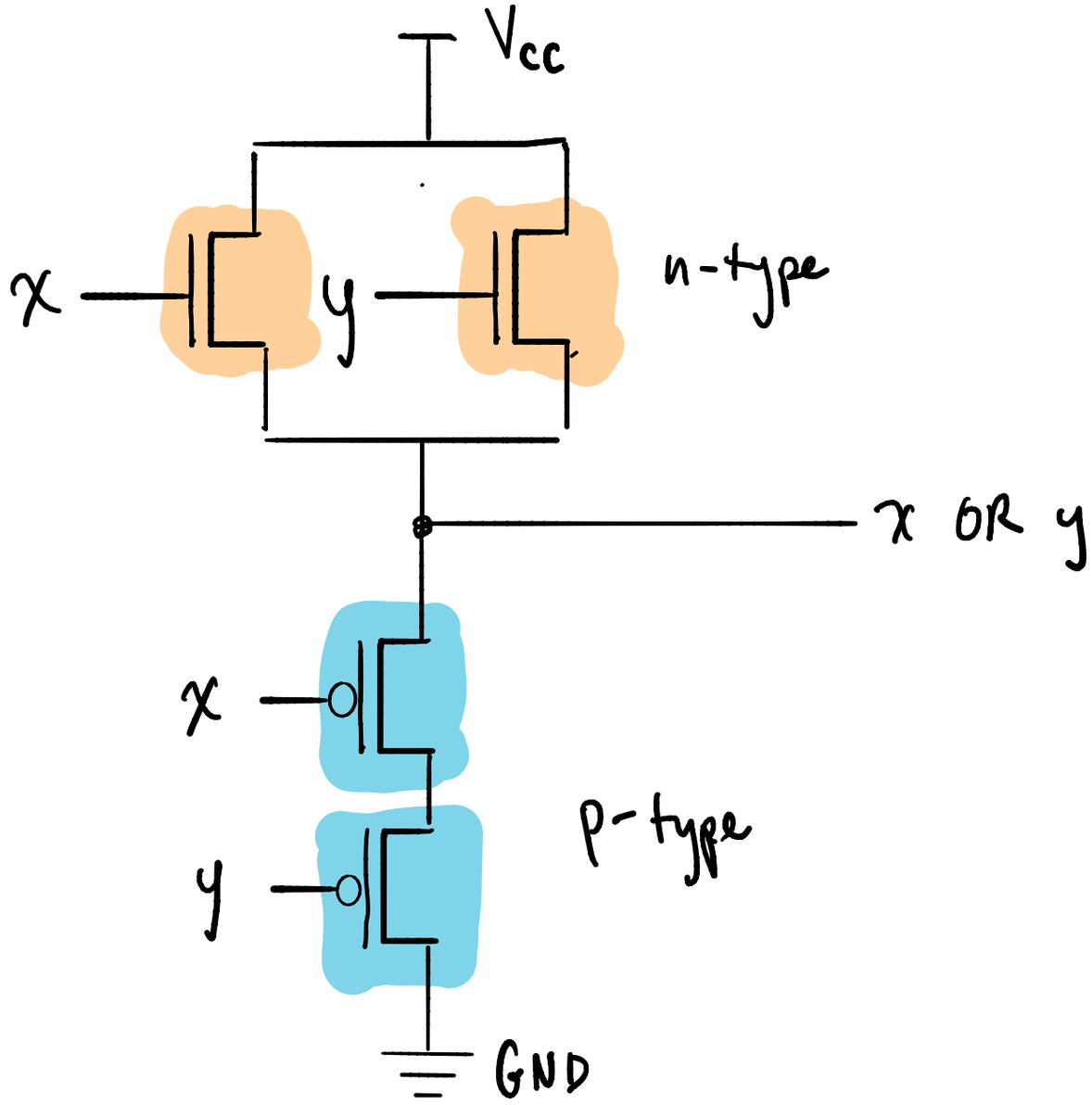
symbol:



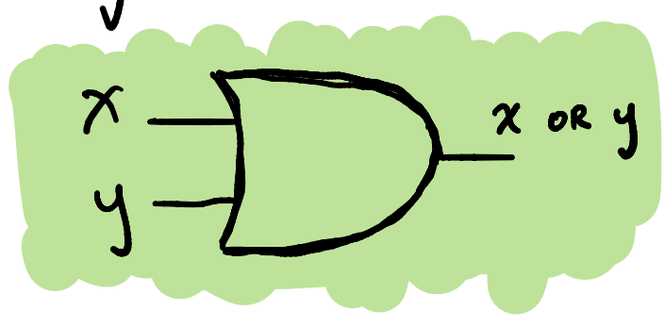
truth table:

x	$\bar{x}$
0	1
1	0

TRANSISTOR LOGIC: OR GATE



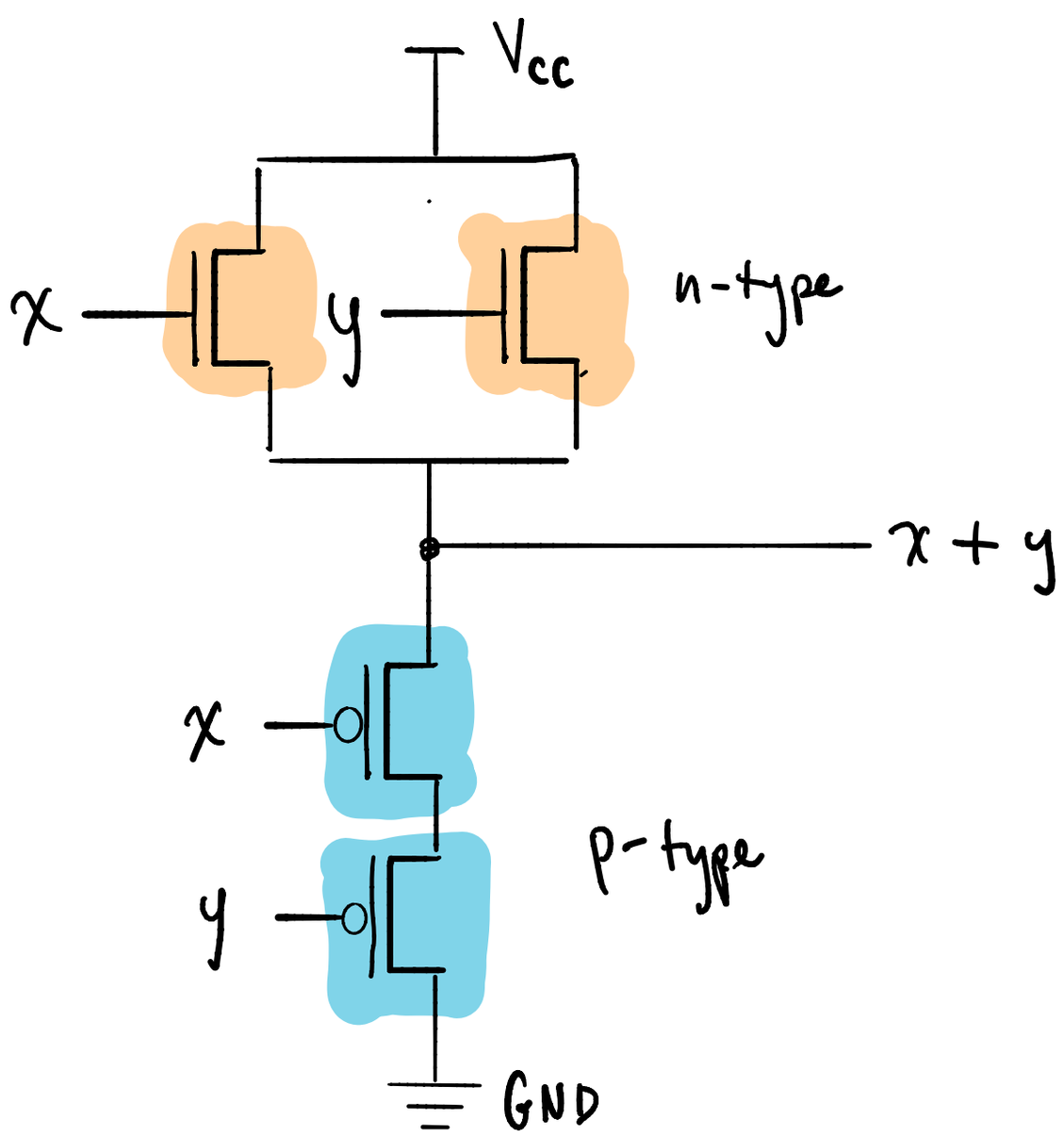
symbol:



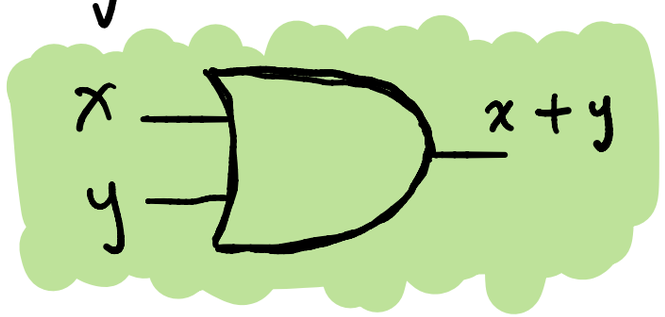
truth table:

x	y	x OR y
Low	Low	Low
Low	HIGH	HIGH
HIGH	Low	HIGH
HIGH	HIGH	HIGH

TRANSISTOR LOGIC: OR GATE



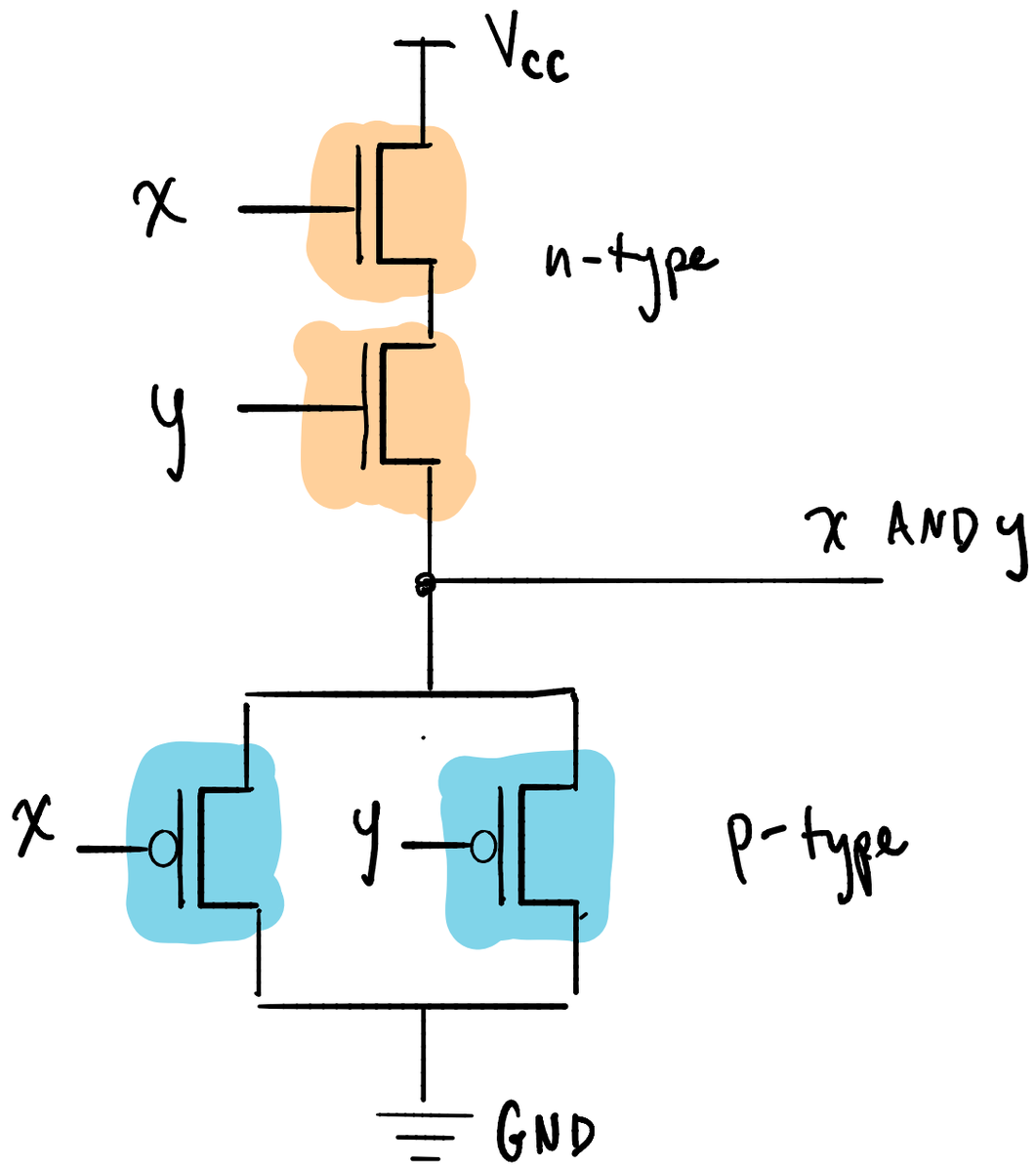
symbol:



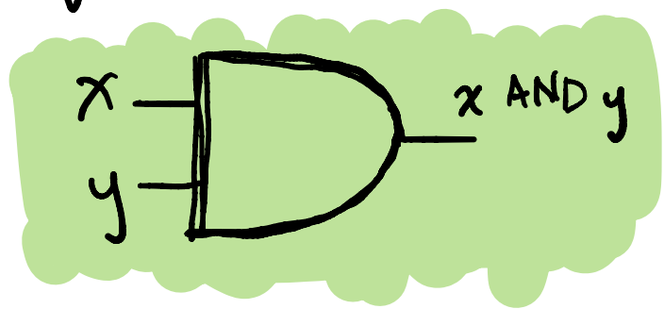
truth table:

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

TRANSISTOR LOGIC: AND GATE



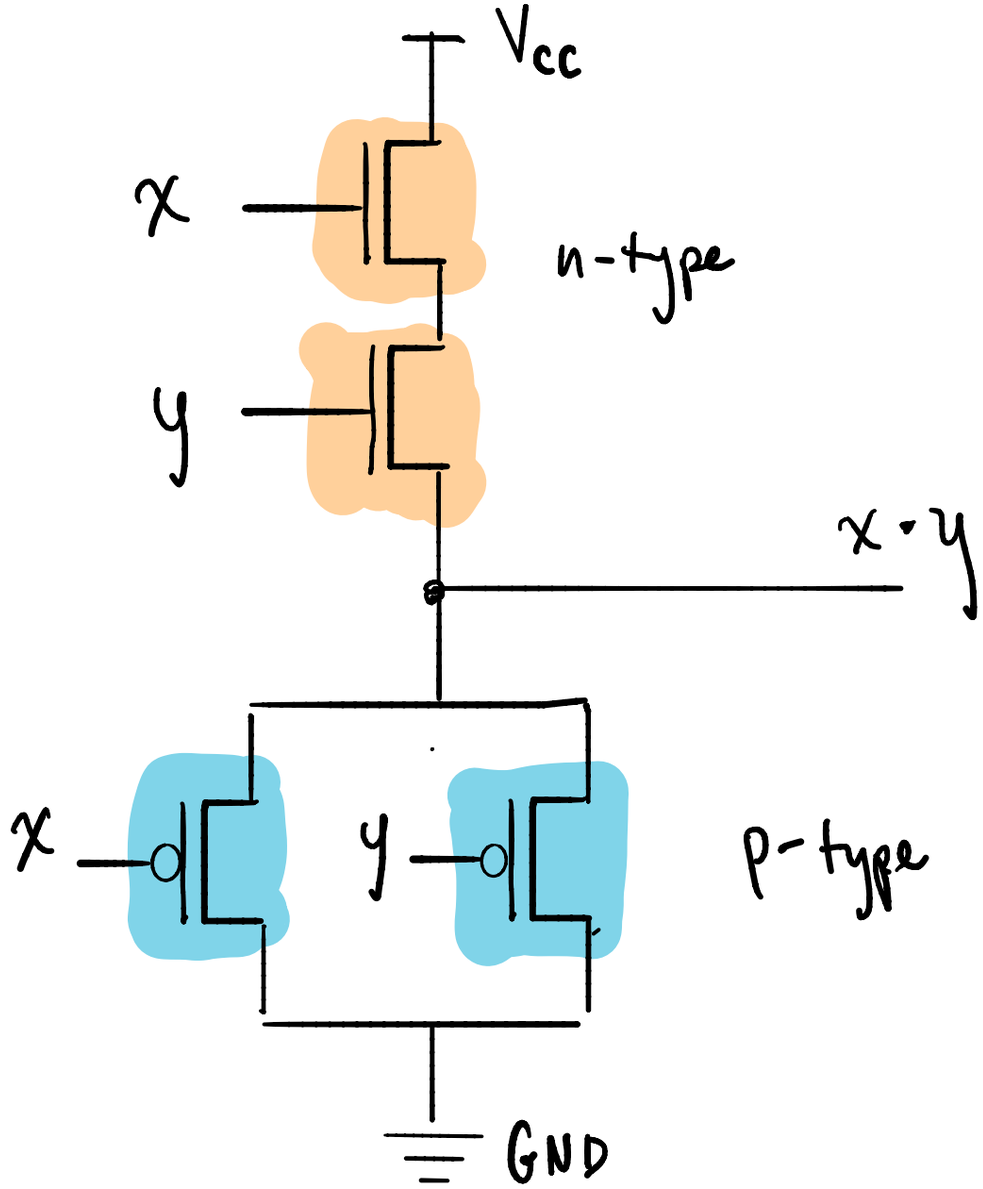
symbol:



truth table:

x	y	x AND y
LOW	LOW	LOW
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	HIGH

TRANSISTOR LOGIC: AND GATE

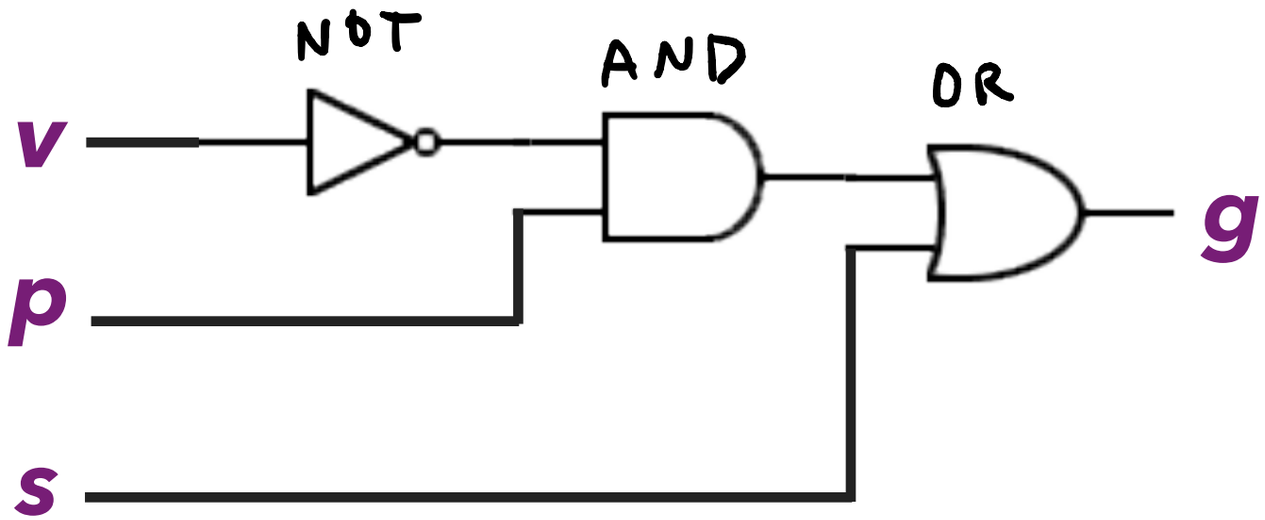


symbol:

truth table:

x	y	x·y
0	0	0
0	1	0
1	0	0
1	1	1

REACTOR CONTROL CIRCUIT



truth table

v	p	s	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Boolean expression:

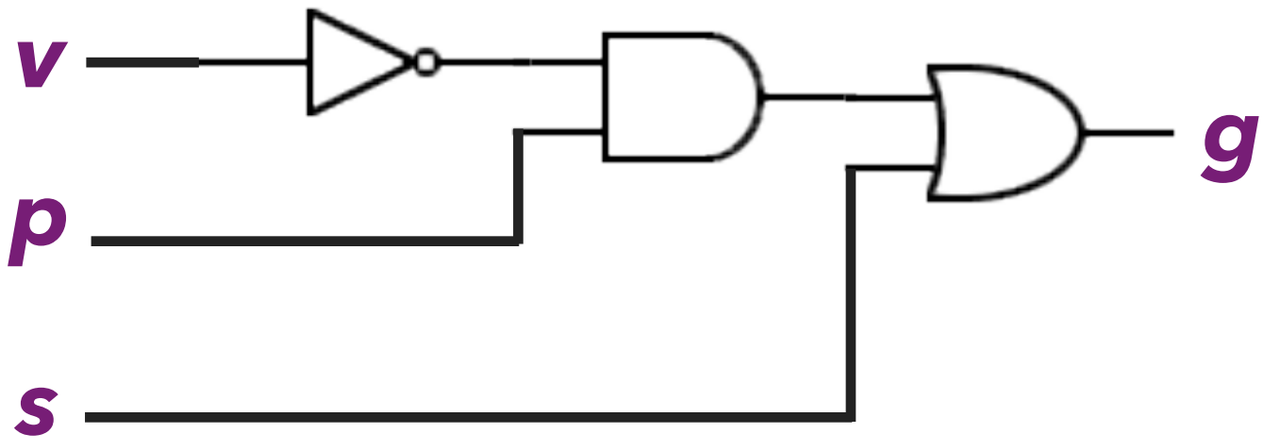
((NOT v) AND p) OR s "pythonic"

((!v) && p) || s C++

(¬v ∧ p) ∨ s CS theory

v · p + s digital design

REACTOR CONTROL CIRCUIT



truth table

v	p	s	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

g := ((NOT v) AND p) OR s

Alternatively...

$g := \bar{v} \cdot p + s$

EXCLUSIVE OR CIRCUIT

- logical OR is inclusive
  true when both inputs true or either is true

inclusive OR t.t.

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

- exclusive OR
  true when exactly one input is true

exclusive OR t.t.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR CIRCUIT

Let's express the logic of XOR using AND, OR, NOT...

exclusive OR t.t.

x	y	$x \oplus y$	
0	0	0	
0	1	1	⇐
1	0	1	
1	1	0	

Consider rows where output is one. Describe their input configuration...

2nd row: $\bar{x} \cdot y$

EXCLUSIVE OR CIRCUIT

Let's express the logic of XOR using AND, OR, NOT...

exclusive OR t.t.

x	y	$x \oplus y$	
0	0	0	
0	1	1	⇐
1	0	1	⇐
1	1	0	

Consider rows where output is one. Describe their input configuration...

2nd row: $\bar{x} \cdot y$

3rd row: $x \cdot \bar{y}$

EXCLUSIVE OR CIRCUIT

Let's express the logic of XOR using AND, OR, NOT...

exclusive OR t.t.

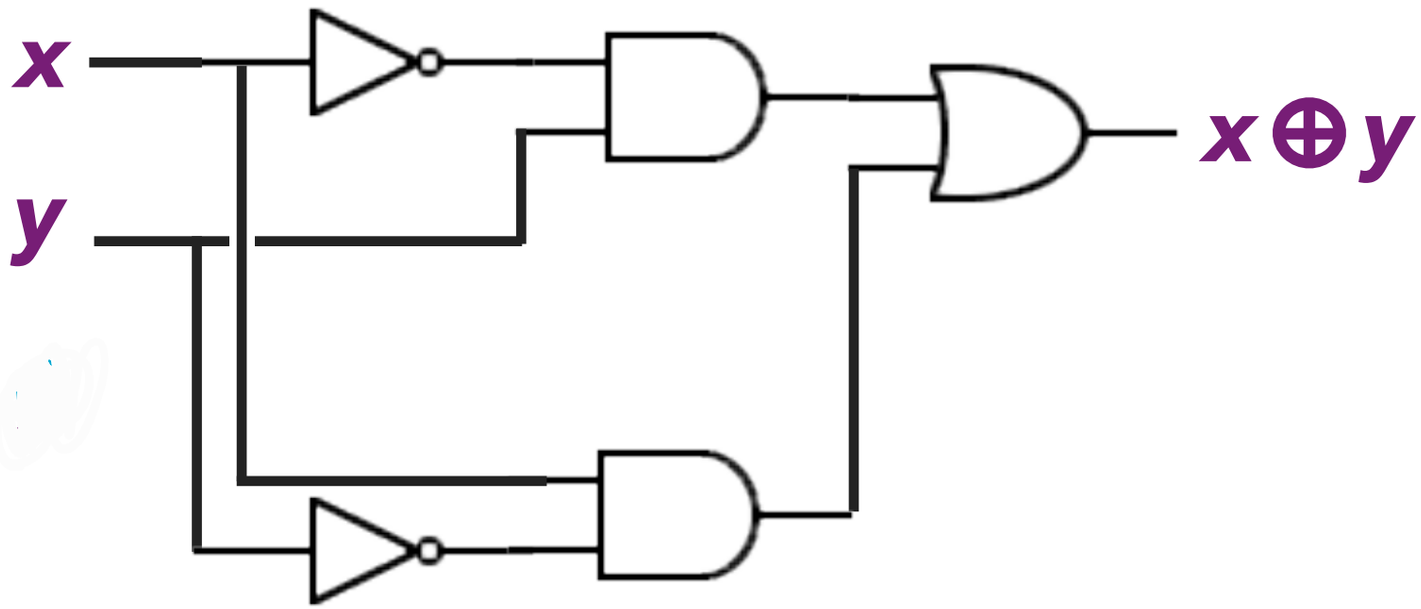
x	y	x ⊕ y	
0	0	0	
0	1	1	⇐
1	0	1	⇐
1	1	0	

Boolean expression is just inclusive or of those cases...

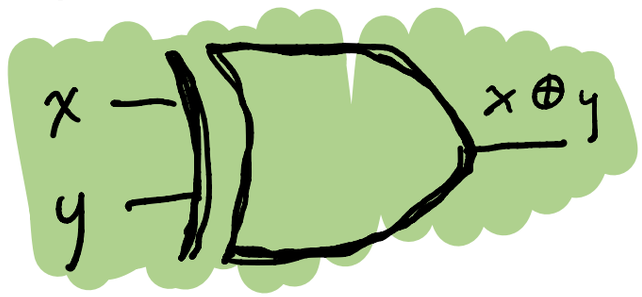
$$\bar{x} \cdot y + x \cdot \bar{y}$$

This method always yields a sum of products.

EXCLUSIVE OR CIRCUIT



symbol:

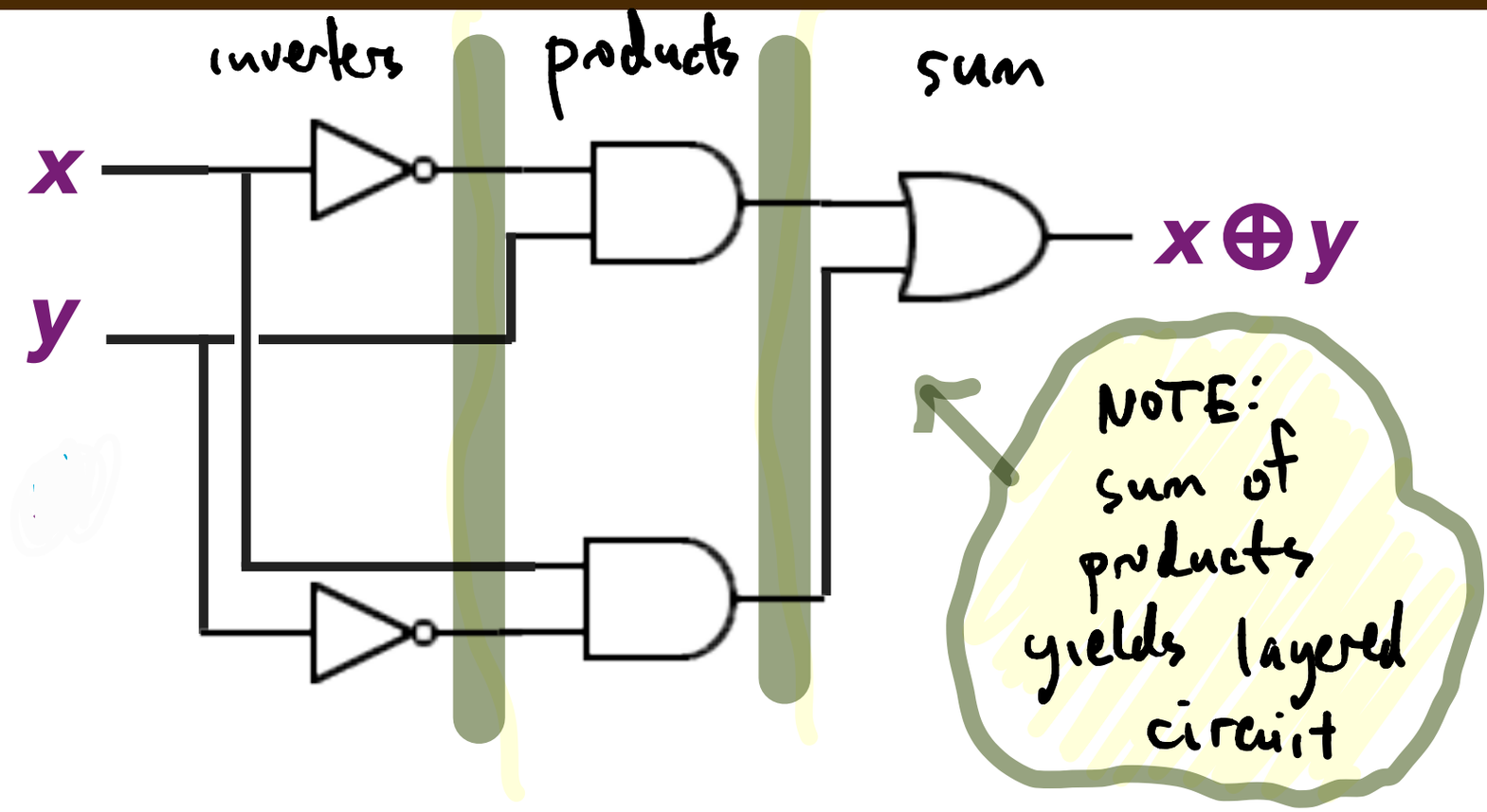


$$x \oplus y := \bar{x} \cdot y + x \cdot \bar{y}$$

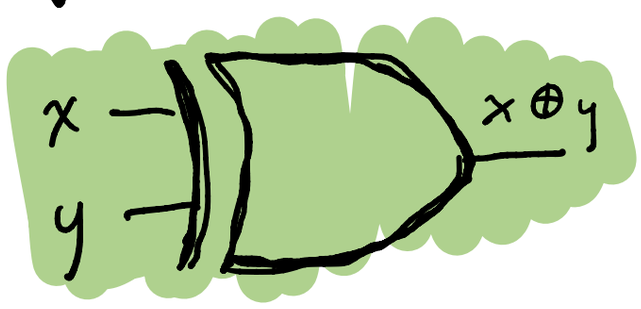
truth table:

x	y	x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR CIRCUIT



symbol:



truth table:

x	y	x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

$$x \oplus y := \bar{x} \cdot y + x \cdot \bar{y}$$

In various boolean expression notations...

((NOT x) AND y) OR (x AND (NOT y))

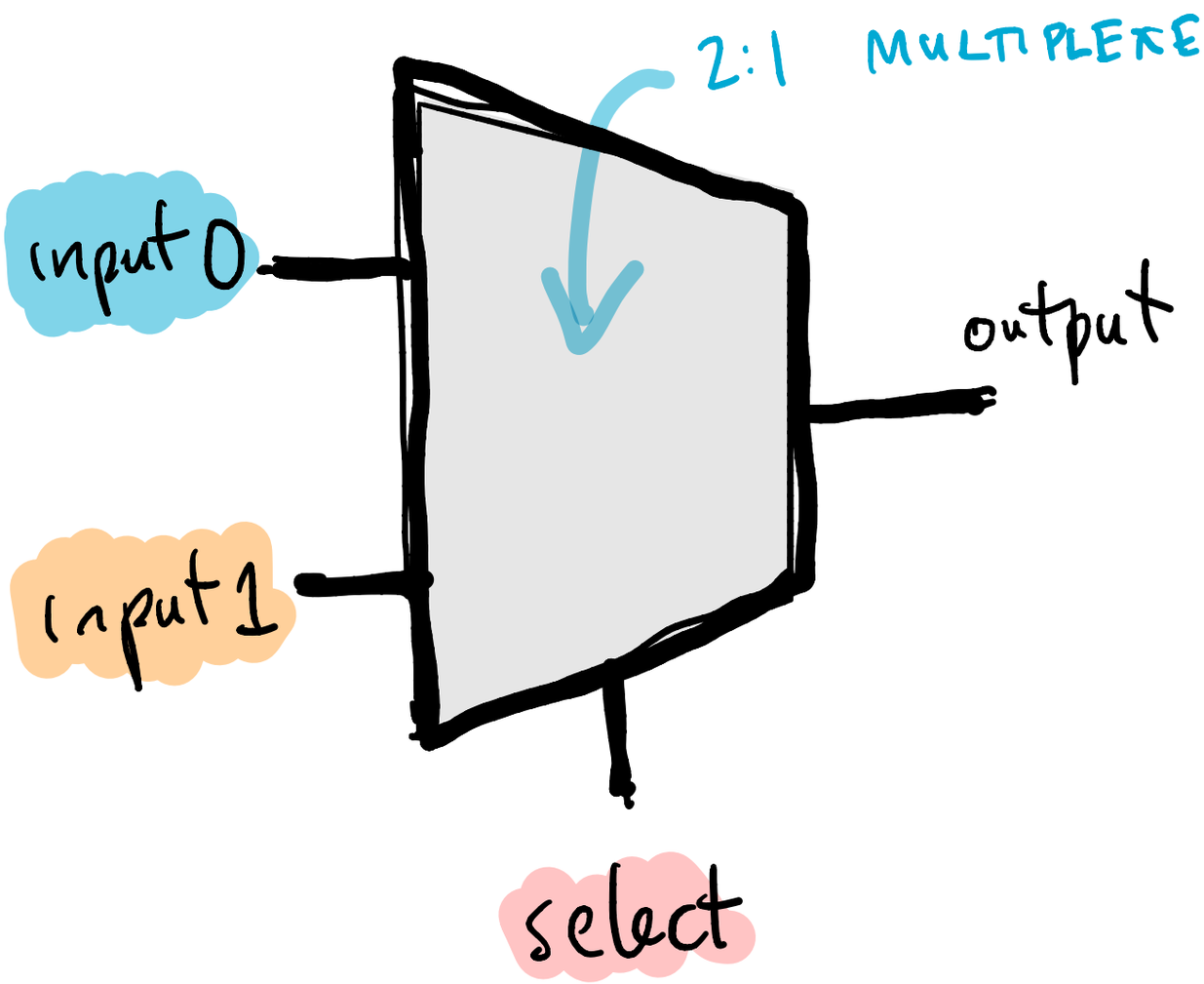
((!x) && y) || (x && (!y))

((¬x) ∧ y) ∨ (x ∧ (¬y))

$\bar{x} \cdot y + x \cdot \bar{y}$

2:1 MULTIPLEXER CIRCUIT

Can create a "switch-like circuit":

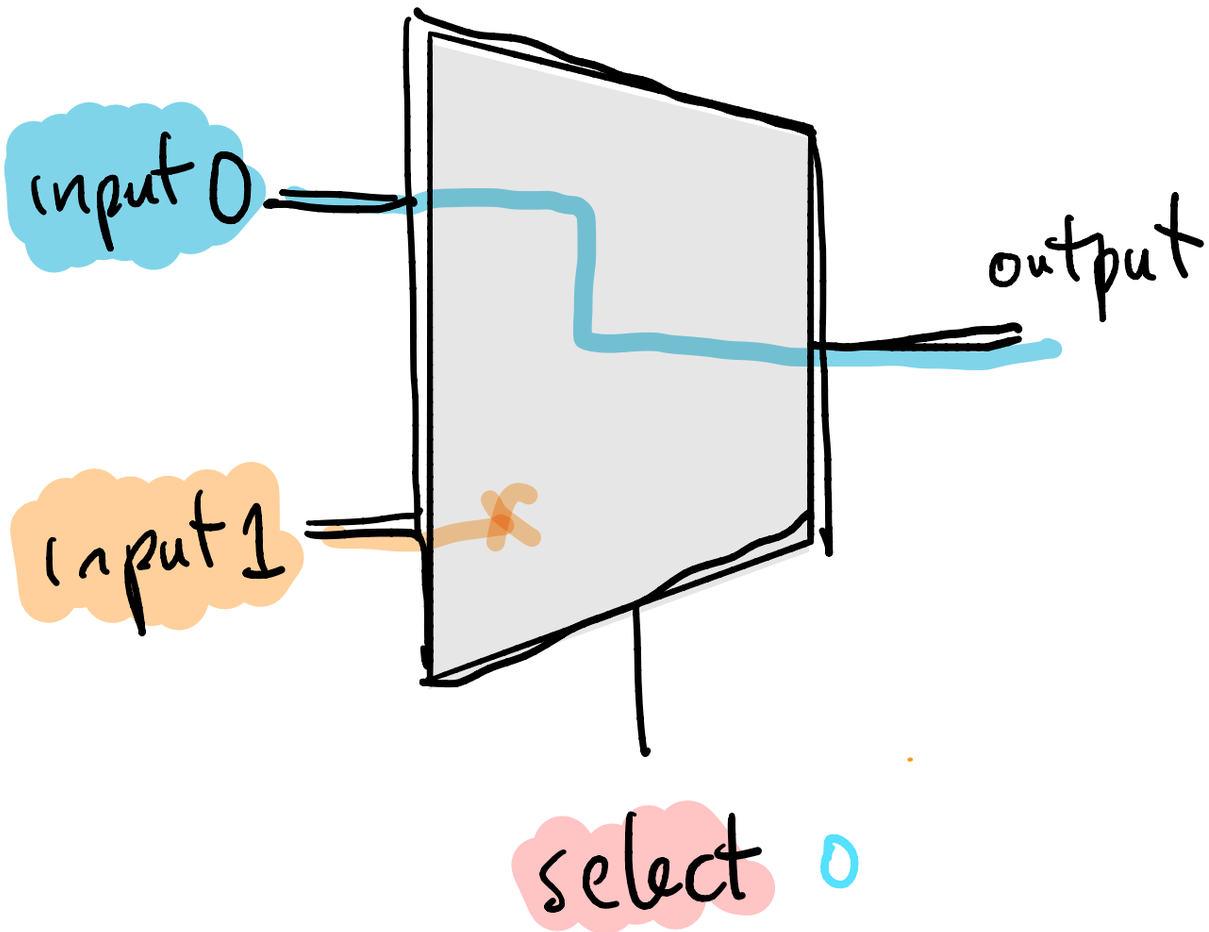


S	i_0	i_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2:1 MULTIPLEXER CIRCUIT

Can create a "switch-like circuit":

With select set to 0, outputs input 0...

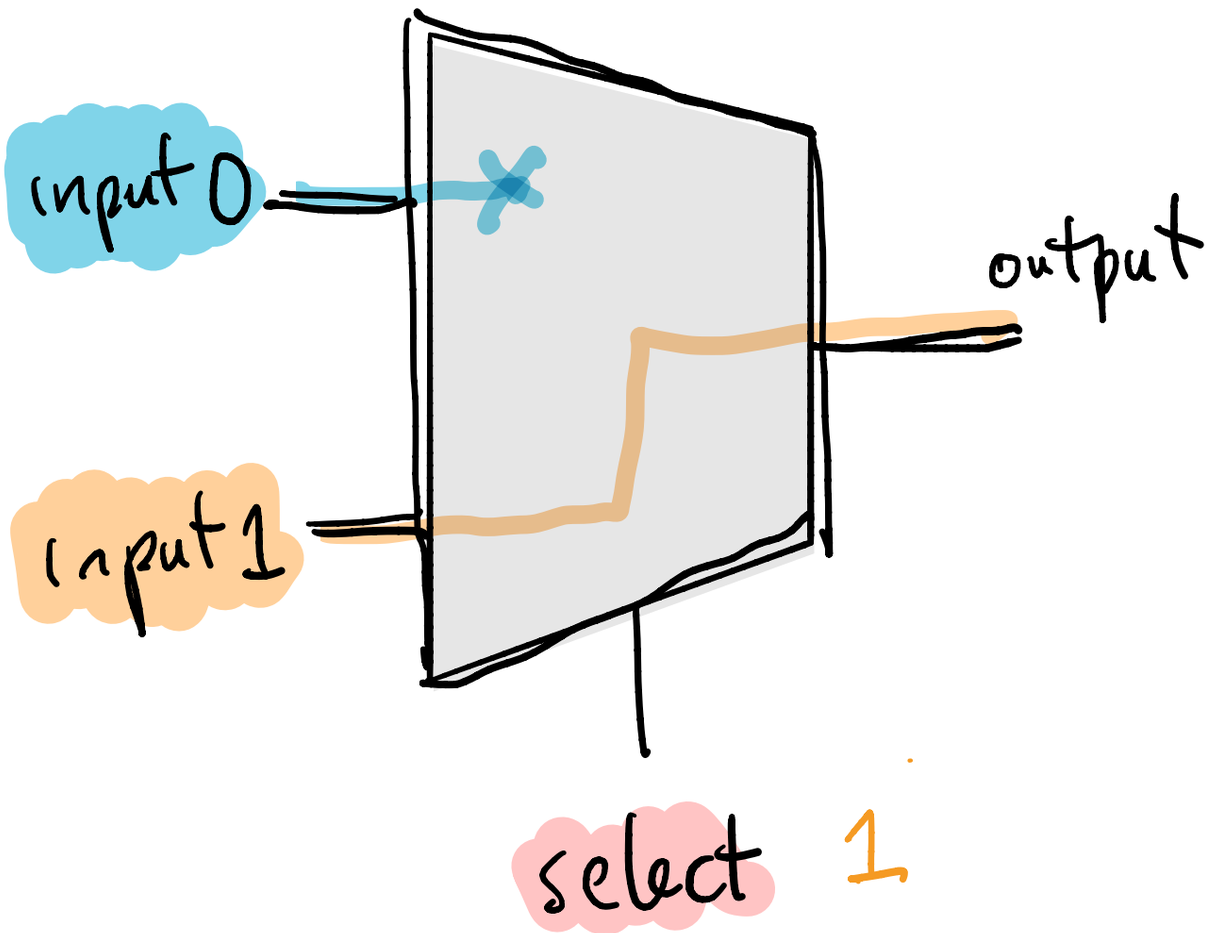


s	i_0	i_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2:1 MULTIPLEXER CIRCUIT

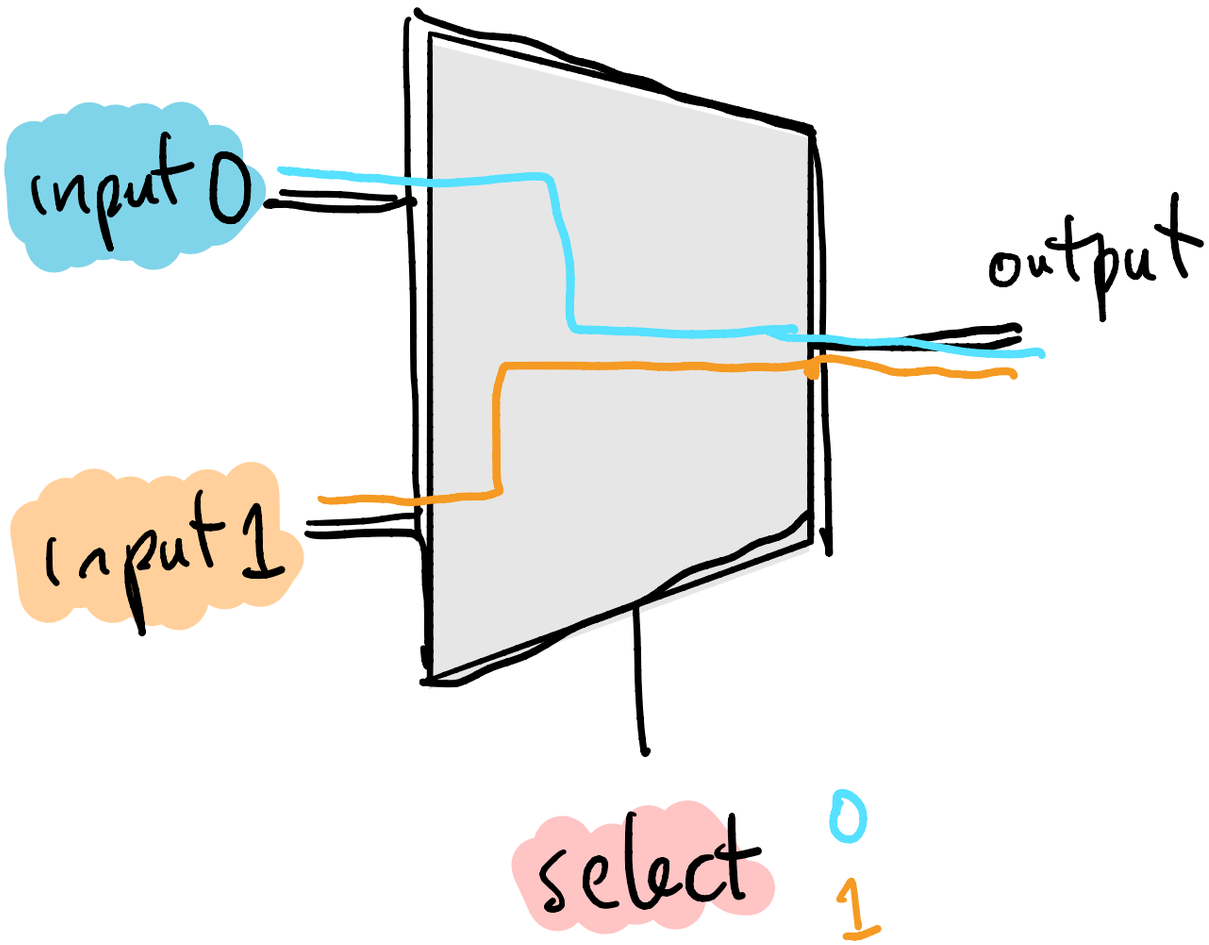
Can create a "switch-like circuit":

With select set to 1, outputs input 1 ...



s	i_0	i_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2:1 MULTIPLEXER CIRCUIT



S	i_0	i_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2:1 MULTIPLEXER CIRCUIT

Let's express the logic using AND, OR, NOT...

$$O := \bar{S} \cdot i_0 \cdot \bar{i_1} + \bar{S} \cdot i_0 \cdot i_1 + S \cdot \bar{i_0} \cdot i_1 + S \cdot i_0 \cdot i_1$$

S	i_0	i_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2:1 MULTIPLEXER CIRCUIT

Let's express the logic using AND, OR, NOT...

$$o := \begin{cases} \bar{s} \cdot i_0 \cdot \bar{i}_1 + \bar{s} \cdot i_0 \cdot i_1 \\ + \\ s \cdot \bar{i}_0 \cdot i_1 + s \cdot i_0 \cdot i_1 \end{cases} = \bar{s} \cdot i_0 + s \cdot i_1$$

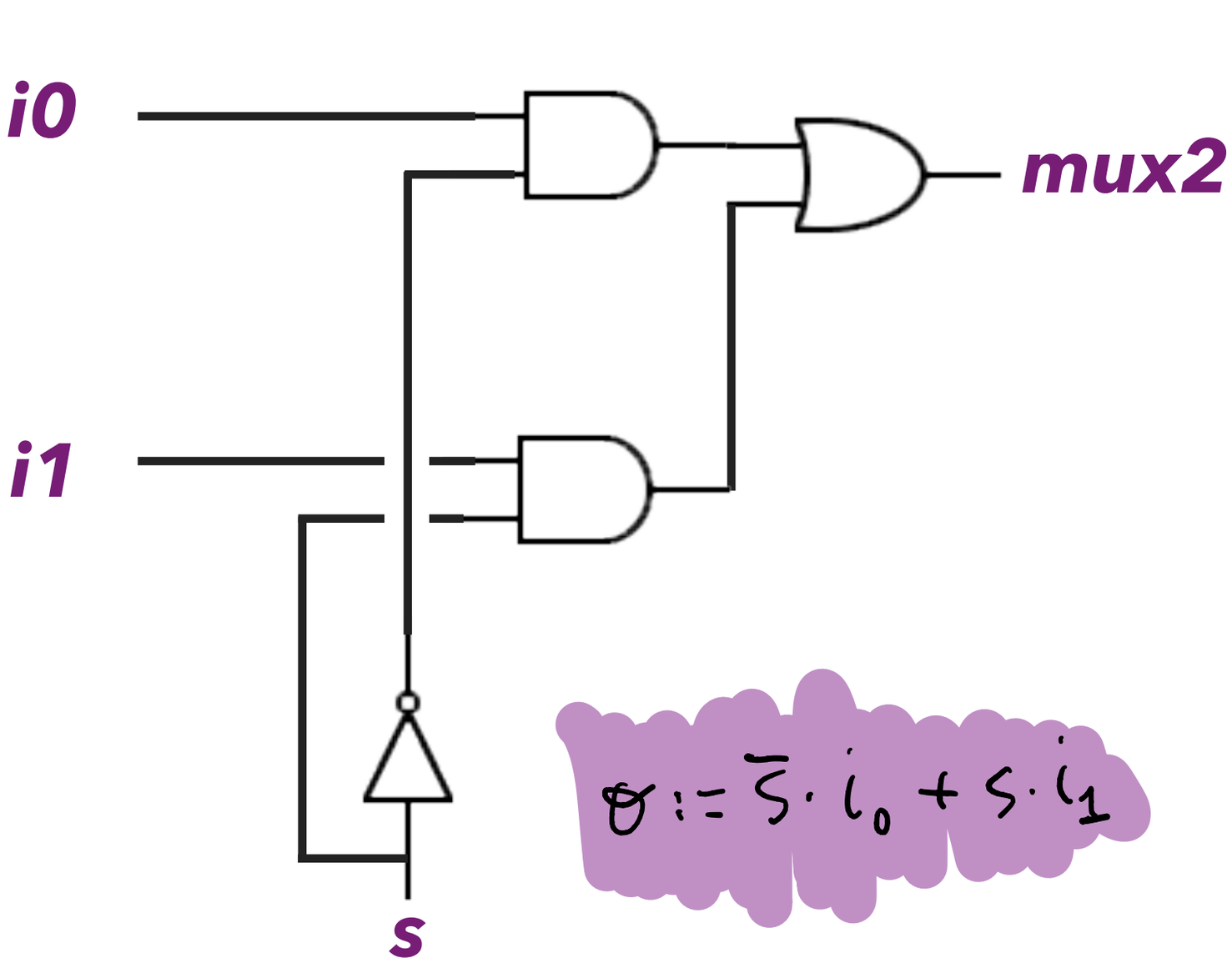
$\bar{s} \cdot i_0$ — see below

$s \cdot i_1$ — similar

Note that:

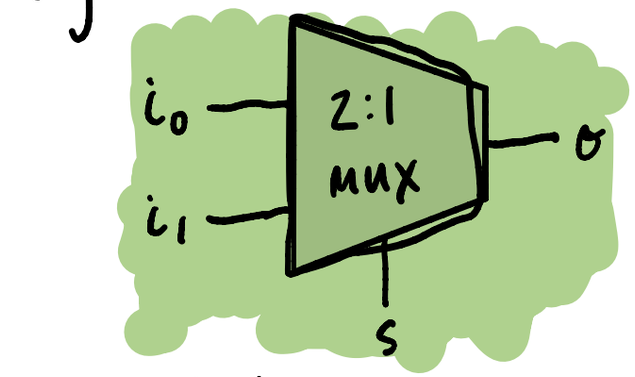
$$\bar{s} \cdot i_0 \cdot \bar{i}_1 + \bar{s} \cdot i_0 \cdot i_1 = \bar{s} \cdot i_0 (\bar{i}_1 + i_1) = \bar{s} \cdot i_0 \cdot 1 = \bar{s} \cdot i_0$$

2:1 MULTIPLEXER CIRCUIT



$$o := \bar{s} \cdot i_0 + s \cdot i_1$$

symbol:



truth table:

s	i_0	i_1	o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

We'll look at some more boolean algebra rules on Wednesday...

We'll also use MUX ad adders to build a calculator circuit.