Basic Electronics and Digital Logic
Computer Science 240

Laboratory 1

- **Administrivia**
- **Lab Environment**
- **Basic Electronics (Ohm's law, transistors, logic gates)**
- **Truth Tables, Sum-of-Products**
- **Boolean Identities**
- **Universal Gates**
- **Integrated circuits**
- **Protoboard**
- **LogicWorks**

Lab Environment

- All lab exercises and reports will be *Google Docs*, and should be shared with lab partner and the instructor
- Bring a laptop to lab if you have it (helpful to use a second computer for the lab report)
- From lab machine booted to Linux, you can enter Linux commands using a **terminal/shell**
- You can also use a terminal from either

Mac (*Terminal*) or PC (*PuTTY*) to open a remote connection to a Linux machine for command-line entry

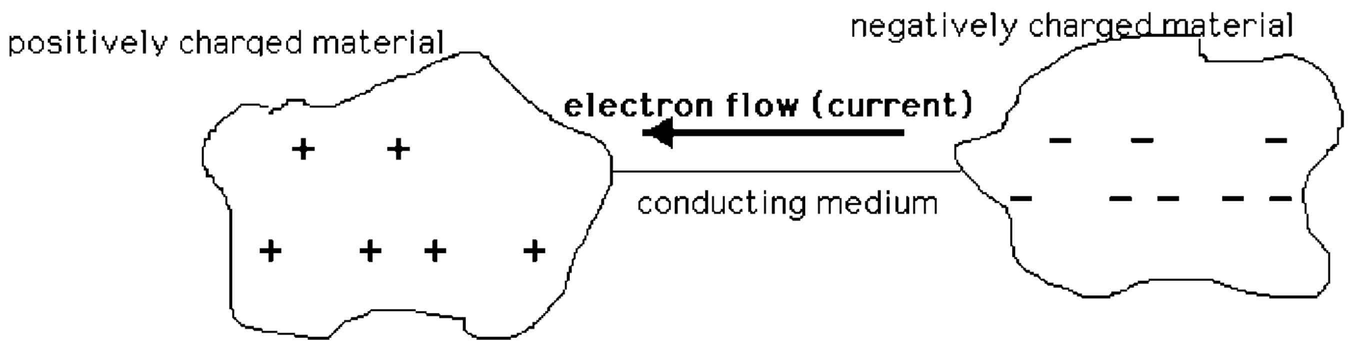
NOTE: for some exercises and assignments, you will be required to use the lab machines to compile and run your programs

Basic Concepts of Electricity

Electricity = **the movement of electrons** in a material

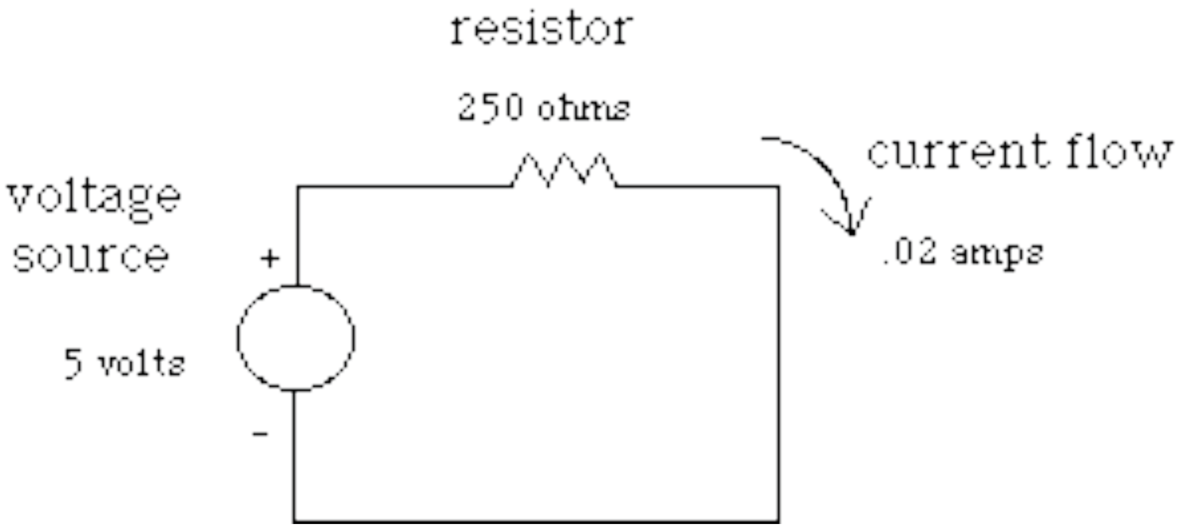
Materials tend to have a net negative or positive charge

Difference of charge between two points = **potential difference (V)**



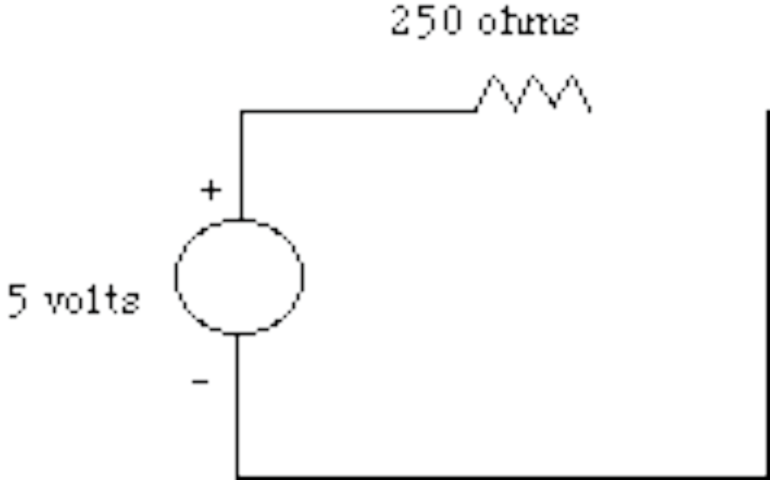
Rate at which electrons flow through = **current (A)**.

Ease of conduction, or current flow = **resistance (Ω)**

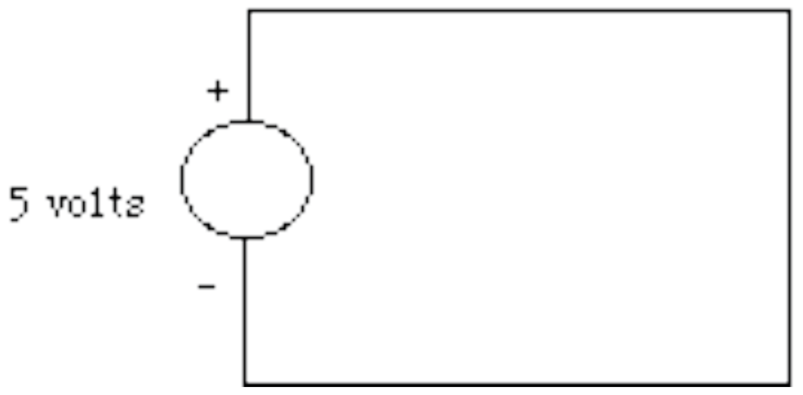


Ohm's Law, V = IR.

Open circuit = no current

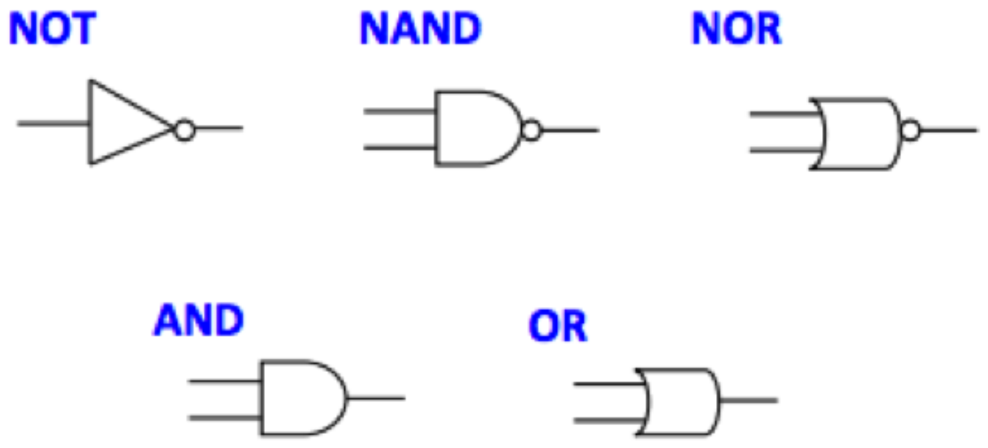


Short circuit = infinite current, since V/0 = infinite current:



Infinite current swiftly results in the destruction of the circuit!

Basic Gate Symbols



Notation and Truth Tables for Basic Logic Gates

NOT
F = A′

A	F
0	1
1	0

NAND
F = (AB)′

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR
F = (A+B)′

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

AND
F = AB

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

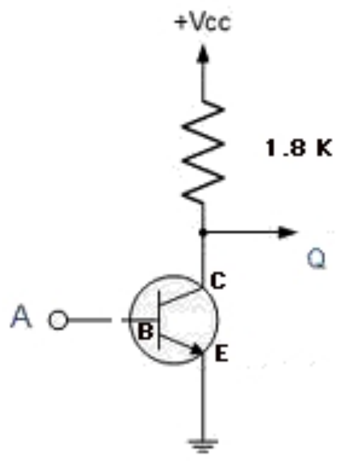
OR
F = A + B

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

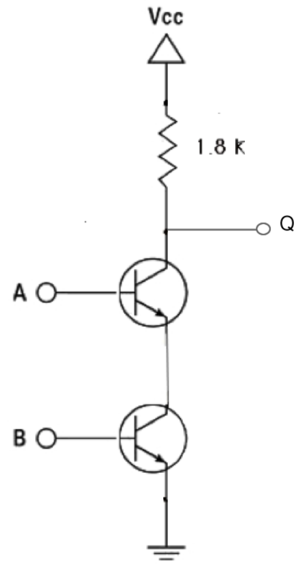
Basic Gates are built using Transistors

You have seen the circuits for NOT and NAND in lecture:

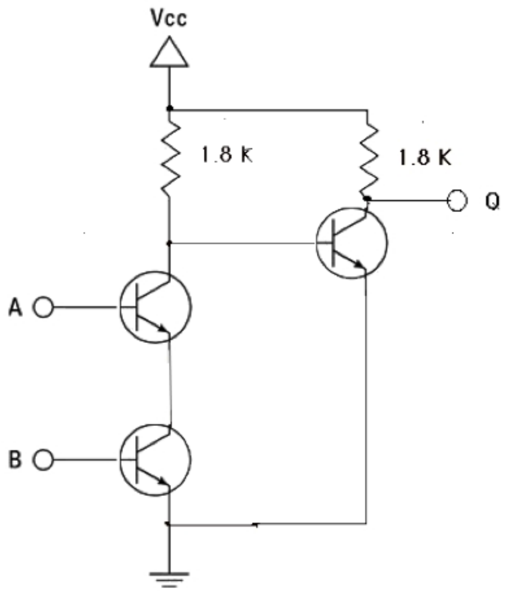
NOT – 1 transistor



NAND - 2 transistors



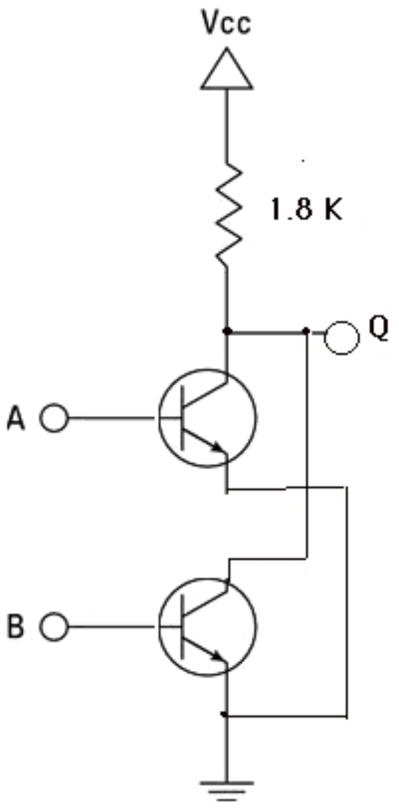
AND – uses 3 transistors (send the output of a NAND through another transistor acting as a NOT gate to complement the result):

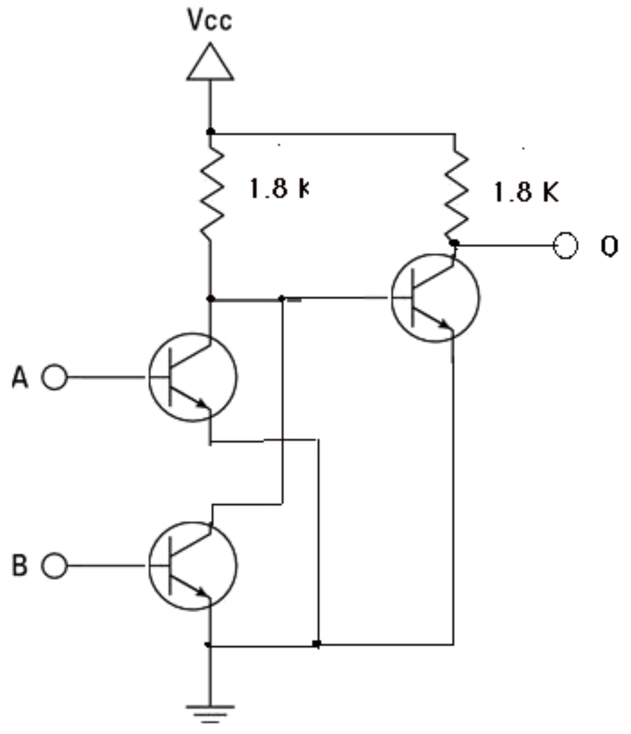


Similarly, these are the transistor circuits for a NOR and OR gate:

NOR – 2 transistors

OR – 3 transistors





Truth Tables and Sum-of-Products

Truth tables specify the output for all the given input combinations of a function.

An input combination can be expressed by ANDing together the inputs (each input or its' complement is used in the expression, depending upon which combination is being expressed)

A function can then be expressed as a **sum-of-products** by ORing together the input combinations which make the function true.

A	B	A'B'	A'B	A'B' + A'B
0	0	1	0	1
0	1	0	1	1
1	0	0	0	0
1	1	0	0	0

A	B	A'	A'B	A' B'	A'+A'B+A'B'
0	0	1	0	1	1
0	1	1	0	0	1
1	0	0	0	0	0
1	1	0	0	0	0

F = A'B' + A'B   Q = A' + A'B + A'B'

F and Q are equivalent (produce the same function) when they have the same truth table.

When there is an equivalent circuit that uses fewer gates, transistors, or chips, it is preferable to use that circuit in the design

Identities of Boolean Algebra

Equivalency can also be proved using the identities of Boolean algebra

-	Identity law	$1A = A$	$0 + A = A$
-	Null law	$0A = 0$	$1 + A = 1$
-	Idempotent law	$AA = A$	$A + A = A$
-	Inverse law	$AA' = 0$	$A + A' = 1$
-	Commutative law	$AB = BA$	$A + B = B + A$
-	Associative law	$(AB)C = A(BC)$ $(A + B) + C = A + (B + C)$	
-	Distributive law	$A + BC = (A + B)(A + C)$ $A(B + C) = AB + AC$	
-	Absorption law	$A(A + B) = A$ $A + AB = A$	
-	De Morgan's law	$(AB)' = A' + B'$ $(A + B)' = A'B'$	

Example:

$F = A'B' + A'B$	$Q = A' + A'B + A'B'$
$= A'(B' + B)$ distributive	$= A' + A'B'$ absorption
$= A'(1)$ inverse	$= A'$ absorption
$= A'$ identity	

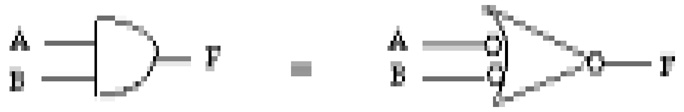
Universal Gates

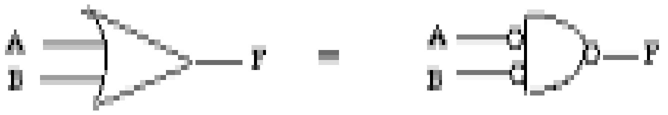
Any Boolean function can be constructed with only NOT, AND, and OR gates

But also with either only NAND or only NOR gates = **universal gates**

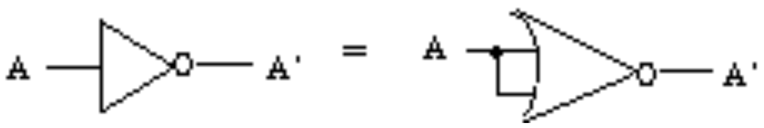
DeMorgan's Law shows how to make **AND** from NOR (and vice-versa)

AB = (A' + B')' (**AND** from NOR)
A + B = (A'B')' (**OR** from NAND)

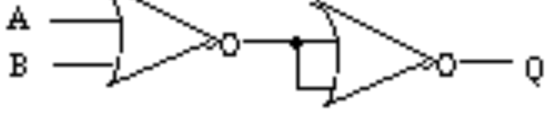




NOT from a NOR



OR from a NOR

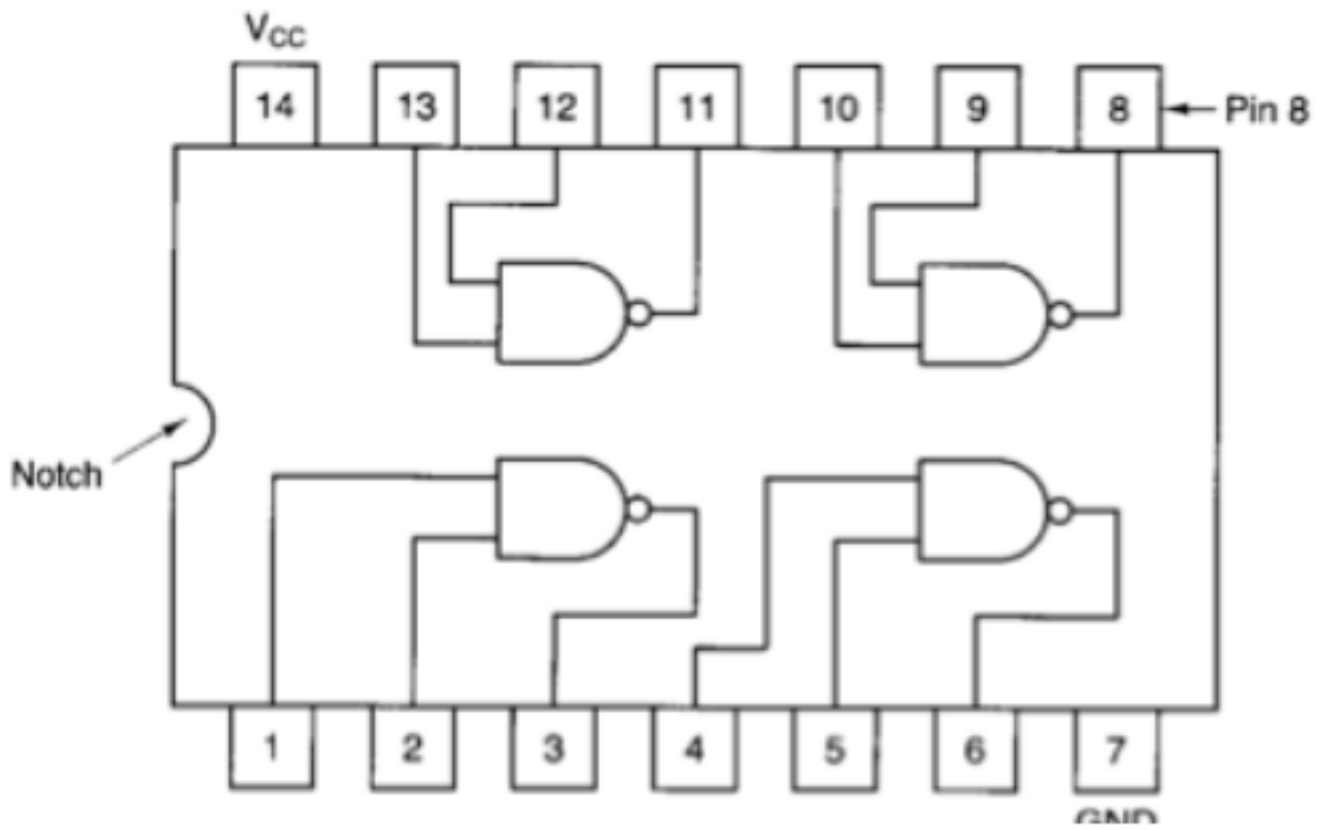


To implement a function using only NOR gates:

- apply DeMorgan's Law to each AND in the expression until all ANDs are converted to NORs
- use a NOR gate for any NOT gates, as well.
- remove any redundant gates (NOT NOT, may remove both)

Implementing the circuit using only NAND gates is similar.

Integrated Circuits (Chips)



Protoboard demo

LogicWorks demo