

Basic Electronics and Digital Logic Computer Science 240

Laboratory 1

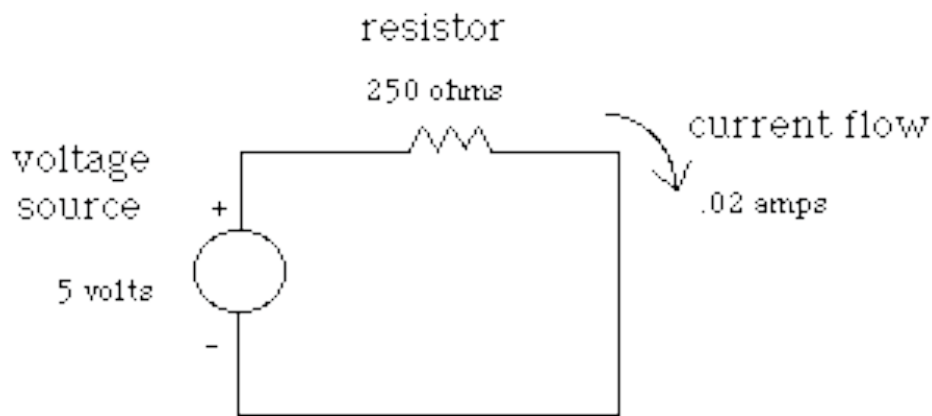
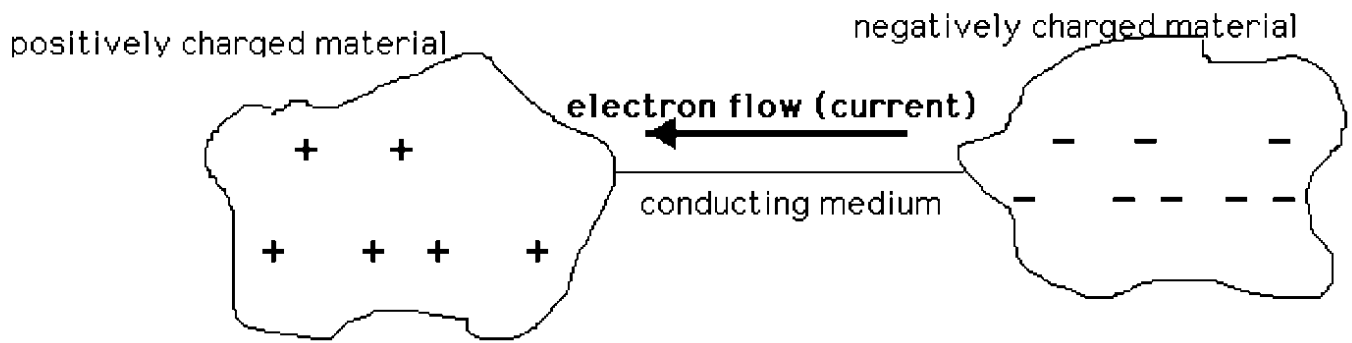
- **Administrivia**
- **Lab Environment**
- **Basic Electronics (Ohm's law, transistors, logic gates)**
- **Sum-of-Products and Equivalence**
- **Integrated Circuits**
- **Protoboard (for building physical circuits)**
- **LogicWorks (for simulating circuits)**

Lab Environment

- All lab exercises and reports will be *Google Docs*, and should be shared with lab partner and the instructor.
- You will switch partners each lab.
- To log in to a machine booted to Windows, use your Wellesley network username and password.
- To log in to a machine booted to Linux, use your CS username and password (should be the same as your account for CS 111 and/or CS 230).

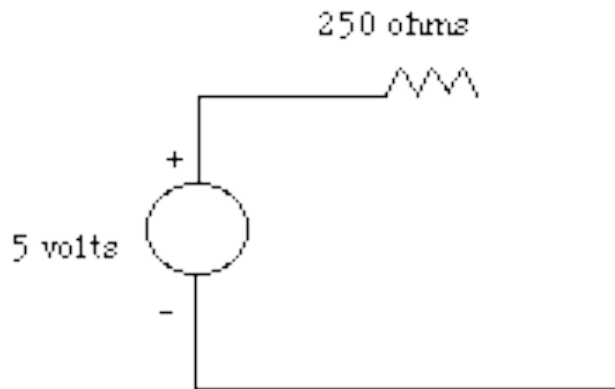
Basic Concepts of Electricity

- 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/voltage** (**V**, measured in **Volts**)
- When you connect two materials with a potential difference using a conducting medium (such as a wire), the electrons will flow to try to balance the charge
- Rate at which flow of electrons is called **current I** (measured in **Amps**).
- The conducting material has an integral ease of conduction to the flow of electrons called **resistance R** (measured in Ohms **Ω**)

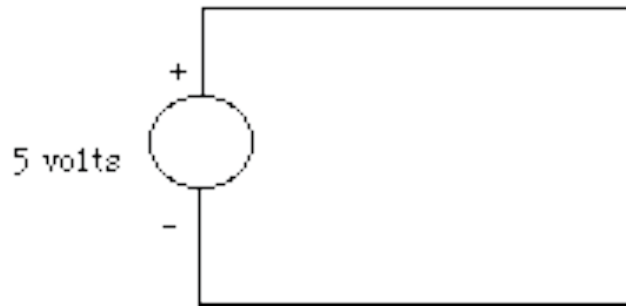


Ohm's Law, $V = IR$.

Open circuit = no current:



Short circuit = infinite current, since $V/0 = \text{infinite current}$:



Infinite current swiftly results in the destruction of the circuit!

The basis of electronic computers is that we can specify a voltage measured in a circuit as either *high* (close to the voltage source) or *low* (close to 0 volts, or ground).

Therefore, using electronic circuits, we can represent Boolean values (high = true, low = false), and we can also represent numbers using the binary number system, with high = 1 and low = 0.

Basic Gates and Truth Tables

A	B	F	
0	0	0	A truth table specifies the output for all the given input combinations of a Boolean function. We represent a value of <i>true</i> with a 1 and <i>false</i> with a 0.
0	1	0	
1	0	0	
1	1	1	

If a function has two inputs **A** and **B** (called *literals*), and the function is true when both input are true, for example:

when **A = 1 AND B=1**

that can be represented by the *minterm* **AB**, which implies **A AND B**, since an **AND** operation is implied by two inputs placed next to one another.

AB is only true when **A = 1 AND B = 1**

For a function that is true only when one of the inputs is true and the other is false, for example:

when **A = 1 and B = 0**






we use the *inverse* of **B** in the minterm, **AB'** (**B'** means **NOT B**).

B' is true only when **B = 0**

AB' means **A AND NOT B**, and is only true when **A = 1 and B = 0**.

An **OR** operation is expressed by the + operator (such as **A + B**, meaning **A OR B**).

There are several basic logic functions which are fundamental to our study of digital electronics (including symbols used to represent the function):

NOT	NAND	NOR	AND	OR																																																																		
$F = A'$	$F = (AB)'$	$F = (A+B)'$	$F = AB$	$F = A + B$																																																																		
<table border="1"> <thead> <tr> <th><u>A</u></th> <th><u>F</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	<u>A</u>	<u>F</u>	0	1	1	0	<table border="1"> <thead> <tr> <th><u>A</u></th> <th><u>B</u></th> <th><u>F</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	<u>A</u>	<u>B</u>	<u>F</u>	0	0	1	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th><u>A</u></th> <th><u>B</u></th> <th><u>F</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	<u>A</u>	<u>B</u>	<u>F</u>	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <thead> <tr> <th><u>A</u></th> <th><u>B</u></th> <th><u>F</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	<u>A</u>	<u>B</u>	<u>F</u>	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <thead> <tr> <th><u>A</u></th> <th><u>B</u></th> <th><u>F</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	<u>A</u>	<u>B</u>	<u>F</u>	0	0	0	0	1	1	1	0	1	1	1	1
<u>A</u>	<u>F</u>																																																																					
0	1																																																																					
1	0																																																																					
<u>A</u>	<u>B</u>	<u>F</u>																																																																				
0	0	1																																																																				
0	1	1																																																																				
1	0	1																																																																				
1	1	0																																																																				
<u>A</u>	<u>B</u>	<u>F</u>																																																																				
0	0	1																																																																				
0	1	0																																																																				
1	0	0																																																																				
1	1	0																																																																				
<u>A</u>	<u>B</u>	<u>F</u>																																																																				
0	0	0																																																																				
0	1	0																																																																				
1	0	0																																																																				
1	1	1																																																																				
<u>A</u>	<u>B</u>	<u>F</u>																																																																				
0	0	0																																																																				
0	1	1																																																																				
1	0	1																																																																				
1	1	1																																																																				
																																																																						

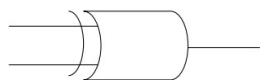
Although NOT, AND, and OR are the only functions needed for expressing sum-of-products, it turns out that NAND (NOT AND, the opposite of AND) and NOR (NOT OR, the opposite of OR) are also very useful.

In addition, the Exclusive-OR function (XOR) is also considered a basic logic function, because it can be used for comparison of bits, which is quite useful for many tasks, including addition!

XOR

$$F = AB' + A'B$$

<u>A</u>	<u>B</u>	<u>F</u>
0	0	0
0	1	1
1	0	1
1	1	0



Sum-of-Products

Boolean functions be expressed in a **sum-of-products** form, which uses AND, OR, and NOT basic functions. For example, given the following truth:

<u>A</u>	<u>B</u>	<u>F</u>
0	0	0
0	1	1
1	0	1
1	1	1

By sum-of-products, F is true if:

$$A = 0 \text{ AND } B=1 \quad (A'B)$$

-OR-

$$A = 1 \text{ AND } B= 0 \quad (AB')$$

-OR-

$$A = 1 \text{ AND } B=1 \quad (AB)$$

$$\text{so, } F = A'B + AB' + AB$$

Any Boolean function can be expressed using only NOT, AND, and OR basic functions using sum-of-products.

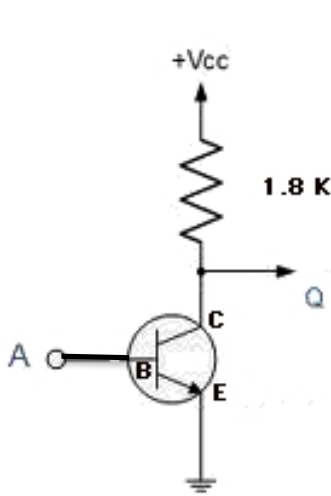
Therefore, if we had an electronic circuit which could produce these basic functions (assuming a high voltage measurement can represent true and a low voltage measurement can represent false), that means that we can build a circuit for any Boolean function.

Transistors

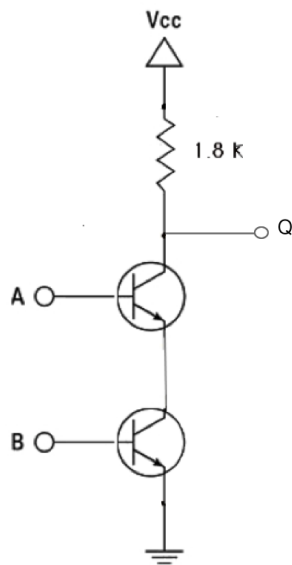
A key to the development of modern computers was the invention of devices that can act like a switch (can be turned on or off). Although the early devices were large (such as vacuum tubes), and used a variety of technologies, eventually **transistors**, miniature electric switches, were developed.

In addition to acting as a switch, transistors can be used to produce the basic logic functions:

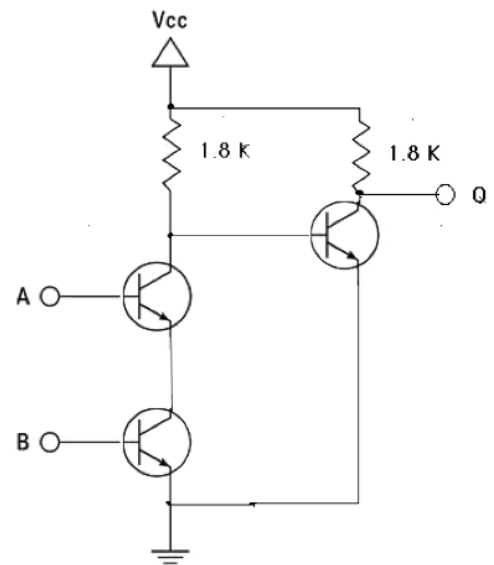
NOT – 1 transistor



NAND – 2 transistors



AND – 3 transistors

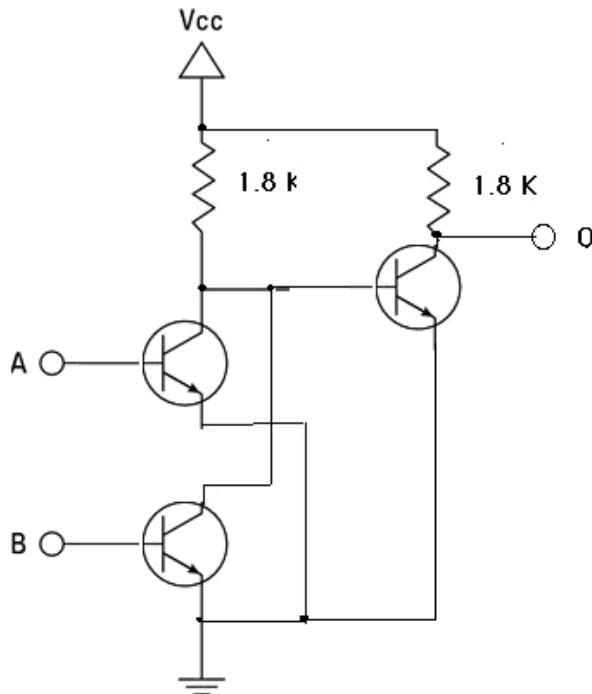
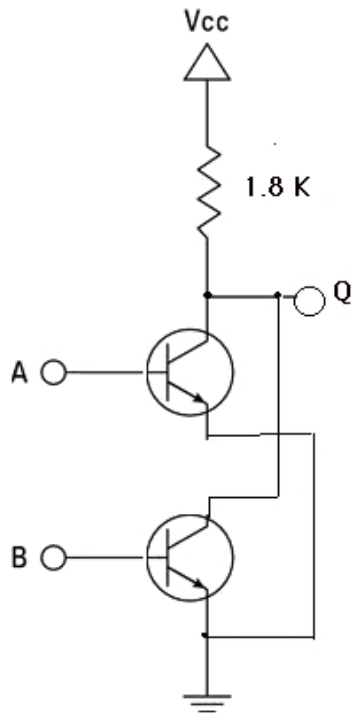


The **AND** gate uses 3 transistors and is basically a **NOT NAND** (it sends the output of a NAND through another transistor acting as a NOT gate to complement the result):

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

NOR – 2 transistors

OR – 3 transistors



Equivalence

Two functions which produce the same truth table are considered *equivalent*.

For example, the functions F and Q can be shown to be equivalent:

$$F = A'B' + A'B$$

$$Q = A' + A'B + A'B'$$

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

When there is an equivalent function/circuit that uses fewer gates, transistors, or chips, it is preferable (cheaper and faster) to use that circuit in a design.

Equivalence can also be proven through use of Boolean algebra, which you will soon be covering in lecture.

Integrated Circuits

Integrated circuits (chips) are devices that contain transistors which perform specific functions.

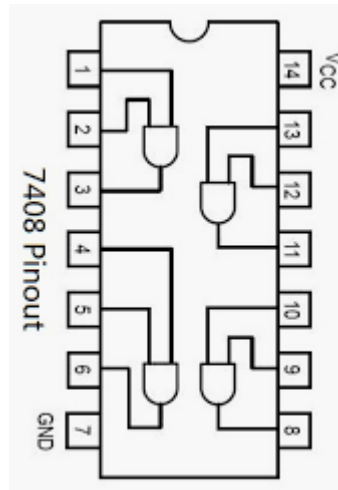


The **pinout** (found in a datasheet from a TTL Data Book or online) shows the physical layout of the pins and the purpose of the device:

Pins are numbered, starting with “1” at the top left corner and incremented counter-clockwise around the device.

Top left pin is pin 1, always to left of notch in chip, is often marked with a dot.

Bottom left pin is often connected to the negative terminal of the power supply (called **Ground**, and assumed to be 0 Volts).

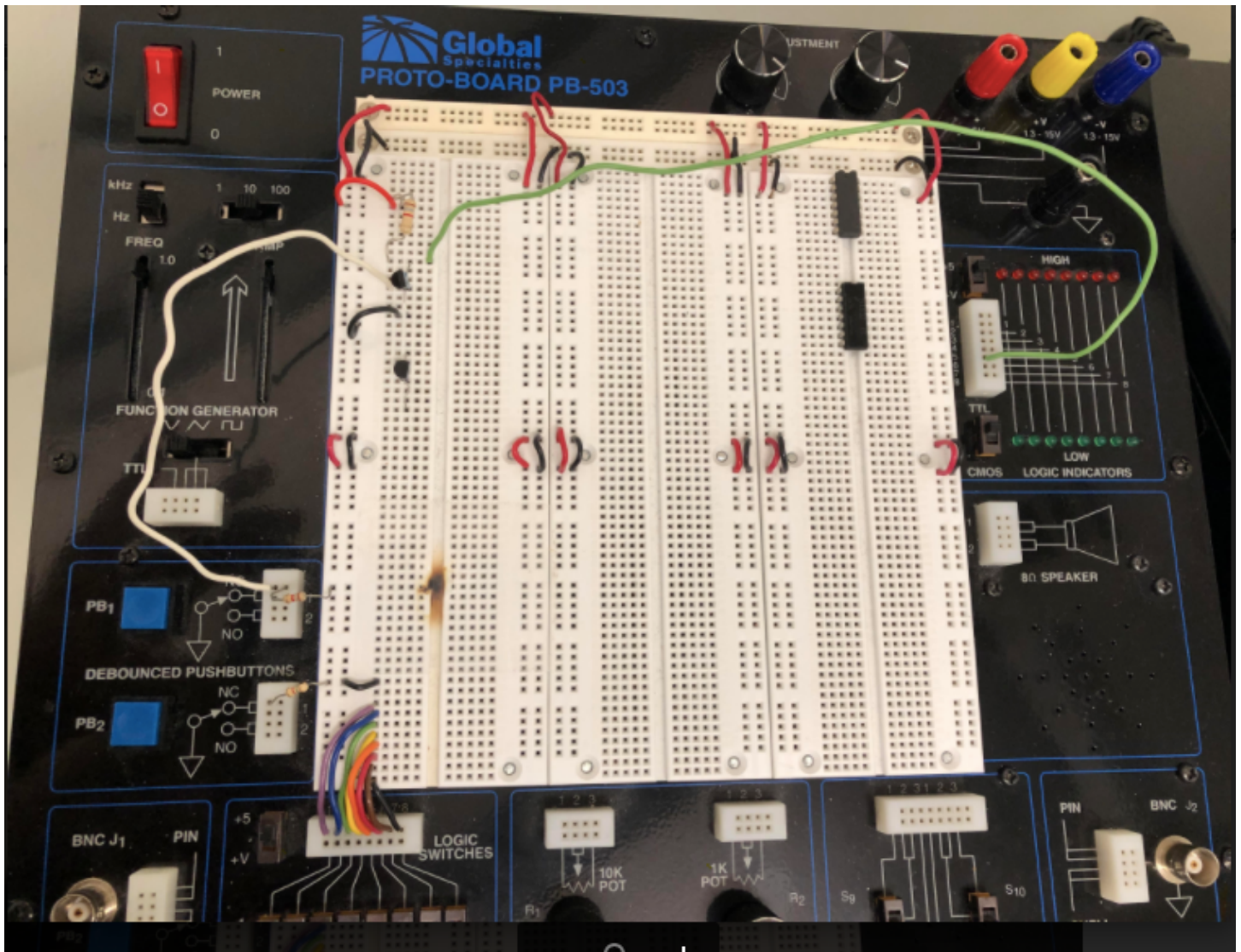


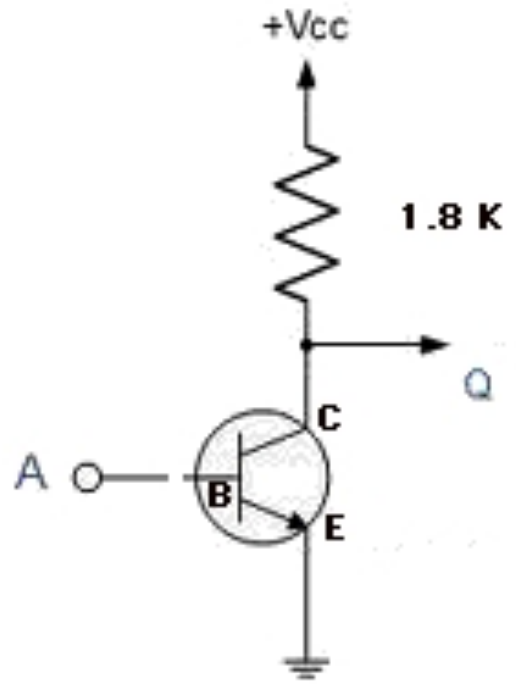
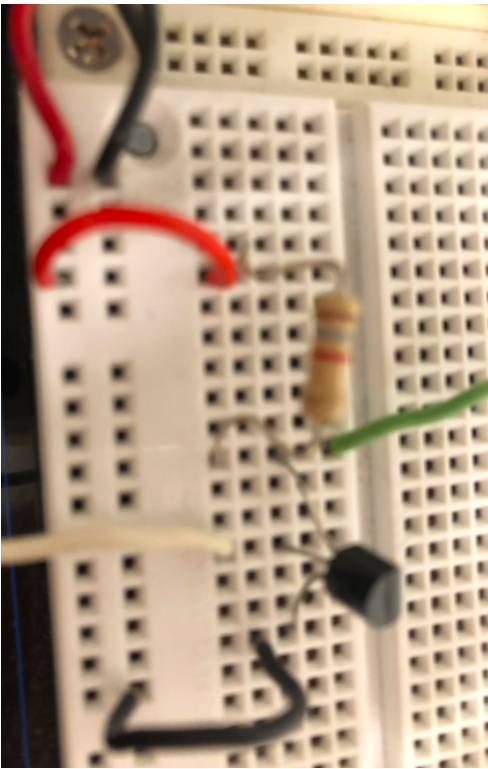
Top right pin is often connected to the positive terminal of the power supply (called **Vcc**, and assumed to be +5 Volts for our experiments).

The chip will not work if it is not connected to power and ground!

Protoboard for building circuits

A protoboard is a tool to create prototype circuits. It contains a built-in power supply, switches to supply inputs to circuits, Logic Indicators to display outputs of circuits, and an array of holes/tie points in which components and wires can easily be inserted to connect circuits:





Transistor circuit for a NOT gate

Circuit Simulation/LogicWorks

The screenshot displays the LogicWorks 5 interface for a circuit simulation. The main workspace shows a schematic of a 4-bit adder implemented using a 74181 ALU chip. The chip's inputs are connected to switches, and its outputs are connected to LEDs. The simulation window at the bottom shows a timing diagram with a 1 ns scale. The right-hand panel displays a list of components available in the library, including various power supplies and logic components.

LogicWorks 5 - [C:\Documents and Settings\jherbst\Desktop\adder.cct]

File Edit View Schematic Simulation Window Help

1 ns

2

3

4

adder.cct

ns

200 400 600 800 1000

Timing

ALL LIBRARIES

Filter:

Preview

- +12V
- +15V
- +5V
- 12V
- 15V
- 5V
- 74_00
- 74_02
- 74_03
- 74_04
- 74_08
- 74_10
- 74_100
- 74_101
- 74_102
- 74_103
- 74_103.a
- 74_103.b
- 74_104
- 74_105
- 74_106
- 74_106.a
- 74_106.b
- 74_107
- 74_107.a
- 74_107.b
- 74_107A
- 74_107A.a
- 74_107A.b
- 74_108
- 74_109
- 74_109.a
- 74_109.b

Ready

start Desktop : FirstClass Mailbox : FirstClass LogicWorks 5 - [C:\D...

NUM 6:08 PM