

# **Basic Electronics and Digital Logic Computer Science 240**

## **Laboratory 1**

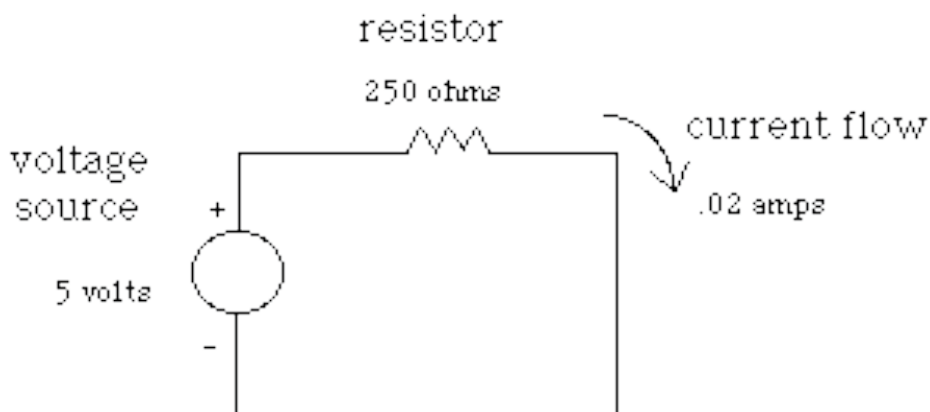
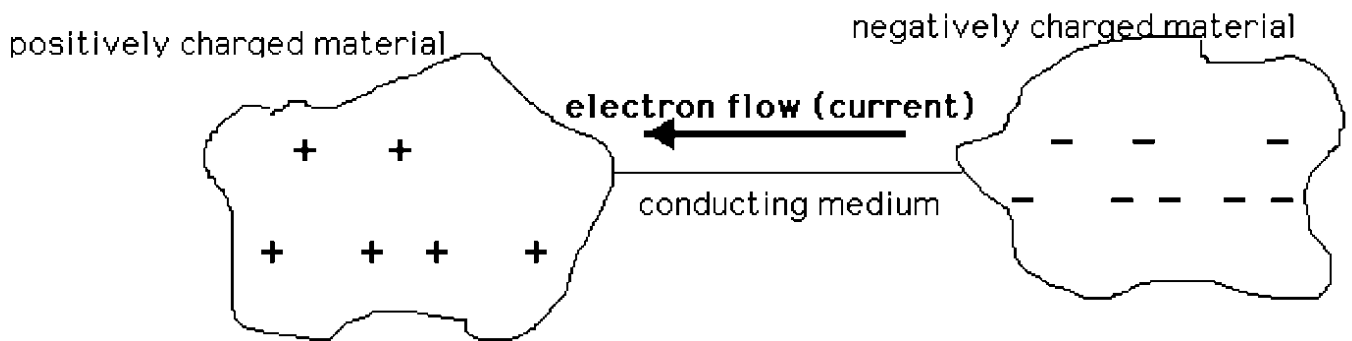
- **Administrivia**
- **Lab Environment**
- **Basic Electronics (Ohm's law, transistors, logic gates)**
- **Sum-of-Products and Equivalence**
- **Integrated Circuits**
- **Protoboard (for building physical circuits)**
- **LogicWorks (for simulating circuits)**

## Lab Environment

- All lab exercises and reports will be *Google Docs*, and should be shared with lab partner and the instructor.
- You should only use the machine in the seats specified for use (distanced 6 feet apart) during lab, which are bootable to either Windows or to Linux.
- To log in to a machine booted to Windows, use your Wellesley network username and password.
- At the beginning and end of lab, clean your hands with the antiseptic wipes. Wear gloves during lab.
- For the first four labs, you will be using a protoboard to build some physical circuits.
- When you are working with a partner, only one of the partners will build the circuit and touch the devices and wires (the other partner will sit across the table and be able to see the board and results of the experiments).
- Each person working on the physical circuits will have their own set of wires that will not be shared between lab sections.
- Keyboards and protoboards will be cleaned between lab sections.
- We will switch partners each lab and make sure that everyone has a chance to build some circuits.

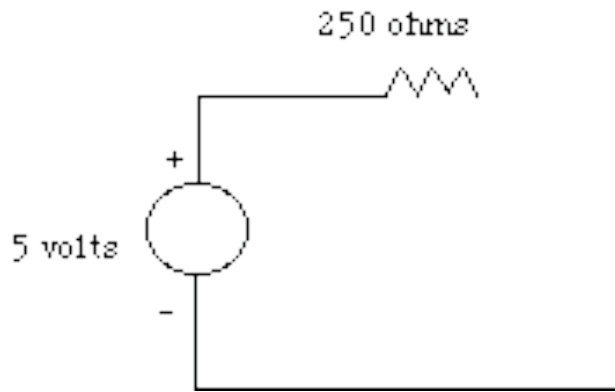
## Basic Concepts of Electricity

- 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  $\Omega$ )

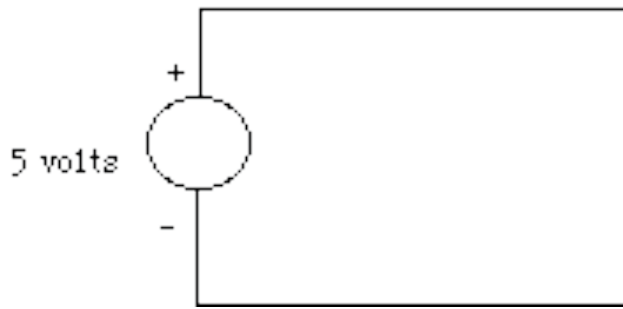


**Ohm's Law,  $V = IR$ .**

**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

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

A **truth table** specifies the output for all the given input combinations of a Boolean function. We represent a value of *true* with a 1 and *false* with a 0.

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 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.

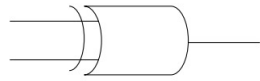


In addition, the Exclusive-OR function (XOR) is also considered a basic logic function, because it can be used for comparison of bits, which is quite useful for many tasks, including addition!

### 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-

$$A = 1 \text{ AND } B=1 \quad (AB) \quad \text{so, } F = A'B + AB' + AB$$

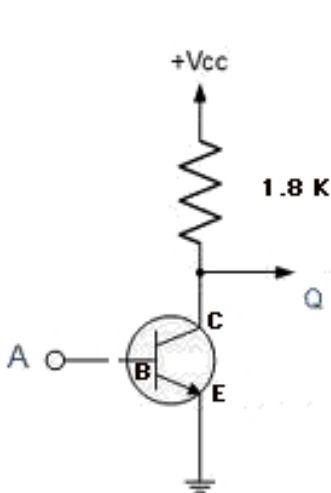
Since a Boolean function can be expressed using only NOT, AND, and OR basic functions using sum-of-products, if we had an electronic circuit which could produce these basic functions (assuming a high voltage measurement can represent true and a low voltage measurement can represent false), that means that we can build a circuit for any Boolean function.

# Transistors

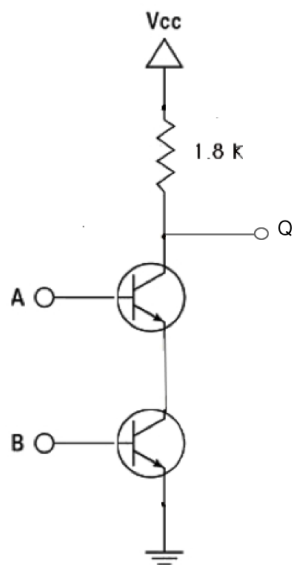
A key to the development of modern computers was the invention of devices that can act like a switch (can be turned on or off). Although the early devices were large (such as vacuum tubes), and used a variety of technologies, eventually **transistors**, miniature electric switches, were developed.

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

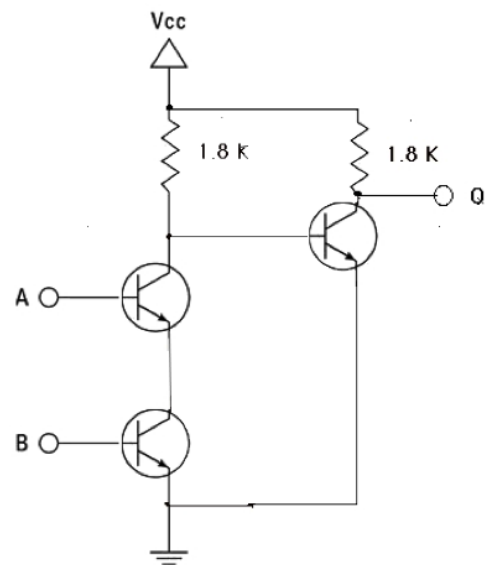
**NOT** – 1 transistor



**NAND** – 2 transistors



**AND** – 3 transistors



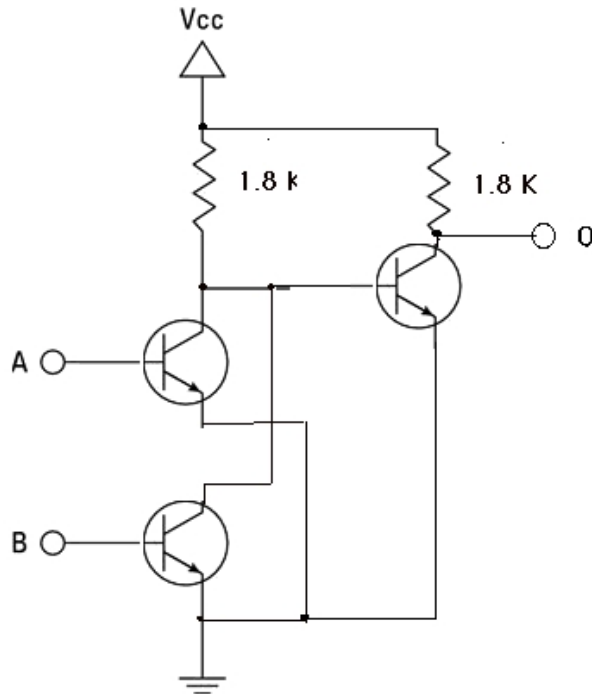
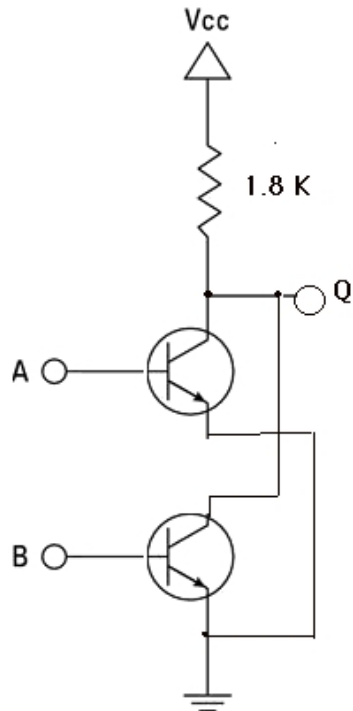
The **AND** gate uses 3 transistors and is basically a **NOT NAND** (it sends the output of a NAND through another transistor acting as a NOT gate to complement the result):



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 has a set of laws or identities:

### Boolean Laws Reference Sheet

Name of Law / Theorem	Form	Equivalent/Dual form (interchange AND and OR, and 0 and 1)
<b>Identity</b>	$0 + A = A$	$1 * A = A$
<b>Inverse (or Complements)</b>	$A\bar{A} = 0$	$A + \bar{A} = 1$
<b>Commutativity</b>	$A + B = B + A$	$AB = BA$
<b>Associativity</b>	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
<b>Idempotent</b>	$A + A = A$	$AA = A$
<b>Null (or Null Element)</b>	$0A = 0$ <b>(the Zero Law)</b>	$1 + A = 1$ <b>(the One Law)</b>
<b>DeMorgan's</b>	$\overline{A + B + C + \dots} = \bar{A}\bar{B}\bar{C}\dots$	$\overline{A * B * C * \dots} = \bar{A} + \bar{B} + \bar{C} + \dots$
<b>Absorption (or Covering)</b>	$A + AB = A$	$A(A + B) = A$
<b>Involution (or double negation)</b>	$\overline{\bar{A}} = A$	none
<b>Distributive</b>	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
<b>Combining</b>	$AB + \bar{A}B = A$	$(A + B)(A + \bar{B}) = A$
<b>Consensus</b>	$AB + \bar{A}C + BC = AB + \bar{A}C$	$(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$

To prove equivalence using the Boolean laws, transform one of the functions using the Boolean laws until both functions have the same expression.

For example, given the previous equivalent functions:

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

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

$$A' + A'B = A' \quad \text{(absorption)}$$

$$A'B + A'B' = A'B' + A'B \quad \text{(commutativity)}$$

## Universal Gates

As we have seen using sum-of-products form, any Boolean function can be constructed using only NOT, AND, and OR gates

But any function can also be produced using only NAND gates or only NOR gates. For that reason, NAND and NOR are called **universal gates**.

One of the identities of Boolean algebra, **DeMorgan's Law**, states how to make an **AND** using a **NOR** (and vice-versa):

$$AB = (A' + B)'$$

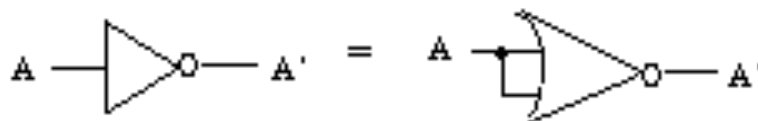
(**AND** from **NOR**)

$$A + B = (A'B)'$$

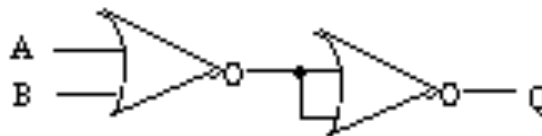
(**OR** from **NAND**)



**NOT** from a **NOR** (just tie inputs together)



**OR** from a **NOR** (just NOT the NOR)



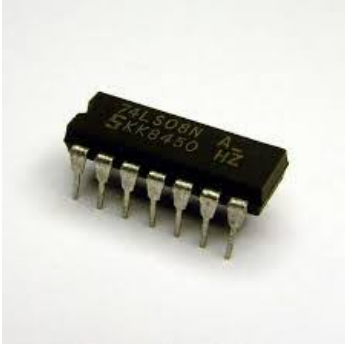
**To implement a function using only NOR gates:**

- 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

Integrated circuits (chips) contain transistors which perform a specific function.

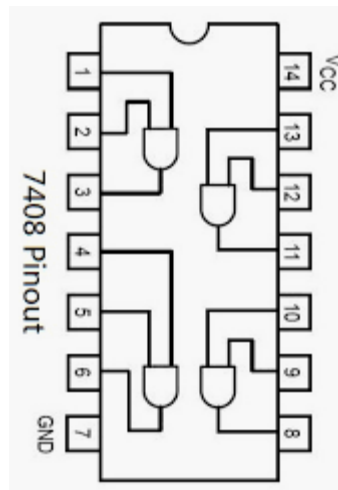


The **pinout** (found in a datasheet from a TTL Data Book or online) shows the physical layout of the pins and the purpose of the device:

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).

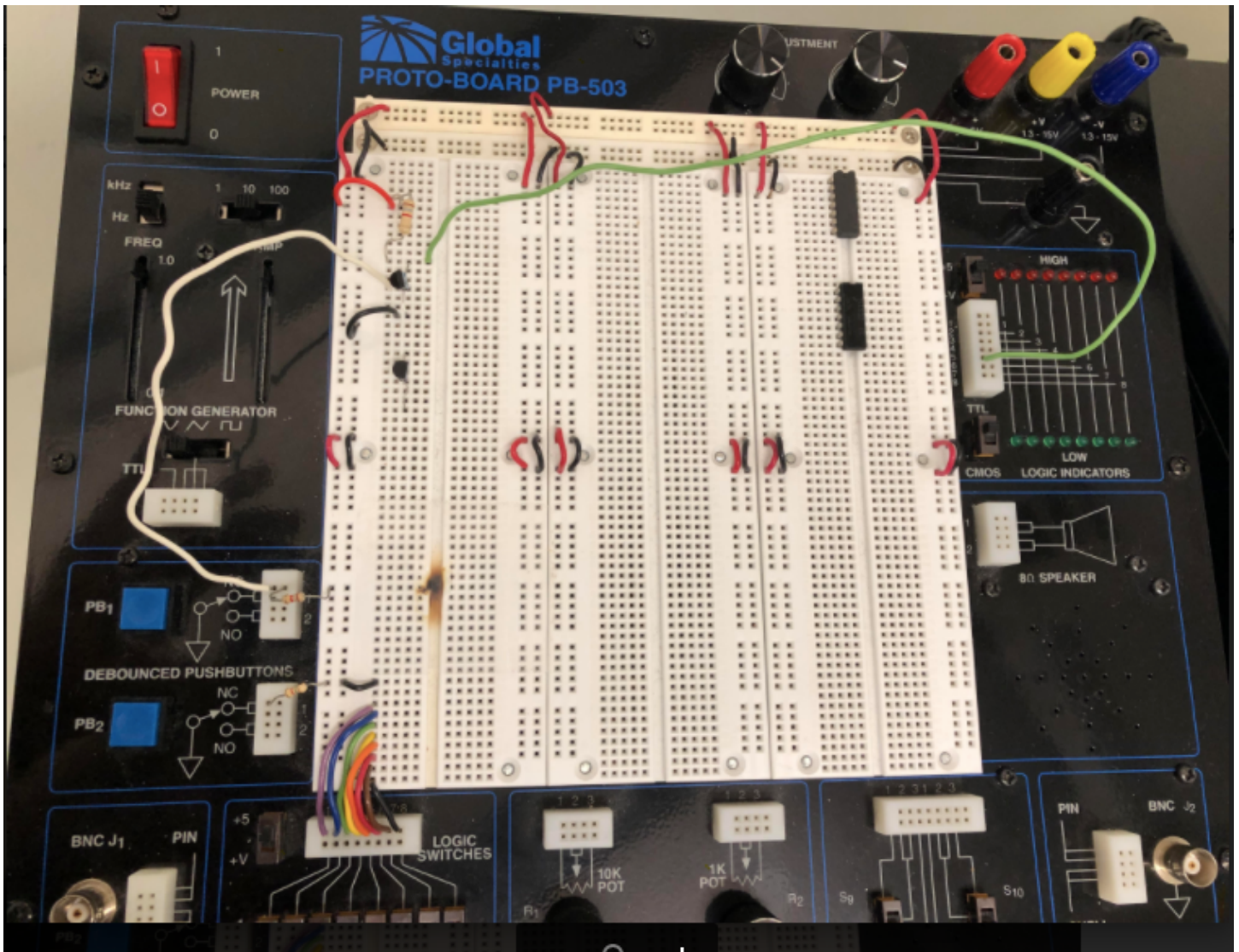


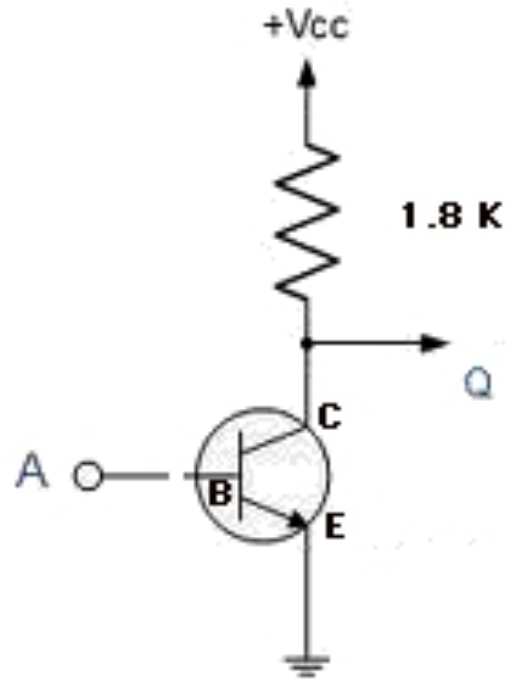
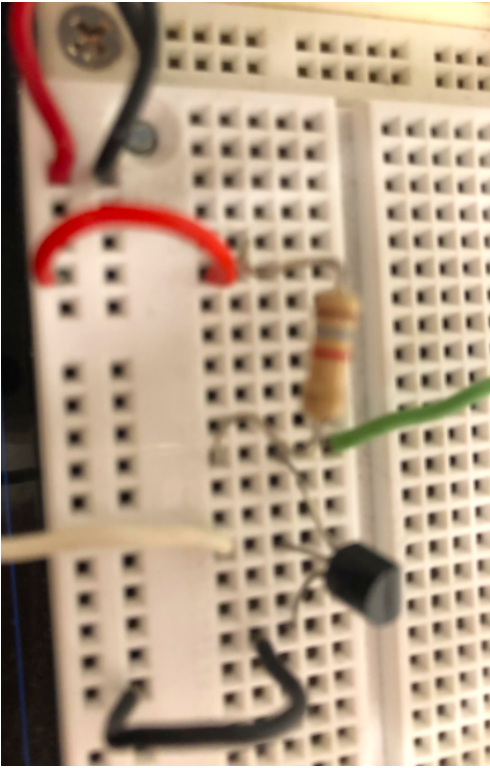
Top right pin is often connected to the positive terminal of the power supply (called **Vcc**, and assumed to be +5 Volts for our experiments).

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 component (labeled '83'). The component has four input bits (A0, A1, A2, A3) and four input bits (B0, B1, B2, B3). The outputs are four sum bits (S0, S1, S2, S3) and a carry-out bit (CO). The inputs are connected to switches, and the outputs are connected to LEDs. The simulation is running at 1 ns.

The right-hand side of the interface shows a library of components, including power supplies (+12V, +5V, -12V, -5V) and various logic components (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).

The bottom of the interface shows a timing diagram window with a scale of 1 ns. The Windows taskbar at the bottom indicates the system is ready and the time is 6:08 PM.