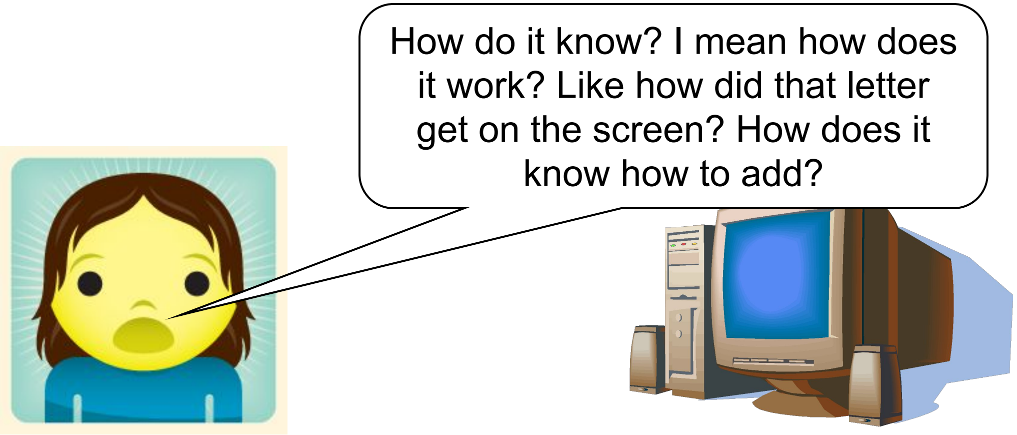
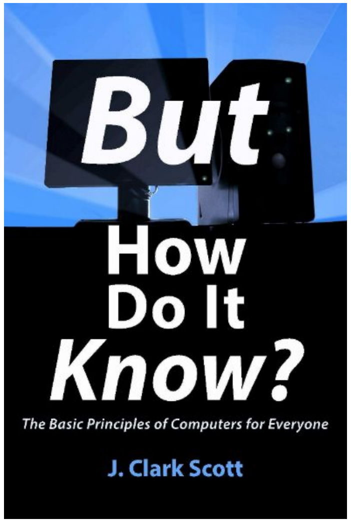
Lesson One: How Do It Know?

by: tjjenk2

How Do Computers Work?


How do it know? I mean how does it work? Like how did that letter get on the screen? How does it know how to add?



But How Do It Know?

The Basic Principles of Computers for Everyone

J. Clark Scott

This guy knows his stuff and can explain it so I can understand it! ☺

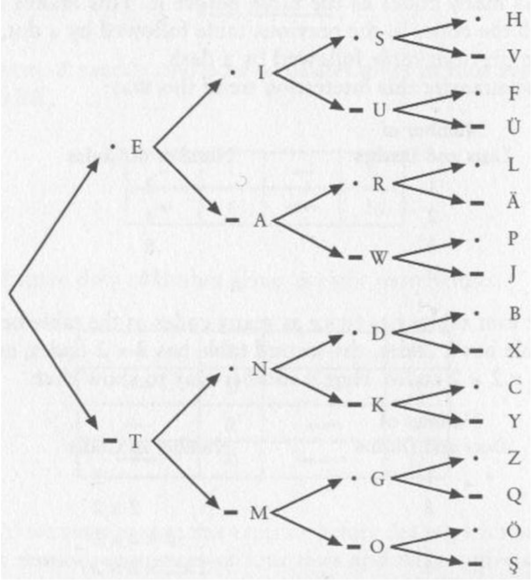
Early Forms of Communication

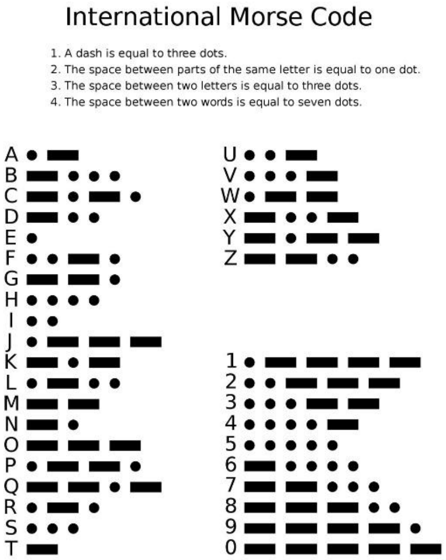
- Smoke signals – Very slow, distance and weather can be an issue.

- Light – Faster than smoke, but still slow because it requires manual interaction. Must go in straight line so cannot communicate as earth curves.

Telegraph – Faster yet, but only as fast as the person could click the switch. Can cover long distances using relay switch and can go around things.

- One type of communication - Morse Code – Two States – a dot (DIT) or a dash (DAH)





International Morse Code

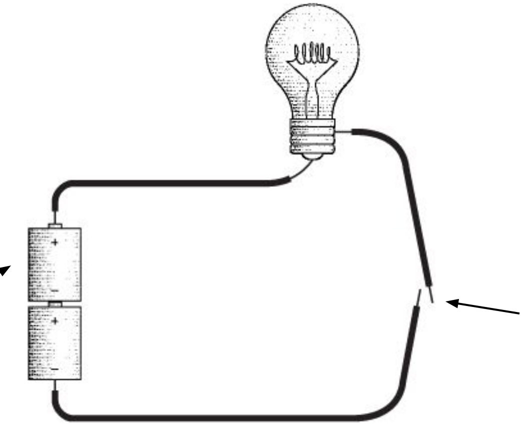
1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

A	● ▬
B	▬ ● ● ●
C	▬ ● ▬ ●
D	▬ ● ●
E	●
F	● ● ▬ ●
G	▬ ▬ ●
H	● ● ● ●
I	● ●
J	● ▬ ▬ ▬
K	▬ ● ▬
L	● ▬ ● ●
M	▬ ▬
N	▬ ●
O	▬ ▬ ▬
P	● ▬ ▬ ●
Q	▬ ▬ ● ▬
R	● ▬ ●
S	● ● ●
T	▬
U	● ● ▬
V	● ● ● ▬
W	● ▬ ▬
X	▬ ● ● ▬
Y	▬ ● ▬ ▬
Z	▬ ▬ ● ●
1	● ▬ ▬ ▬ ▬
2	● ● ▬ ▬ ▬
3	● ● ● ▬ ▬
4	● ● ● ● ▬
5	● ● ● ● ●
6	▬ ● ● ● ●
7	▬ ▬ ● ● ●
8	▬ ▬ ▬ ● ●
9	▬ ▬ ▬ ▬ ●
0	▬ ▬ ▬ ▬ ▬

How else can we use electricity to communicate?

Electricity

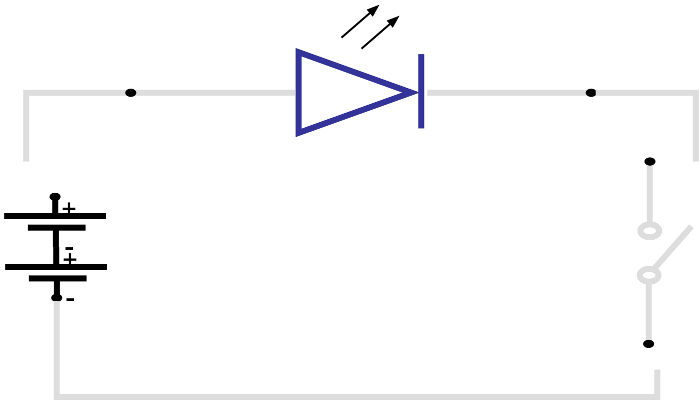
Anatomy of the Flashlight
(Electricity)



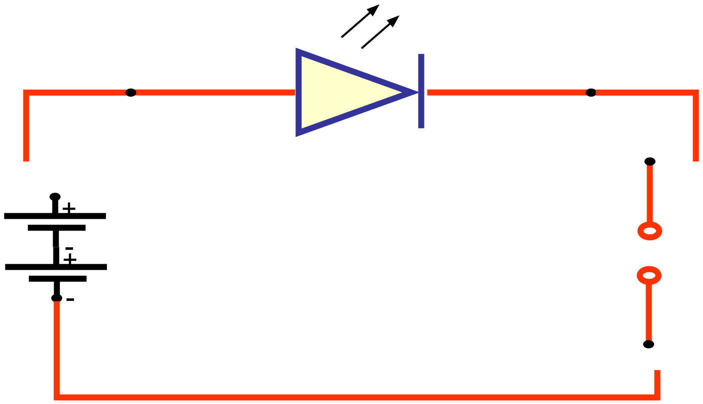
Battery
Chemical reaction inside
causes charge to gather
on opposite ends when
the switch is closed.

Switch
Open – Light bulb is off
Closed – Light bulb is on

Flashlight Schematic (OFF)

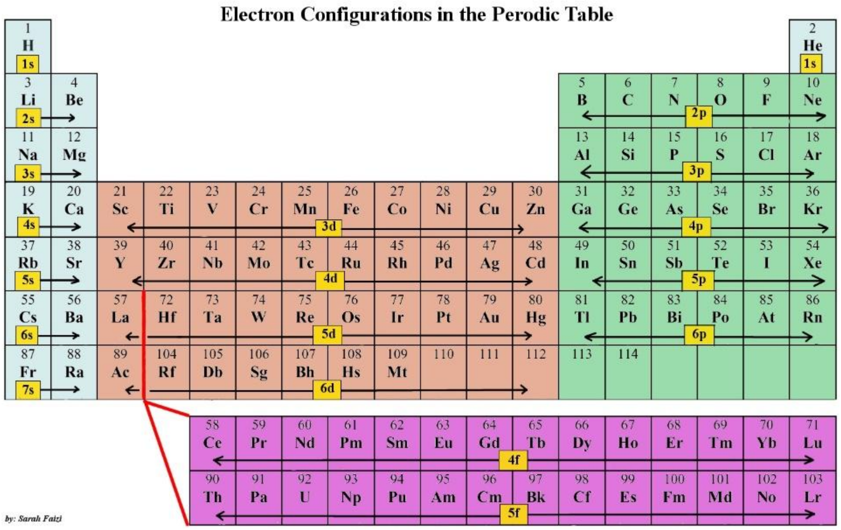


Flashlight Schematic (ON)



Chemistry

Element – A substance that is made entirely from one type of atom. It cannot be broken down by chemical means. There are a finite number.

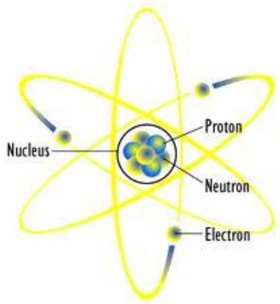


Electron Configurations in the Perodic Table

1 H 1s																	2 He 1s
3 Li 2s	4 Be											5 B	6 C	7 N 2p	8 O	9 F	10 Ne
11 Na 3s	12 Mg											13 Al	14 Si	15 P 3p	16 S	17 Cl	18 Ar
19 K 4s	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn 3d	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As 4p	34 Se	35 Br	36 Kr
37 Rb 5s	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc 4d	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb 5p	52 Te	53 I	54 Xe
55 Cs 6s	56 Ba	57 La	72 Hf	73 Ta	74 W	75 Re 5d	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi 6p	84 Po	85 At	86 Rn
87 Fr 7s	88 Ra	89 Ac	104 Rf	105 Db	106 Sg	107 Bh 6d	108 Hs	109 Mt	110	111	112	113	114				
				58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd 4f	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
				90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm 5f	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

by: Sarah Faizi

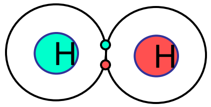
Atom – The basic building blocks of ordinary matter. The smallest quantity of an element that can take place in chemical reactions.



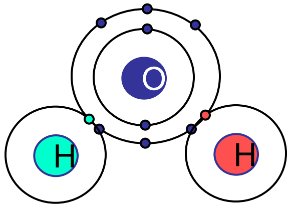
Molecule – Formed when two or more atoms join together chemically

Compound – (are molecules) It is a molecule that contains at least two different elements.

H2 Molecule



H2O Compound



Physics - More About The Atom

- The nucleus makes up about 99% of the mass and there is mostly empty space outside of it.

- Electrons orbit around the nucleus largely because of the force of attractions to the protons.

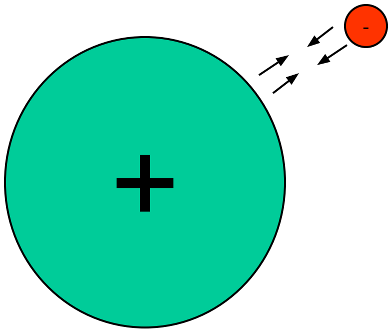
- The electrons are particles that have negative charge and behave like waves when they orbit.

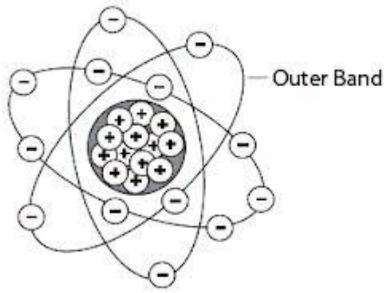
- Some electrons are free to move. The protons and neutrons are bound to the nucleus by a very strong force and are thus not free to move.

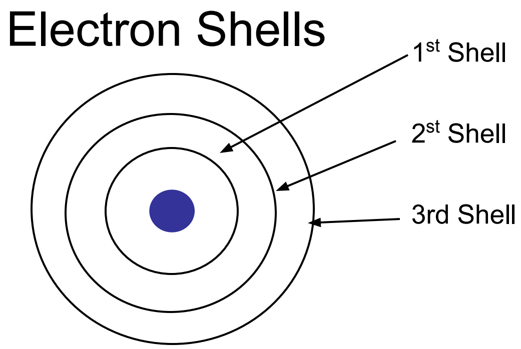
- One Proton and One Electron attract each other. Although the Proton is significantly larger than the Electron, the two together become neutral. Quantum physics keeps the electron from crashing into the proton; hence it orbits.

- Each atom has a certain number of electron orbit shells. The electrons on the inner shells experience a greater force of attraction than those on the outer shells. Each shell has a limit to the number of electrons that can orbit on it.

- Valence Electron – an electron that can participate in the formation of a chemical bound. They typically are in the outer most shell and are the most likely to become free.





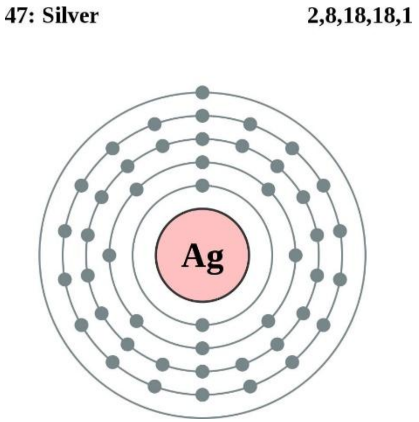


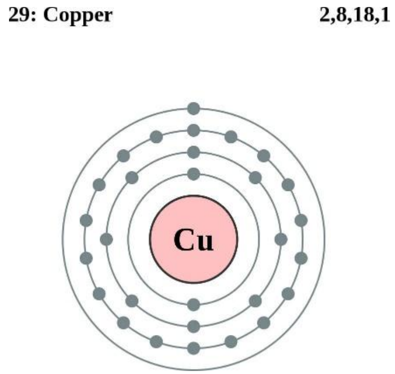
Conductor

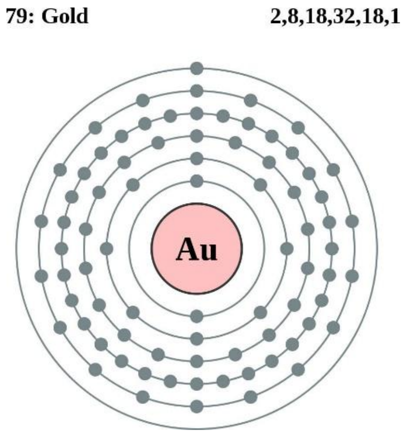
- Conductors permit the flow of electrons in one direction or another. All conductors have a very low resistance to electron flow.

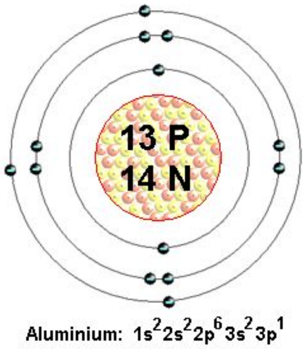
- The top four conductors are Silver, Copper, Gold, Aluminum

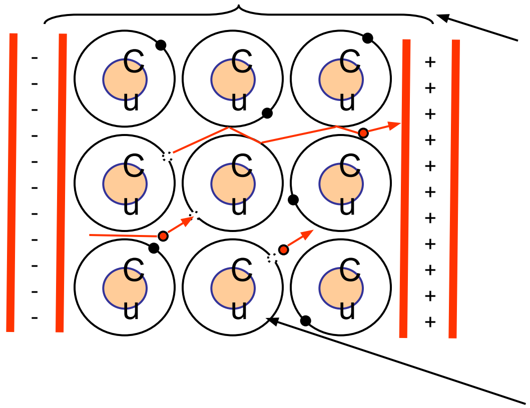
- Notice how there is only one electron in the outer most shell of the first three.











A charge difference exists between these two points and this causes the electrons to want to flow from negative to positive (left to right in this case).

When this atom loses its electron, it becomes positively charged. An atom with a charge is known as an Ion.

Another Look at the Flashlight

- When we close the switch, something known as a “complete circuit” is made. A complete circuit means the positive side of the battery is connected to the negative side of the battery and this causes electrons to flow from negative to positive and this is known as Current.

- Since the battery is part of a complete circuit, it supplies an electromotive force known as Voltage. This force would not be there if the switch were open, even with the battery still connected to the light on one end.

- The current will pass through the light and the light will turn on. Both the light and the wire provide some Resistance to the electron flow.

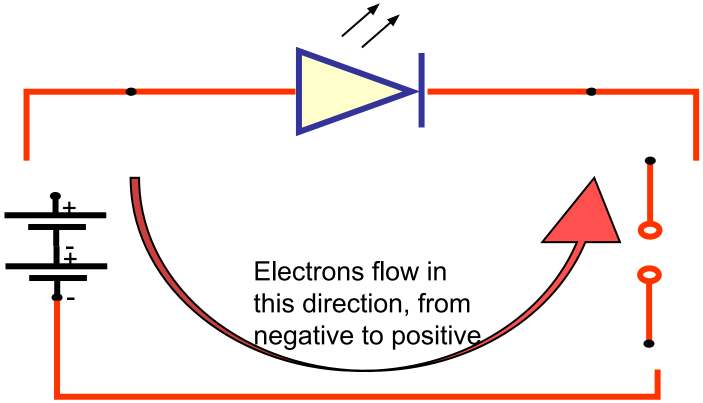
- As it turns out, Voltage, Current, and Resistance can affect each other. When you increase one, another could decrease. For example, increasing the resistance will cause less current to flow.

- Electrical Current – It is the flow of electric charge. When an electric potential is applied to a conductor, the electrons will flow randomly from negative to positive. These electrons flow at nearly the speed of light. Usually expressed in Amps (i).

- Resistance – Opposition to the passage of electric current (r).

- Voltage – Electromotive force or electric potential difference between two points. Usually expressed in volts (v). (v = i · r)

- Power – How much work can be performed in a given amount of time (p). Usually expressed in Watts. (p = i · v = i^2 · r).

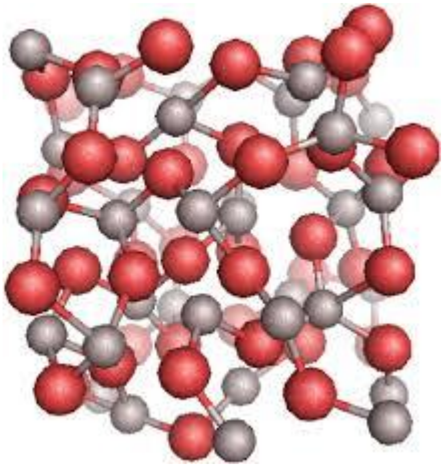


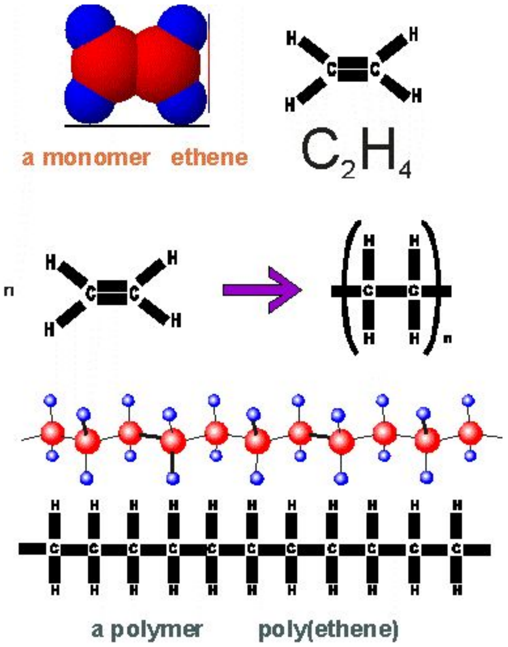
Insulator

- An electrical insulator is a material whose internal electric charges do not flow freely and therefore does not conduct an electric current under the influence of an electric field (wikipedia).

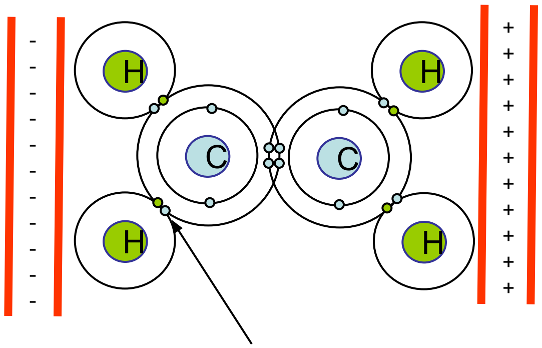
- Very good insulators: glass, paper, Teflon, PVC (polyvinyl chloride), rubber-like polymers. They have very high resistivity.

- Many compounds tend to make good insulators because their valence electrons are associated in the chemical bond and are therefore not free to move. Water, on the other hand, is a compound is good at conducting electricity.





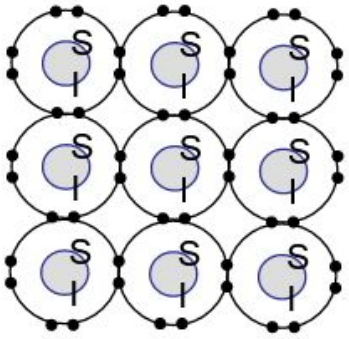
Applying an Electric Field
The atoms will vibrate, but very few, if any, electrons will flow through it.



Carbon Electron Says: I want to move, but I'm stuck in a bond with Hydrogen.

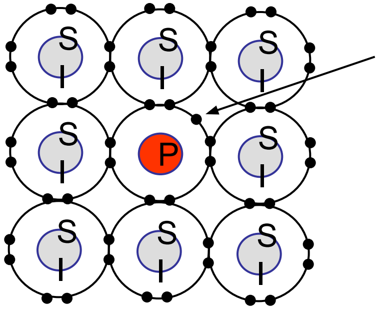
Semiconductor

Pure Silicon



- Pure silicon cannot be found in nature. However, it can be extracted from silicon dioxide which is found in the earth's crust. It is the 2nd most abundant after oxygen.
- Has some conductive properties, better than an insulator, but not as good as conductors (e.g. metals)
- 4 electrons in outer shell allows it to form tetrahedral with 4 of it's neighbors
- Few electrons ever get enough energy to escape bonds and travel through the lattice
- Can make two types using "doping" – inject a small amount of an element (1 part per million)

N-Type Silicon

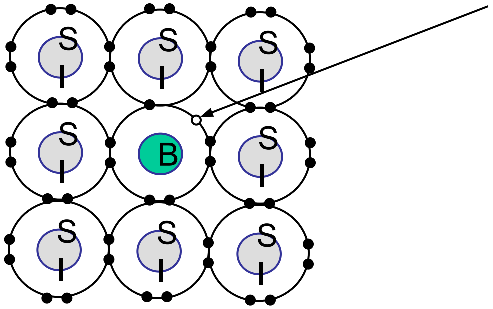


The Phosphorous atom contains an extra electron (i.e. 5 valence electrons) so it conducts electricity better

n

N-Type: Neutral, but electrons will move as negative charge.

P-Type Silicon

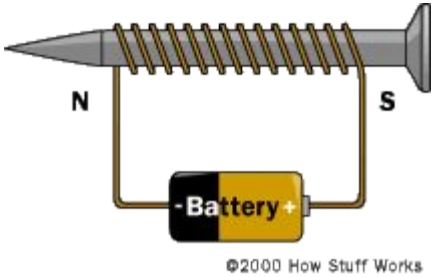


The Boron atom is missing an electron (i.e. 3 valence electrons) so creates a hole – the hole acts as a positive charge

p

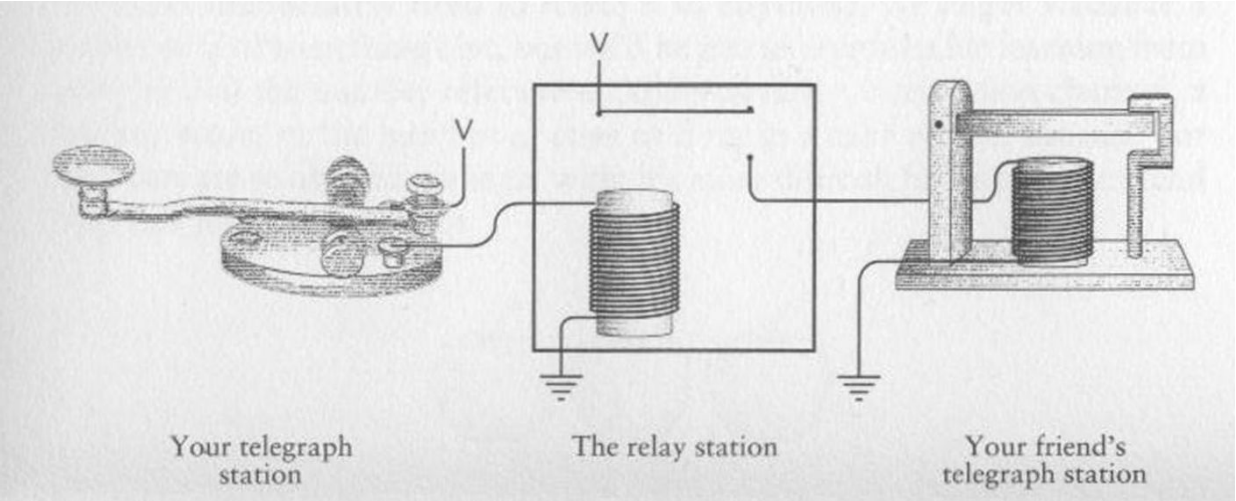
P-Type: Neutral, but holes will move as positive charge.

Another Look At the Telegraph



When wire is wrapped around a piece of metal and a current flows through it, it becomes an electro-magnet. When the switch is off, the metal does not act like a magnet.

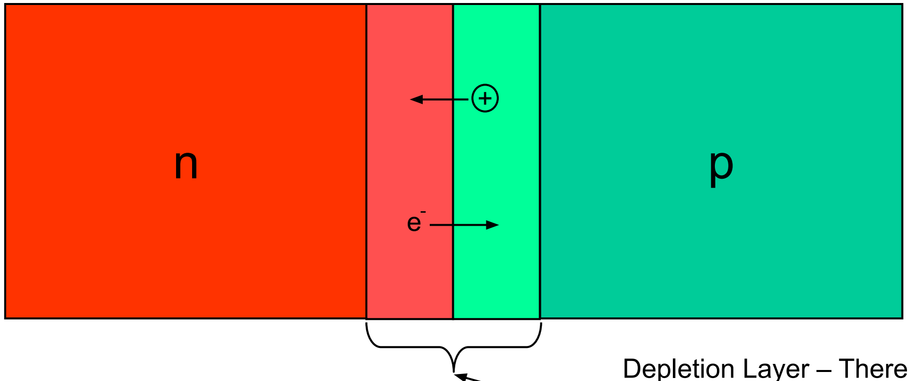
Because of resistance in wires, electricity cannot travel very long distances without losing its signal. The addition of a relay station can repeat the signal and thus increase the distance. When you tap the telegraph, you close the switch and the current flows through the wire causing the metal to be magnet. At the relay station, the switch will close connecting to your friends telegraph station which also has an electro-magnet that causes the metal above it to pull towards it. When you are done tapping, the metal at your friend's station goes back up making a clicking sound (dit). The dits can be combined to make letters (AKA Morse Code).



The Transistor

What happens when you bring N-Type & P-Type together?`

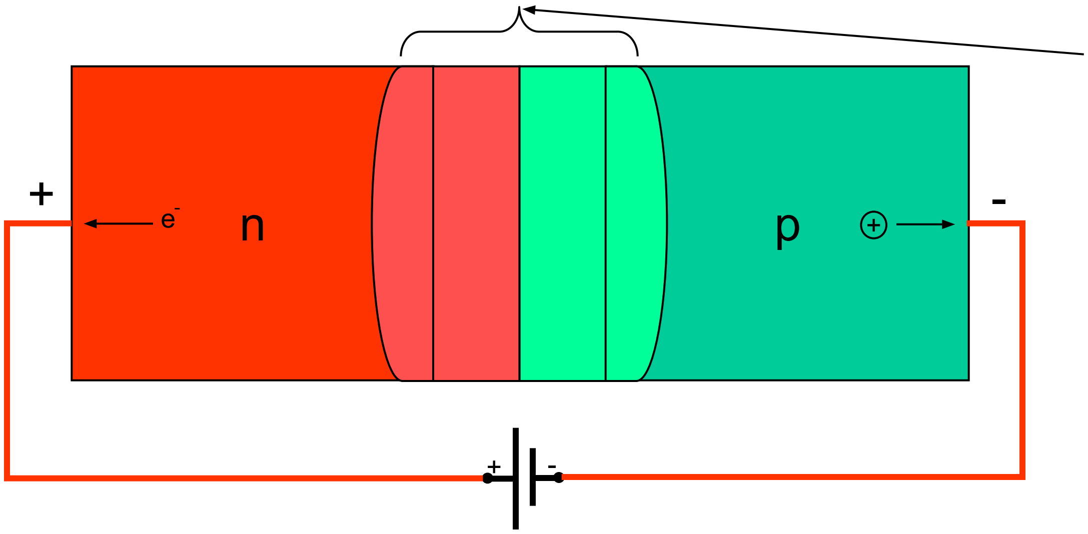
N-Type has a surplus of electrons that are attracted to P-Type.



P-Type has a surplus of positive holes that are attracted to N-Type.

Depletion Layer – There will be a little bit of region where the electrons in the N-Type will fill the holes in the P-Type reducing the force of attraction between the two types.

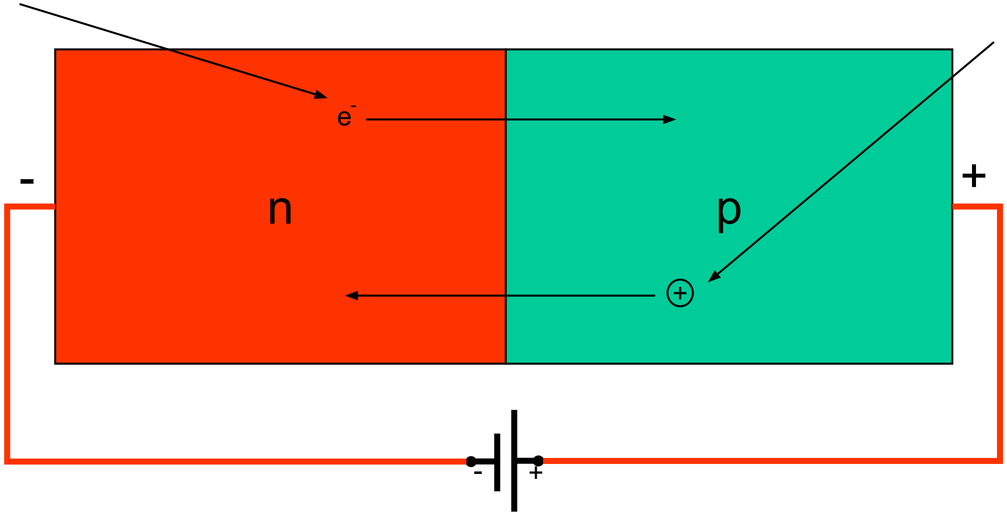
What happens when you apply an electric potential?



The Depletion Layer will grow because the free electrons fill the holes in the p-type and leave no more electrons in n-type because the battery is attracting the other electrons. Thus the depletion layer acts as a barrier and no current will flow.

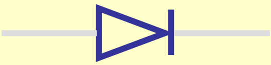
Reverse the polarity

Electrons go across the P-Type leaving positive holes and are replaced by electrons from the battery.



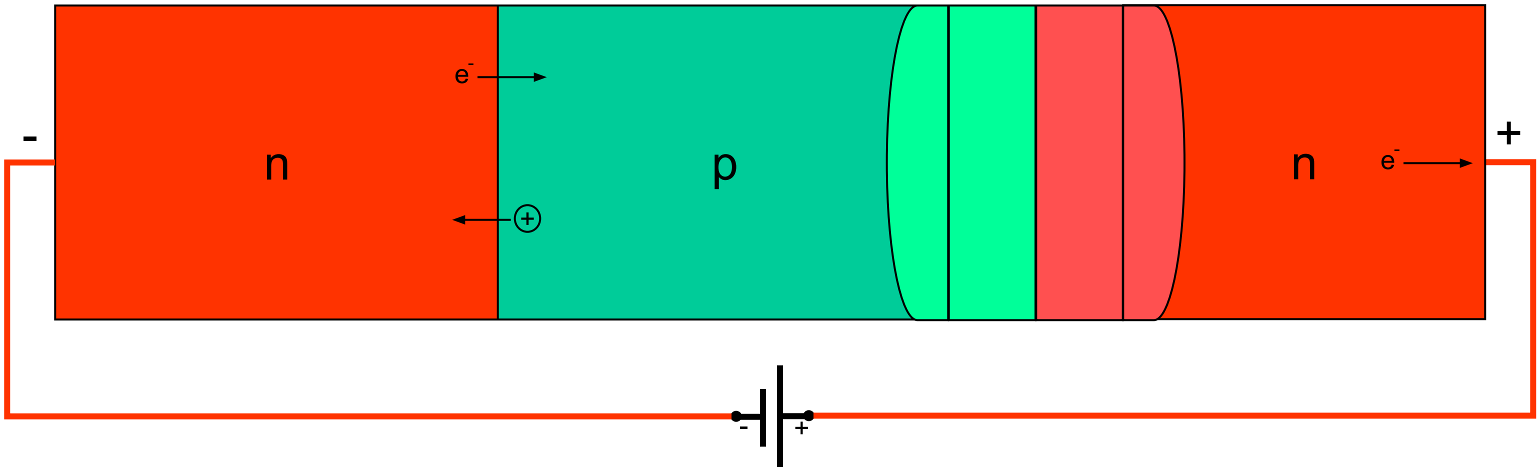
The positive holes move towards the N-Type.

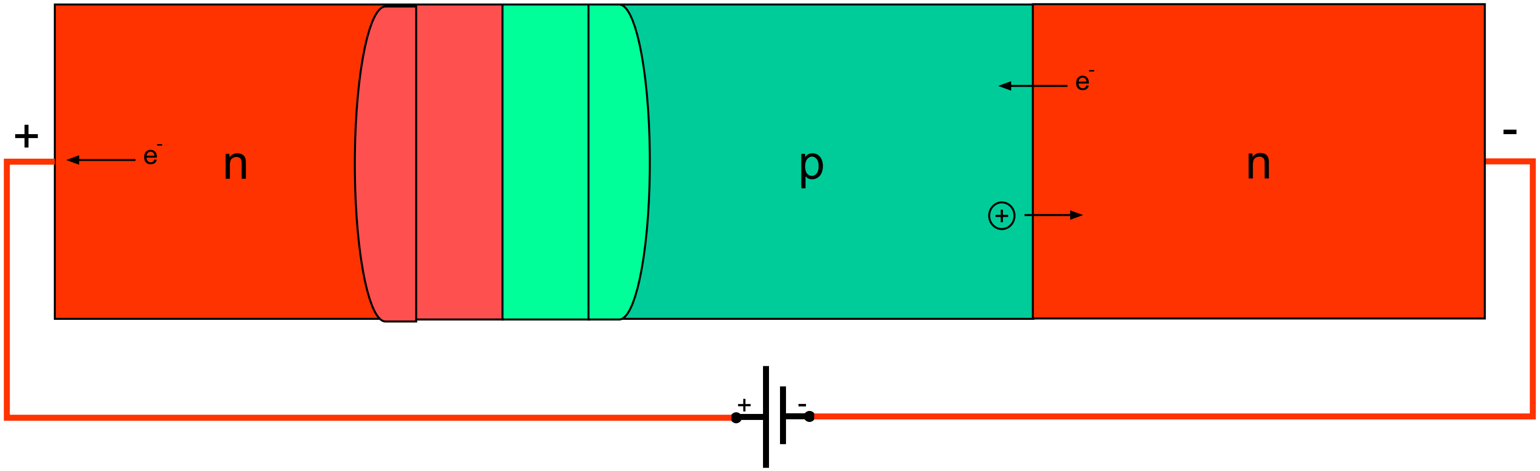
Schematic Symbol



Thus, current flows and it can only flow in one direction. This is known as a Diode.
This is also the technique used to convert Alternating Current to Direct Current.

What happens when we try to connect three types?

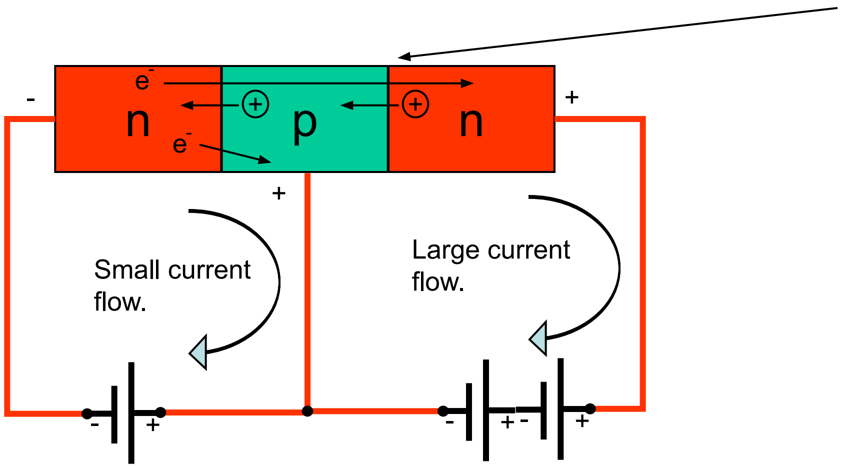




No current flows either way. This is pointless

Let’s attach a small electric potential to the P-Type?

Electrons move in massive numbers to the right and in small numbers down the wire coming out of the P-Type.



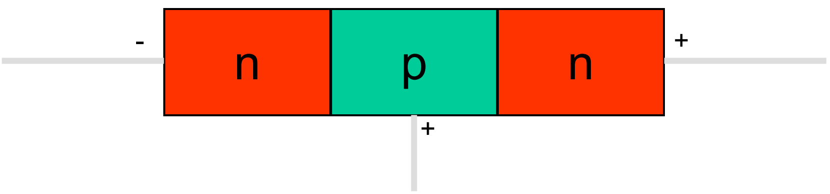
Small current flow.

Large current flow.

There would be a depletion layer here without the larger battery.

Definitions for new component?

Emitter or Source



Collector or Drain

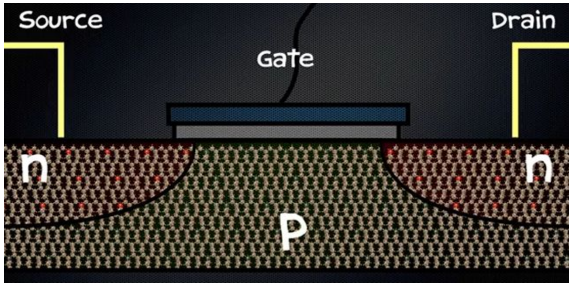
Gate or Base

How would our new component behave if we could choose when to apply the voltage to the base?

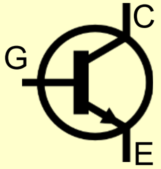
NPN Transistor = Switch

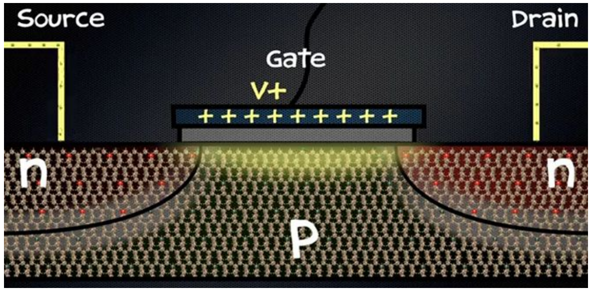
(Push button normally open - implies applying power to base turns on the switch)

- Electrical switch that doesn’t have any moving parts
- It can be turned on/off at nearly the speed of light
- Many Types – npn, pnp, pchannel mosfet, nchannel mosfet, jfet, etc.
- They can be very tiny, 22 nm wide (about 50 atoms across) – getting smaller
- Phones - around 100 million transistors; Computers - over a billion transistors
- Can group them to form logic gates, which is the basis for every component in the computer

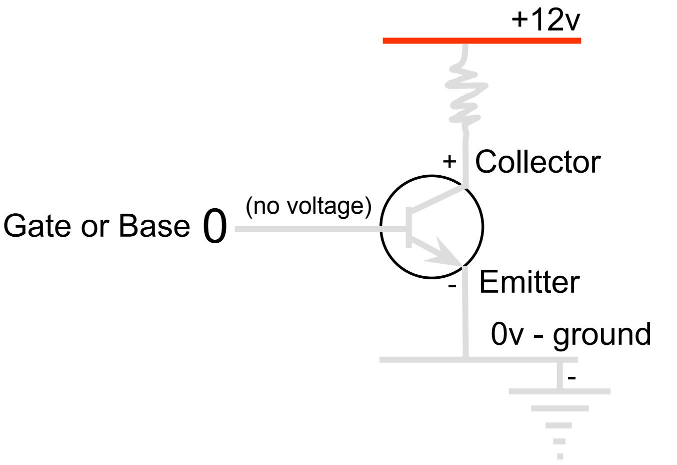


Schematic Symbol

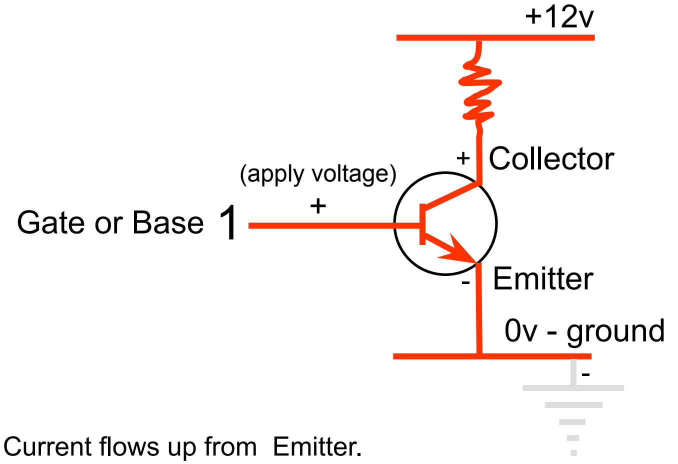




SWITCH IS OFF



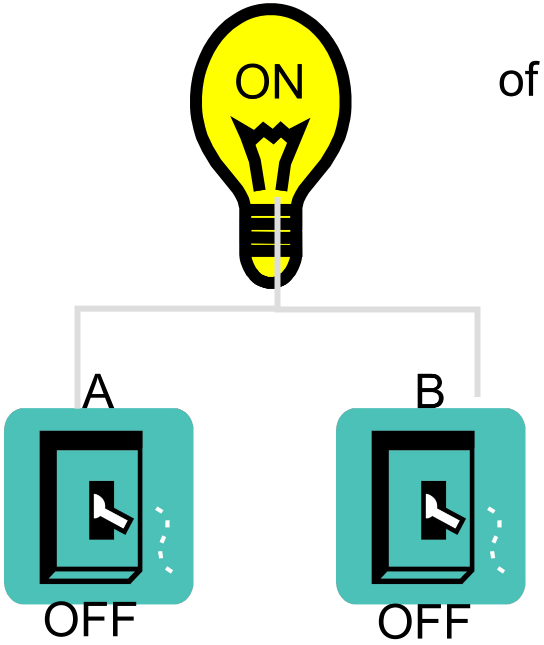
SWITCH IS ON

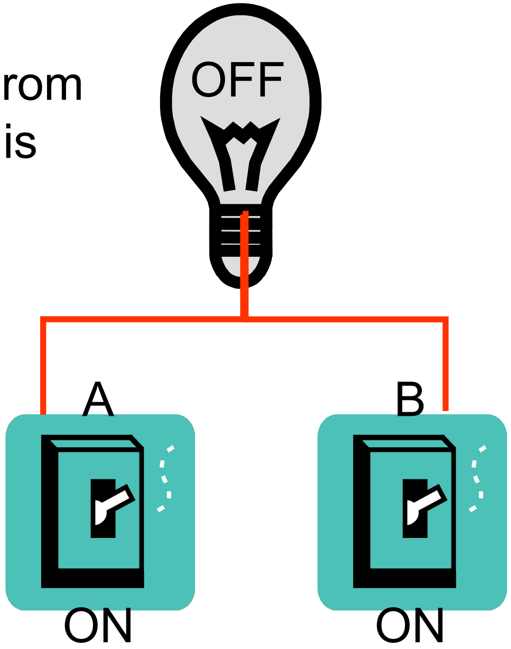


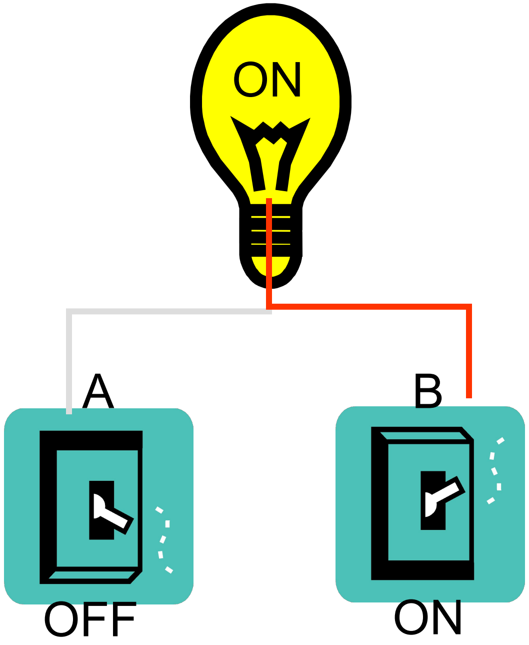
Current flows up from Emitter.

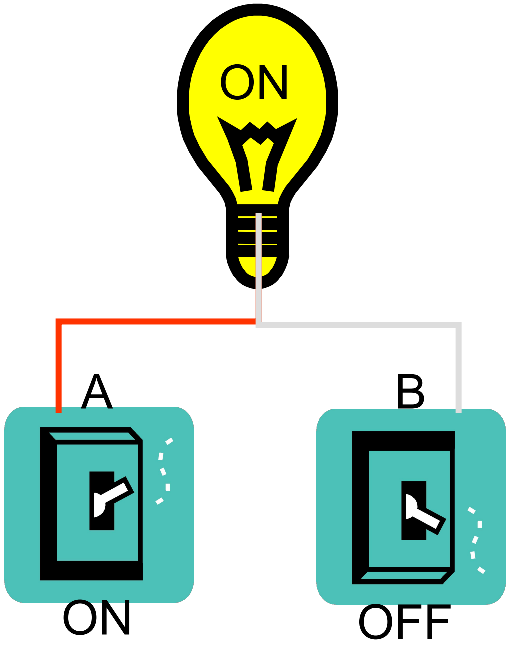
Logic and Math

The Most Basic Parts of a Computer Are Made From Something Similar To This







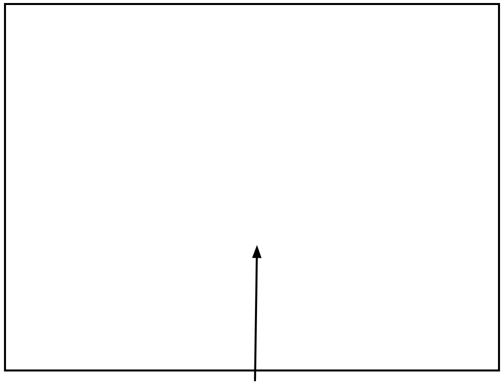


Logic With Two States

On a school playground, some children are wearing a **green shirt**, some are wearing **blue shoes**, some are wearing both, and some are not wearing either.

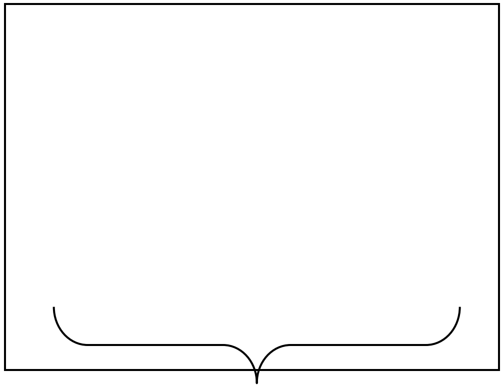
Venn Diagrams

AND



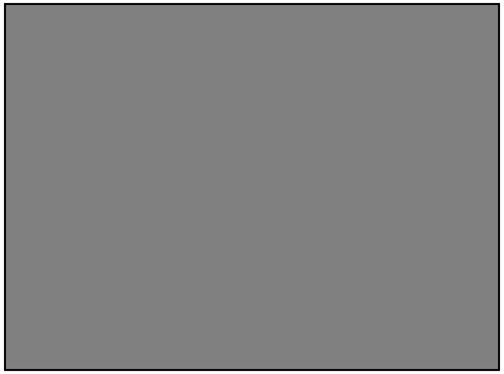
green shirt AND **blue shoes**

OR



green shirt OR **blue shoes**

NOT



NOT (**green shirt** OR **blue shoes**)

Truth Tables

AND

green	**blue**	RESULT
False	False	False
False	True	False
True	False	False
True	True	True

OR

green	**blue**	RESULT
False	False	False
False	True	True
True	False	True
True	True	True

NOT

Green or **blue**	Result
False	True
True	False

Boolean Algrebra – Just the Basics

(George Boole 1815-1864)

- Basic Boolean Operators
  - AND: $Z = A \cdot Y$ (this can also be written as $Z = XY$)
  - OR: $Z = A + Y$
  - NOT: $Z = \bar{Y}$
  - **T** implies True
  - F implies False

- Identity
  - $A \cdot \mathbf{T} = A$
  - $Y + F = Y$

- Zero Property
  - $A \cdot F = F$
  - $A + \mathbf{T} = \mathbf{T}$

- Commutative:
  - $A \cdot Y = Y \cdot Y$
  - $A + Y = Y + A$

- Associative
  - $A(XY) = (AX)Y$
  - $A + (X + Y) = (A + X) + Y$

- Distributive
  - $A(X + Y) = AX + AY$
  - $(A + Y)(W + X) = AW + AX + YW + YX$

- Demorgan's Laws
  - $\bar{Y} + \bar{A} = \bar{Y}\bar{A} = \overline{YA}$

Number Systems – Just the Basics

- Base 10 – Decimal (Arabic Numerals)
    - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (we call them digits – gee we call our fingers digits too)
    - $a \cdot 10^n + \cdot\cdot\cdot + b \cdot 10^3 + c \cdot 10^2 + d \cdot 10^1 + e \cdot 10^{\circ}$

- Base 2 – Binary
    - Symbols: 0, 1 (we call them Bits)
    - This is the number system of the computers
    - Convert to base 10: $a \cdot 2^n + \cdot\cdot\cdot + b \cdot 2^3 + c \cdot 2^2 + d \cdot 2^1 + e \cdot 2^{\circ}$ where a-e are either 1 or 0

- Base 8 – Octal
    - Symbols: 0, 1, 2, 3, 4, 5, 6, 7
    - A single value represents a group of 3 Bits (e.g. 000 through 111)
    - We use this because binary is too long and hard to read
    - Was used in computing systems such as IBM Mainframes which had 12-bit, 24-bit, and 36 bit Words. Each Word size is divisible by three bits.
    - Convert to base 10: $a \cdot 8^n + \cdot\cdot\cdot + b \cdot 8^3 + c \cdot 8^2 + d \cdot 8^1 + e \cdot 8^{\circ}$ where a-e are 0-7

- Base 16 – Hexadecimal
    - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
    - A single value represents a group of 4 bits (e.g. 0000 through 1111) – also known as a Nibble
    - 2 Hexadecimal numbers = 8 Bits = 1 Byte (e.g. 10011101 = (1001)(1101) = 9D)
    - Modern computer platforms use 16-bit, 32-bit, and 64-bit Words
    - Convert to base 10: $a \cdot 16^n + \cdot\cdot\cdot + b \cdot 16^3 + c \cdot 16^2 + d \cdot 16^1 + e \cdot 16^{\circ}$ where a-e are 0-F

Definition: Word represents the number of bits (inputs or outputs) that can be wired to a component.

Sample Conversion of Base 2 to Base 10 and Base 16

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

$= 1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$

$= 1\cdot128 + 0\cdot64 + 0\cdot32 + 1\cdot16 + 1\cdot8 + 1\cdot4 + 0\cdot2 + 1\cdot1$

$= 128 + 0 + 0 + 16 + 8 + 4 + 0 + 1$

$= 128 + 16 + 8 + 4 + 1$

$= 157$

$\therefore$ 10011101 in base 2 is 157 in base 10

Make 4-bit groups. Each group represents 1 Hexadecimal number. 2 Hexadecimal numbers represent one Byte.

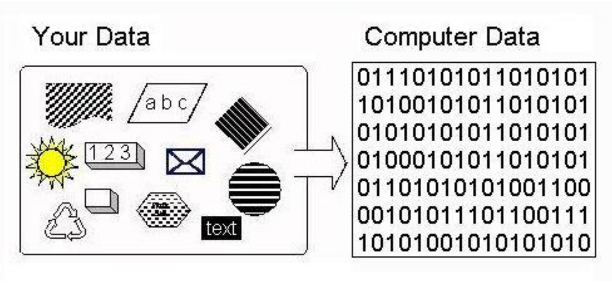
$$\overbrace{\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \end{array}\right)}\overbrace{\left(\begin{array}{cccc} 1 & 1 & 0 & 1 \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \end{array}\right)}$$

$= (1\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0)\ (1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0)$

$= (1\cdot8 + 0\cdot4 + 0\cdot2 + 1\cdot1) \quad (1\cdot8 + 1\cdot4 + 0\cdot2 + 1\cdot1)$

$= (9)\ (13) = 9D = 157$

Does grouping bits together mean something to the computer?



- As it turns out, in general, grouping bits doesn't mean much to the computer. However, the groupings do mean something to us and we can instruct the computer when to use them one way and when to use them another. Below are some of the most common representations of the groupings.

- Positive Numbers: We've already seen that grouping 8 bits together can represent a positive number from 0 to 255. You can group more bits together to make bigger numbers.

- Negative Numbers: We could say the first bit is the sign bit (e.g. 11100111) . If it is a 1, then that grouping of 8 bits would be a negative number, if it is a 0, then the grouping would represent a positive number. However, using that bit for the sign changes the possible numbers to something like -128 to 127 (although it again is up to us to define it and not the computer). The trick is how do we represent 0, it could be 00000000 or +0, or it could be 10000000 or -0. Using what's know as "Two's Complement", we are going to define it as:
  - 0 = 00000000
  - 1 = 00000001
  - 127 = 01111111
  - -128 = 10000000
  - -127 = 10000001

- Floating Point Numbers: These are numbers that have a decimal in them. They are also known as real numbers. (e.g. 123.456)

Does grouping bits together mean something to the computer?

- Letters and Symbols: What if we want to have them represent a letter, or character, similar to what you are reading here. There are only 26 lower case letters, 26 uppercase letters, and a bunch of of symbols in English language. How could we represent these using our grouping of bits. As it turns out, a committee formed a while back to create a character set that was standard for every. It is known as ASCII and it was originally only 7-bits for a possibility of 128 different characters. Later it was extended to 8-bits and that allowed for 256 different characters. The first 32 characters are not even visible (i.e. non-printing) characters and are known as control characters. They represents things like tab, return, delete, etc. There is even a code that means nothing (or the lack of something) called NULL (0x00).

- Other Languages: While ASCII is nice, what about other languages, such as Chinese or Arabic. They have tons more symbols. How do we represent all those symbols when we only have 8-bits. Another standard known as UTF-8, which includes ASCII as a subset, was invented for this very purpose.

- Fonts: A font is a particular size, weight and style of a typeface. I used a font called Arial to create this slide.

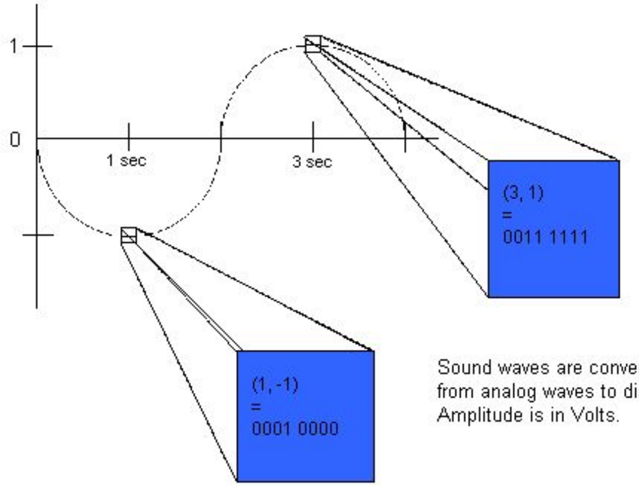
- Addresses: The grouping of bits can also represent something that we use as an address, similar to the addresses to our homes. Such as an address to data in memory or an address to a device.

- Pixels: We can use them to draw pixels on a screen to make pictures. I pixel is the smallest addressable element in a display device. The more bits we use, the more color the pixel can have; this is also known as the number of bits per pixel (bpp).
  - 1 bpp = 2 colors (monochrome)
  - 2 bpp = 4 colors
  - 3 bpp = 8 colors
  - 8 bpp = 256 colors
  - 16 bpp = 64k colors (HighColor)
  - 24 bpp = 16 million colors (Truecolor)

Does grouping bits together mean something to the computer?

- Pictures: Such as a JPEG, BITMAP, GIF, PNG, etc. They also are made of pixels.

- Sound: Sound occurs naturally as an analog wave. The analog wave is converted to digital using a process called sampling. The height of the wave is sampled at regular intervals of time, often small fractions a second. If one byte is used to hold a single sample of analog wave, then the wave can be 1 of 256 different heights for that sample. These heights represent the decibel levl of the sound. Thus a spoken word might occupy several hundred bytes. See below.

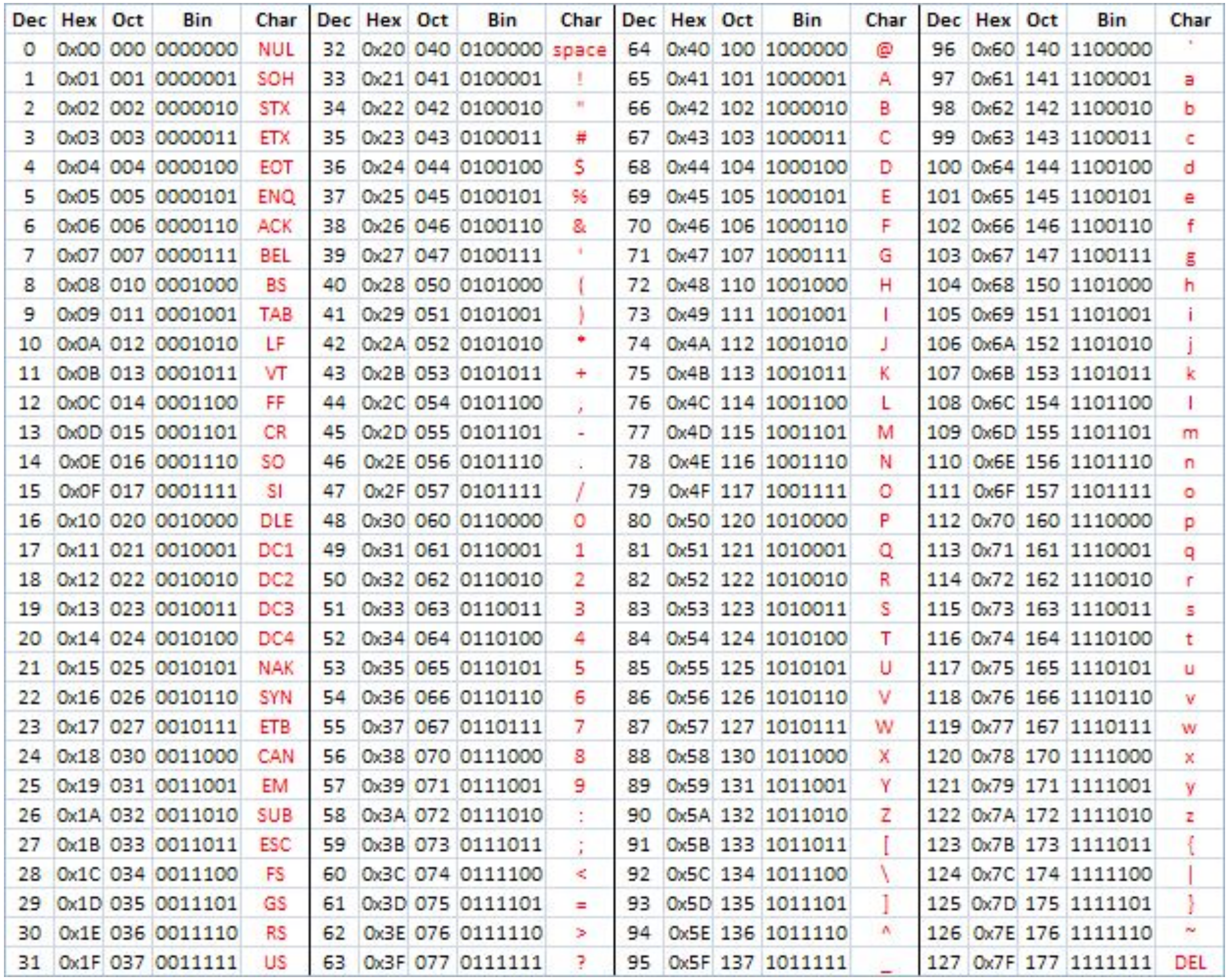


Sound waves are converted
from analog waves to digital data.
Amplitude is in Volts.

- Program Instruction or Data: A grouping of bits can tell the CPU what to do as an instruction, such as to load some data from RAM.

ASCII

32 control characters – They are non-printable.



Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

Logic Gates

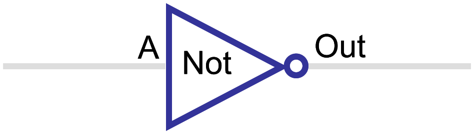
Logic Gates

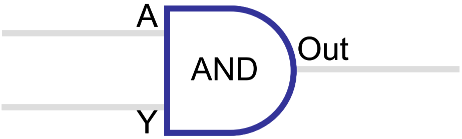
Logic Gates translate that “Boolean Logic” into an electronic form

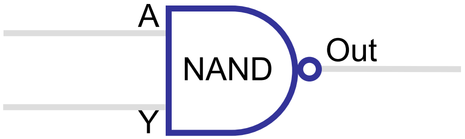
- There are seven fundamental Logic Gates. With the exception of the NOT gate, they take two inputs and produce one output. If a voltage is applied to a line it represents a 1; no voltage represents 0.

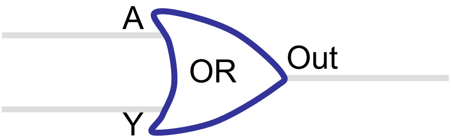
- The cheapest and most fundamental of all is the NAND gate. All other Logic Gates and nearly all other computer components can be made with this gate.

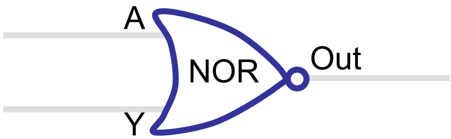
- We can use logic gates in various connected-patterns to make nearly every functional piece of a computer. A Gate, or any component made from Gates, must have power connected to it in order to operate.

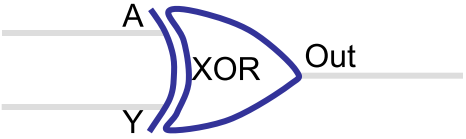


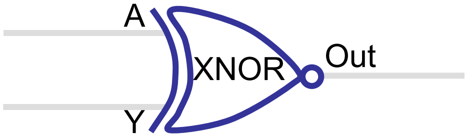




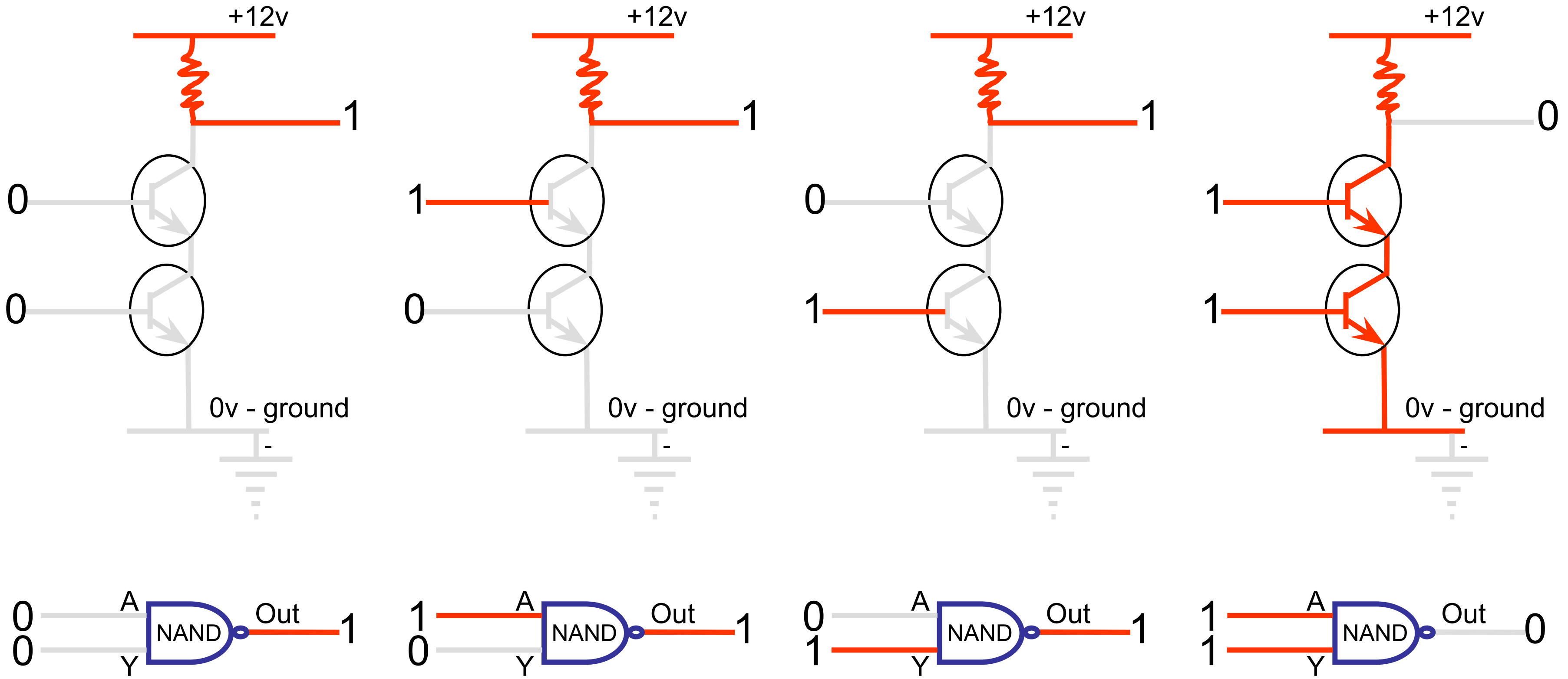








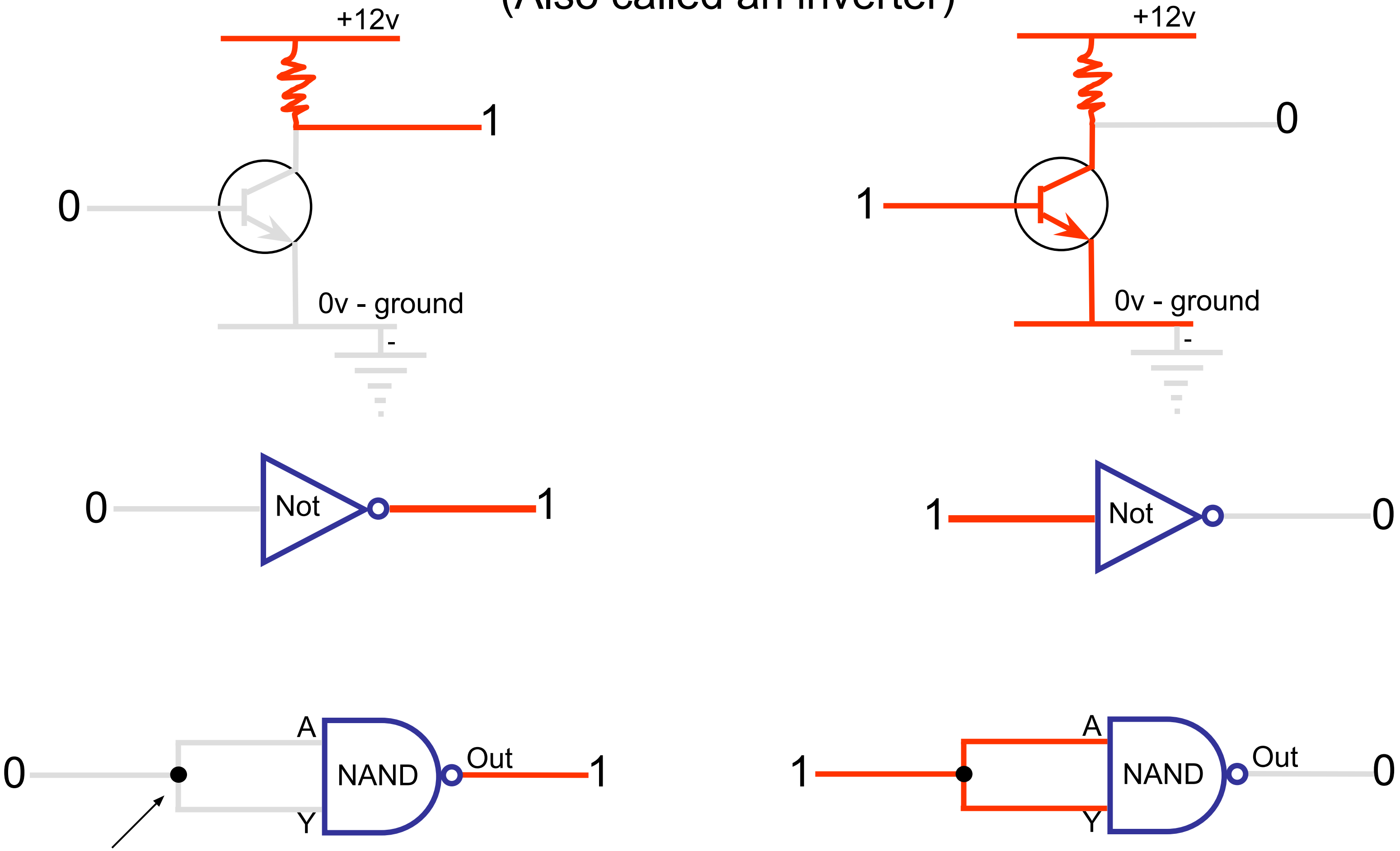
The NAND Gate – (all the gates can be built from this)



NAND Truth Table

A	Y	RESULT
0	0	1
0	1	1
1	0	1
1	1	0

The NOT Gate
(Also called an inverter)

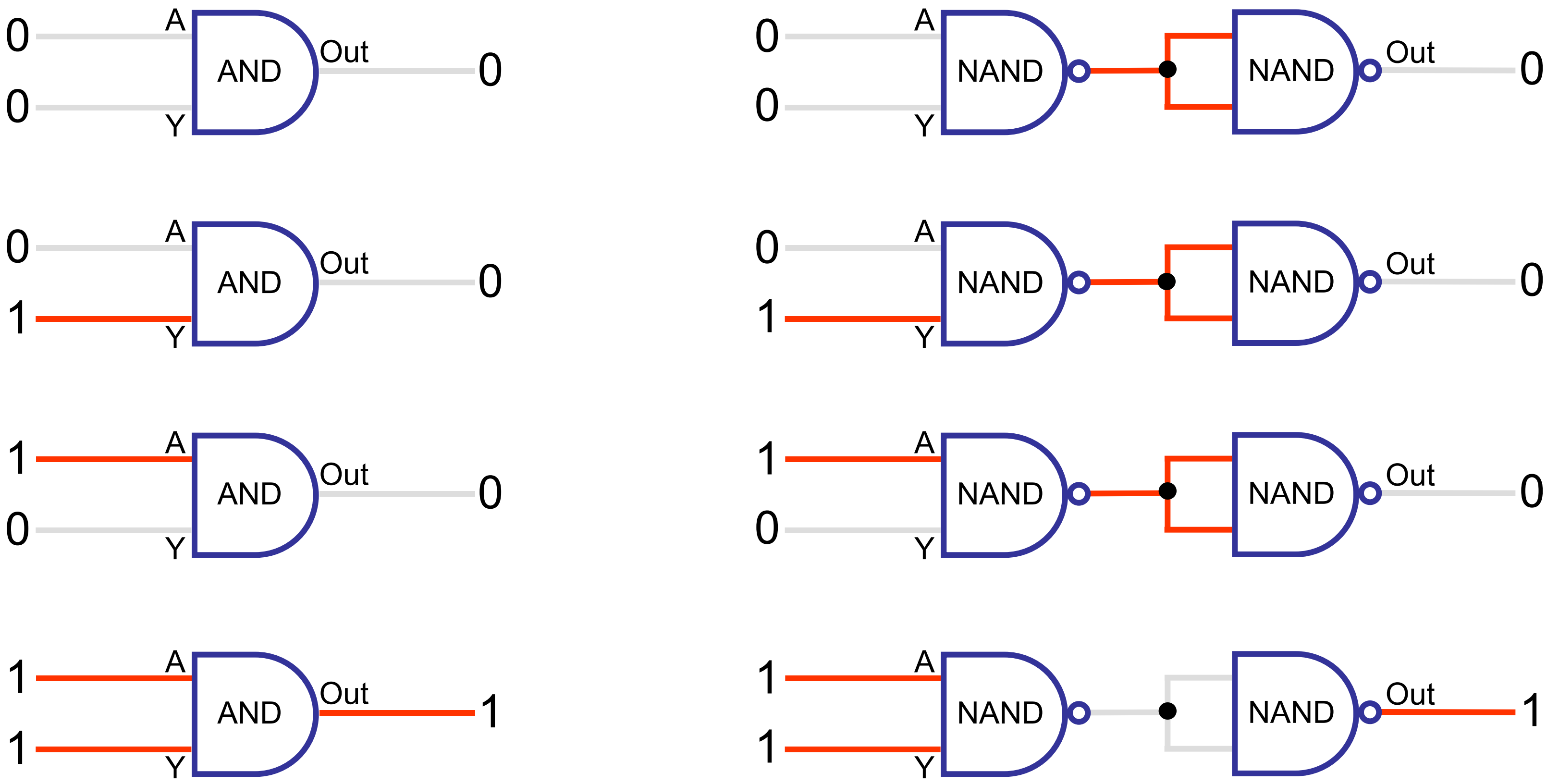


The BLACK circle means the lines are connected.

NOT Truth Table

A	Result
0	1
1	0

The AND Gate

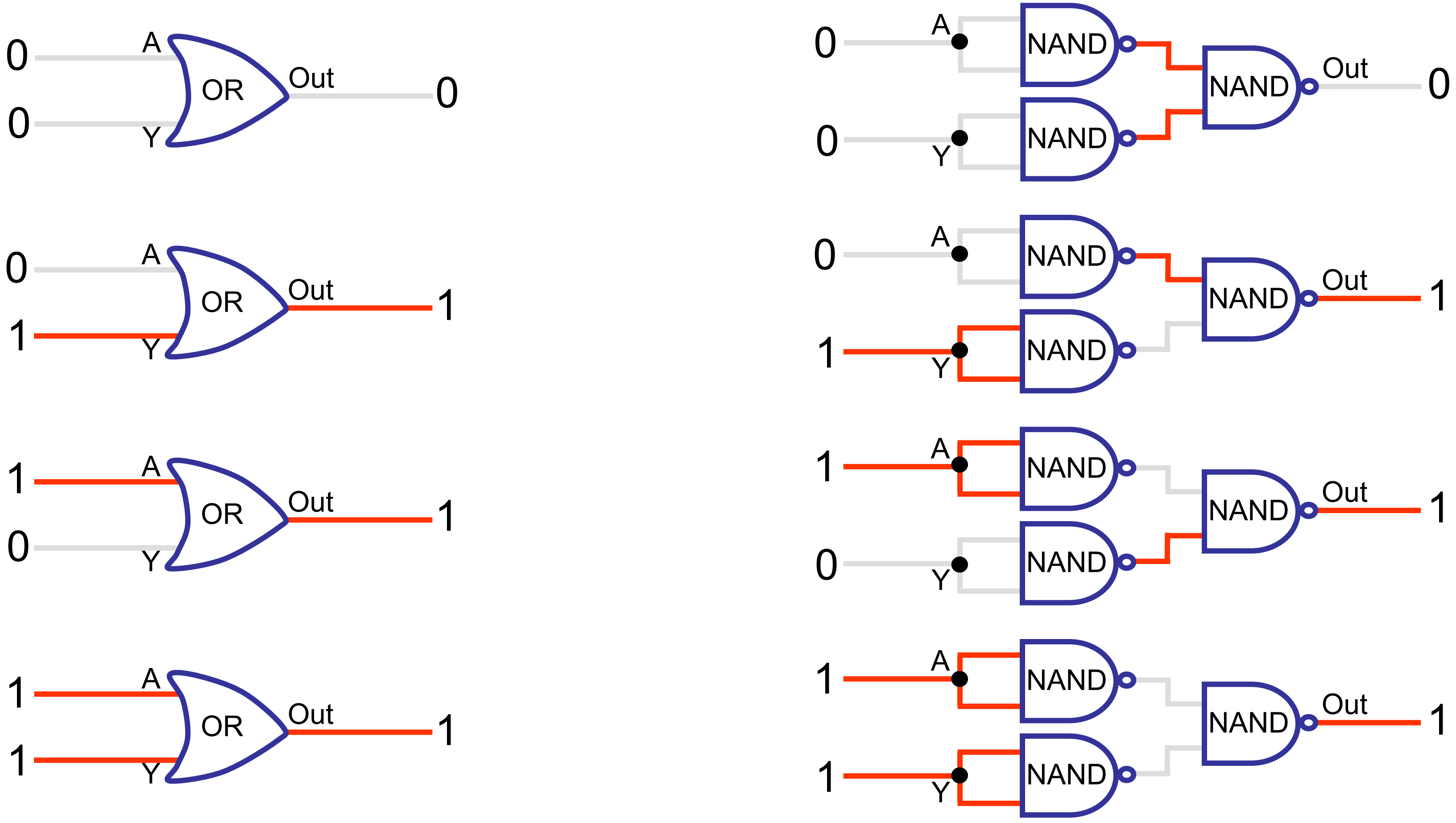


AND Truth Table

A	Y	RESULT
0	0	0
0	1	0
1	0	0
1	1	1

Notice how it is the opposite of the NAND gate. You could connect a NOT gate to a NAND gate to invert it.

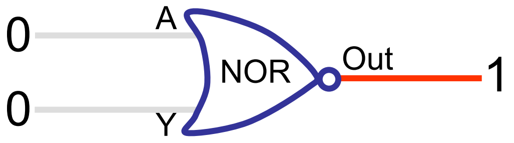
The OR Gate

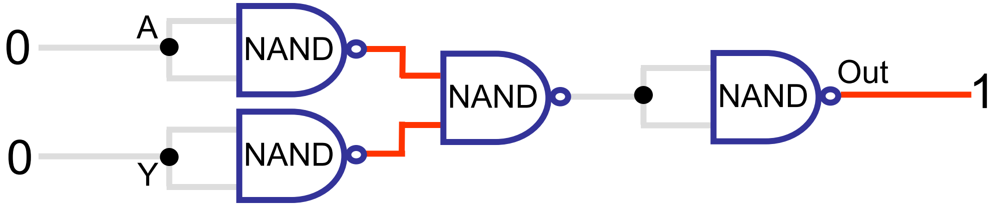


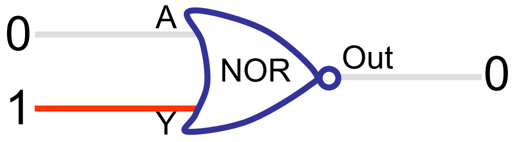
OR Truth Table

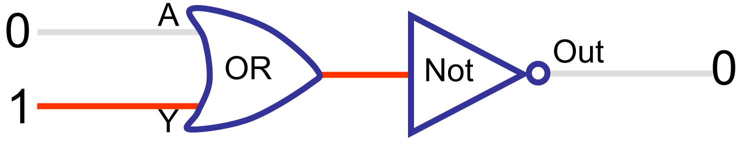
A	Y	RESULT
0	0	0
0	1	1
1	0	1
1	1	1

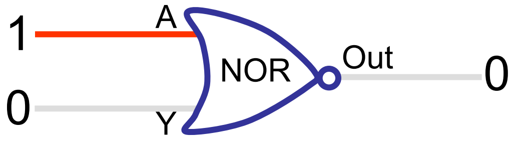
The NOR Gate

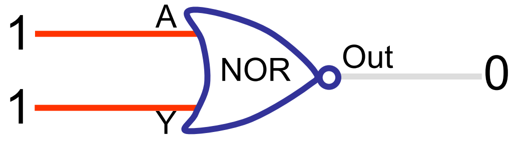










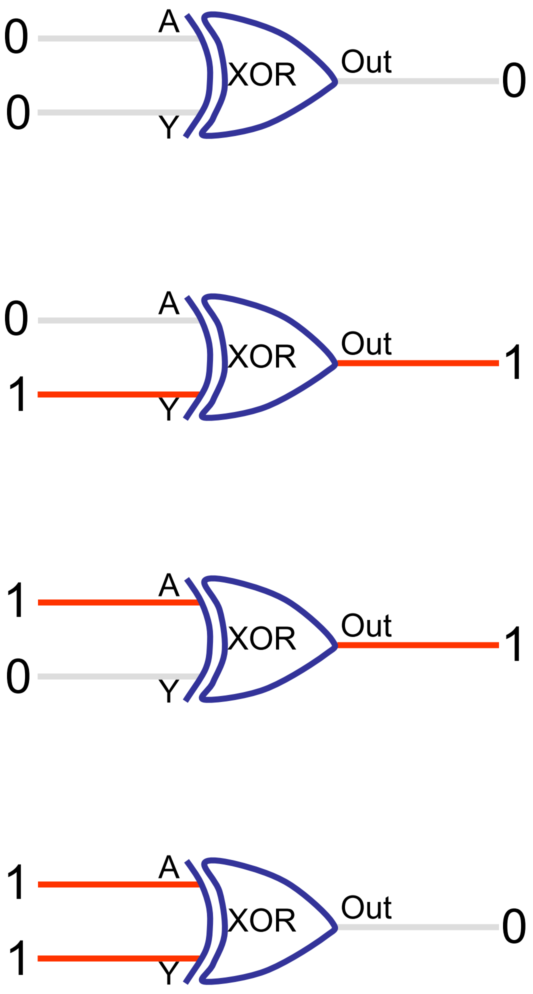


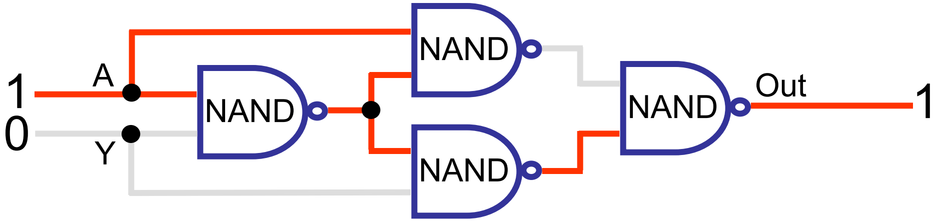
NOR Truth Table

A	Y	RESULT
0	0	1
0	1	0
1	0	0
1	1	0

The XOR Gate

(Exclusive OR – one or the other, but not both)

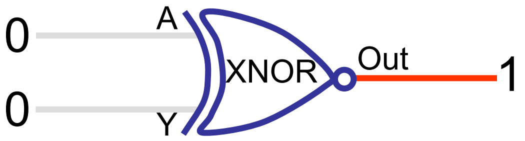


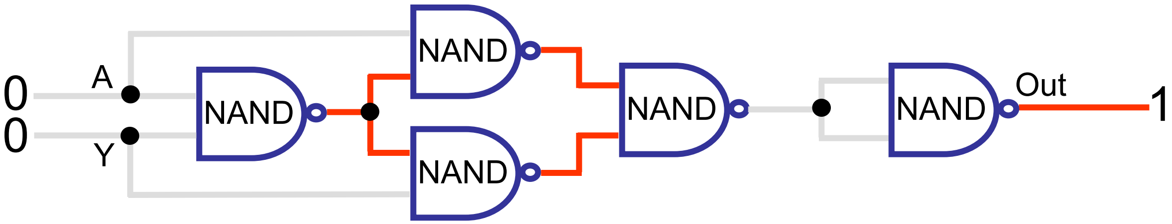


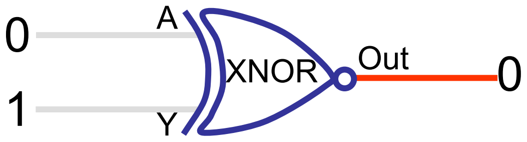
XOR Truth Table

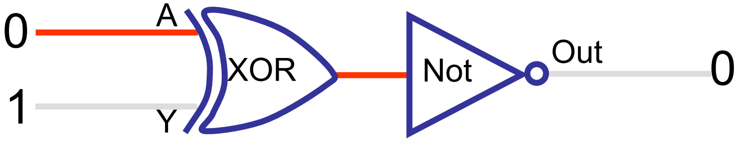
A	Y	RESULT
0	0	0
0	1	1
1	0	1
1	1	0

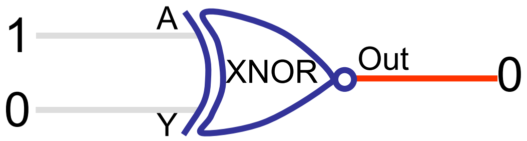
The XNOR Gate

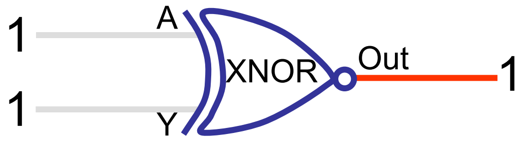








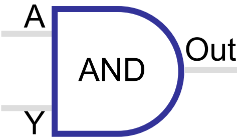
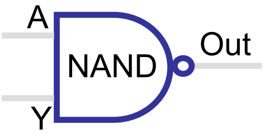
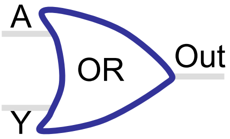
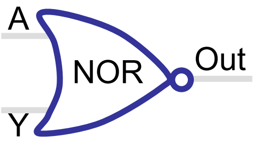
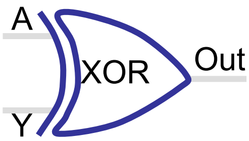
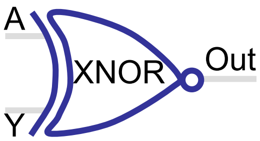


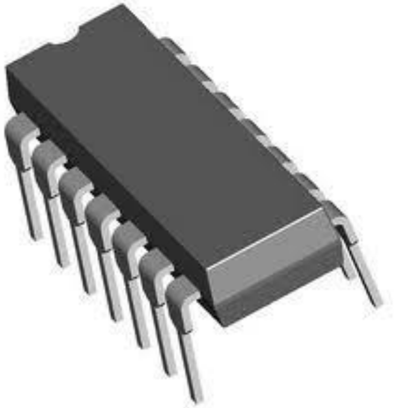


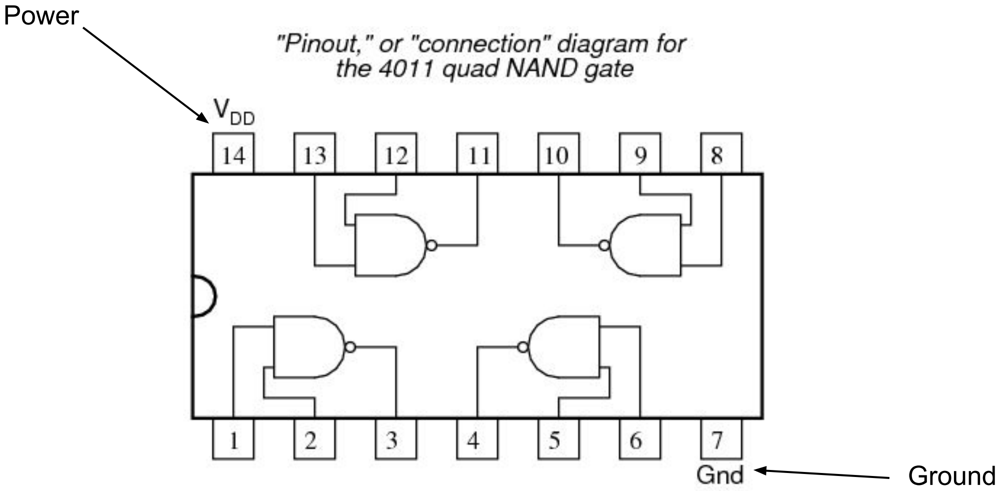
XNOR Truth Table

A	Y	RESULT
0	0	1
0	1	0
1	0	0
1	1	1

Gate Cheatsheet

A	Y						
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1



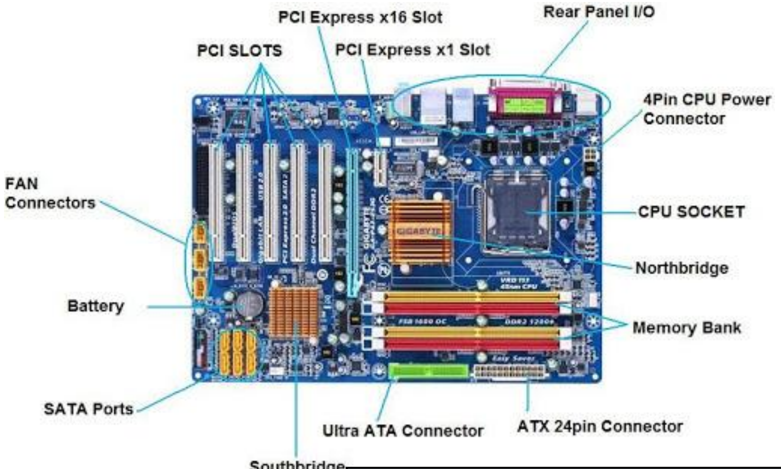


"Pinout," or "connection" diagram for the 4011 quad NAND gate

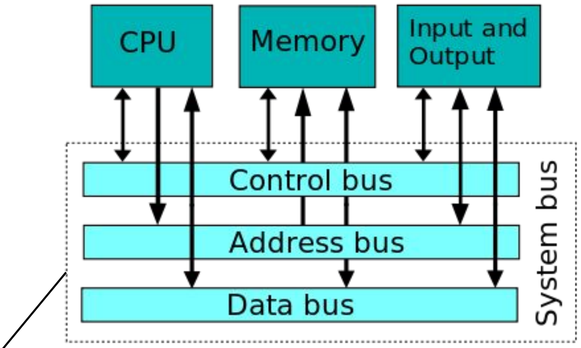
The Computer / Hardware

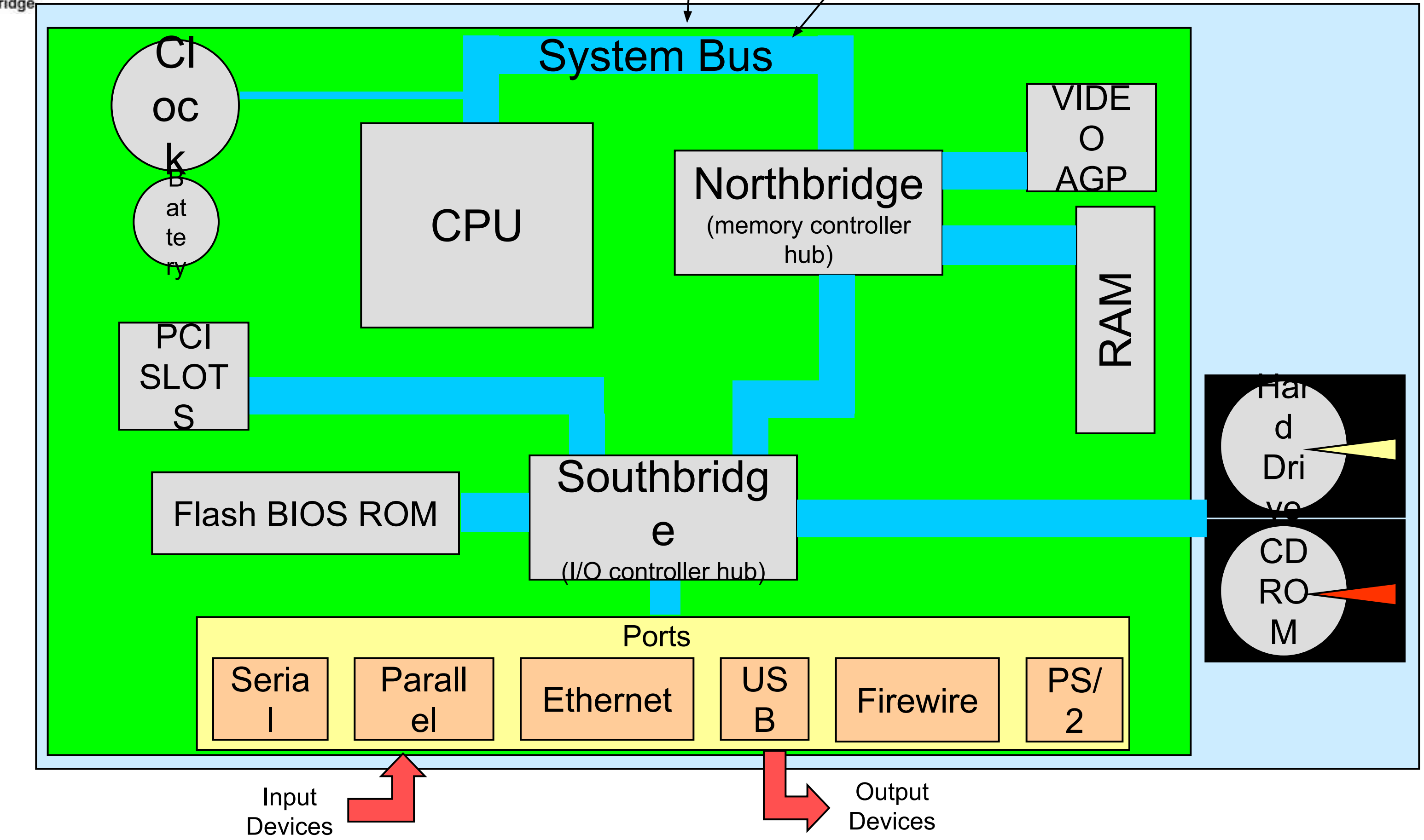
For the most part, we are going to talk about a “Made-Up” computer based upon J Clark Scott’s book. Real-World computers will have the same basic functionality.

Open Up!



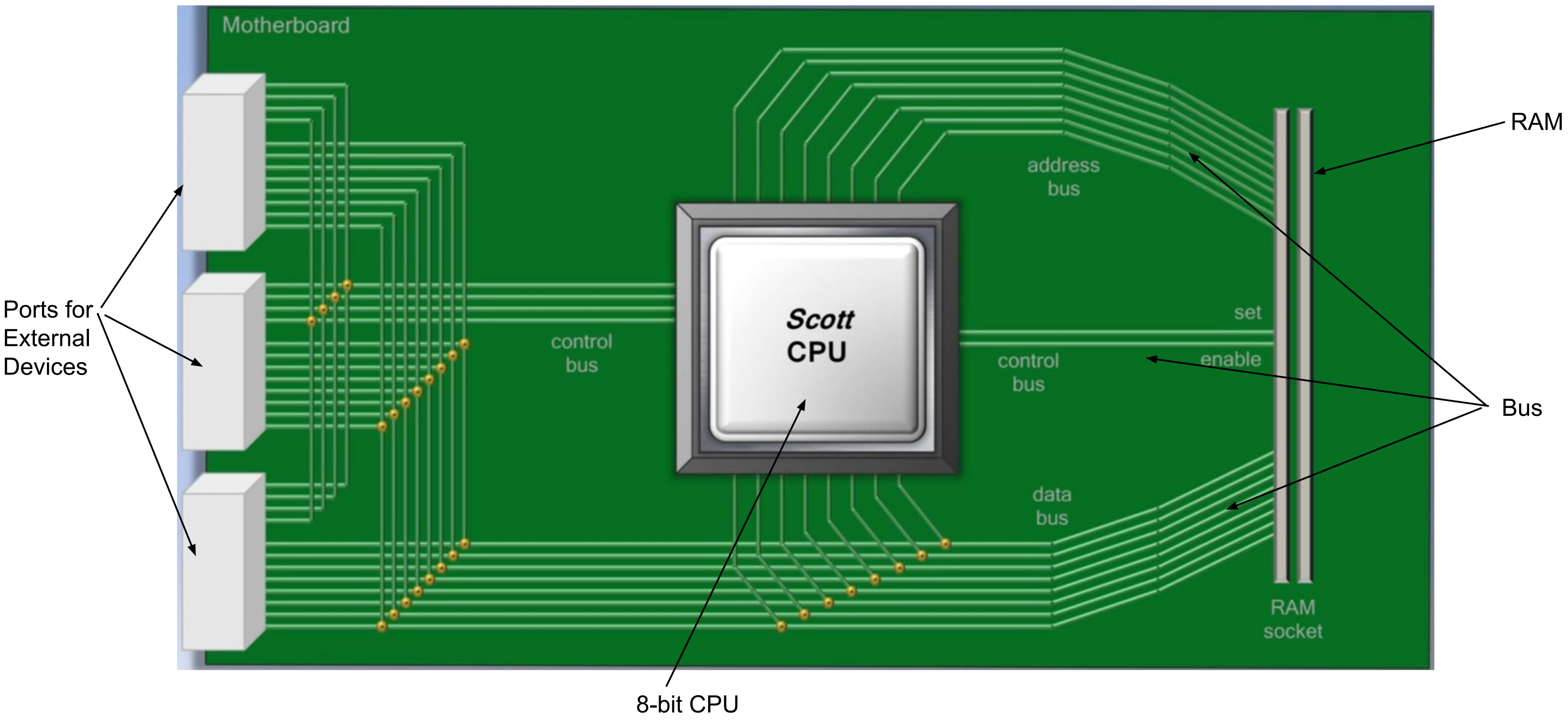
The bus that connects to the CPU is also known as the Front-Side Bus.





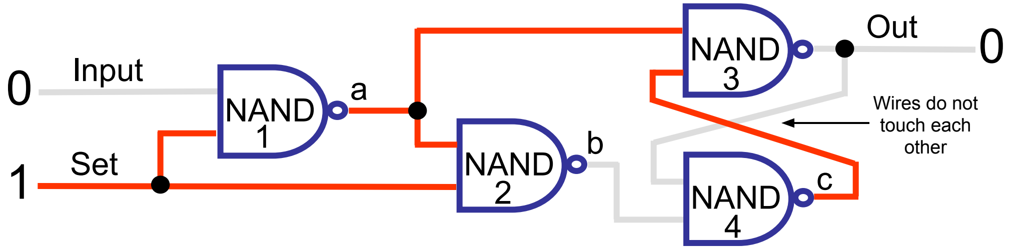
The Computer

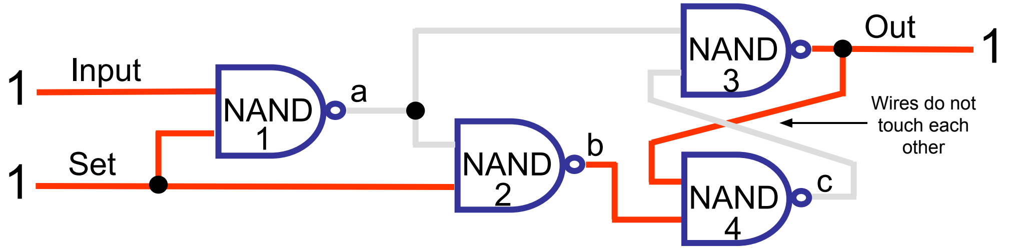
- In simplistic terms, the computer can be broken down into Four parts.
- The first part is memory, also known as Random Access Memory (RAM), and it will maintain information as long as it has power.
- The second part is the Central Processing Unit (CPU). It contains an Arithmetic Logic Unit (ALU) and several Registers for temporary storage (as a reminder, a register is a word of memory).
- The third part is known as the data bus. It represents a word of bits that travel from the CPU to RAM or external devices. The CPU will also have a data bus so that bits can travel from CPU to Registers or ALU, etc. Lastly, a word of bits can travel to external devices using the data bus.
- The last part are external devices, such as the keyboard, monitor, parallel port, serial port, USB port, etc. Each of these devices are connected to the data bus in some fashion.



Part One: Memory

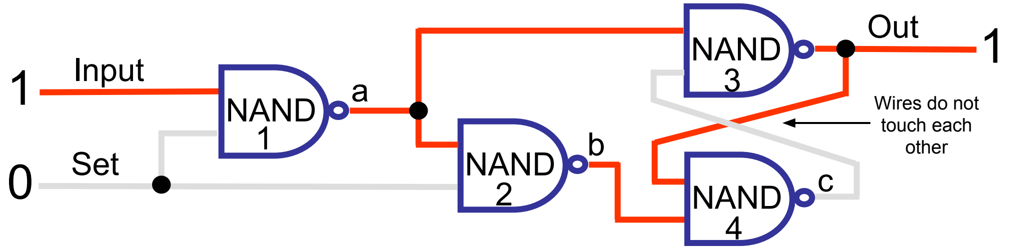
1 Bit of Memory

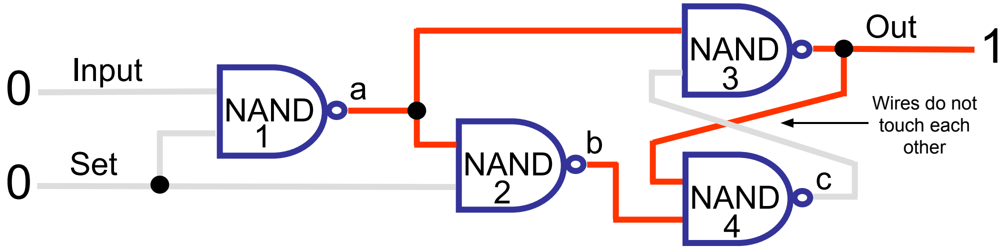




When the **Set** bit is ON, **Out** will always equal whatever the input is.

The value will be remembered as long as the power is on.



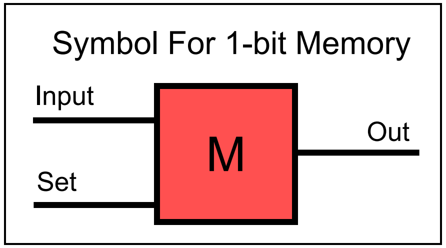


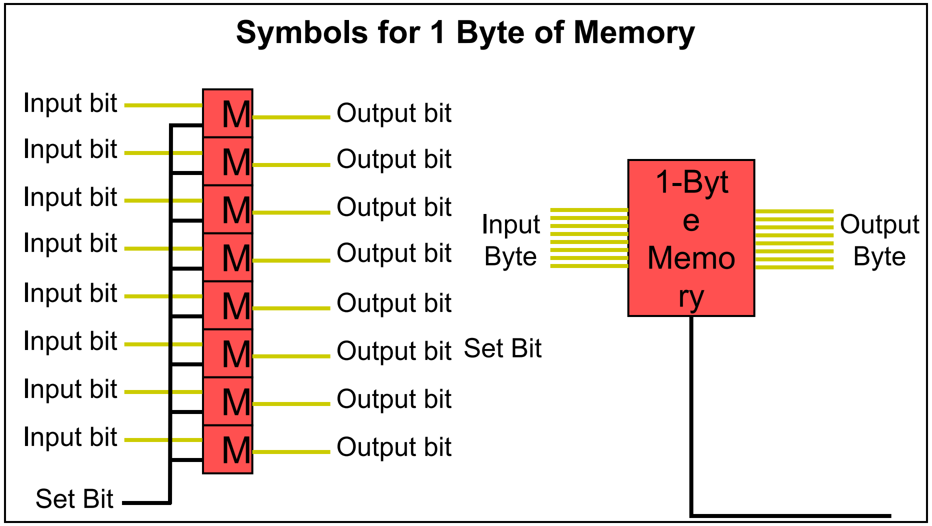
When the **Set** bit is OFF, **Out** will remain whatever it was before and cannot be changed no matter what input is.

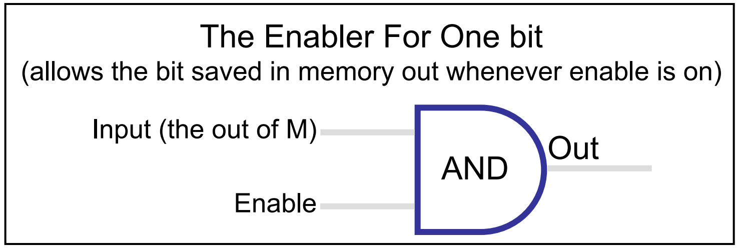
The value will be remembered as long as the power is on.

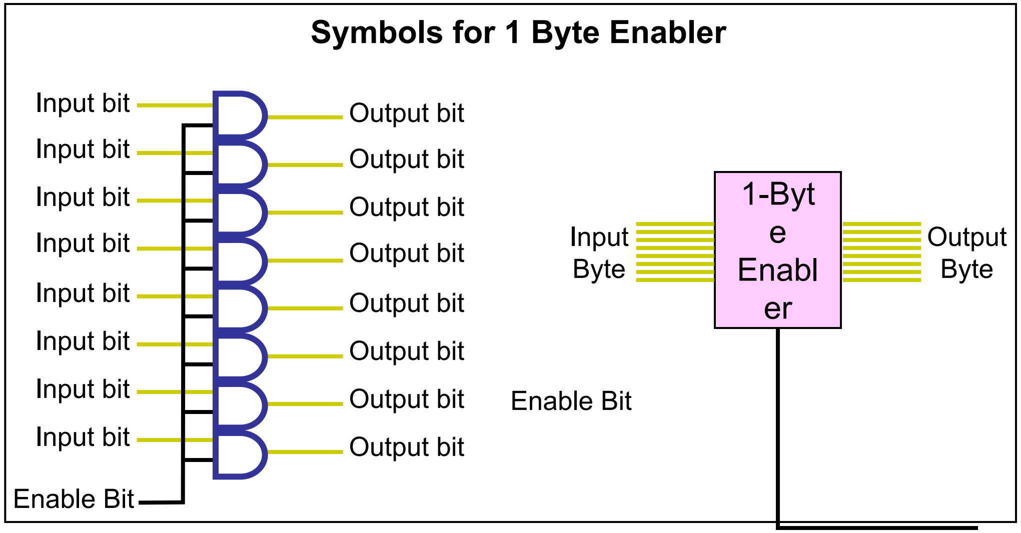
If **Input** and **Out** were on before **Set** was turned off, gate 3 had both inputs off and gate 4 had both on, so c will be off and out will be 1. When **Set** goes off, **a** and **b** will always be on, **c** will continue to be off. Changing input will continue to have the same structure, therefore nothing happens when **Set** is off and **Input** is changed.

1 Byte Memory / 1 Byte Enabler
(Memory's output would flow onto bus without enabler.)


Symbol For 1-bit Memory

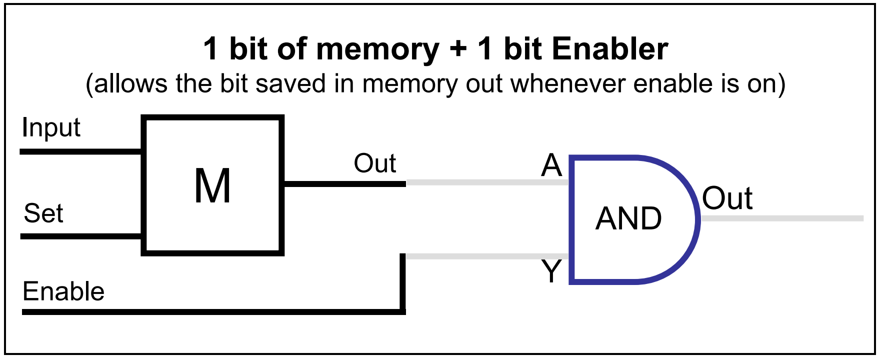

Symbols for 1 Byte of Memory

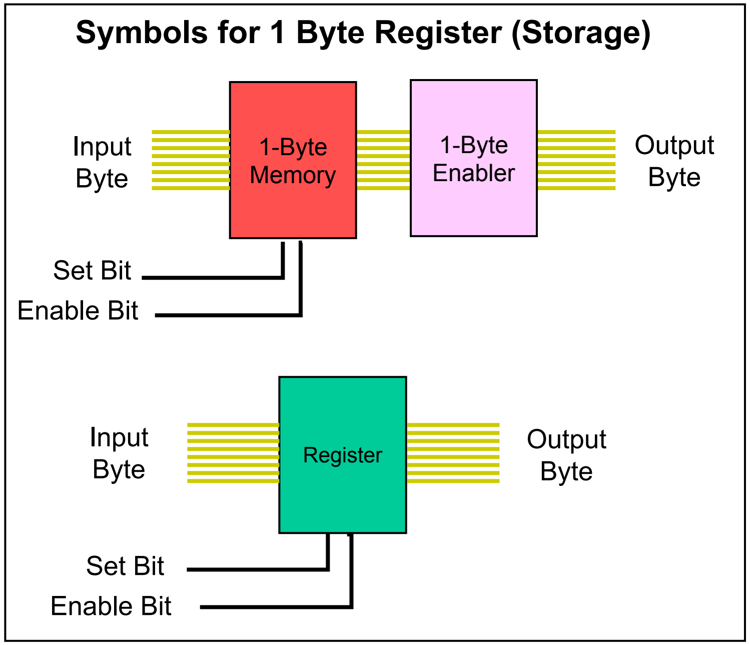

The Enabler For One bit
(allows the bit saved in memory out whenever enable is on)


Symbols for 1 Byte Enabler

The Register
(1 Byte in our case)

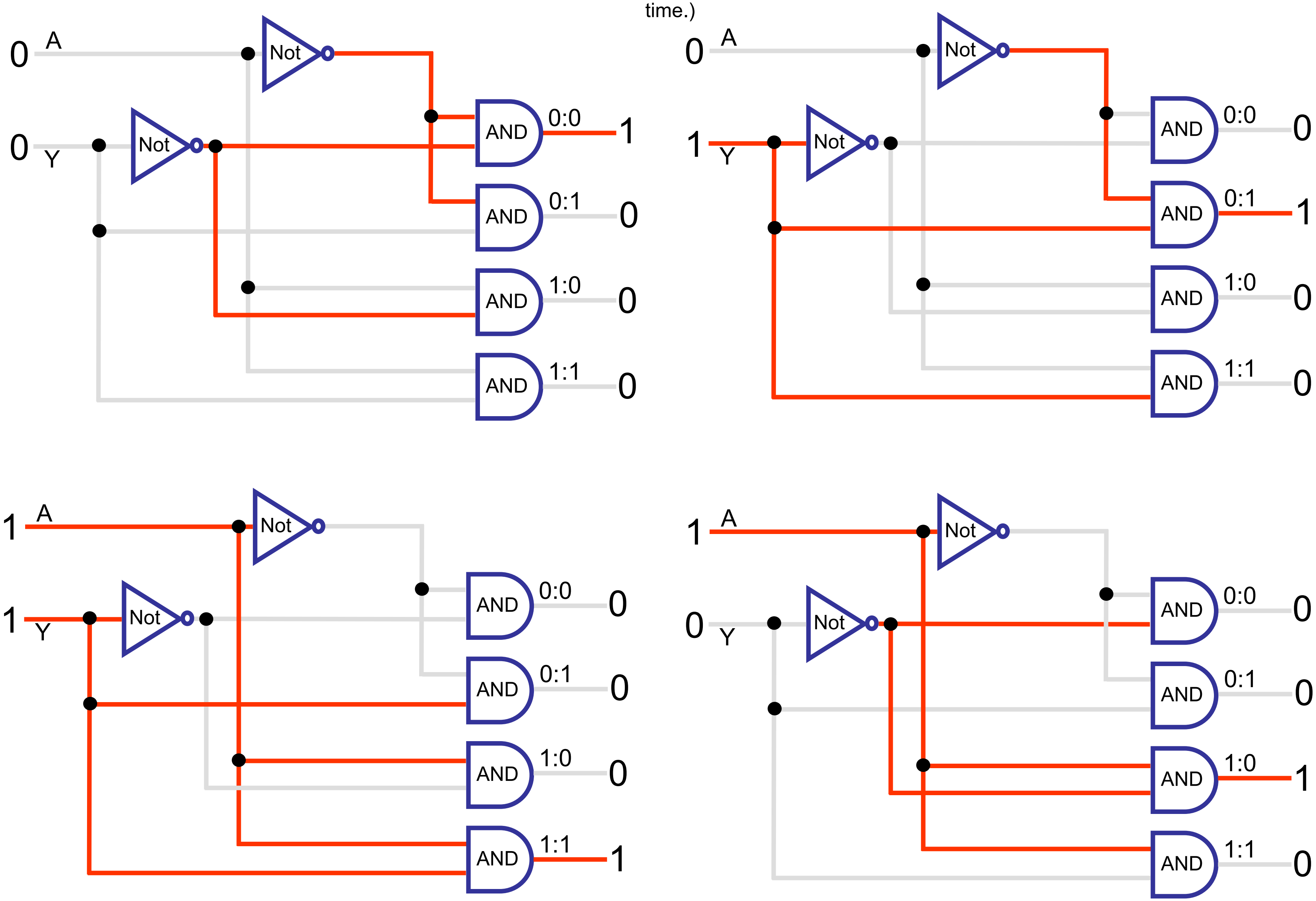
Connecting the 1-bit memory to 1-bit enabler allows for the bit to be saved in M when Set bit is on and released onto bus when Enable bit is on. The same thing is true when grouping 8 bits together to form 1-byte Register. The size of the Register is dependent upon the CPU bit size. The scott CPU is an 8-bit CPU, so all of its Registers are 8-bits.


1 bit of memory + 1 bit Enabler
(allows the bit saved in memory out whenever enable is on)
Input
Set
Enable
M
Out
A
Y
AND
Out



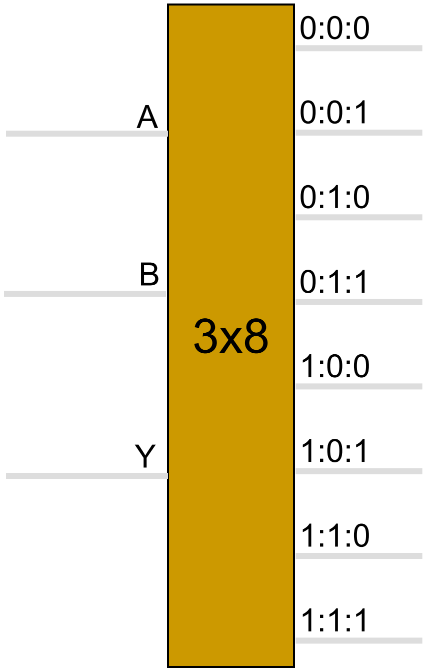
The Decoder

(2x4 Decoder shown: 2 inputs implies 4 possible combinations or outputs, but only one will output will be on at a time.)



More Decoder

(they have any size: 2x4, 3x8, 4x16, 5x32, 6x64, 7x128, 8x256, …, nx2^n)



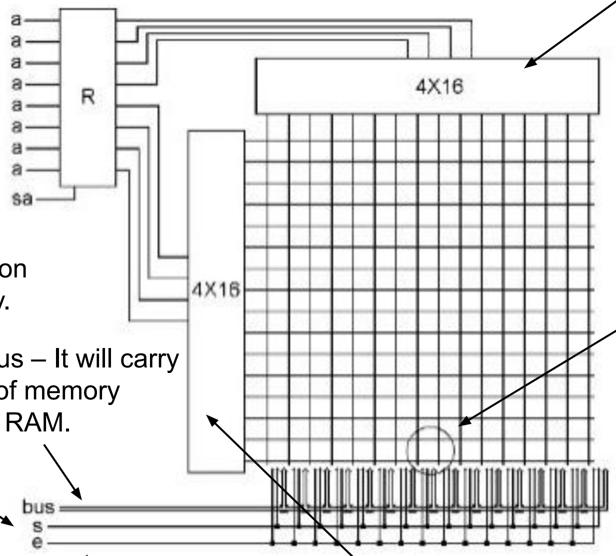
A	B	Y	0:0:0	0:0:1	0:1:0	0:1:1	1:0:0	1:0:1	1:1:0	1:1:1
0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Addressable Memory

(Data remains as long as power is on)

Memory Address Register – Pass in 8 bits and it will map to one of the 256 boxes of memory (1-byte). 8-bit register implies we can only address 256 Bytes of RAM (2^8).

4x16 decoder – allows us to select the column.



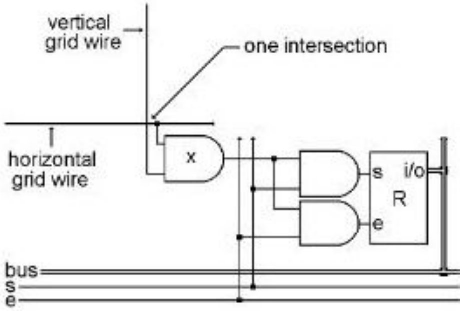
The circle is around one Byte of memory and it looks like this underneath. It too is must from simple logic gates.

Set bit – It needs to be on to set a Byte of memory.

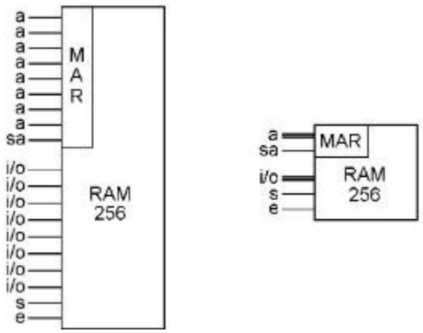
Data Bus – It will carry a byte of memory to/from RAM.

Enable bit – It needs to be on to retrieve a byte from memory.

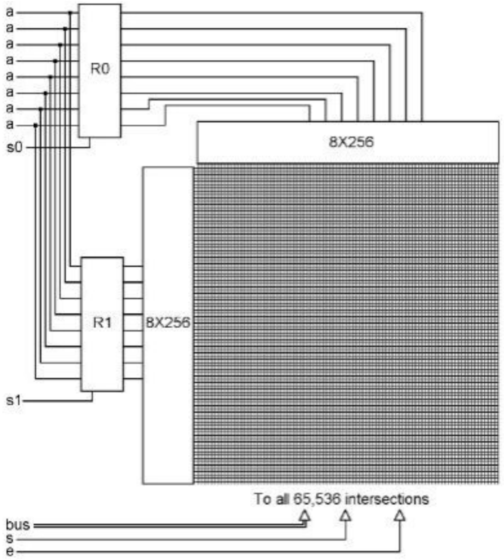
4x16 decoder – allows us to select the row.



New Symbols For 256 Byte RAM



Sample of using 2 8-bit registers and 2 8x256 decoders to address 64k of memory.



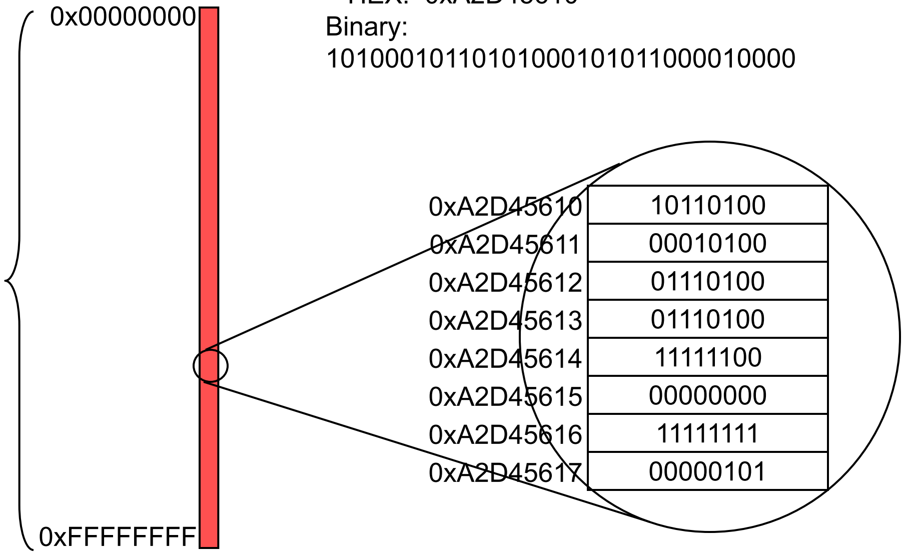
Why is it called Random Access Memory (RAM)?

- Data for a computer used to be stored on Tape Drives. Using this type of memory meant that data (or programs) had to be written in a contiguous/consecutive/sequential fashion. This means that every piece of data was written one right after the other without skipping around. Likewise, data had to be read from this type of memory in the same order it was written. The type of data is permanent, but very slow.

- Modern internal computer data storage allows data to be stored or retrieved in any order simply by supplying the memory address for the location of the data. This type of memory is much faster than the Tape Drive but is not permanent. Because you can jump around in memory, this means you can randomly access it. Hence, it is called Random Access Memory or RAM for short.

HEX: 0xA2D45610
Binary:
10100010110101000101011000010000

Total Addressable memory for 32-bit CPU. The first address is 0x00000000 in Hex and the last address is 0xFFFFFFFF. Each Address points to a Byte and it behaves similary to how our home addresses work. 2^{32} = 4294967296 total addressable bytes (i.e. 4 GB).

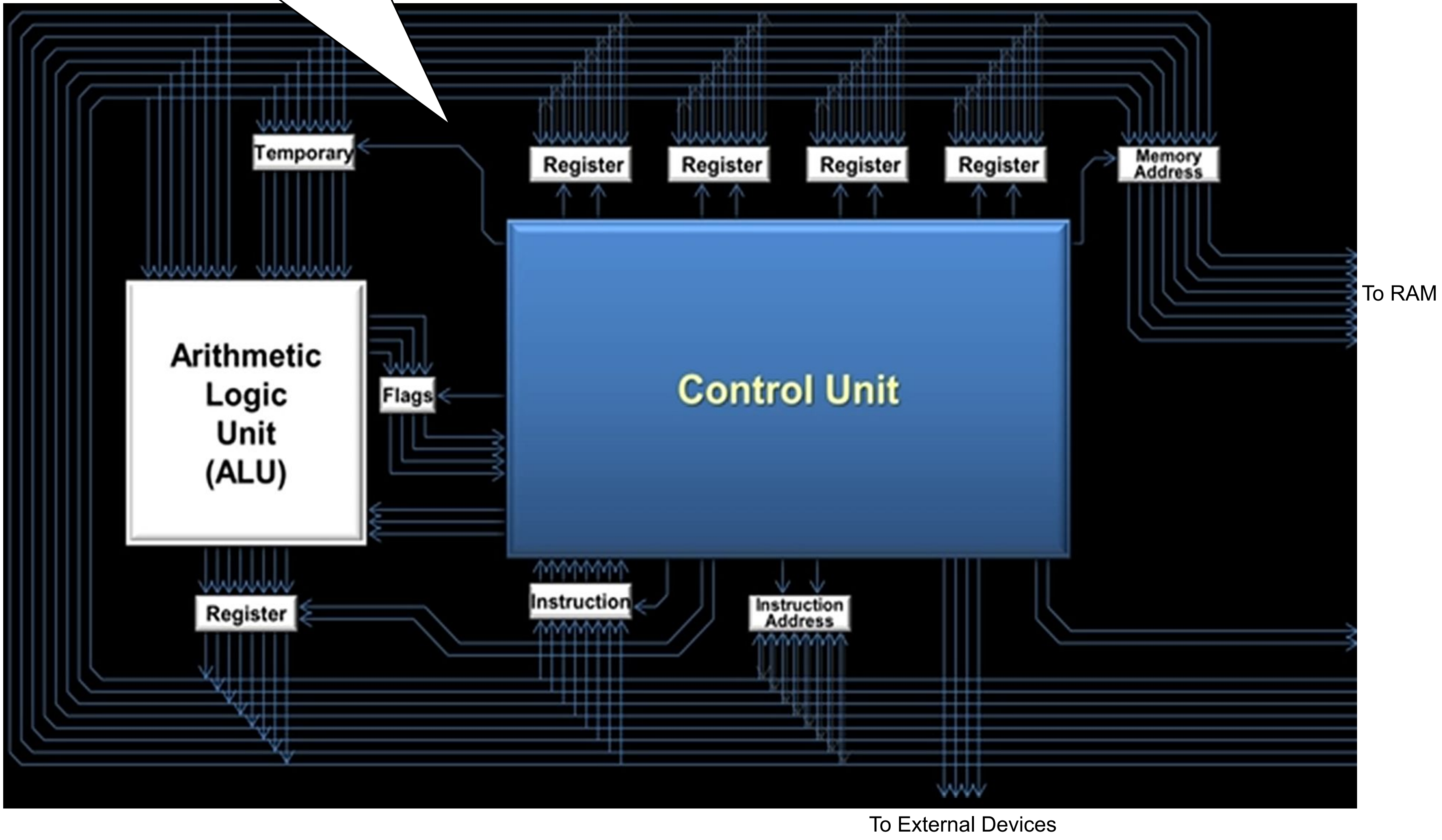


0xA2D45610	10110100
0xA2D45611	00010100
0xA2D45612	01110100
0xA2D45613	01110100
0xA2D45614	11111100
0xA2D45615	00000000
0xA2D45616	11111111
0xA2D45617	00000101

Part Two: CPU

Central Processing Unit (CPU)

I am the brains of this entire operation? Ehhem..or so they say.



Anytime the CPU needs to load/store data to RAM or an external device, it takes much, much longer than sending it to an internal cache or to a register.

Inside the CPU

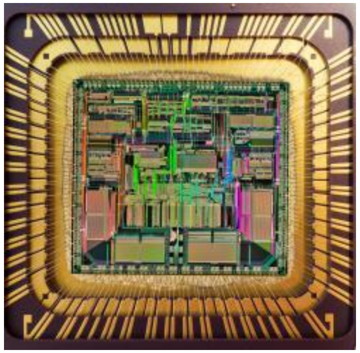
- A System Bus for sending / receiving bytes of data to/from ALU, Registers, IAR, external MAR, ...

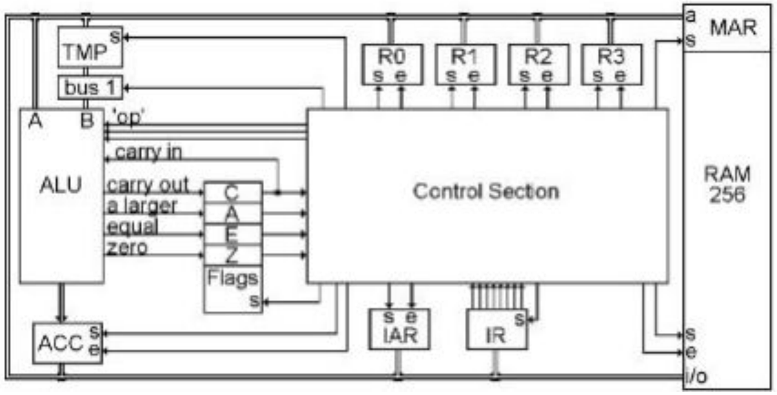
- Registers for temporary storage of Bytes.

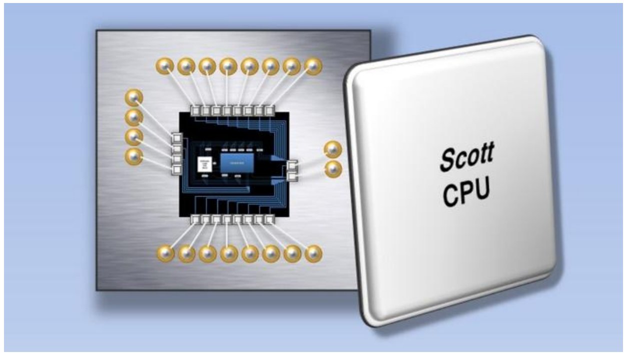
- The Arithmetic Logic Unit – the include it's own registers for flags, arithmetic and temporary storage
  - The ACC and TMP registers are associated with it

- A Control Unit, which contains:
  - The stepper – To ensure a program instruction gets executed in steps. It is in sync with the clock.
  - The accumulator (i.e. bus 1) - which adds the Byte representing the number one to the bus. This will be added to an address to get the next address of a program.
  - Sets and Enable wires that are connected to
    - Several CPU Registers that are used for temporary storage
    - The ALU
    - The Memory Address Register (MAR) of external RAM
    - The Instruction Address Register (IAR)
    - The set bit for the Instruction Register – holds the currently executing instruction
    - The set bit for various flags

RAM is not in the CPU.

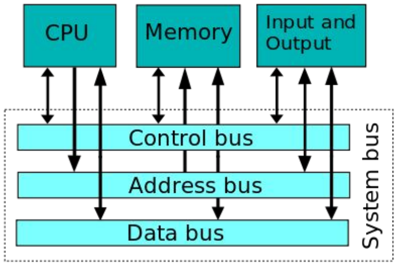






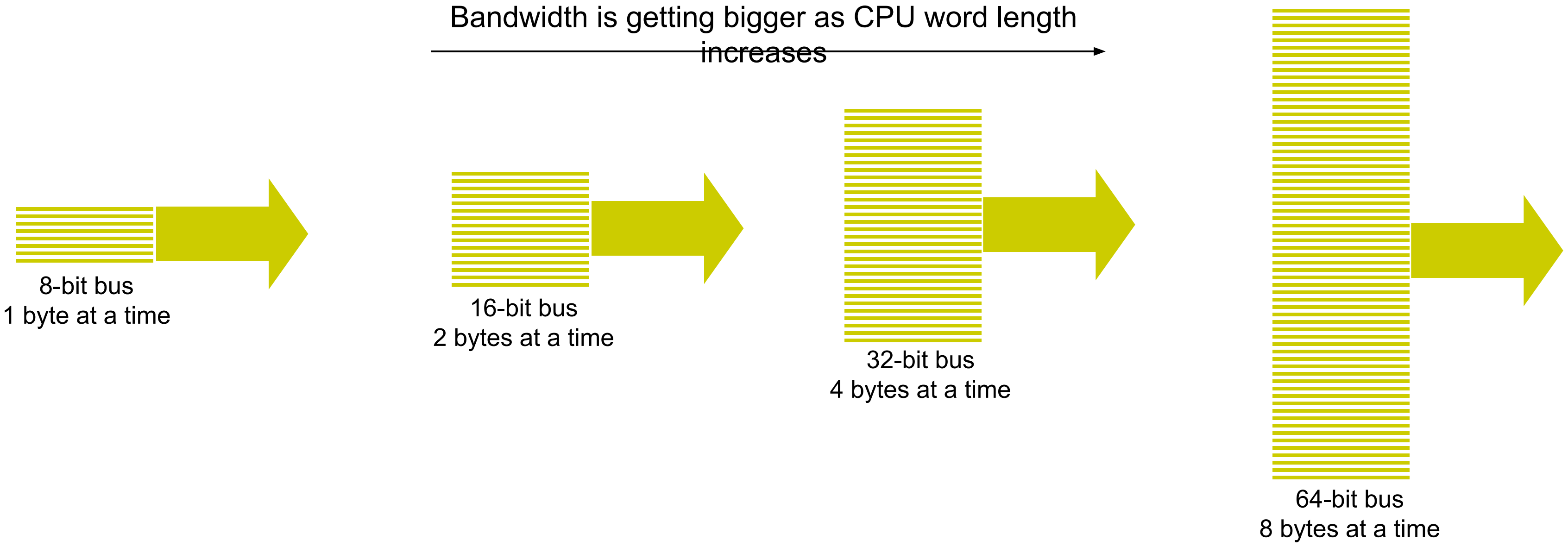
Part Two (A): The System Bus

System Bus



- The System Bus connects the CPU to other components in the computer with a group of wires. It allows for signals to move in groups of bytes, depending on the bandwidth of the bus. The word length (i.e. number of bits) generally determines the bandwidth of the bus.

- It is usually separated into three different buses so that signals won't collide:
  - **Address Bus**: This bus will be used for addressing memory or I/O devices. It is unidirectional. The bandwidth of this bus determines how much memory can be addressed (8-bit implies 256 Bytes, 16-bit implies 64 Kilobytes, 32-bit implies 4 Gigabytes, 64-bit implies 16 Exabytes).
  - **Data Bus**: Any data to be loaded / stored into memory or elsewhere will move on this bus. It is bidirectional.
  - **Control Bus**: (also called the timing bus) – This is the bus where the set and enable signals are sent (timing signals). It allows for the data to be set into memory or leave memory and to be placed onto the data bus.



Bandwidth is getting bigger as CPU word length increases

8-bit bus
1 byte at a time

16-bit bus
2 bytes at a time

32-bit bus
4 bytes at a time

64-bit bus
8 bytes at a time

Part Two (B): Registers

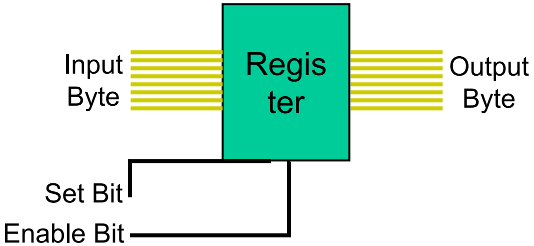
CPU Registers

- Recall that registers are generally 1-word of memory. In the Scott CPU, the word size is 8-bits (Byte); thus a register is 1 Byte.

- To a CPU, temporary storage (i.e. RAM) is far away, and permanent storage (i.e. Hard Drive) is farther still. Let's compare this to the following scenario. Let's assume you have three friends, one that lives next door to you, one that lives 10 blocks down the road, and one that lives in the next state (say 50 miles away). Your friend that is the neighbor is similar to a Register. The one that lives 10 blocks away is similar to RAM. And the one that lives in the next state is similar to the Hard Drive (or any external storage). As you can see, if the CPU wants to load/store any type of data, it would be fastest for it to use the memory that is inside of it. Going to the Hard Drive is very, very slow because the Hard Drive is a mechanical device that spins in order to find its data. In other words, data loading/storing no longer goes near the speed of light.

- Most modern CPU's have between 16 and 64 General Purpose Registers. In the Scott CPU, the Registers it can use are:
    - For temporary storage: R0, R1, R2, and R3.
    - For ALU Input/Output: TMP, ACC
    - For Condition Codes: Flags Register (also used in conjunction with the ALU)
    - For Addressing Memory: MAR
    - For executing programs: IAR and IR

- The contents of a Register are:
    - Set when the Set bit is on with data from the bus.
    - Released to the bus when the Enable bit is on.
    - Lost when power is turned off.

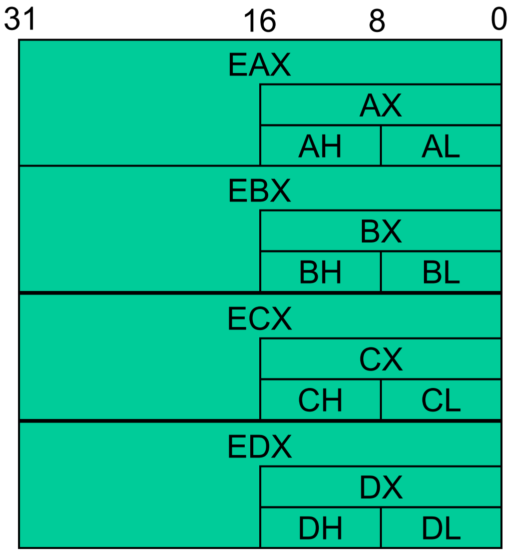


Registers
R0, R1, R2, R3

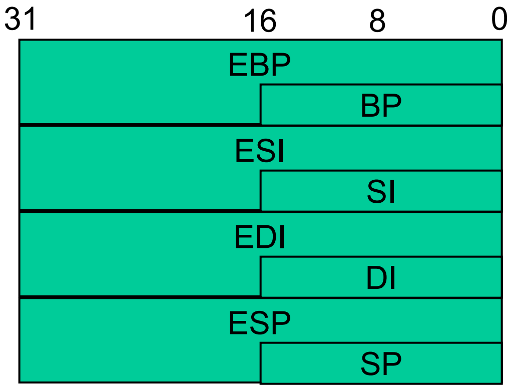
- Registers R0-R1 in the Scott CPU are available to store any transient data required by a program. For example, for adding two numbers:
    - use an instruction that will take a number from RAM and put it into R0
    - use an instruction that will take a number from RAM and put it into R1
    - use an instruction that will add the contents of R0 to R1

- Since Registers are part of the CPU, it is faster to temporarily store data in these than it is to store them in RAM (or even worse on Disk).

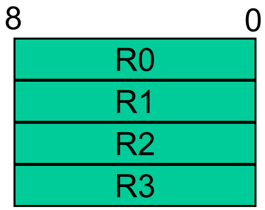
X86-32 Bit Type 1 Registers



X86-32 Bit Type 2 Registers

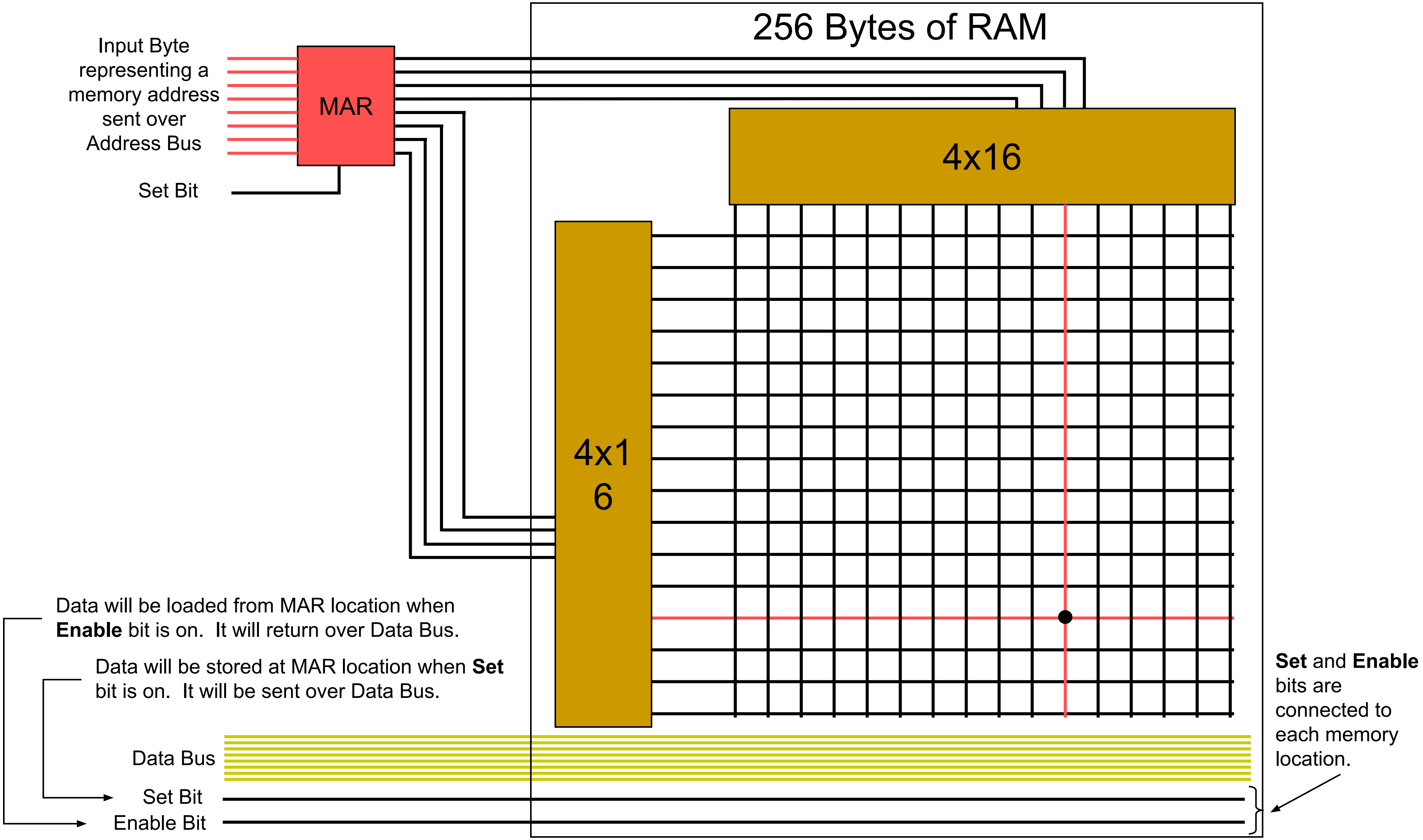


Scott CPU



The Memory Address Register (MAR)

- Use to address a particular word in memory. When you put a number in the MAR using the address bus by turning the set bit on, it selects the word of data in memory according to the number in the MAR.



256 Bytes of RAM
Input Byte representing a memory address sent over Address Bus
MAR
Set Bit
4x16
4x1
6
Data Bus
Set Bit
Enable Bit

Data will be loaded from MAR location when **Enable** bit is on. It will return over Data Bus.

Data will be stored at MAR location when **Set** bit is on. It will be sent over Data Bus.

Set and **Enable** bits are connected to each memory location.

The Instruction Register (IR)

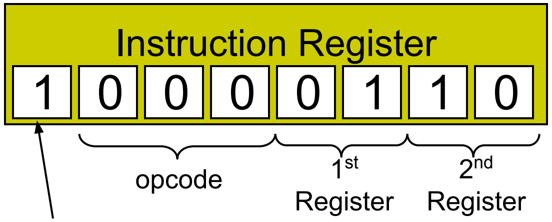
- Register that stores the instruction that is currently being executed (or decoded). That is, the bits from this Register "instruct" the CPU in what to do.

- Some instructions have to be decoded before they can be executed. That means to break the instruction into groups of bits, such as the first 4 bits representing the opcode and the last four bits representing Register locations.

- Each instruction to be executed is loaded into this Register, decoded, and finally broken into steps (see Stepper) for execution.

- The bit pattern for the instruction is the LOWEST form of instruction the CPU is familiar with. This is called the Machine Language or CPU Instruction Set. In the beginning most programmers actually coded using Machine Code.

The entire instruction bit pattern is loaded into the IR and the bits are decoded to get the opcode, registers to use, etc.



This Bit indicates whether or not its an ALU instruction.
1 = Yes, 2 = No.

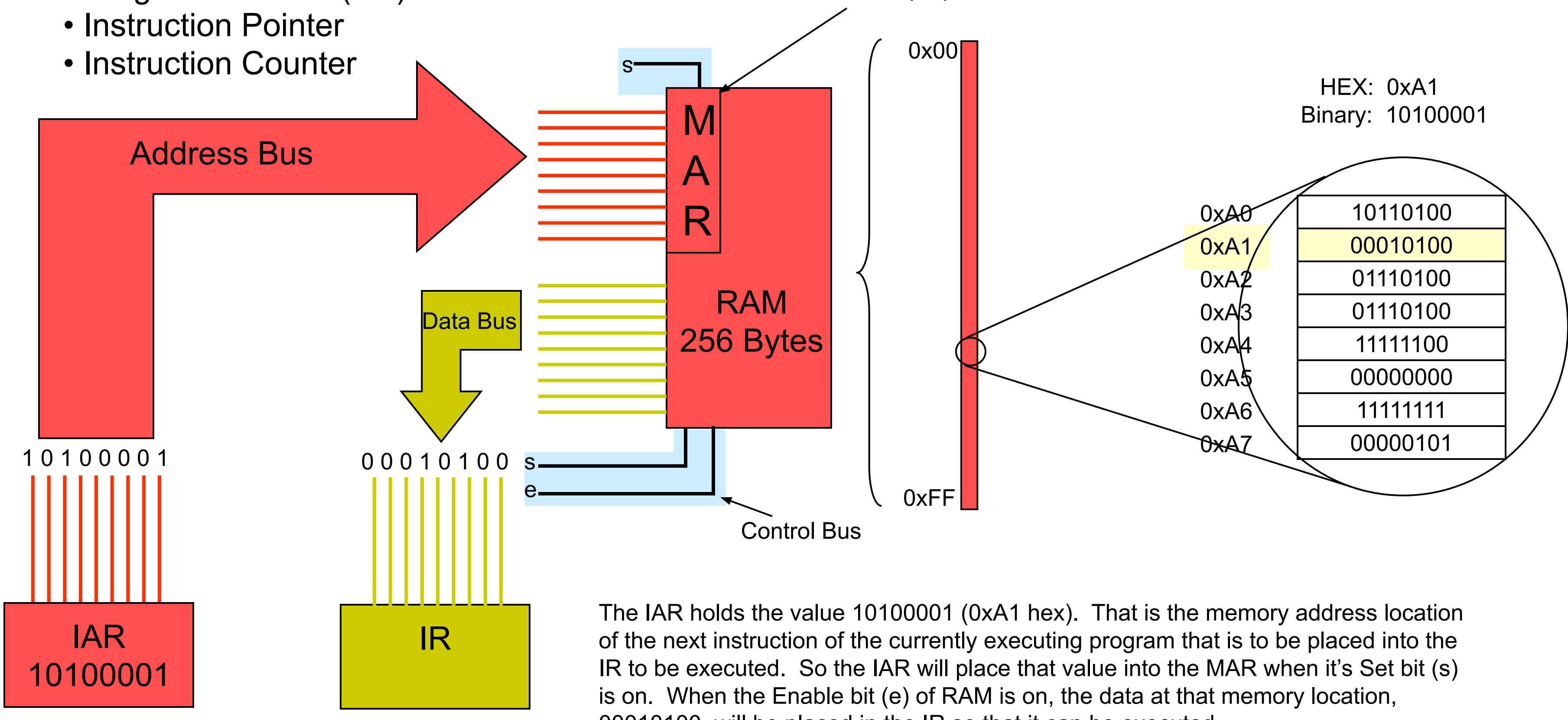
The Instruction Address Register (IAR)

- The purpose of the IAR is to hold the memory address location of the next executable instruction in the currently executing program. In essence, this means it points to a place in memory that holds the next instruction to be executed.

- Instructions for a program are usually fetched sequentially in memory, but certain instructions, such as jump/branch, could transfer control to a program instruction somewhere else in memory.

- The IAR is also known as the following:
  - Program Counter (PC)
  - Instruction Pointer
  - Instruction Counter

The MAR is a Register inside the CPU. Just drawn like this for demonstration purposes.



HEX: 0xA1
Binary: 10100001

0xA0	10110100
0xA1	00010100
0xA2	01110100
0xA3	01110100
0xA4	11111100
0xA5	00000000
0xA6	11111111
0xA7	00000101

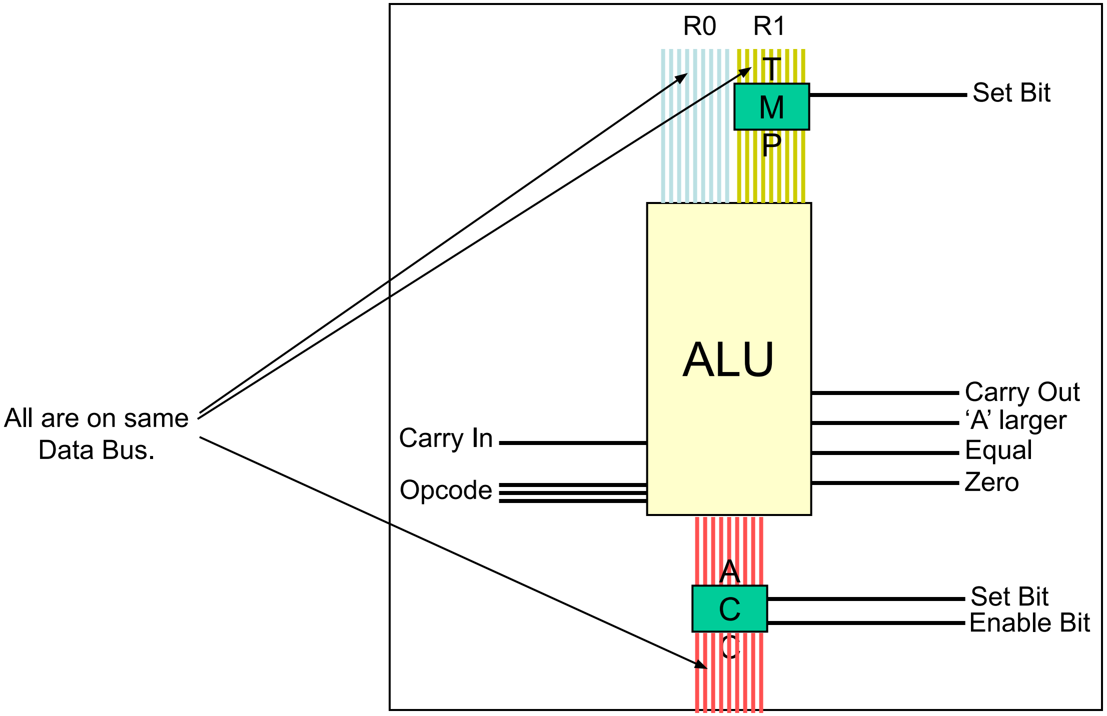
The IAR holds the value 10100001 (0xA1 hex). That is the memory address location of the next instruction of the currently executing program that is to be placed into the IR to be executed. So the IAR will place that value into the MAR when it's Set bit (s) is on. When the Enable bit (e) of RAM is on, the data at that memory location, 00010100, will be placed in the IR so that it can be executed.

The TMP and ACC Registers

- The TMP Register is used by the ALU when it needs to operate on two bytes. This is because only one byte can be on the data bus at a time. Thus, one byte is moved from a register to the TMP register before the opcode is passed to the ALU. The 2nd byte will be on the data bus and the ALU will perform its action on the value in the TMP register and the value from the Register that has its value enabled on the Data Bus. TMP only has a set bit.

- The ACC Register is used by the ALU to store its results. Because again, only one word of data can be on the data bus at one time.

- For example, if two numbers are to be added with the 1st in R0 and the 2nd in R1. The contents of R1 will be moved to TMP. The contents of R0 will be sent to the ALU along with the opcode and the two numbers will be added and the result will be placed into the ACC Register.

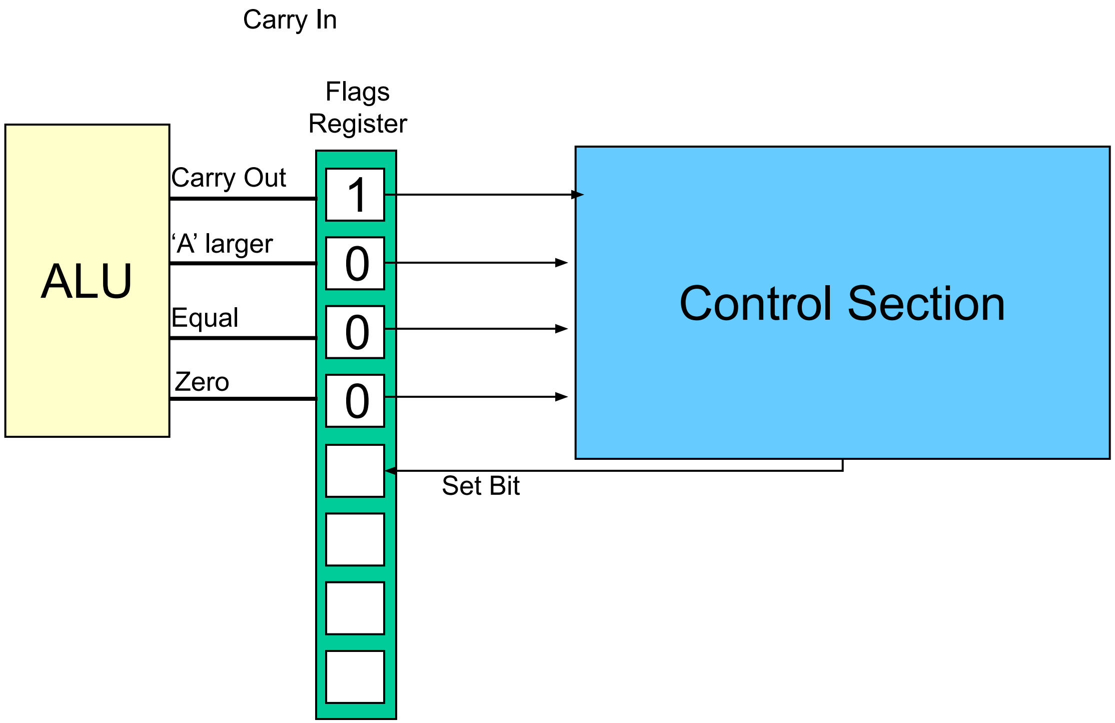


The Flags Register

- The Flags Register is connected to the ALU and will hold all the statuses that come out of it:
  - Carry Out bit
  - "A" Larger bit
  - Equal bit
  - Zero bit

- A Register is needed so that these values can be stored when they come out of the ALU and fetched at a time when the CPU needs them (possibly a different step of the executing instruction).

- In the Scott CPU, only part of the Register is used to store the bits; that is, we only need 4 bits to store the flags. Notice how the Carry In is connected to the Carry Out. The Set bit is used to clear the Register.

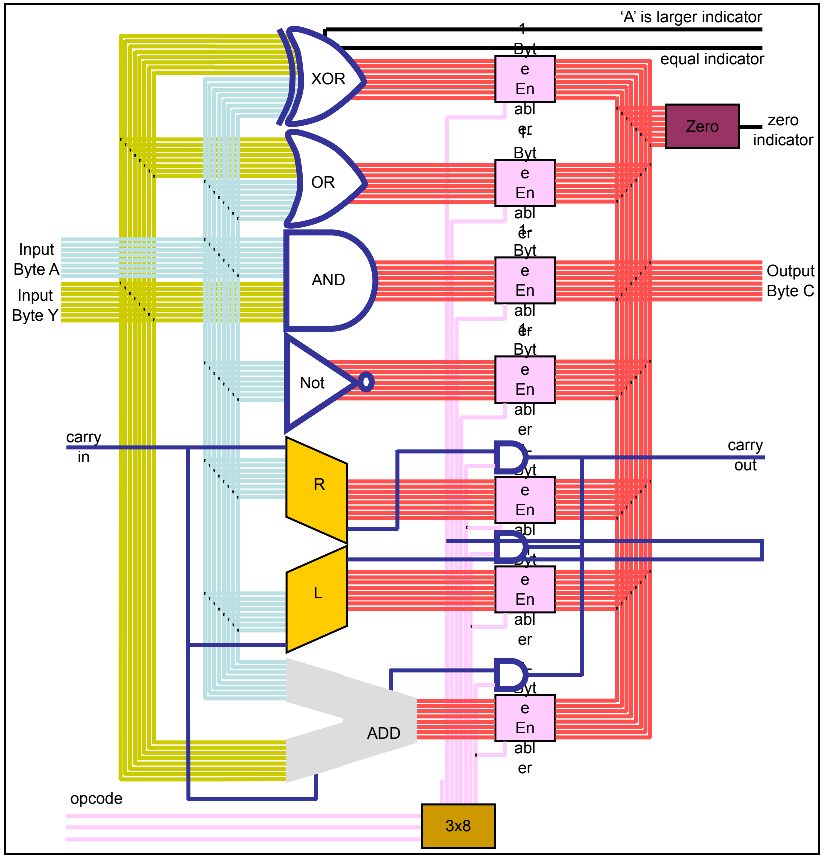


Part Two (C): The ALU

The Arithmetic and Logic Unit
(ALU)

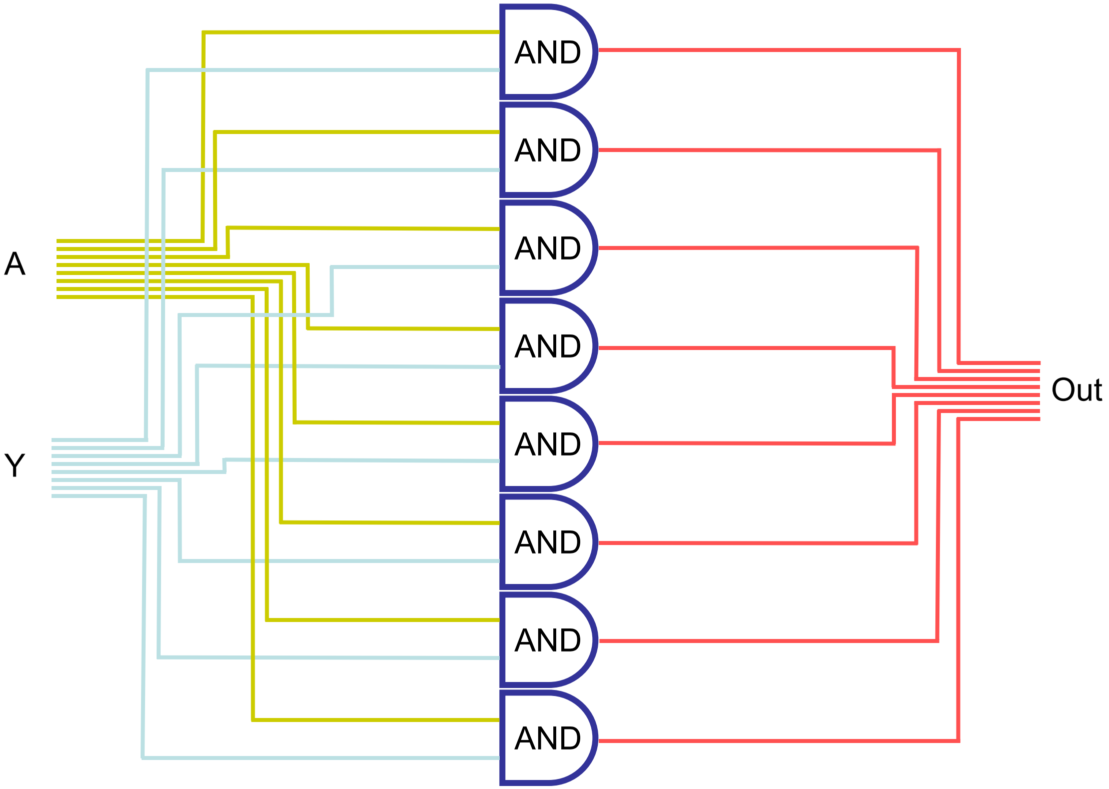
- The CPU will delegate all arithmetic, such as addition, logic, such as ANDING, and other computations to this unit.

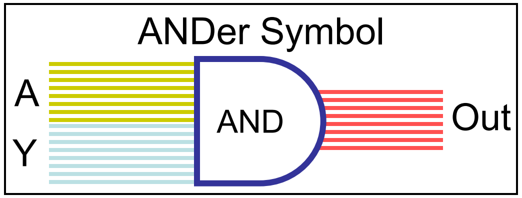
- The Scott CPU is an 8-bit computer with the following ALU operations:
  - Logic:
    - XOR one Byte with another Byte
    - OR one Byte with another Byte
    - AND one Byte with another Byte
    - NOT one Byte
  - Arithmetic
    - Add one Byte to another Byte
  - Bit Shifting:
    - Bit shift Left one Byte
    - Bit shift Right one Byte



The ANDer

- The ANDer takes two input bytes and ANDs them together, bit-by-bit, to produce a third byte.
- One use for doing this is to turn an ASCII letter code from lower case to upper case.



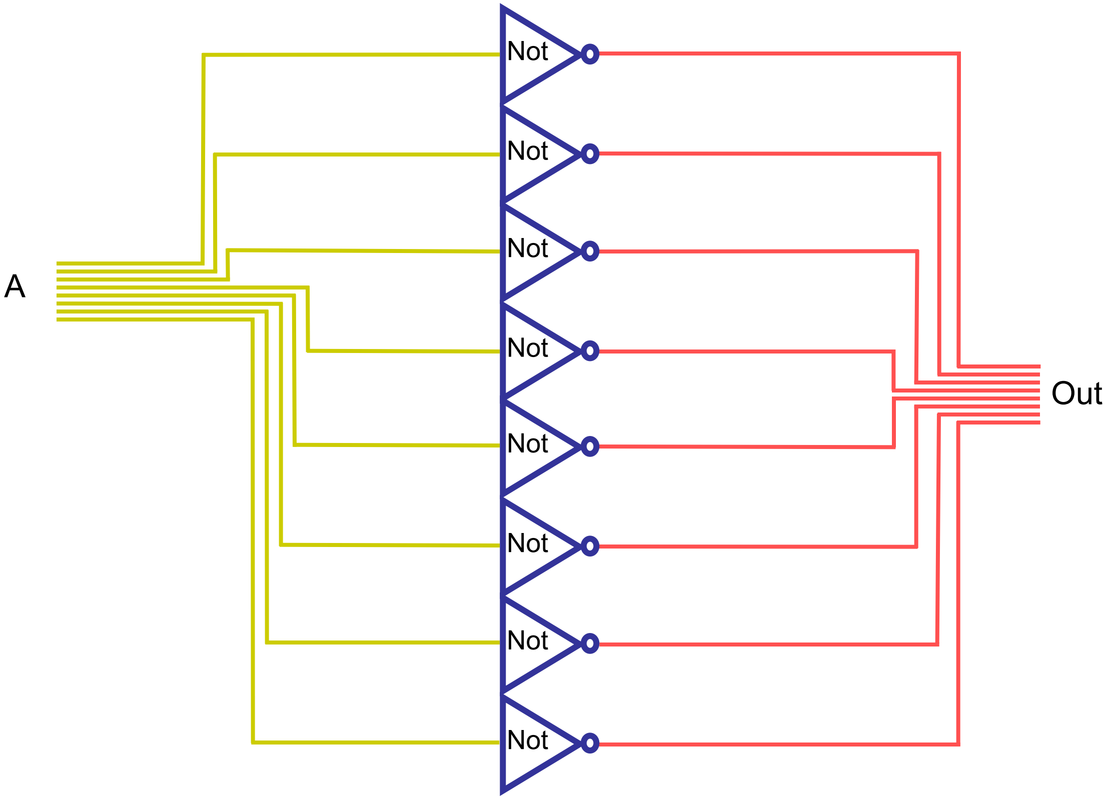


A   0 1 1 0 0 0 0 1   Represents lower case a in ASCII
Y   1 1 0 1 1 1 1 1   (the 3rd bit from left needs to be flipped)

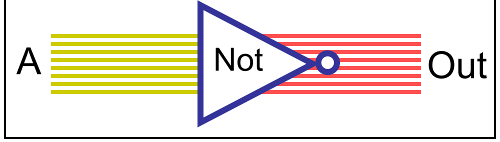
Out   0 1 0 0 0 0 0 1   Represents upper case A in ASCII

The NOTter

- The NOTter takes a byte and flips all of its bits.
- One use for this is to take a true value and make it false (not true) or vice versa. 11111111 □ 00000000.



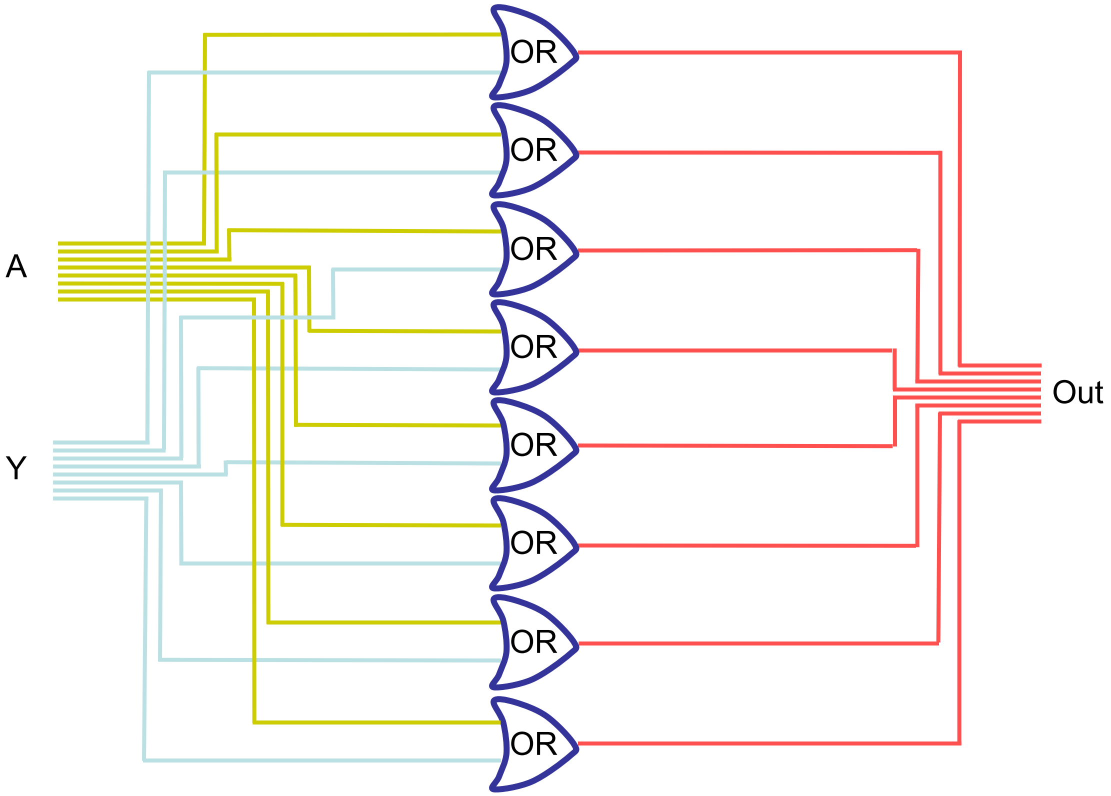
NOTter Symbol

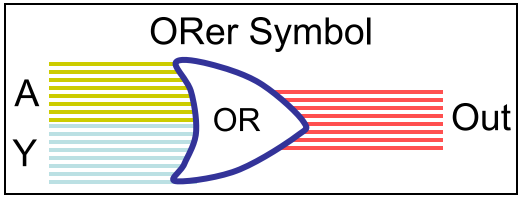


```
A      1 1 1 1   1 0 0 1
-------------------------------
Out    0 0 0 0   0 1 1 0
```

The ORer

- The ORer takes two input bytes and ORs them together, bit-by-bit, to produce a third byte.
- One use for doing this is to turn an ASCII letter code from upper case to lower case.



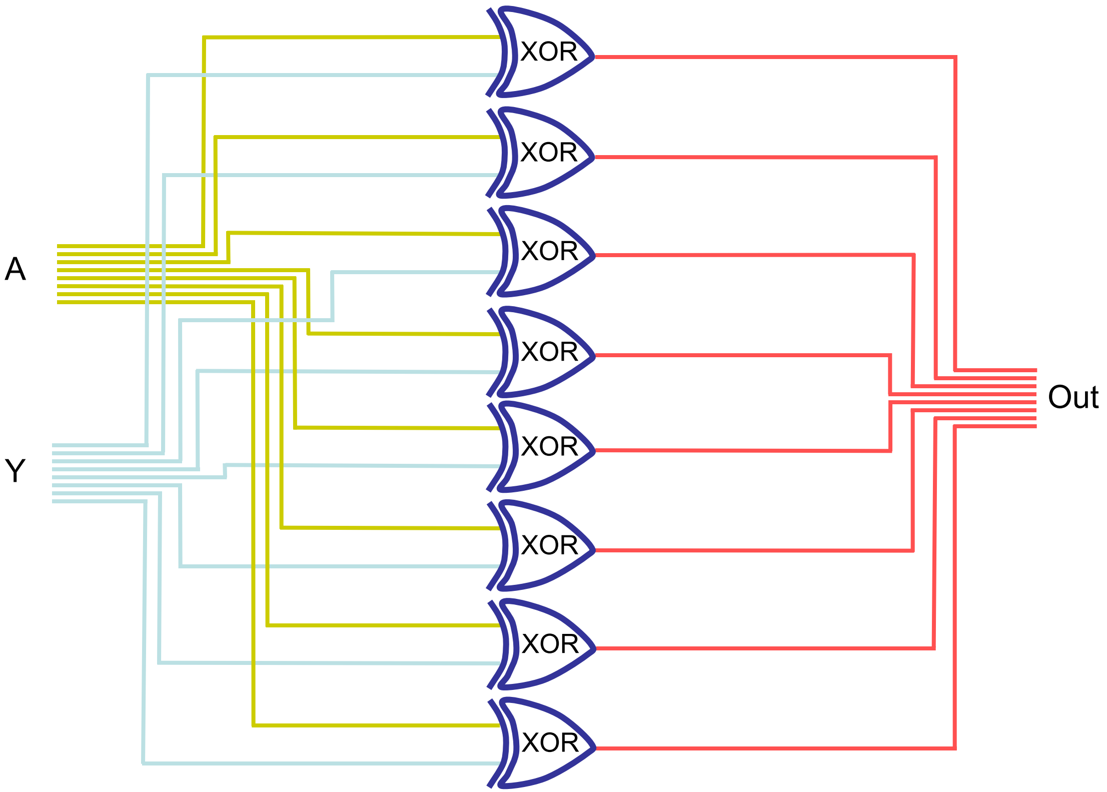


A	0 1 0 0 0 0 0 1	Represents upper case A in ASCII
Y	0 0 1 0 0 0 0 0	(the 3rd bit from left needs to be flipped)

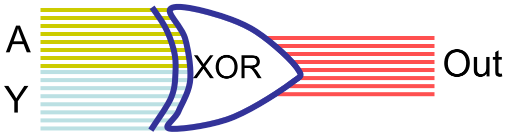
Out	0 1 1 0 0 0 0 1	Represents lower case a in ASCII

The Exclusive ORer

- The XORer takes two input bytes and XORs them together, bit-by-bit, to produce a third byte.
- One use for doing this is to compare if two values are equal. If they are equal, the 3rd byte will be 0.



XORer Symbol



```
A       1010  1001
Y       1010  1001
---------------------------------
Out     0000  0000
```

(Since output is zero we know they are equal)

The Left and Right Shifters

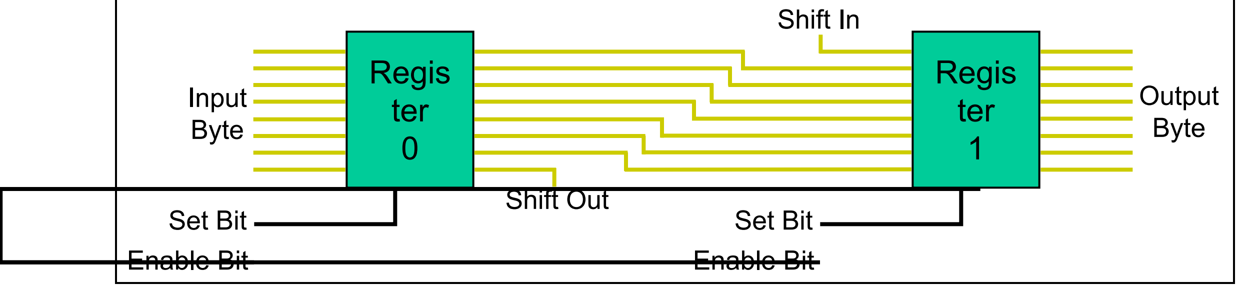
- The shifter doesn’t use any gates at all. It just wires up the bus so that the bits of one register will get shifted, left or right, into another register. The “Shift Out” is often connected to the “Shift In”, but it is hardware dependent. When they are connected, the bit that gets shifted out will move to the shifted in position.

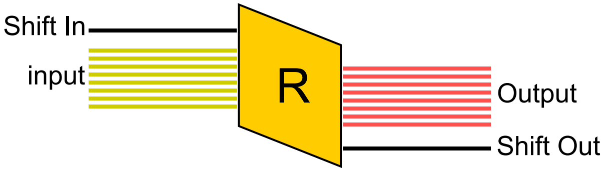
- One possibly use is to multiply or divide by 2. Right Shift is a divide and Left Shift is a multiply.

Right Shifter

When the Enable Bit is on in R0 and the Set bit is on in R1, the contents of R1 are copied into R2, but they are shifted over one position to the right.

1001 0111 will change to 0100 1011.

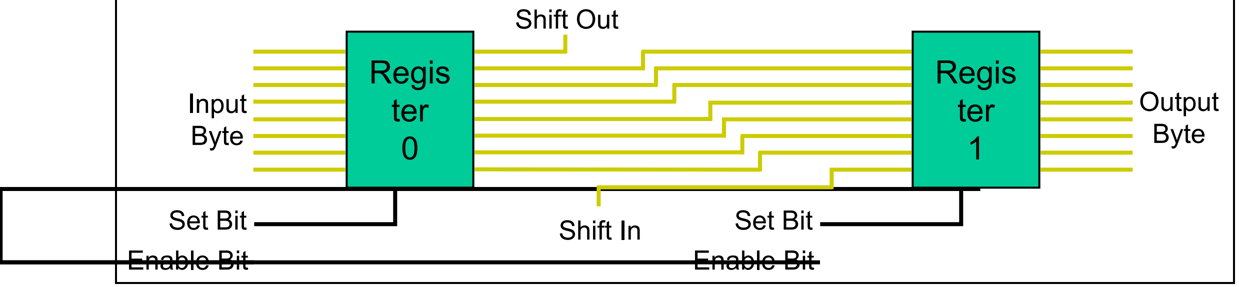


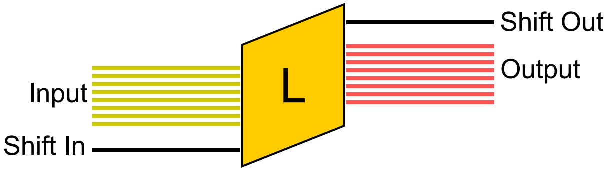


Left Shifter

When the Enable Bit is on in R0 and the Set bit is on in R1, the contents of R1 are copied into R2, but they are shifted over one position to the right.

1001 0111 will change to 0010 1110.





The Adder (Bit Input)

- Whenever we add two numbers together and their sum is greater than 10, we carry the 10 over. The same thing happens when we add to binary numbers where the sum is equal to two; a 1 is carried over.

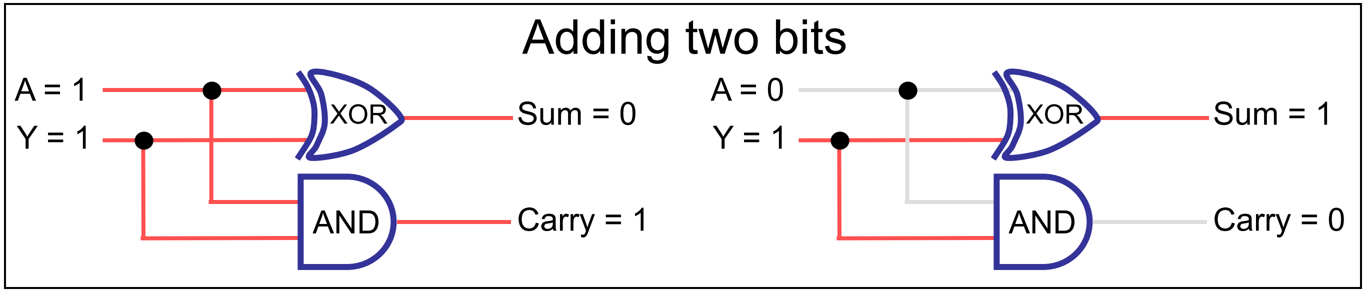
Base 10

No Carry	Carry
3	6
+ 4	+ 6
7	12

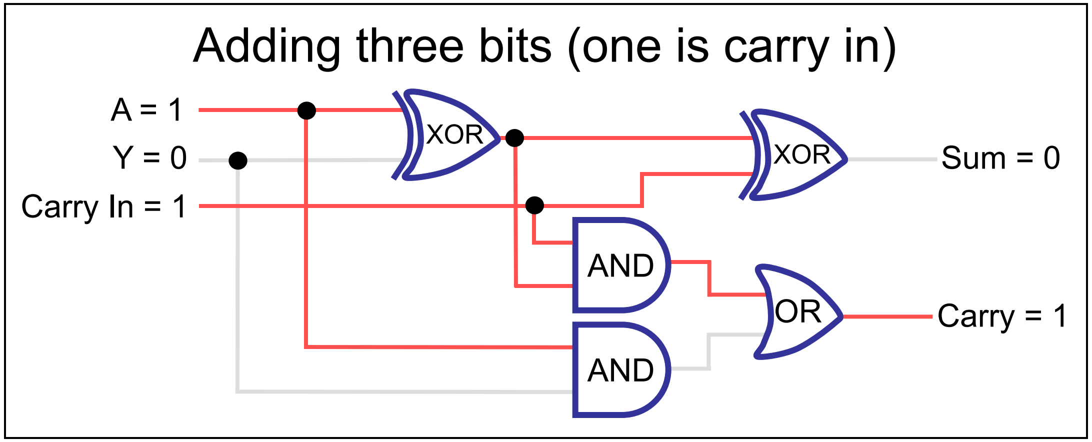
Base 2

No Carry	Carry
1	1
+ 0	+ 1
1	10

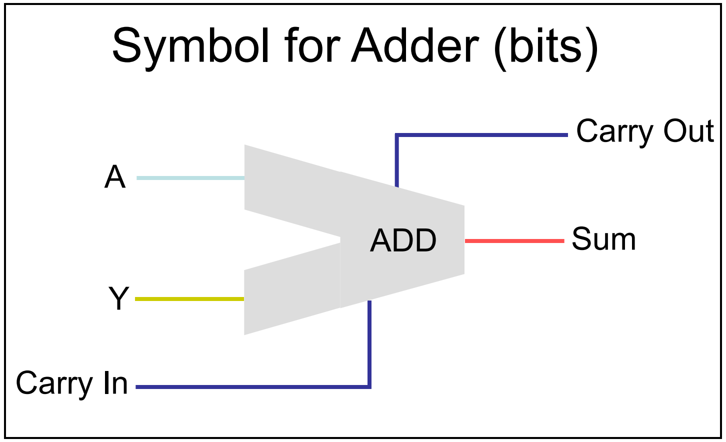
Adding two bits



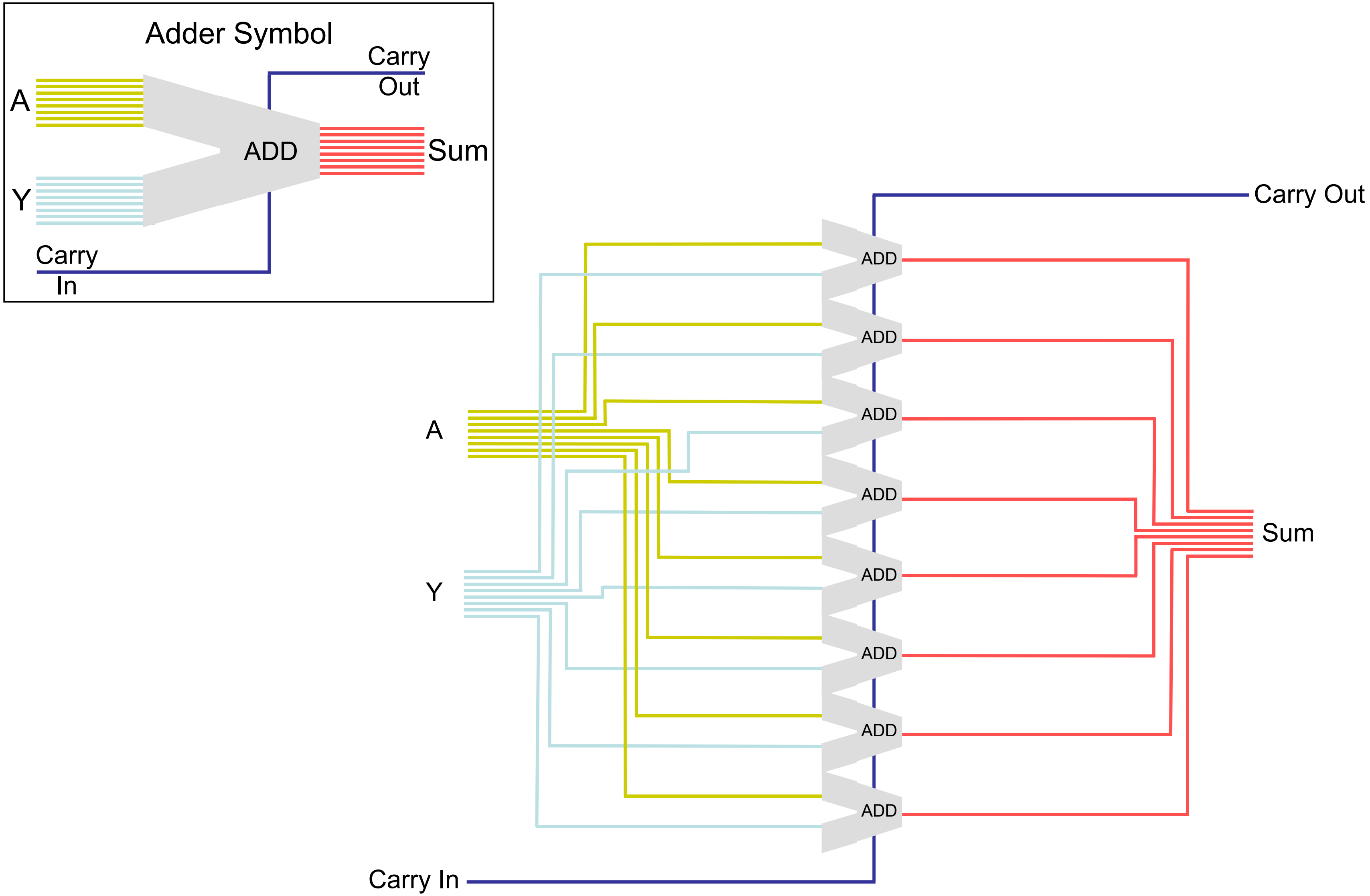
Adding three bits (one is carry in)



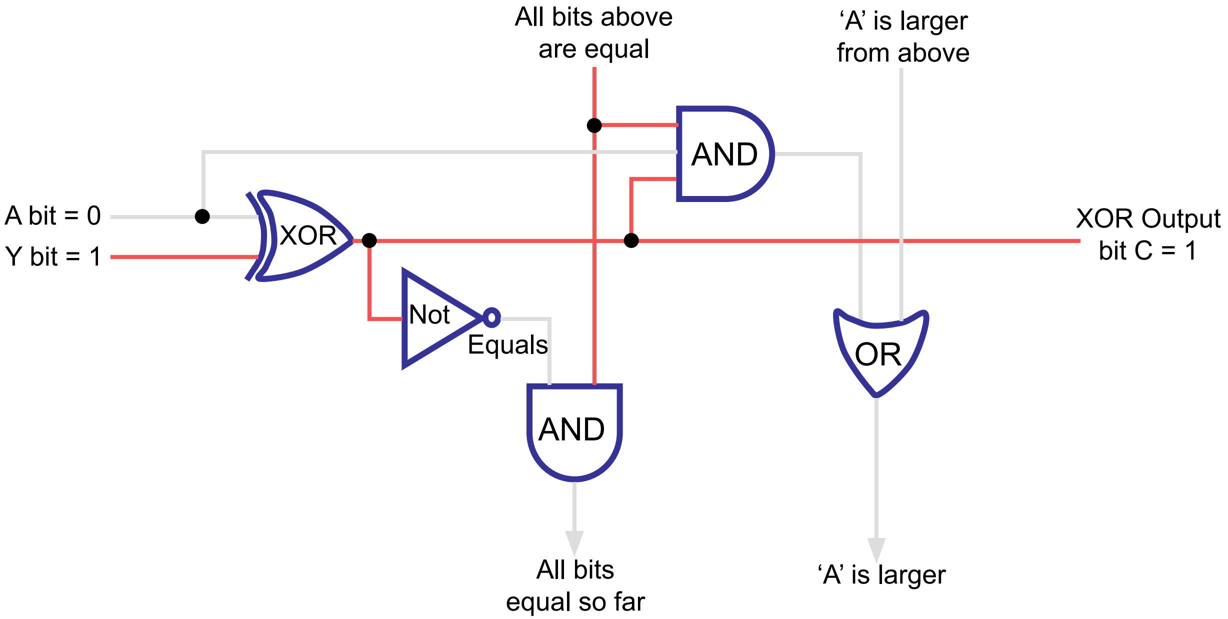
Symbol for Adder (bits)



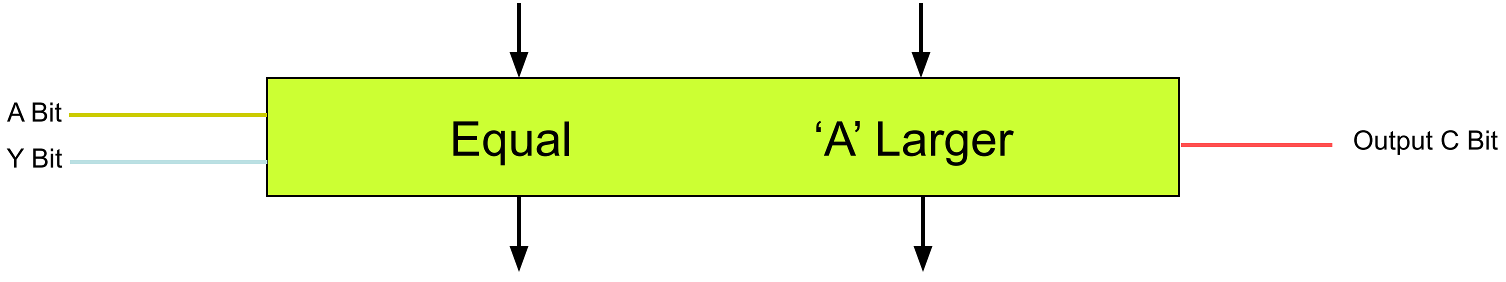
The Adder (Byte Input)



The Comparator (Bit input)



Symbol for Comparator (two bits)

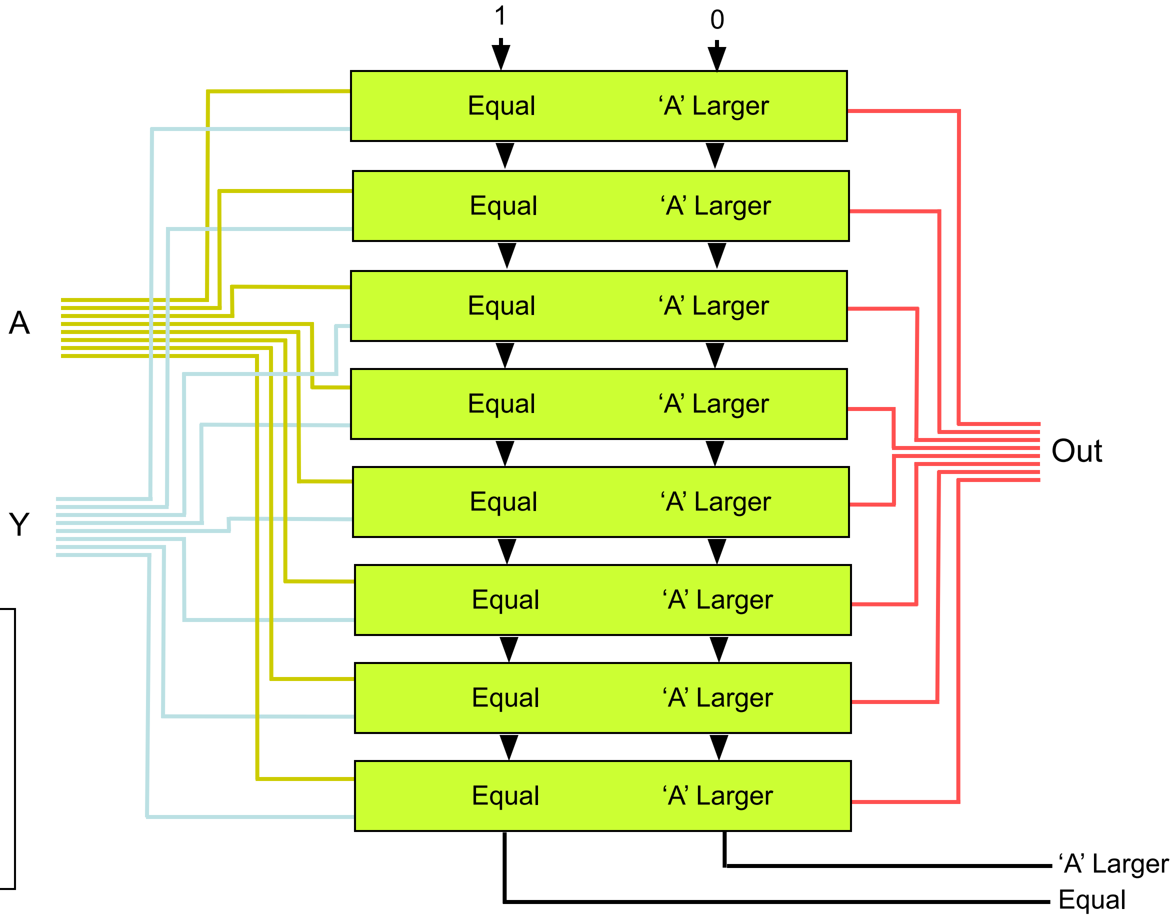


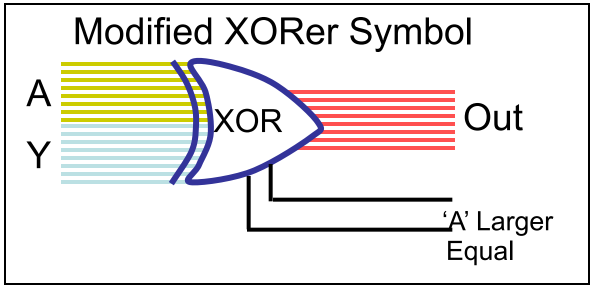
The Comparator (Byte input)

- The comparator is built into the XOR gate in the ALU because it makes use of the gates that are already there. It generates two bits for output, "'A' Larger" and "Equals".

- The purpose of it is to find out whether two bytes are exactly equal to each other. If they aren't, it will find out whether Byte A is larger than Byte B.

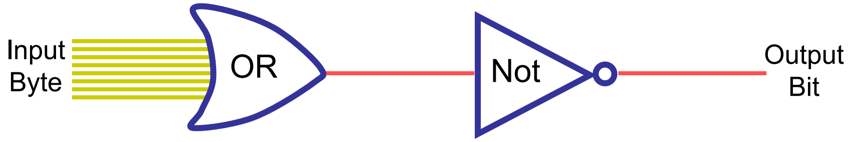
- If the output of an XOR gate is zero (i.e. 00000000), then the inputs are equal. Determining the larger Byte is a little harder.

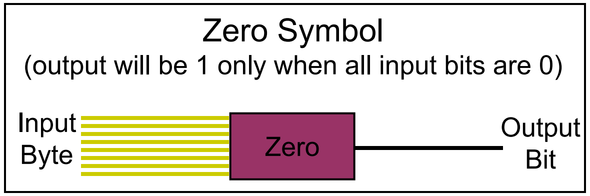




The Zero indicator

- Takes a byte as input and generates only a single bit output. The output turns on when all the bits in the input are off (i.e. 00000000). Thus, it indicates whether or not the input is the zero byte.

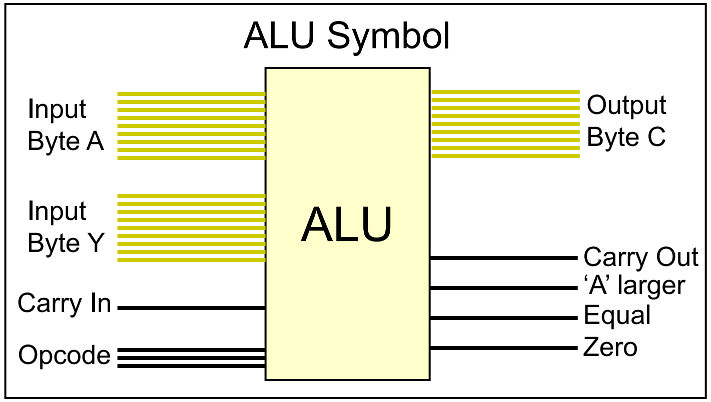



Zero Symbol
(output will be 1 only when all input bits are 0)

Scott CPU - ALU OP Codes

- Since three bits are used as the input (called “op” in our symbol), there are 2^3 or 8 possible codes that we can pass in that tell the ALU what action to take. These codes are known as operational codes or opcodes. When the ALU receives a particular bit-pattern (see table below), it will perform the corresponding action.

- For example, if we pass in 01101111 into **A** and 11000100 into **Y** and 100 into **opcode**, then an AND will be performed and **output C** will be 01000100.

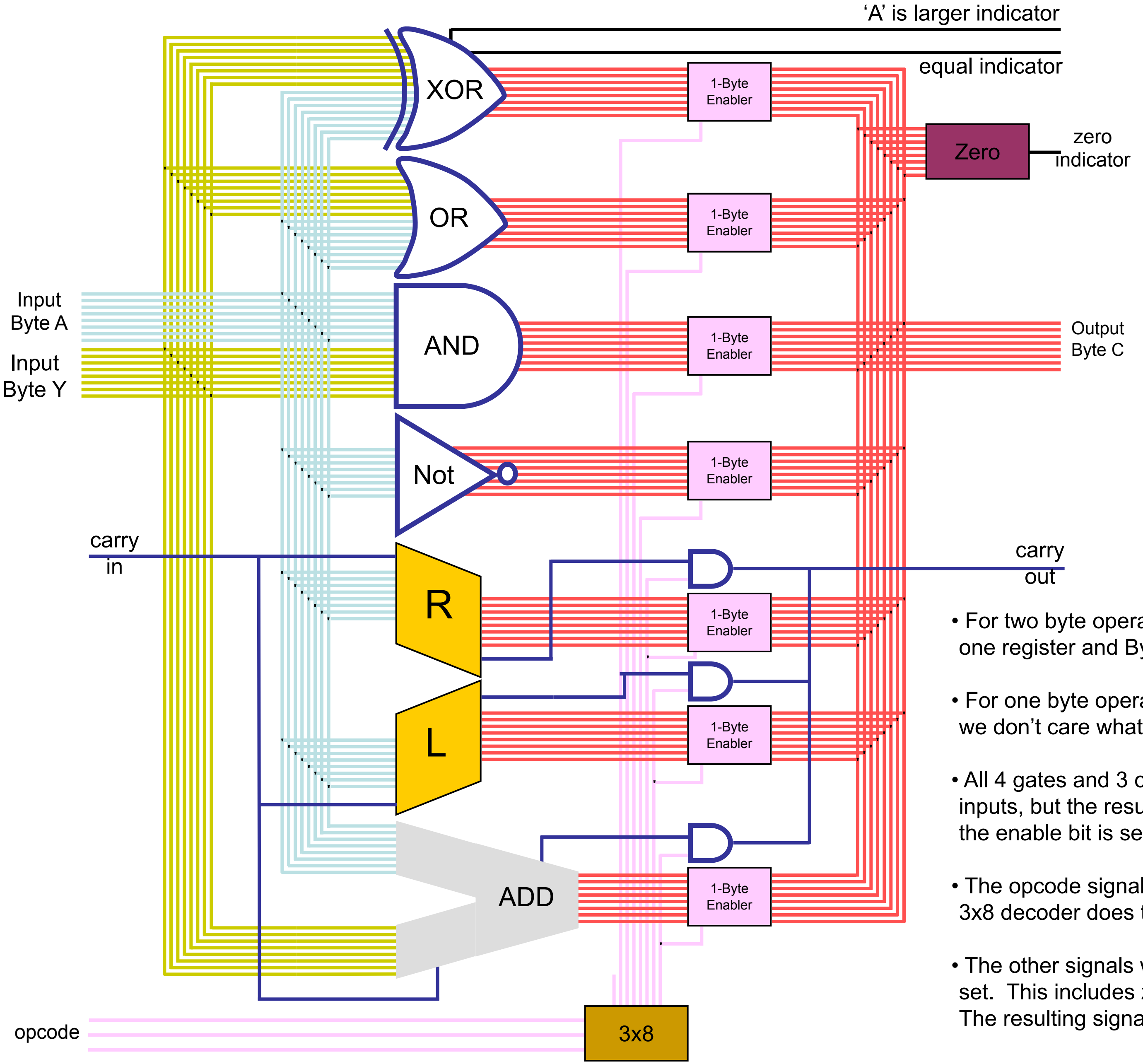


ALU OPCODE	ACTION NAME	Description
000	ADD	Add 2 bytes
001	SHR	Shift Right on 1 byte
010	SHL	Shift Left on 1 byte
011	NOT	NOT a byte
100	AND	AND 2 bytes
101	OR	OR 2 bytes
110	XOR	XOR 2 bytes
111	CMP	Compare 2 bytes

Definitions of input in the ALU Symbol

A Input for A byte
Y Input for Y byte
carry in The carry in bit
opcode The three opcode bits
C The output result
carry out The carry out bit
‘A’ larger The bit that indicates byte A was larger than byte Y
equal The bit that indicates A and Y were equal
zero The indicating that the last operation resulted in a zero

The Scott CPU - ALU



- For two byte operations, Byte A is typically connected to one register and Byte Y another.

- For one byte operations, Byte A will be the active signal, we don't care what Byte Y is.

- All 4 gates and 3 components will get the signal from all inputs, but the result of them will not be outputted unless the enable bit is set for that gate/component.

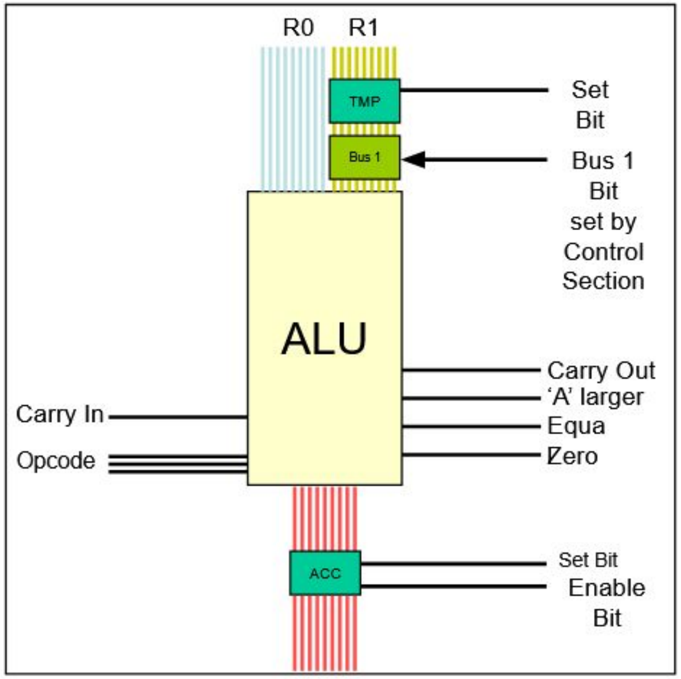
- The opcode signal determines which enable bit is set. The 3x8 decoder does the work of selecting a single enabler.

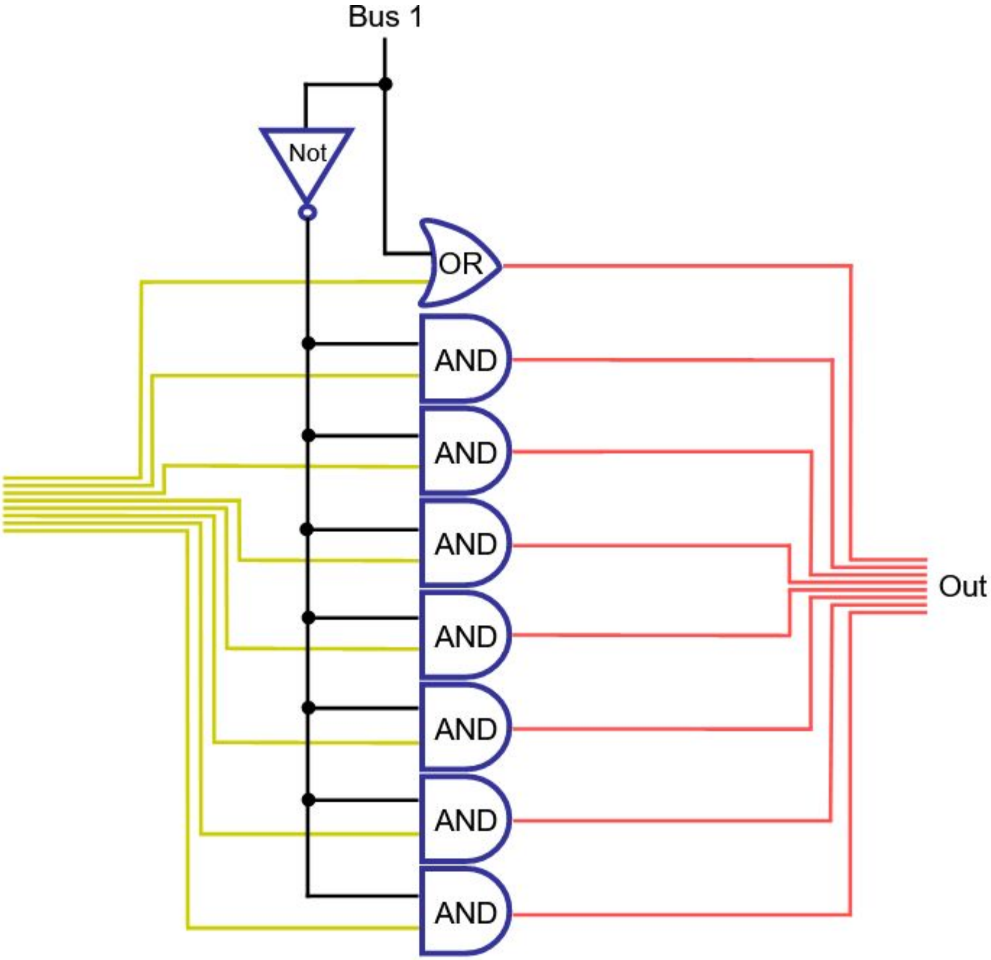
- The other signals will occur no matter which enable bit is set. This includes zero, equal, 'A' larger, and carry out. The resulting signal is dependent upon the input(s).

Bus 1 – The accumulator

- The purpose of the Bus 1 is to add the binary number 1 to the bus. This is useful for incrementing the next address in the IAR by one so that the next programming instruction can be fetched from RAM and placed into the IR.

- The single bit input ‘Bus 1’ determines what happens when a byte tries to pass through this device. When it’s off, all bits from A pass through to Out unchanged. When it’s on, the input byte is ignored and the output byte will be 00000001, which is the number 1 in binary.





Part Two (D): The Controller

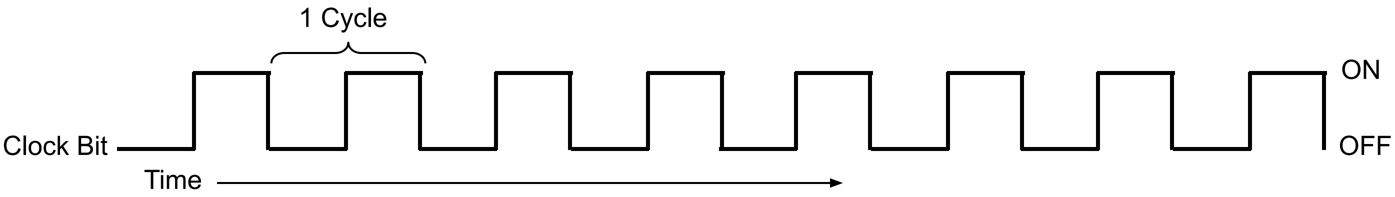
The Clock

(Technically it is not in the Controller or even the CPU – although the CPU does have a multiplier)

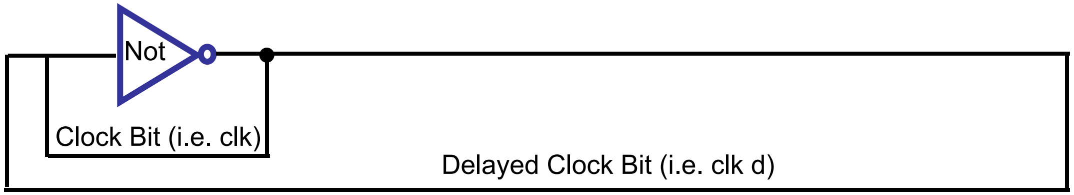
- The clock generator is a circuit that produces a timing signal, known as a clock signal (i.e. Bit). It basically oscillates from on to off and it does this millions/billions of times per second.

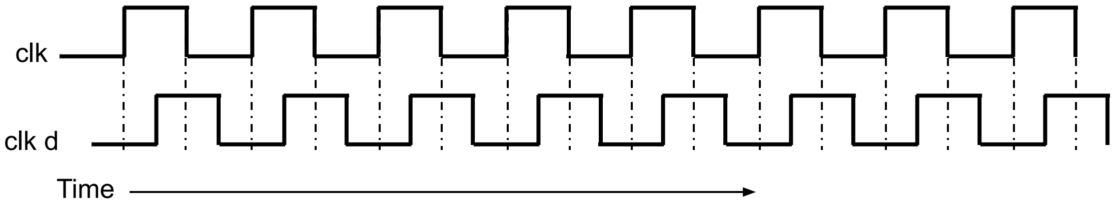
- It's purpose is to turn the control bits (i.e. set and enable) on/off at the appropriate times.

- Because the clock turns on/off in regular time intervals, it is cyclic, like a circle. A complete cycle is 1 part On + 1 part Off. The number of cycles that occur per second is measured in Hertz (after Heinrich Hertz). 500 Hz means 500 times / second. Most modern day computers are in GigaHertz (gHz).



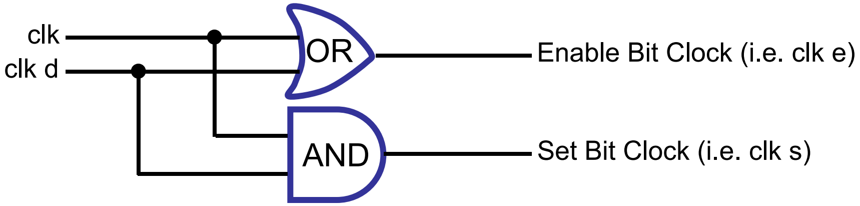
- You can wire a clock simply by connecting a NOT gate's output back to its input. And to get a longer signal (i.e. a delayed signal), just lengthen the line. Notice how the Delayed Clock Bit cycle happens slightly after the Clock Bit's cycle.



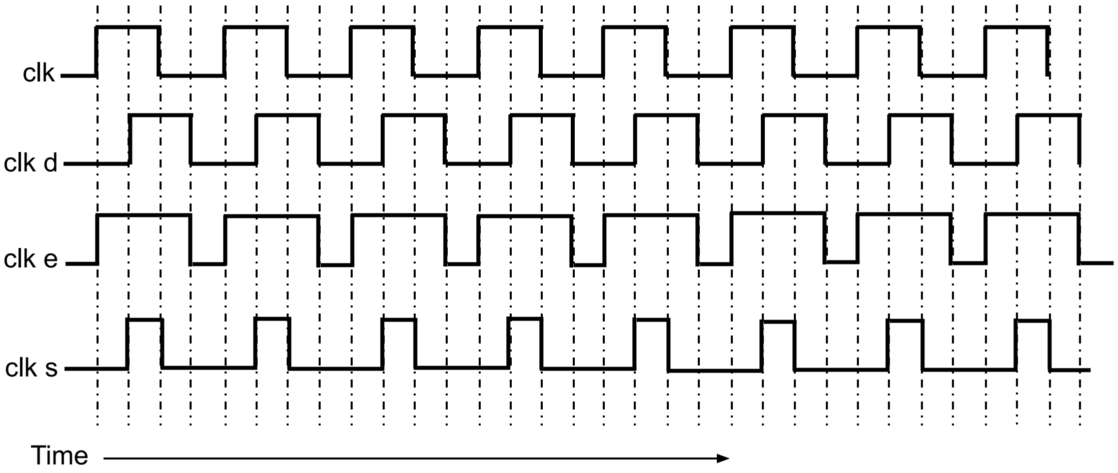


The Clock (Continued)

- We can use the clock to time when the Set and Enable Bits are set. We want the signals for each of these types of bits to happen in a certain order and slightly delayed from one another. Thus, let's wire the Clock Bit and Delayed Clock Bit together using some Gates:



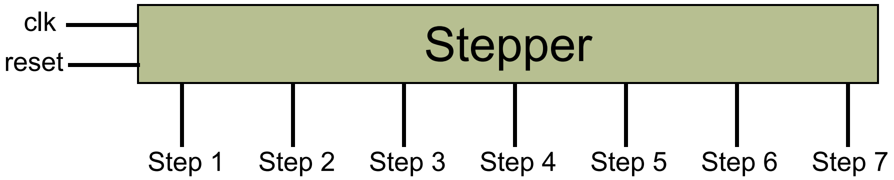
- The two new clock bits will have the following timings. Notice how the "clk e" cycle encompasses the "clk s" cycle. Thus, this makes for perfect timing for moving a byte of data from one register, across the bus, and into another register by enabling the first register then setting the second register. The enable will occur and stay on while the set bit comes on then off, then the enable goes off.



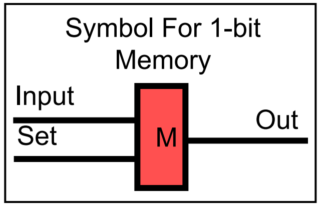
The Stepper

- The Stepper has two inputs:
  - The clock bit (clk) – need a bit to turn on/off to change steps.
  - The reset bit – returns the Stepper back to Step 1.

- It's purpose is to iterate through a series of step bits turning each bit on/off one-by-one in order. The Scott CPU will have 7 steps. Each step will turn on for one clock cycle. When the last step occurs, it stays on until the reset occurs. Alternatively, you could connect the last step to the reset, so that it resets itself.



- The Stepper is built by stringing together the same 1-bit memory gate that was used to make a Register.



Inside the Controller

- The

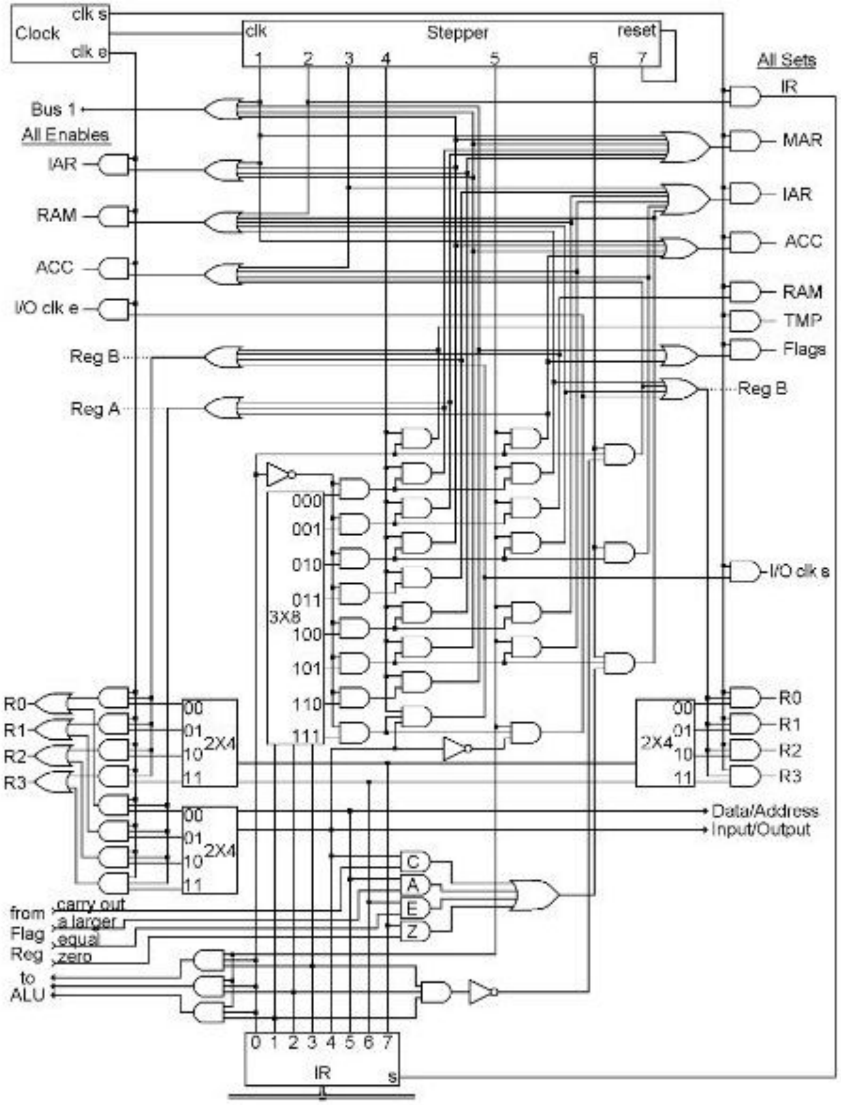
Doing something useful with the Controller

- Let’s say we want to add two bytes together and that one byte is in Register R0 and the other in R1.

The Completed Controller

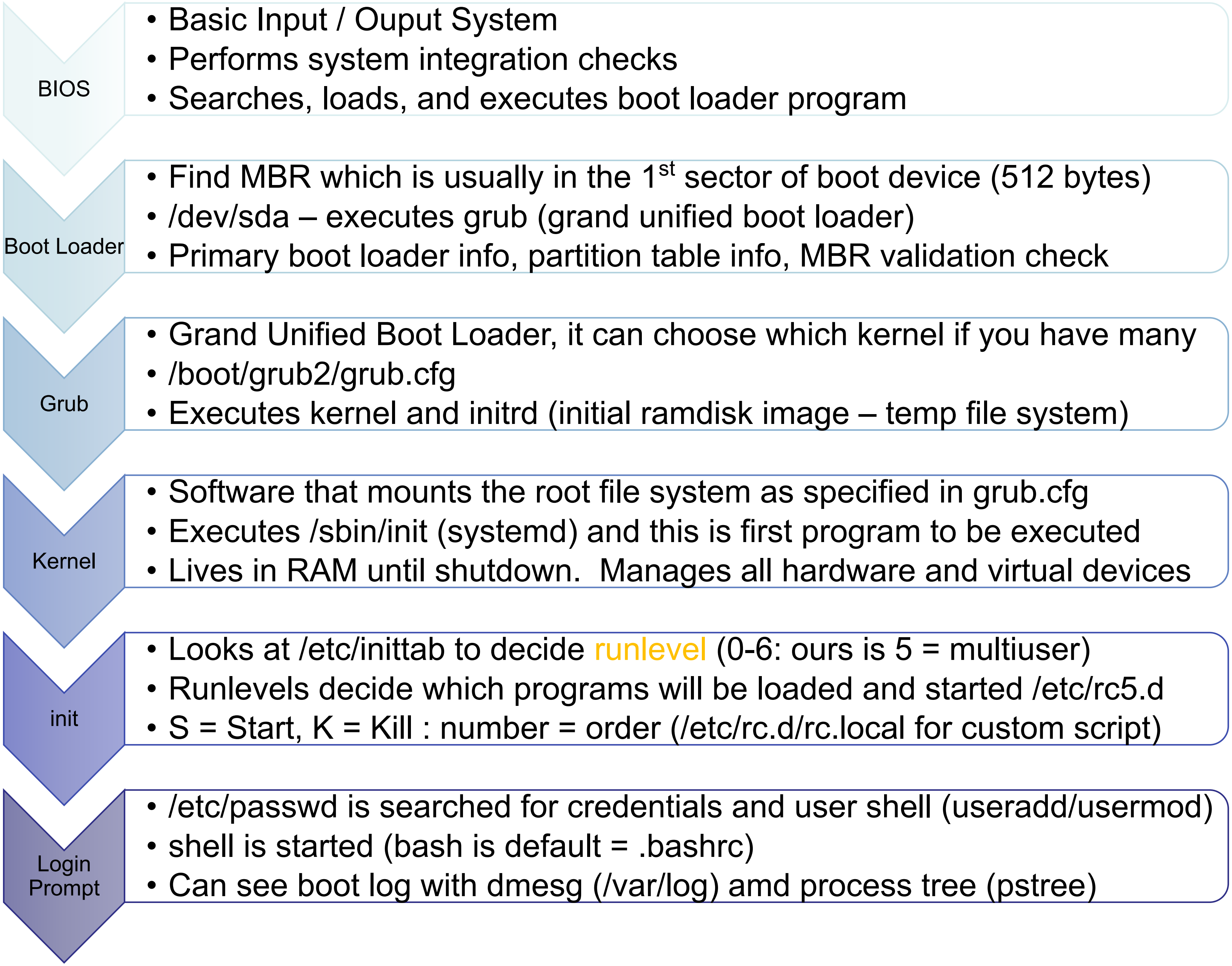
(Scott CPU)

We'll discuss the pieces of it in the CPU Instruction Set section.



Part Two (D): Other Parts

The Linux Boot Process



BIOS

- Basic Input / Ouput System
- Performs system integration checks
- Searches, loads, and executes boot loader program

Boot Loader

- Find MBR which is usually in the 1st sector of boot device (512 bytes)
- /dev/sda – executes grub (grand unified boot loader)
- Primary boot loader info, partition table info, MBR validation check

Grub

- Grand Unified Boot Loader, it can choose which kernel if you have many
- /boot/grub2/grub.cfg
- Executes kernel and initrd (initial ramdisk image – temp file system)

Kernel

- Software that mounts the root file system as specified in grub.cfg
- Executes /sbin/init (systemd) and this is first program to be executed
- Lives in RAM until shutdown. Manages all hardware and virtual devices

init

- Looks at /etc/inittab to decide runlevel (0-6: ours is 5 = multiuser)
- Runlevels decide which programs will be loaded and started /etc/rc5.d
- S = Start, K = Kill : number = order (/etc/rc.d/rc.local for custom script)

Login Prompt

- /etc/passwd is searched for credentials and user shell (useradd/usermod)
- shell is started (bash is default = .bashrc)
- Can see boot log with dmesg (/var/log) amd process tree (pstree)

Basic Input/Output System (BIOS)

What came first, the chicken or the egg?
Programs have to be loaded by something, such as another program!
So what program is loaded first that will be responsible for loading other programs?

- The BIOS is a piece of hardware with software on it. It is stored on a Read Only Memory (ROM) chip on the motherboard.

- It contains the first program (i.e. software) that runs when you start up your computer. This software is called firmware and it requires special steps to update it.

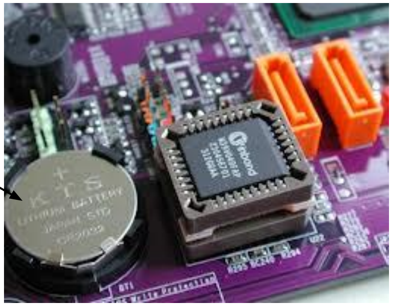
- Some configuration information for the BIOS is not stored in the firmware and is rather stored in the CMOS (Complementary metal-oxide semiconductor) and it is connected to a battery for permanent power. For example it will contain the date/time, passwords, etc.

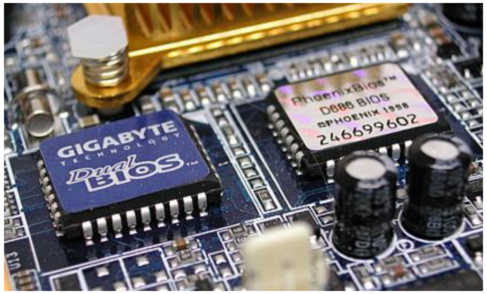
- It will perform the Power On Self Test (POST).

- It is in between hardware and the Operating System and brings all of the hardware to a state where it is ready to boot the Operating System. For example, it will perform a Power On Self Test (POST) and initialize peripheral devices.

- It is then used to find, load, and start an Operating System. The boot loader will be used to locate the boot sector (first 512 bytes – sector) on a device and will load the Operating System from there.

Battery





CPU Instruction Set

CPU Instruction Set

- When a CPU is made, it is designed to understand a certain number of instructions or bit patterns. We call this the CPU Instruction Set or Machine Language and it is the lowest level of programming.

The Arithmetic or Logic Instruction

(Scott CPU)

- d

The Load and Store Instructions

(Scott CPU)

- d

The Data Instruction

(Scott CPU)

- d

The Jump Instruction

(Scott CPU)

- Branch to another location in the program and execute instructions there.

Another Type of Jump

(Scott CPU)

- Indirectly branch to another location, while saving the location of the next instruction as a point to return to. (subroutine call)

The Jump If Instruction

(Scott CPU)

- Conditionally branch to another location if a certain condition holds.

The Clear Flags Instruction

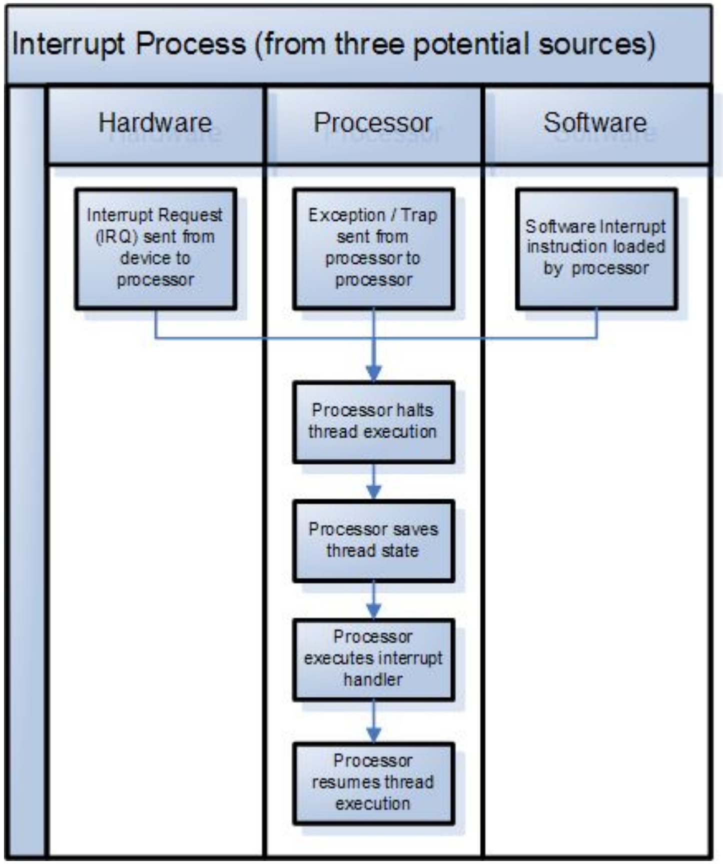
(Scott CPU)

- g

The Interrupt

(Scott CPU)

- An interrupt is a signal sent to the processor indicating an event that needs immediate attention. It can be sent from:
    - Hardware, such as a keyboard
    - Software, such as a program that executes the interrupt instruction
    - The processor itself


Interrupt Process (from three potential sources)
Hardware
Processor
Software
Interrupt Request (IRQ) sent from device to processor
Exception / Trap sent from processor to processor
Software Interrupt instruction loaded by processor
Processor halts thread execution
Processor saves thread state
Processor executes interrupt handler
Processor resumes thread execution

Other Software

Operating System

- Loaded by the BIOS.

- It is software called a Program or Process. Part of it is loaded into the top-most part of RAM and it is designed to run in an infinite loop.

- contains low-level functions for hardware

- protects memory

- designed to know when, how, and where to load other programs. It swap in and out programs so fast that they all appear to be running at the same time.

Other Programs

- Also known as Processes or Applications.

Programming Languages