Logic

The Decimal System

Based on the fact that humans have two hands containing ten digits.

Represented by the ten digits: 0 – 9

i.e. 0,1,2,3,4,5,6,7,8,9

The Decimal System

This system works on a base of 10 and numbers can be represented in this way by the suffix$_{10}$ i.e.

$2_{10} = 2$ $3_{10} = 3$ $44_{10} = 44$ and so on

e.g. $236_{10} = 236$

Counting

When counting in a decimal system we just count to the tenth digit in the first column (called the units) zero to nine. Then we start a second column for the next number which sets the units column to zero and the next column to one, this column is called the Tens column and so on for each addition.

Column 3 Thousands	Column 2 Hundreds	Column 1 Tens	Column 0 Units
0	0	0	0
0	0	0	1
0	0	0	2
0	0	0	3
0	0	0	4
0	0	0	5
0	0	0	6
0	0	0	7
0	0	0	8
0	0	0	9
0	0	1	0
0	0	1	1
0	0	1	2

From this we can see that any digit in the first column in a decimal system is said to be multiplied by 10 to the power of zero or unity which equals 1

So the digit 7 in the first column can be expressed in a decimal system as:

$$7_{10} \text{ or } 7 \text{ X } 10^0$$

What would a digit 4 in column 2 be expressed as?

$$4 \text{ X } 10^2 = 4(10 \text{ x } 10) = 4(100) = 400$$

Binary system

Machines can only deal with two states.
A machine is either on or off.

So machines can only work to a base of 2

This is called a Binary System

So the machine can only count from 0 to 1 then it needs a new column

Column 3	Column 2	Column 1	Column 0 Units
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0

Binary system

This system works on a base of 2 and

numbers can be represented in this way by the suffix$_2$ i.e.

$0_2 = 0$ $1_2 = 1$

8

e.g. $1000_2 =$

Binary Digits (bits)

If binary digits 1 and 0 are inserted into columns as previously seen.

How many binary columns and therefore bits will it take to represent a decimal number 7 ?

3 Columns 3 bits

How to read binary numbers

Exponent	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
ON/OFF	0	1	1	0	1	0	0	0

The next example is 11111111 in binary, the maximum 8-bit value of 255. Again, reading right to left we have 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255.

Value	128	64	32	16	8	4	2	1
ON/OFF	1	1	1	1	1	1	1	1

Binary Coded Decimal

Although machines use binary numbers internally, operators still prefer to enter numbers via keyboards in decimal.

Thinking about binary columns, our decimal numbers 0-9 would require at least 4 columns

Column 3	Column 2	Column 1	Column 0 Units
1	0	0	1

Binary Coded Decimal

To represent any number from 0-9 we would require a table with 4 columns, so to enter a two digit decimal number into a keyboard we need two tables with 4 columns each.

i.e. 89 in BCD would be:-

8			
1	0	0	0

9			
1	0	0	1

Where in Pure Binary 89 would be:-

89						
1	0	1	1	0	0	1

Binary Logic

In binary logic there are only two states off and on.

We call these states:

Logic 0 Off
Logic 1 On

Truth Functions

Logic states can be expressed in different ways:

On	Off
In	Out
Up	Down
High	Low
True	False

The same can be said for states that have not been operated i.e.

Not On	=	Off		Not Off	=	On
Not In	=	Out		Not Out	=	In
Not Up	=	Down		Not Down	=	Up
Not High	=	Low		Not Low	=	High
Not True	=	False		Not False	=	True

TTL and CMOS Logic Gates

There are two main types of logic gates

TTL (transistor Transistor Logic)

Cmos (Complementary Metal-Oxide-Silicon)

Generally speaking, **TTL** logic IC's use NPN and PNP type Bipolar Junction Transistors

CMOS logic IC's use complementary MOSFET or JFET type Field Effect Transistors for both their input and output circuitry.

Logic Voltage Levels

For TTL devices, Vcc is normally 5V

For CMOS circuits Vdd can range from 3-18V

For TTL, logic 0 : 0 - 0.8V
logic 1 : 2 - 5V

For CMOS logic 0 : 0 - 1.5V
logic 1 : 3.5 - 18V

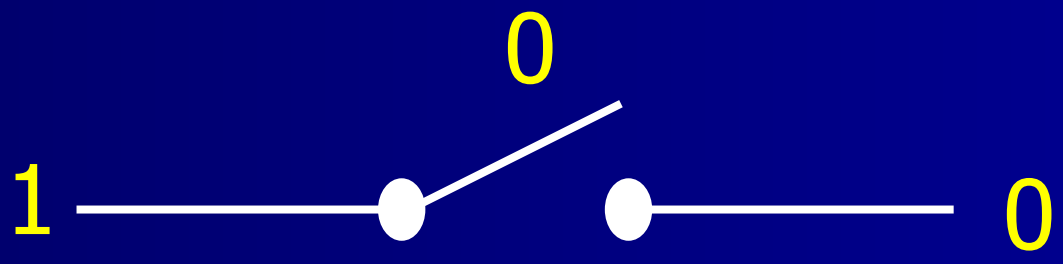
These may vary slightly depending on manufacture, see manufactures data sheet for specifications.

Boolean Algebra

Part 1

Binary Logic State 0

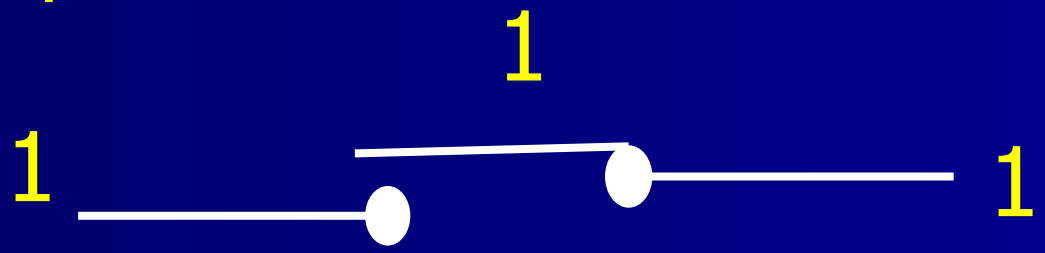
This can be represented electrically as a simple normally open switch:



Though we have a supply of 1 we get 0 out because the switch has not been operated.

Binary Logic State 1

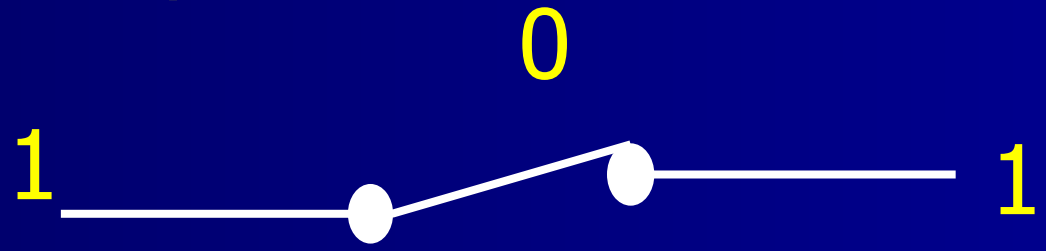
This can be represented electrically as a simple normally open switch that has been operated:



Now because the switch has been operated with a supply of 1 we get 1 out.

Binary Logic State not 0

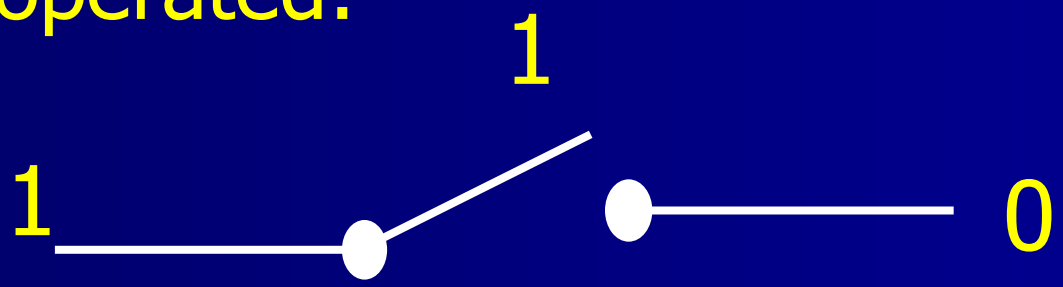
This can be represented electrically as a simple normally closed switch that has not been operated:



Now because the switch has not been operated with a supply of 1 we get 1 out.

Binary Logic State not 1

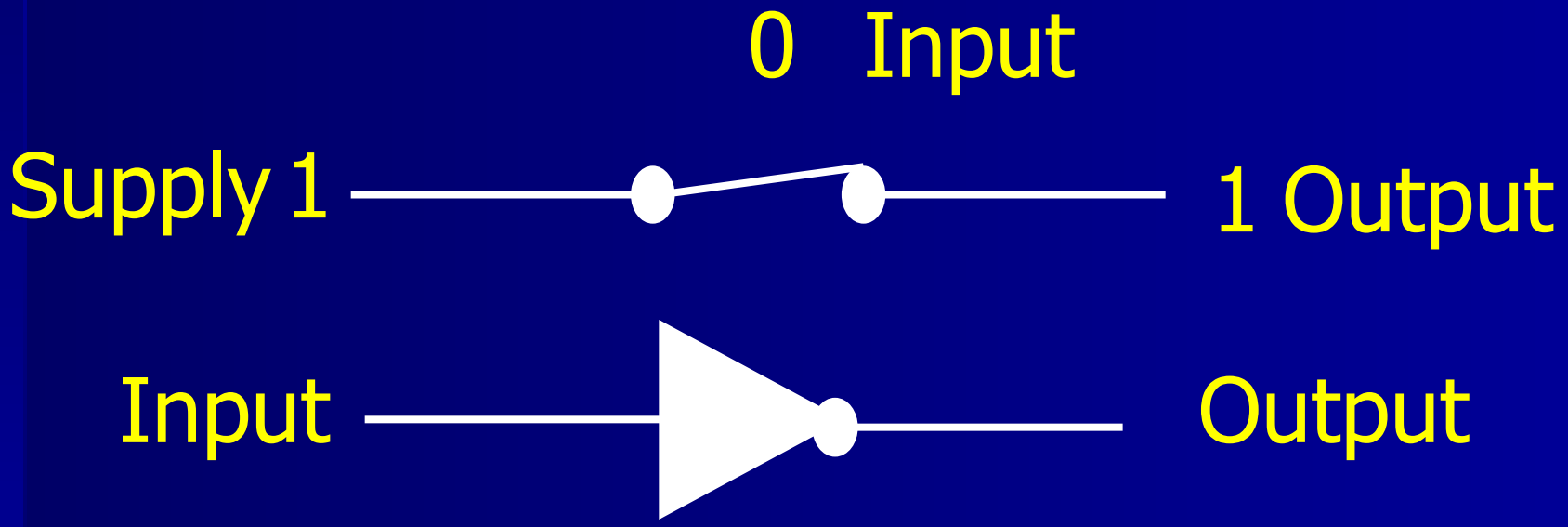
This can be represented electrically as a simple normally closed switch that has been operated:



Though we have a supply of 1 we get 0 out because the switch has been operated.

Binary Logic Symbols

The symbol for the simple not function below is drawn as a triangle with a circle at the end and is now called a logic gate.



This is the symbol for a not gate

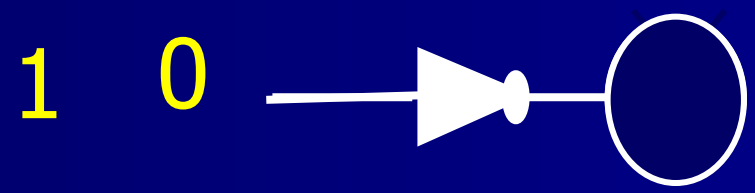
Truth Tables

Consider a device with 3 inputs, A B and C we would have 8 possible variables, so we would need to create a truth table for all combinations

A	B	C	Out
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

Truth Table Inverter

This has only two possible inputs 0 or 1



Input	Output
0	1
1	0

The previous not gate was acting as a simple inverter and was performing a simple not function.

The table it produced is called a truth table and the idea of true and false has been historically linked with algebraic methods of studying binary variables.

Algebra

The algebra we use to study binary variables is called Boolean Algebra.

Boolean Algebra

Boolean algebra variables are denoted the same way as in ordinary algebra.

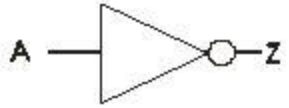
e.g. We use letters A, B, C etc or X, Y, Z

The difference being, in ordinary algebra the letters could be any value, whereas in Boolean they can only be 0 or 1

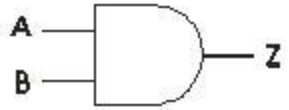
Boolean Expressions

Each logic gate has a corresponding Boolean mathematical formula or expression. The use of these expressions saves us having to draw symbol diagrams over and over again.

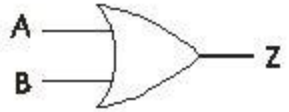
The name Boolean is taken from an English mathematician, George Boole, who founded symbolic logic in the nineteenth century

$Z = \overline{A}$ 

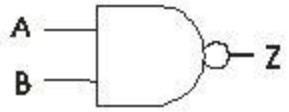
NOT

$Z = A.B$ 

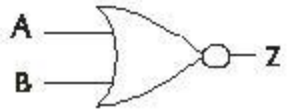
AND

$Z = A+B$ 

OR

$Z = \overline{A.B}$ 

NAND

$Z = \overline{A+B}$ 

NOR

Truth Tables

The operation of not we saw earlier can be written as a single variable Not A or by a bar above A to symbolise a negation of A e.g. $\bar{A}$

A	$\overline{A}$
0	1
1 (on)	0 (off)

Logic Gates

As well as the not gate there are 5 other types of gate, these are termed:

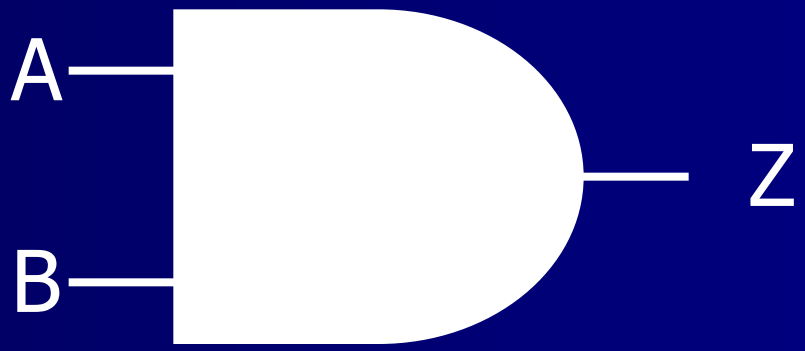
AND
OR
NAND
NOR
EXOR

2 Input AND Gate

This can be represented by two NO electrical switches A **AND** B in series.

Both switch A **AND** B have to be operated to get an output of 1 for an supply of 1.

AND: Symbol & Truth Table



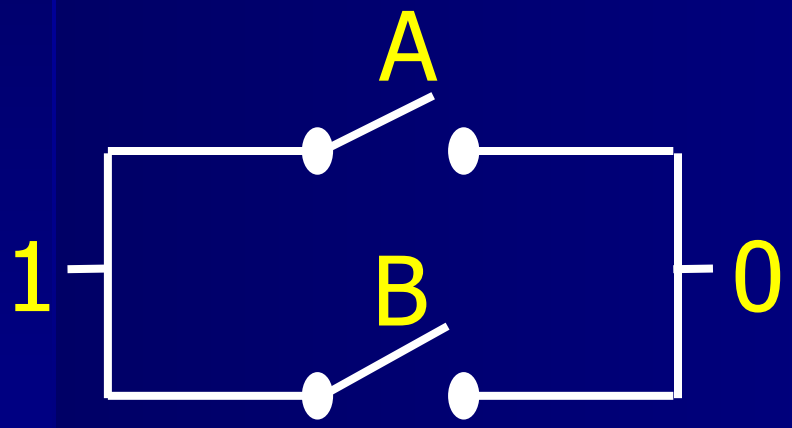
Where A and B are inputs and Z is an output

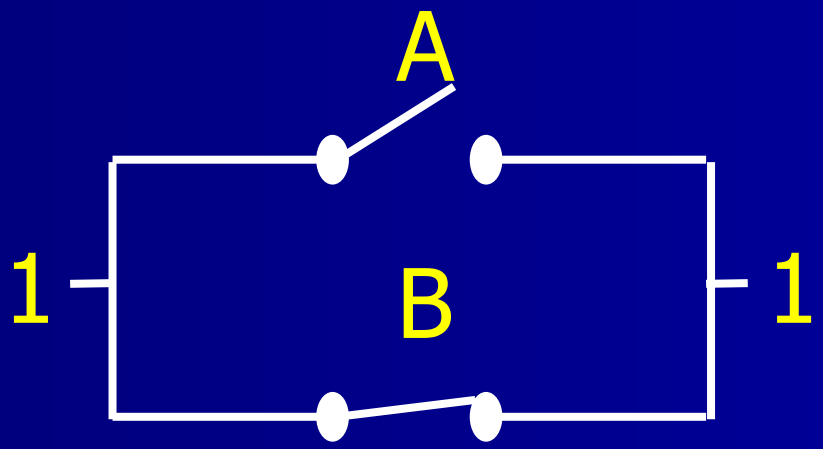
A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

Z = A AND B alternatives Z = A.B also AB

2 Input OR Gate

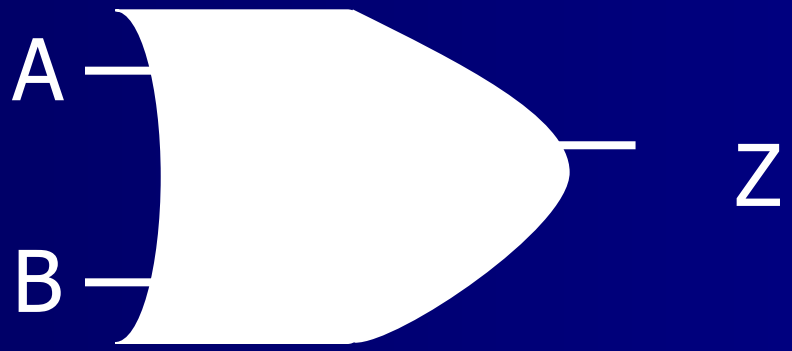
This can be represented by two NO electrical switches A **OR** B in parallel.





Operating A **OR** B will give an output of 1 with a supply of 1

OR: Symbol & Truth Table



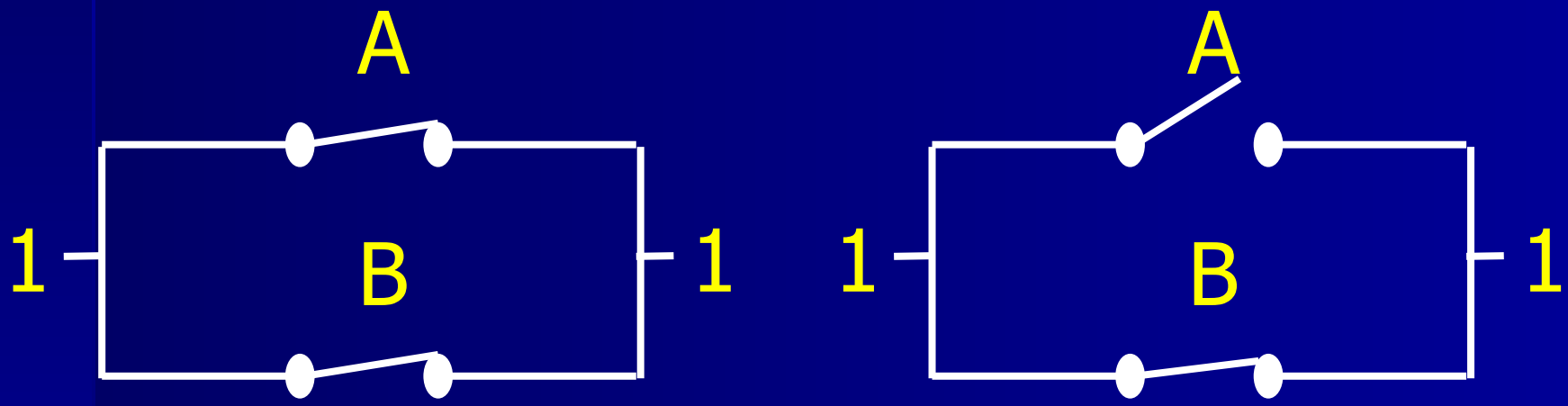
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

Where A and B are inputs and Z is an output

Z = A OR B alternative Z = A + B

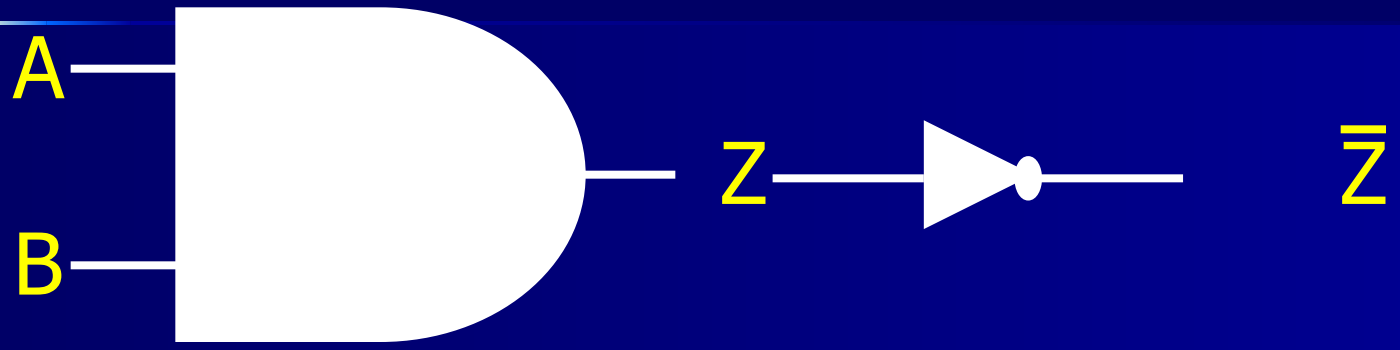
NAND Gate

This can be represented by two NC electrical switches A **and** B in parallel.

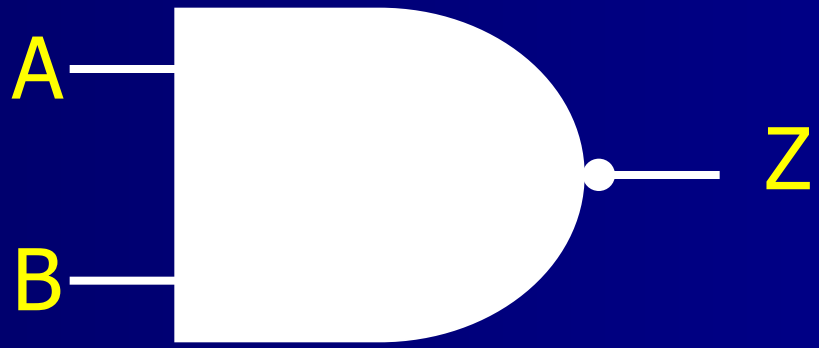


Not operating A **and** B will give an output of 1.

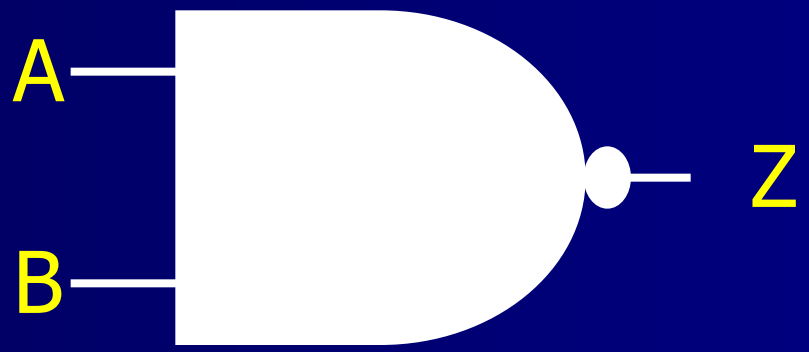
NAND: Symbol



This is basically an AND gate with a not at the output



NAND: Truth Table



Where A and B are inputs and Z is an output

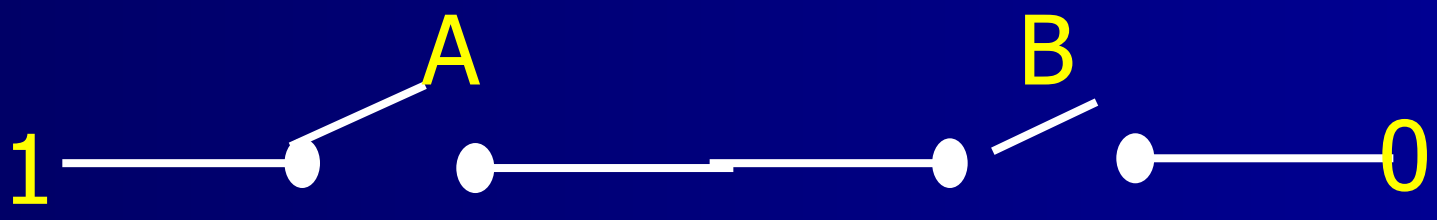
A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

$Z = \overline{A \text{ AND } B}$ alternative $Z = \overline{A.B}$

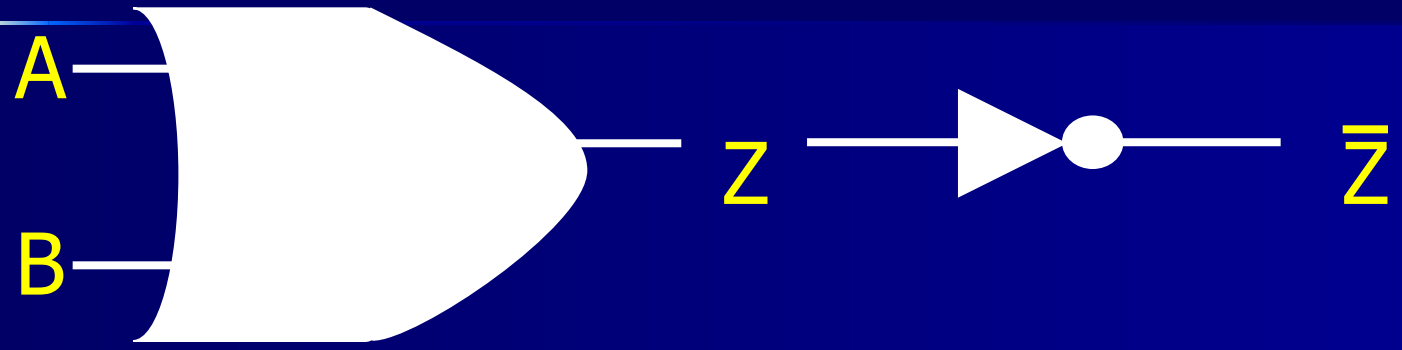
Nor Gate

This can be represented by two NC electrical switches A **AND** B in series.

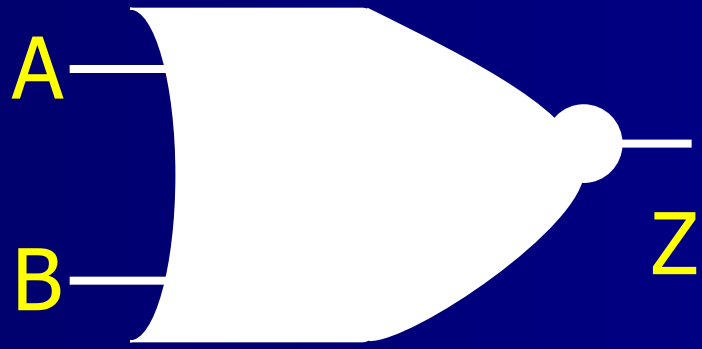
Both switch A **AND** B have to be not operated to get an output of 1 for an supply of 1.



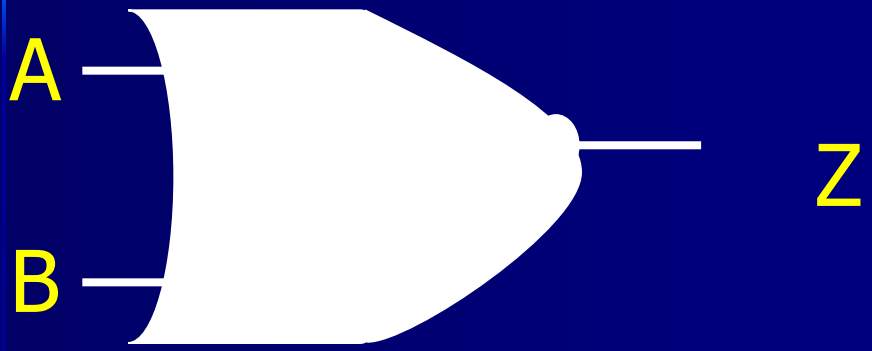
NOR: Symbol



This is basically an OR gate with a not at the output



NOR: Symbol & Truth Table



Where A and B are inputs and Z is an output

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

$Z = \overline{A \text{ OR } B}$ alternative $Z = \overline{A + B}$

Boolean Algebra

Part 2

De Morgan's Theorem

The most important logic theorem for digital electronics, this theorem says that any logical binary expression remains unchanged if we:

Change all variables to their complements.

Change all AND operations to ORs.

Change all OR operations to ANDs.

Take the complement of the entire expression.

De Morgan's Theorem

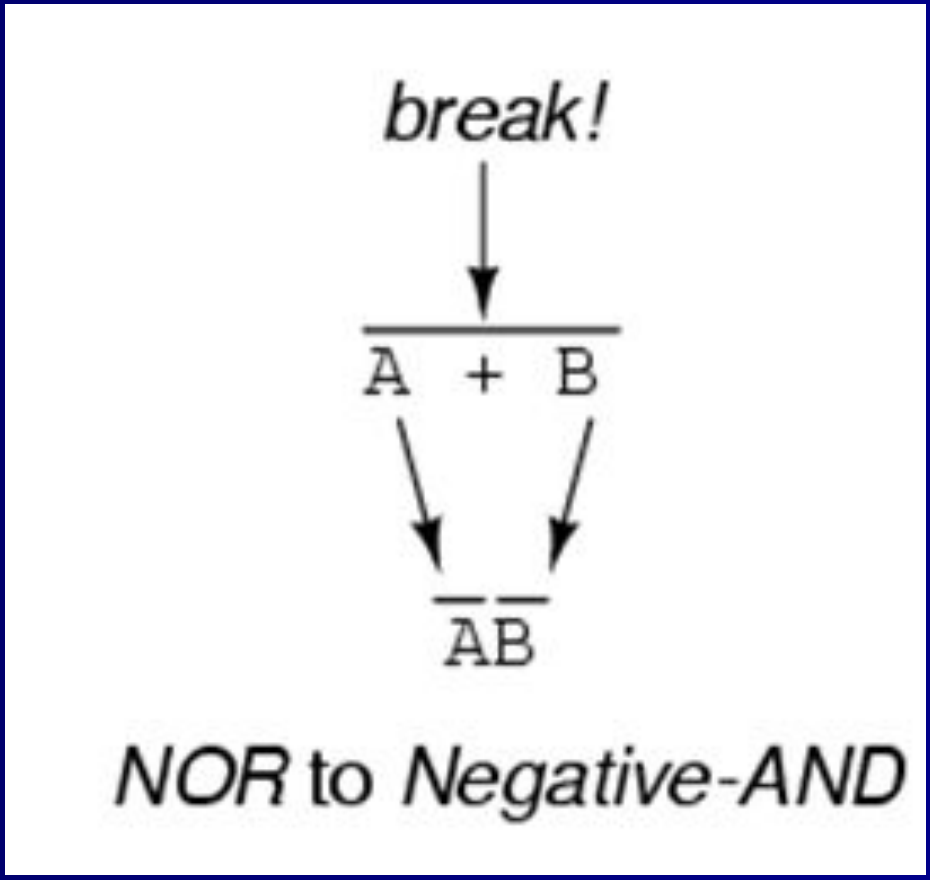
A practical operational way to look at De Morgan's Theorem is that the inversion bar of an expression may be broken at any point and the operation at that point replaced by its opposite.

(i.e., AND replaced by OR or vice versa).

$$\overline{A + B} = \overline{A}.\overline{B}$$

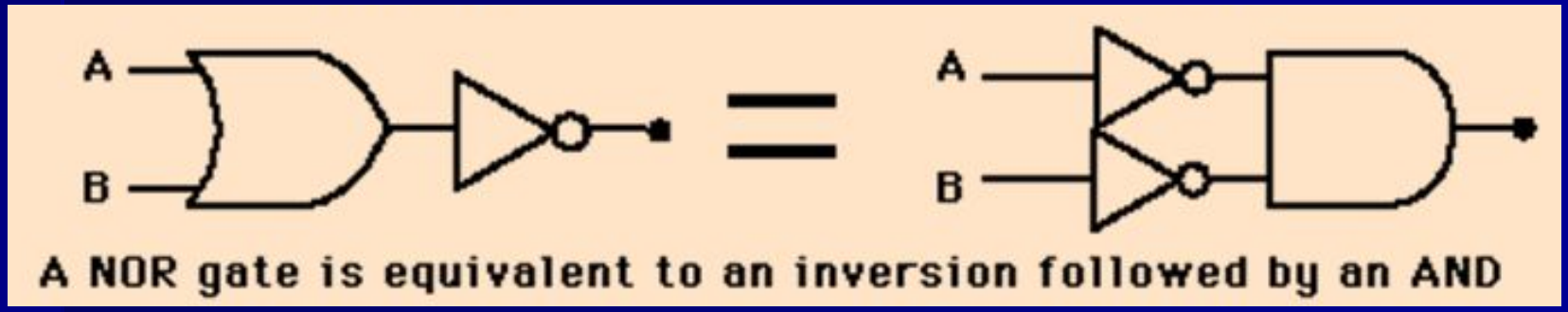
Not A OR B = Not A AND Not B

De Morgan's Theorem

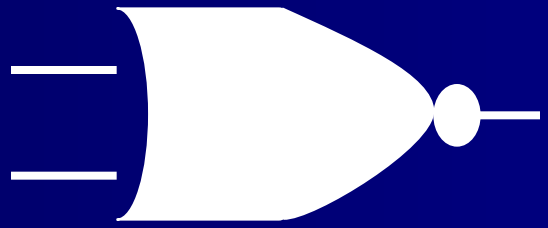


De Morgan's Theorem in Gates

$$\overline{A + B} = \overline{A}.\overline{B}$$

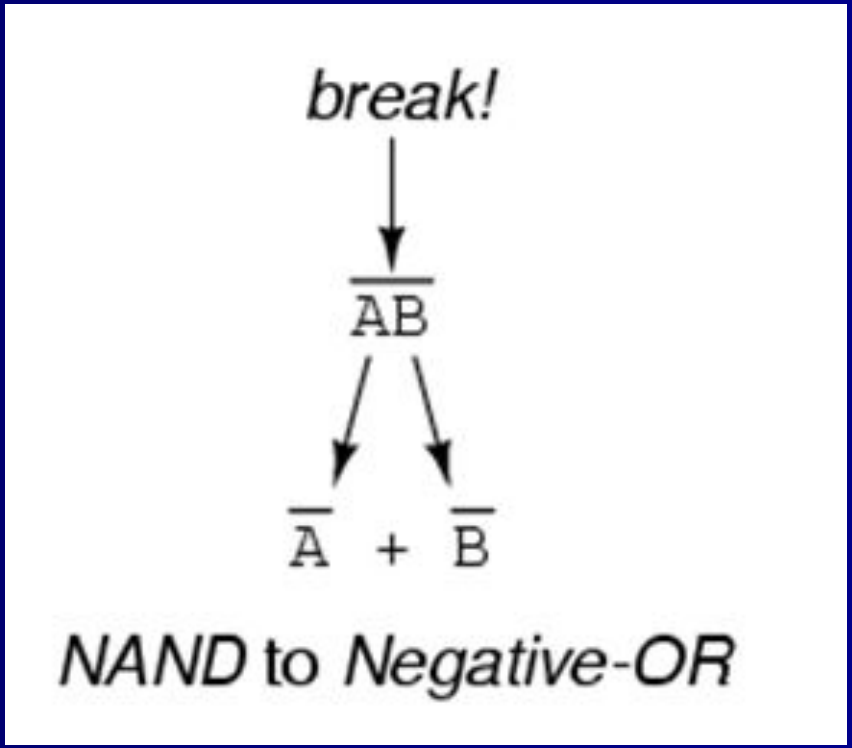


A NOR gate is equivalent to an inversion followed by an AND



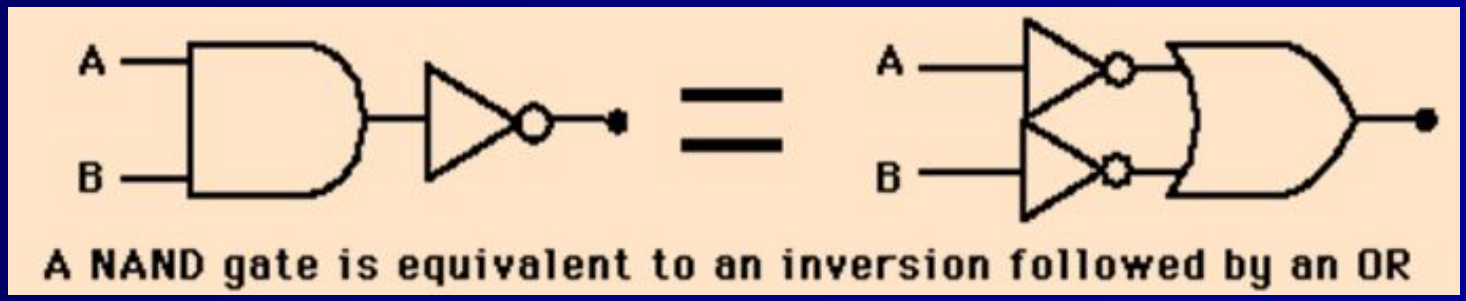
NOR

De Morgan's Theorem

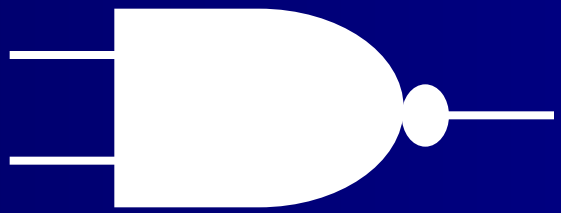


De Morgan's Theorem in Gates

$$\overline{A.B} = \overline{A} + \overline{B}$$



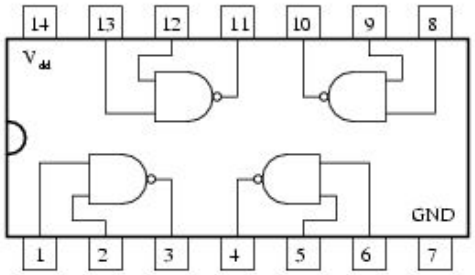
A NAND gate is equivalent to an inversion followed by an OR



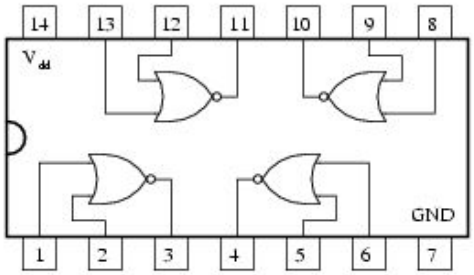
NAND

Integrated Circuits (ICs)

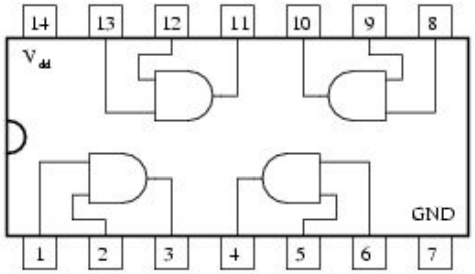
4011
Quad NAND gate



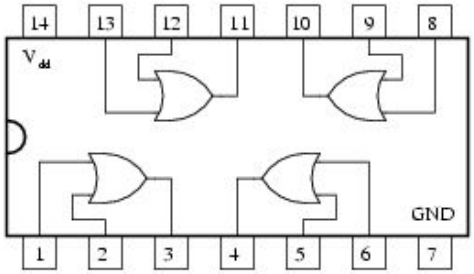
4001
Quad NOR gate



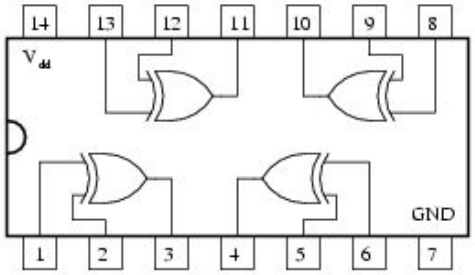
4081
Quad AND gate



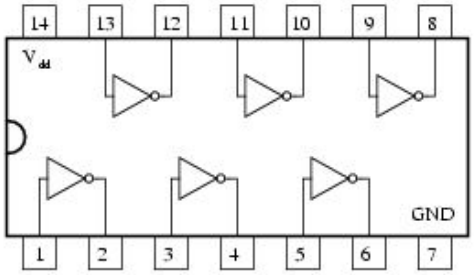
4071
Quad OR gate

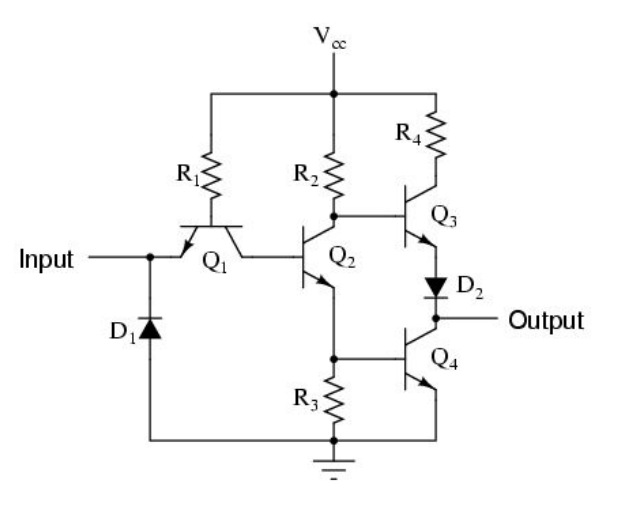


4070
Quad XOR gate

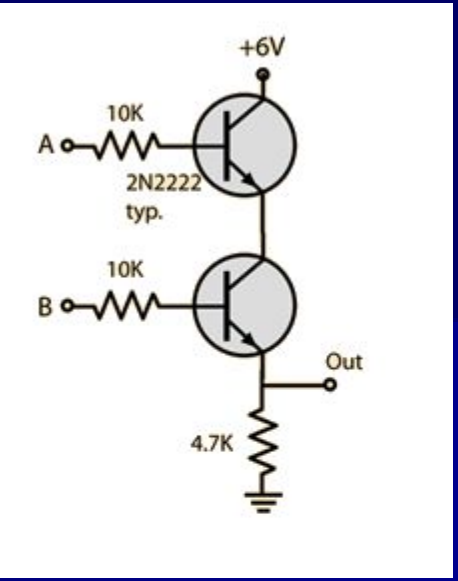


4069
Hex inverter

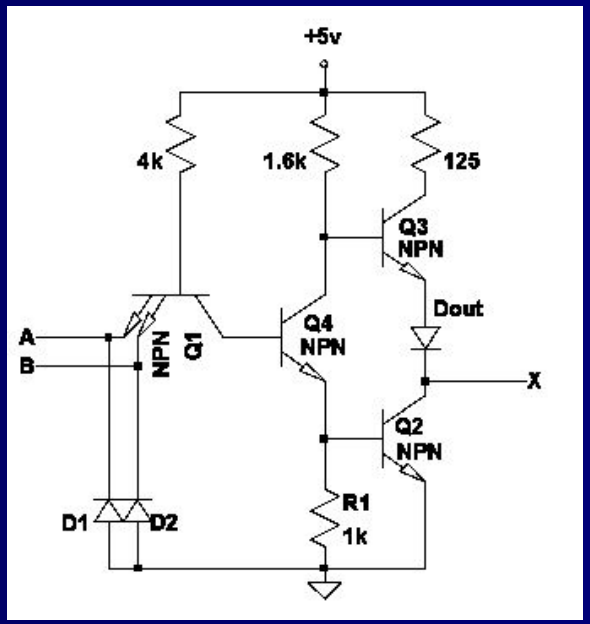




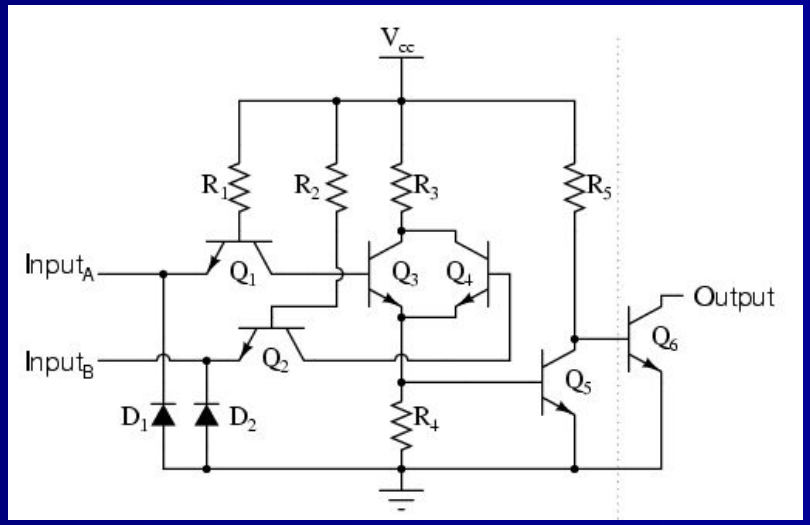
Practical NOT Circuit



Practical AND Circuit

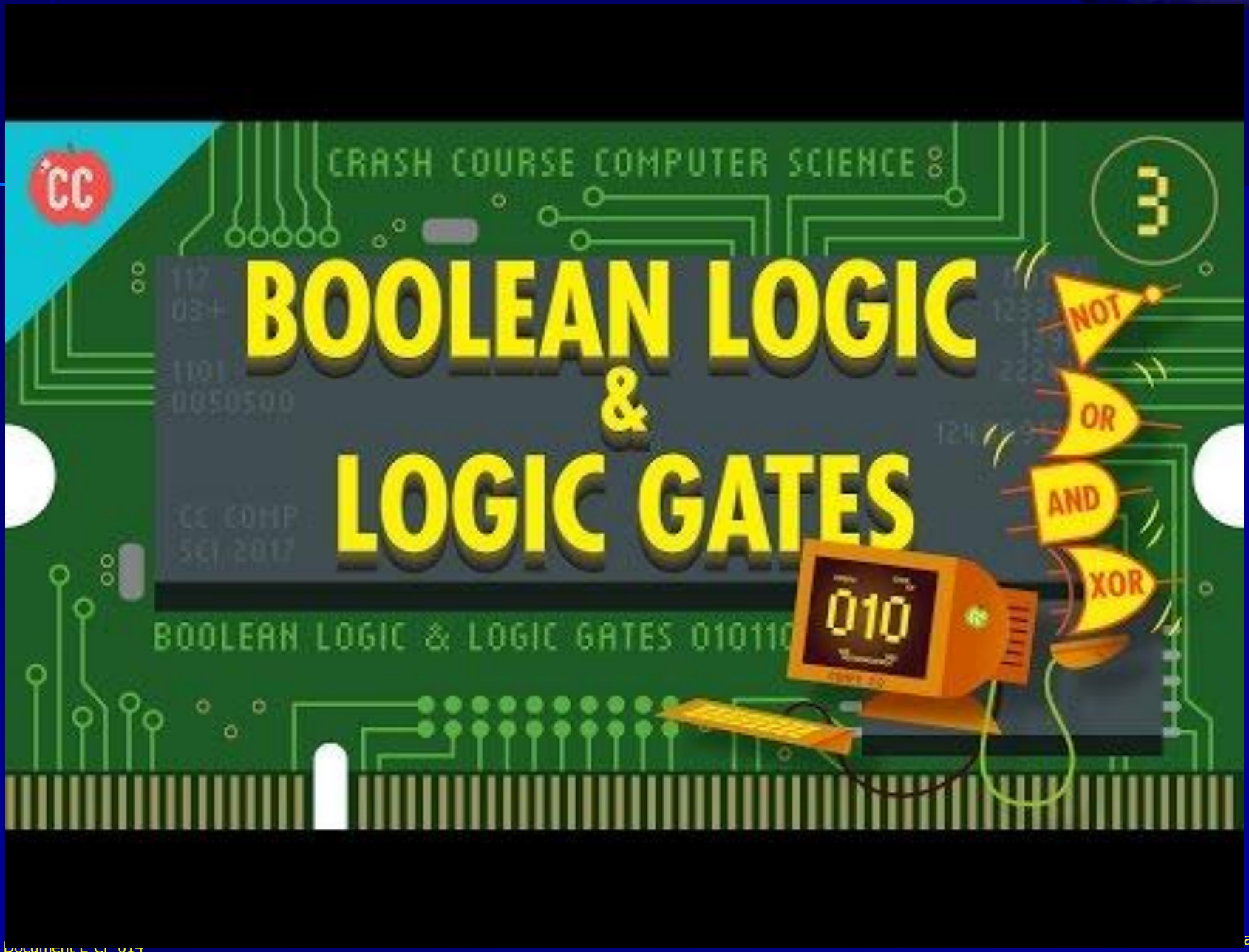


Practical NAND Circuit



Practical NOR Circuit

Recap


CRASH COURSE COMPUTER SCIENCE
3
BOOLEAN LOGIC
&
LOGIC GATES
NOT
OR
AND
XOR
010
BOOLEAN LOGIC & LOGIC GATES