BCN 1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

OUTCOME

- Able to perform operation on Boolean algebra, design simple circuit using gates
- Understand the relationship of Digital logic with computer operation for example in gates vs memory

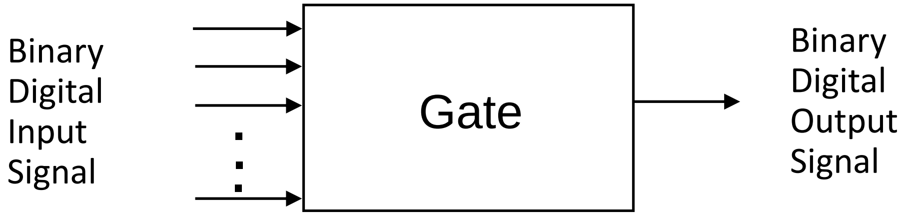
CONTENT

- Logic Gates
- Boolean Algebra
- Combinational Circuits
  - Flip-Flops
- Sequential Circuits
  - Memory Components

DIGITAL COMPUTERS

- Digital computer deals with information that is denoted by binary digits
- Think! Answer!
- Why is it *BINARY* ? Why not Decimal or other number system?

LOGIC Circuit



Basic Logic Circuits

- **Combinational Logic circuit:** output value depends only on the input values (e.g. OR, AND, etc)
- **Sequential Logic Circuit:** output value depends on the input values and the current state
- Gate Functions are described by:
  - Truth Table
  - Boolean Function
  - Karnaugh Map

LOGIC GATES & BOOLEAN ALGEBRA

Truth Tables

Inputs		Outputs
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Inputs		Outputs
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Inputs	Outputs
x	$\overline{x}$
0	1
1	0

xy = x
AND is true only if **both** inputs are true

x + y = x
OR is true if **either** inputs are true

x bar = NOT x
NOT inverts the bit
We will denote x bar as ~x

NOR is NOT of OR

x	y	x **NOR** y
0	0	1
0	1	0
1	0	0
1	1	0

NAND is NOT of AND

x	y	x **NAND** y
0	0	1
0	1	1
1	0	1
1	1	0

XOR is true if both inputs **differ**

x	y	x **XOR** y
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Expressions

- Possible to derive Boolean expressions from Boolean operations
- Consider the expression:
  - F = X + ~Y*Z
- What is it's truth table?

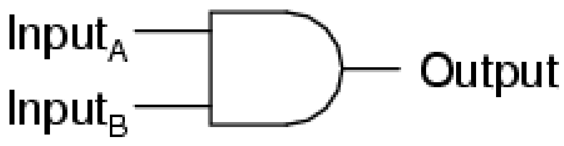
Inputs					Outputs
x	y	z	$\overline{y}$	$\overline{y}z$	$x + \overline{y}z = F$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

Take note: Notice that it is easier to derive the truth table for the entire expression by breaking it into subexpressions
So first we determine ~Y
*next, ~Y * Z*
*finally, X + ~Y*Z*

Truth Table ???

- Is to understand a logic circuit with different permutations of inputs, with logic 1- true and logic 0- false.
- The desired output can be achieved by a combination of logic gates.
- truth table of two inputs is shown

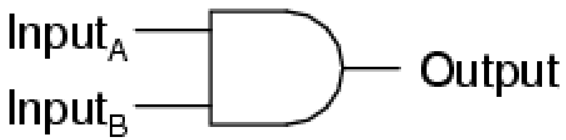
2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table ???

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Basic Boolean Identities

- As with algebra, there will be Boolean operations that we want to simplify
  - We apply the following Boolean identities to help
    - For instance, in algebra, x = y * (z + 0) + (z * 0) can be simplified to just x = y * z

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0+x = x$
Null (or Dominance) Law	$0x = 0$	$1+x = 1$
Idempotent Law	$xx = x$	$x+x = x$
Inverse Law	$x\overline{x} = 0$	$x+\overline{x} = 1$
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
Absorption Law	$x(x+y) = x$	$x+xy = x$
DeMorgan's Law	$(\overline{xy}) = \overline{x}+\overline{y}$	$(\overline{x+y}) = \overline{x}\,\overline{y}$
Double Complement Law	$\overline{\overline{x}} = x$	

Law of Common Identities

$$A \cdot (\overline{A} + B) = AB$$
$$A + (\overline{A}B) = A + B$$

Basic Boolean Identities

Here we have an example specifically to see how DeMorgan’s Law works

x	y	(xy)	$\overline{(xy)}$	$\overline{x}$	$\overline{y}$	$\overline{x}+\overline{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

DeMorgan’s Law states that ~(X*Y) = ~X+~Y
Boolean expressions are proved if the values in truth tables give the same values for left and right side of the equations

Example 1: use algebraic simplification rules to reduce ~abc+~ab~c+ac

= ~abc + ~ab~c + ac

= ~ab(c+~c)+ac (distributive law)

= ~ab(1)+ac (inverse law)

= ~ab+ac (identity law)

*Example 2: ab+~ac+bc = ab+~ac+bc*1 (identity)*

= ab+~ac+bc(a+~a) (inverse)*

= ab +~ac+abc+~abc (distributive)

= ab(1+z)+~ac (b+1) (distributive)

= ab(1)+~ac(1) (null)

*= ab *1+~ ac *1 (absorption)*

= ab +~ac (identity)

Example 3: $(a+b)(\sim a+b)$

$= \sim a(a+b)+b(a+b)$ (distributive)

$= \sim aa+\sim ab+ab+bb$ (distributive)

$= 0+\sim ab+ab+bb$ (inverse)

$= \sim ab+ab+bb$ (identity)

$= b(\sim a+a+b)$ (distributive)

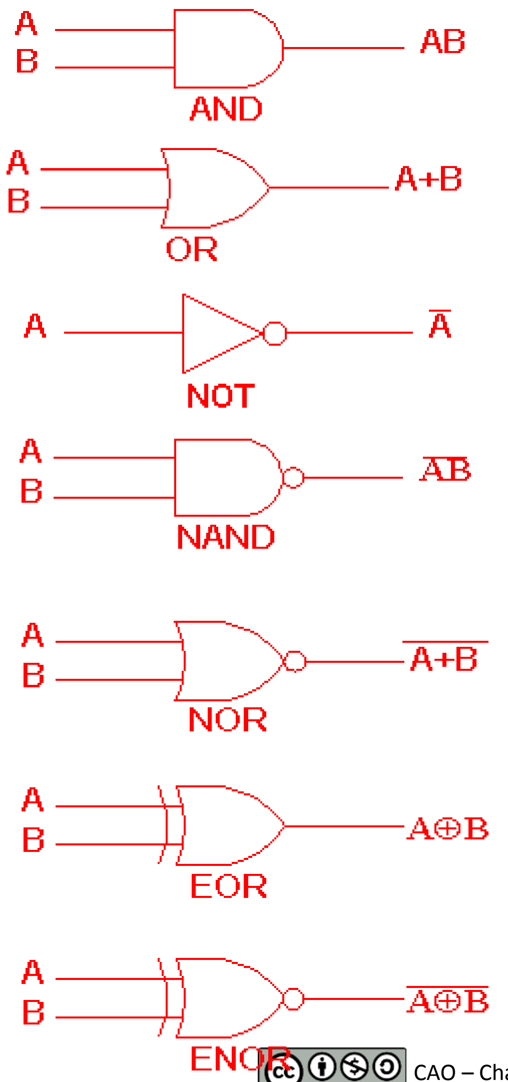
$= b(1+b)$ (inverse)

$= b(1)$ (identity)

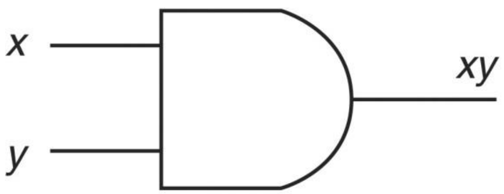
$= b$ (idempotent)

LOGIC GATES

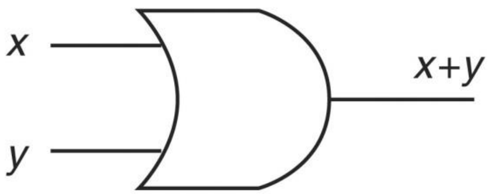
Gate	Description	Truth table
AND Gate	The AND gate is a logic gate that gives an output of '1' only when all of its inputs are '1'. Thus, its output is '0' whenever at least one of its inputs is '0'. Mathematically, $Q = A \cdot B$.	A, B, Output Q 0, 0, 0 0, 1, 0 1, 0, 0 1, 1, 1
OR Gate	The OR gate is a logic gate that gives an output of '0' only when all of its inputs are '0'. Thus, its output is '1' whenever at least one of its inputs is '1'. $Q = A + B$.	A, B, Output Q 0, 0, 0 0, 1, 1 1, 0, 1 1, 1, 1
NOT Gate	The NOT gate is a logic gate that gives an output that is opposite the state of its input. Mathematically, $Q = \bar{A}$.	A, Output Q 0, 1 1, 0
NAND Gate	The NAND gate is an AND gate with a NOT gate at its end. Thus, for the same combination of inputs, the output of a NAND gate will be opposite that of an AND gate. Mathematically, $Q = \overline{A \cdot B}$.	A, B, Output Q 0, 0, 1 0, 1, 1 1, 0, 1 1, 1, 0
NOR Gate	The NOR gate is an OR gate with a NOT gate at its end. Thus, for the same combination of inputs, output of a NOR gate will be opposite that of an OR gate. Mathematically, $Q = \overline{A + B}$.	A, B, Output Q 0, 0, 1 0, 1, 0 1, 0, 0 1, 1, 0
EXOR Gate	The EXOR gate (for 'Exclusive OR' gate) is a logic gate that gives an output of '1' when only inputs is '1'.	A, B, Output Q 0, 0, 0 0, 1, 1 1, 0, 1 1, 1, 0



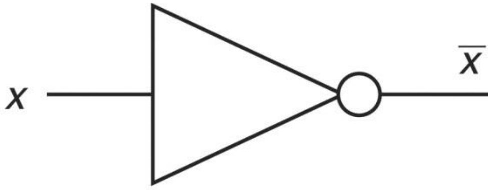
Logic Gates



AND Gate

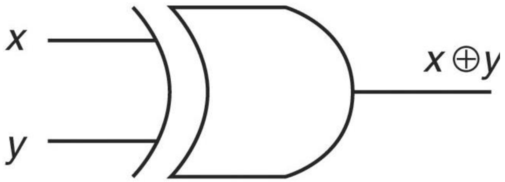


OR Gate

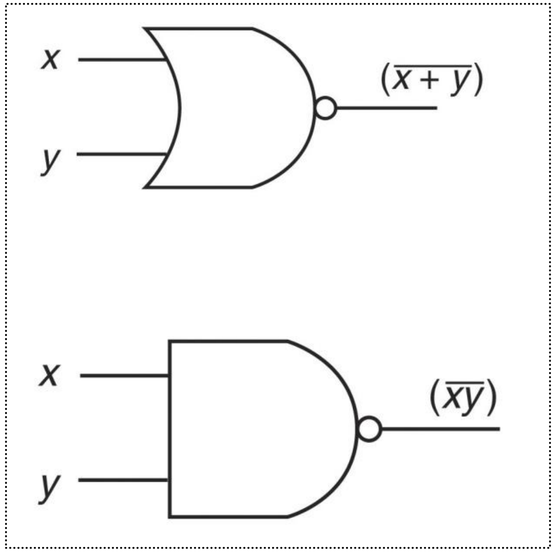


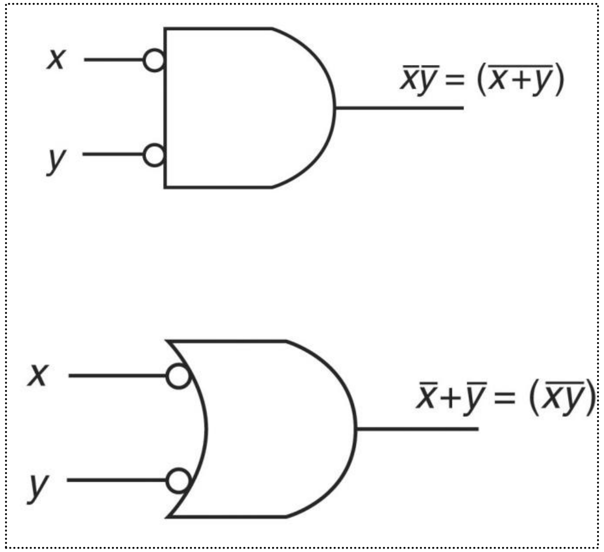
NOT Gate

Here we see the logic gates that represent the Boolean operations previously discussed



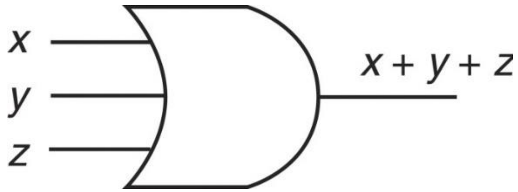
XOR looks like OR but with the added curved line





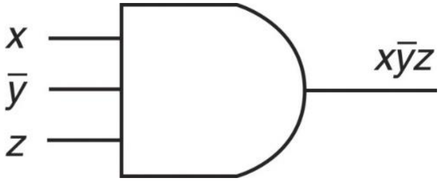
We typically represent NOR and NAND by the two on the left, but the two on the right are also correct

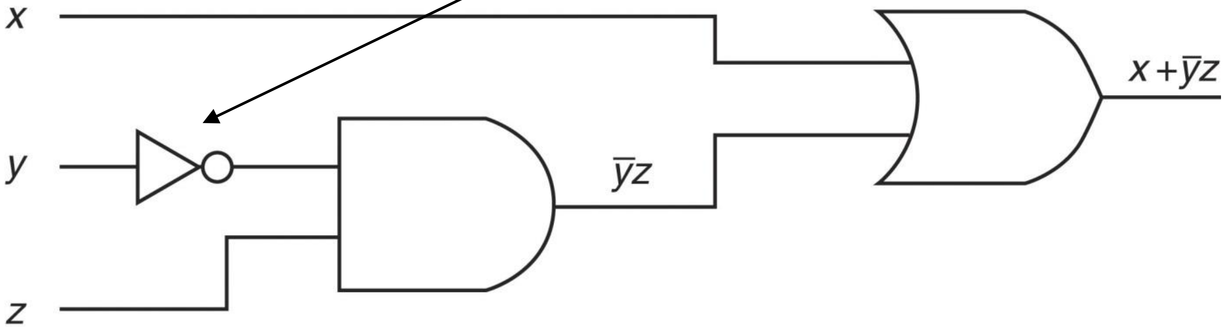
Some example Circuits



AND and OR gates can have more than 2 inputs, as seen here

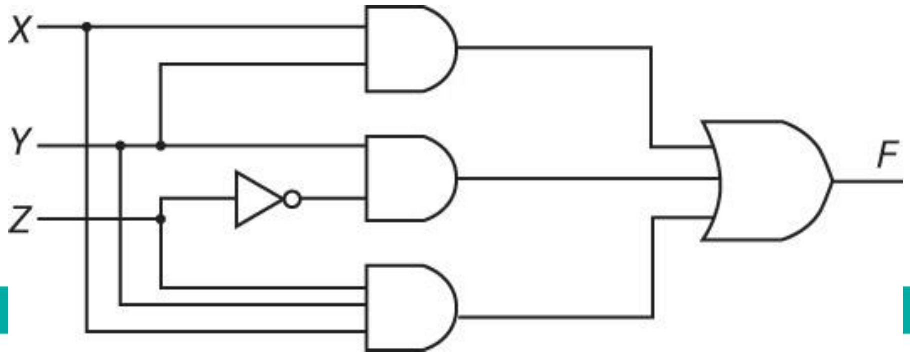
Notice for ~Y, we can either use a NOT gate or simply specify that the value is ~Y





Here is the truth table for this circuit

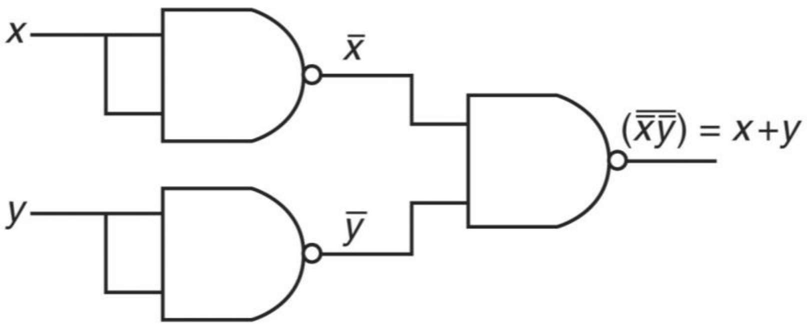
Inputs					Outputs
x	y	z	$\bar{y}$	$\bar{y}z$	$x + \bar{y}z = F$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1



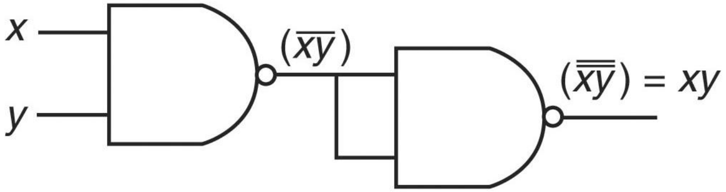
What does this circuit compute? (what is F?)

Using only NAND

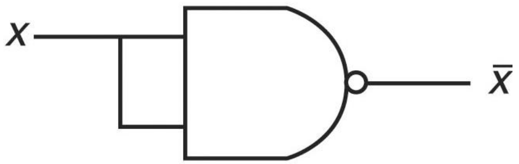
- NAND (and NOR) have unique properties different from the other boolean operations
  - This allows us to use one or more NAND gates (or one or more NOR gates) and create gates that can compute AND, OR and NOT
    - See the examples below



OR Gate

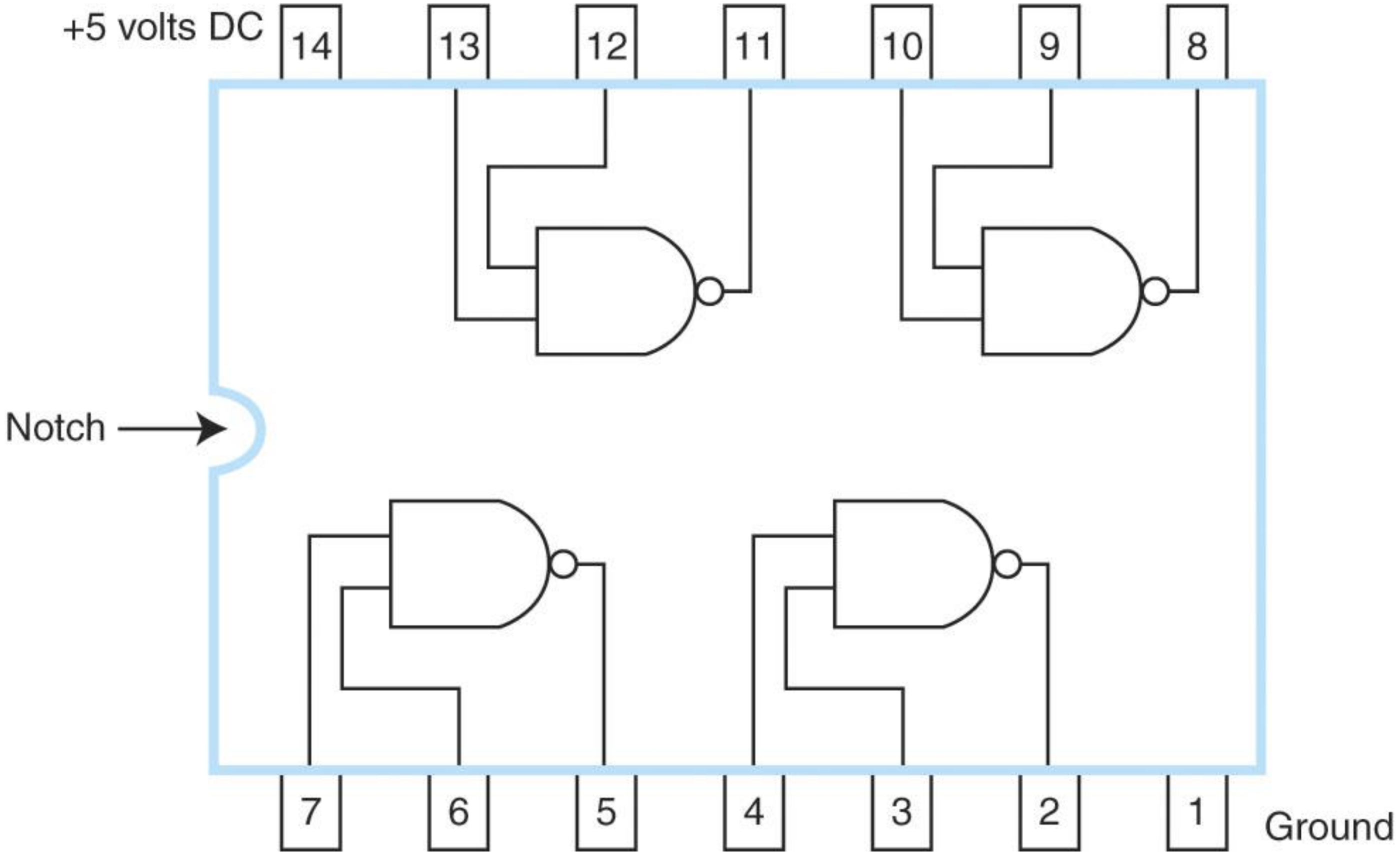


AND Gate



NOT Gate

NAND Chip



This is a NAND chip

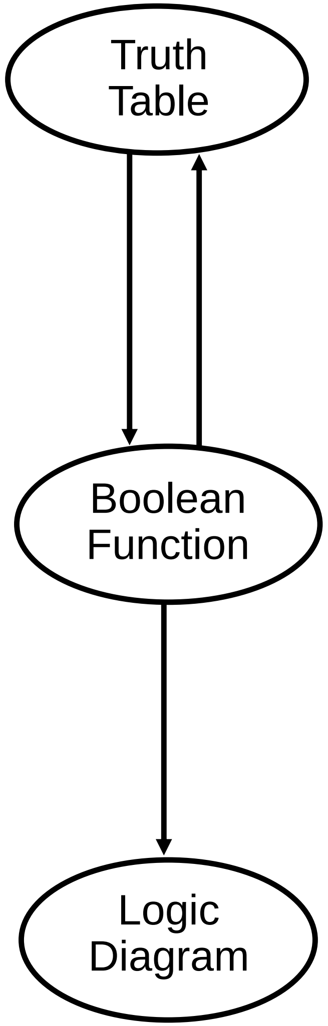
Early integrated circuits were several gates on a single chip, you would connect this chip to other chips by adding wires
between the pins

To do ~(A*B) + ~(C*D)

You would connect A and B to pins 7 and 6, C and D to 4 and 3, and send 5 and 2 to an NAND chip

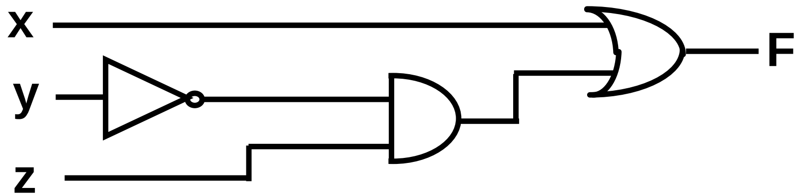
Prove? why did we use NAND

LOGIC CIRCUIT DESIGN



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

F = x + ~yz



Will continue…