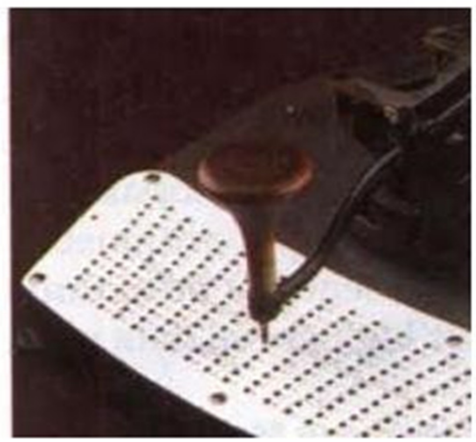
One Bit at a Time…..
said the first I.T. guy ever

CSCI 255



Apcast.com: java-basics-BitsBytes-CensusMachine.jpg

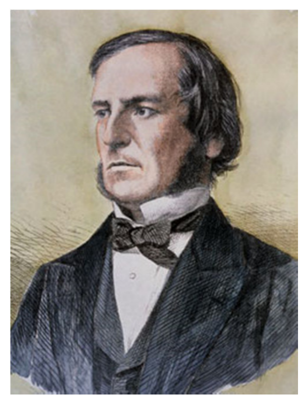
Bit Operations

- What is a bit (or logical) operation?

Is a Boolean procedure between two binary values;
in where, each binary value consists of a single or more bits

- What is a Boolean procedure?

Derived from George Boole.
Boolean is the data type on binary values: on/off, true/false,…discrete
Boolean procedures are how the binary values are evaluated through logical expressions



Without him….
there wouldn't be any
Apple stores

- The Big 3: The basic logical operators/expressions:

AND => Can something be true AND false?
OR => To Be OR Not To Be…
NOT => What is something that is NOT true?

- The logic is derived from plain language to come up with the logical answer
- This is applied to Binary values (voltage values) to work in digital circuits
- Therefore, digital logic => revolution

- How do the logical operators work?

Let's take a plain English sentence:

"There are apples and oranges in each basket"

If a customer checks one basket and sees only apples…and if we assigned:

- Zero/False/off = Not in the basket
- One/True/on = In the basket

, then:

Apples = 1, Oranges = 0

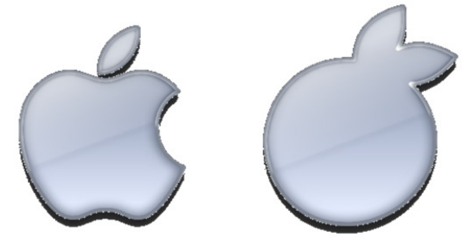
; therefore, the expression for the customer checking that one basket is:

1 AND 0 = False

There are apples in that one basket

There aren't any oranges in that one basket

In conclusion, the above statement is False



Identical sentence:

"There are apples or oranges in each basket"

Customer checks one basket and sees only oranges this time

- Zero/False/off = Not in the basket
- One/True/on = In the basket

, then:

Apples = 0, Oranges = 1

; therefore, the expression for the customer checking that one basket is:

There aren't any apples in that one basket

0 OR 1 = True

There are oranges in that one basket

In conclusion, the above statement is True

The AND-logic:

- The AND operator is used by comparing two values….. so there are four different outcomes:

Scenario	The logic	Outcome	Logic expression
False AND False	Something is false and false	Then, it is false	0 • 0 = 0
False AND True	Something is false and true	Then, it is false	0 • 1 = 0
True AND False	Something is true and false	Then, it is false	1 • 0 = 0
True AND True	Something is true and true	Then, it is true	1 • 1 = 0

The OR-logic:

- The OR operator is used by comparing two values….. so there are four different outcomes:

Scenario	The logic	Outcome	Logic expression
False OR False	Something is false or false	Then, it is false	0 + 0 = 0
False OR True	Something is false or true	Then, it is true	0 + 1 = 1
True OR False	Something is true or false	Then, it is true	1 + 0 = 1
True OR True	Something is true or true	Then, it is true	1 + 1 = 1

The NOT-logic:

- The NOT operator is used by on a single value.....
  so there are two different outcomes:

Scenario	The logic	Outcome	Logic expression
NOT False	Something is not false	Then, it is true	$\overline{0} = 1$
NOT True	Something is not true	Then, it is false	$\overline{1} = 0$

- If you double NOT a single bit – the NOTs cancel out!!

- In combining the 3 logic operators, logical expressions represent the decision process of a circuit/program/computers/embedded systems/etc…

A simple logical expression:

$$(1+1)+\overline{1}\bullet(0\bullet 1)\bullet(0+1)+(0\bullet 1)\bullet\overline{0}$$

$$(1)+0\bullet(0)\bullet(1)+(0)\bullet 1$$

$$(1)+0\bullet(1)+0$$

$$(1)+0+0$$

$$1+0$$

$$1$$

Remember precedence:
1. Perform operations in parenthesis
2. Compliments (NOTs)
3. “Divide/Multiply” (ANDs)
4. “Add/Subtract” (ORs)

- The “+”s are ORs, not addition operations
- The “•”s are ANDs, not multiplication operations

Therefore, the logical expression is **TRUE**

Bit Operations

There's a faster way using same expression:

$$\underbrace{(1+1)}_{(1)} + \underbrace{\overline{1} \bullet (0 \bullet 1) \bullet (0+1) + (0 \bullet 1) \bullet \overline{0}}_{\textit{Anything}}$$

$$(1) + \textit{Anything}\ (\textit{either}\ 0\ \textit{or}\ 1)$$

$$1$$

the logical expression is always **TRUE**

You don't need to evaluate the whole thing if the "1" is OR-ed with any of the two binary values….it will always outcome to "1"

- Although ones and zeros on a logical expression can be reduced to a single outcome, the more complex (most useful) logical expression consists of variables

- Logical expressions with variables describe digital connections within a circuit

- Example of logical expression with variables may look like:

$$(X + Y)\bullet Z+(W\bullet Y)$$

, in where each variable (W, X, Y, Z) can have the binary value 0 or 1

- In order to evaluate this type of logical expression, Boolean algebra is used.

- In Boolean algebra, theorems were formed in order to better evaluate logical expression.

- Example of simple theorem:

$X \bullet 1$ Theorem 1D

When $X = 0$:

$$0 \bullet 1 = 0$$

When $X = 1$:

$$1 \bullet 1 = 1$$

Outcome always equals to the value of X in both instances; therefore:

$$X \bullet 1 = X$$

- Depending on the literature, different notations can be found:
- The logic NOT is A.K.A the COMPLEMENT.
- Some of the notations are:

$$\overline{X} = X'$$

- The logic AND can also have notations as:

$$X \bullet Y \bullet Z = XYZ$$

$$X \bullet (Y \bullet Z) = X(YZ) = XYZ$$ Theorem **7D**

DeMorgan's Theorem (12 & 12D) vs Duality Theorem (13 & 13D)

- DeMorgan's states (in my own words):

When you complement the expression within a parenthesis, each variable is complemented; as well as, the logical operation between variables.

- Duality states (in my own words):

When you complement the expression within a parenthesis, each variable is not complemented; however, the logical operation between variables is complemented.

Meaning:

- The complement of an OR is an AND
- The complement of an AND is an OR

Let's see an example of reducing a logical expression with variables:

$$AB'C' + CD' + BC'D'$$

Apply Theorem 8: X = D', Y=C & Z=BC'

$$AB'C' + D'(C + BC')$$

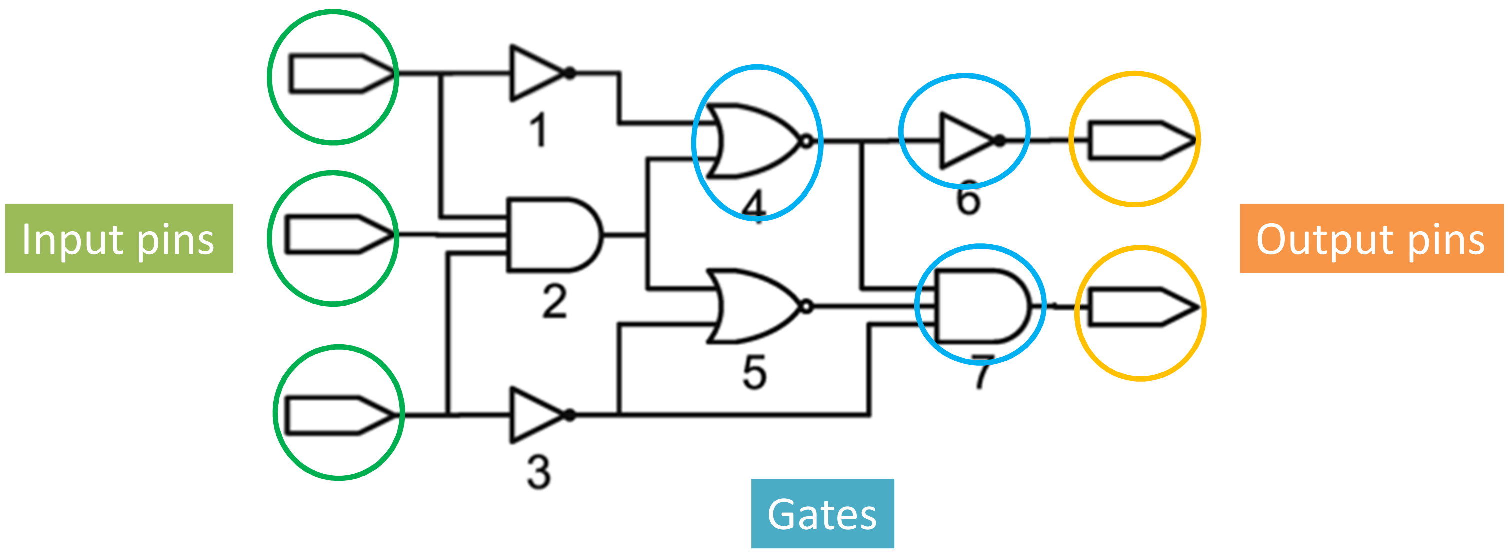
Apply Theorem 11D: X = B, Y=C & Y'=C'

$$AB'C' + D'(C + B)$$

Apply Theorem 8 once more

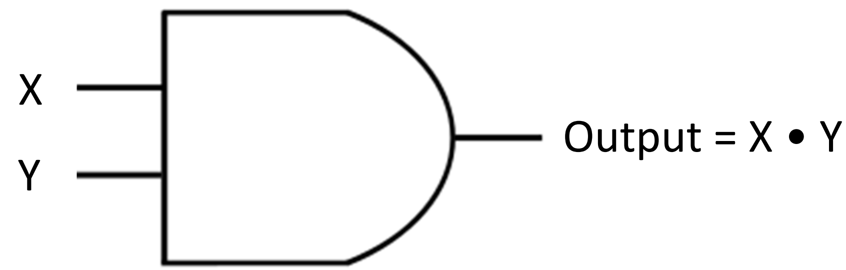
$$AB'C' + CD' + BD'$$

- Previously mentioned, the logical expression describes digital connections within a digital circuit.
- Digital circuitry diagrams consists of symbols (components/gates) that describe functionality/response/decision within its paths.
- A digital circuit diagram may look like this:



AND-GATE

- The big 3 have digital gate representation: First the AND
- We represent the AND-operation with the AND-Gate symbol:

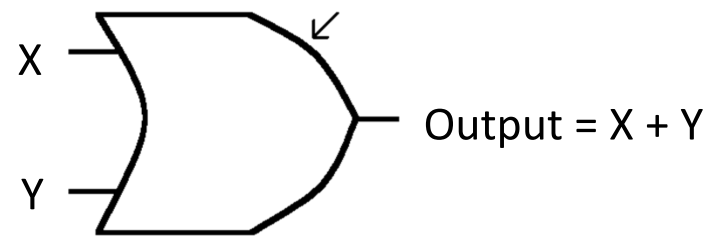


- With it's Truth Table:

INPUT 1 (X-value)	INPUT 2 (Y-value)	OUTPUT (x AND y)
0	0	0
0	1	0
1	0	0
1	1	1

OR-GATE

- We represent the OR-operation with the OR-Gate symbol:

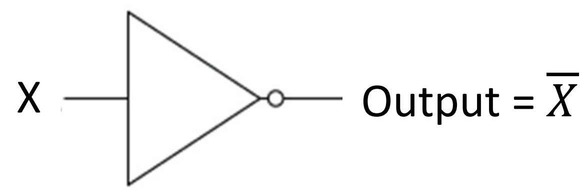


- With it's Truth Table:

INPUT 1 (X-value)	INPUT 2 (Y-value)	OUTPUT (x OR y)
0	0	0
0	1	1
1	0	1
1	1	1

NOT-GATE

- We represent the NOT-operation with the NOT-Gate (INVERTER) symbol:



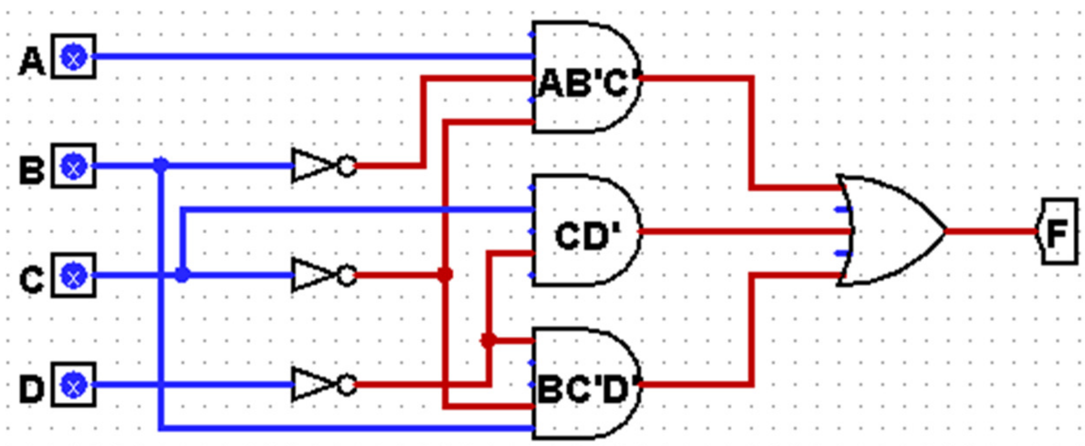
- With it's Truth Table:

INPUT (X-value)	OUTPUT (NOT x)
0	1
1	0

Therefore, when a logical expression is used:

$$F = AB'C' + CD' + BC'D'$$

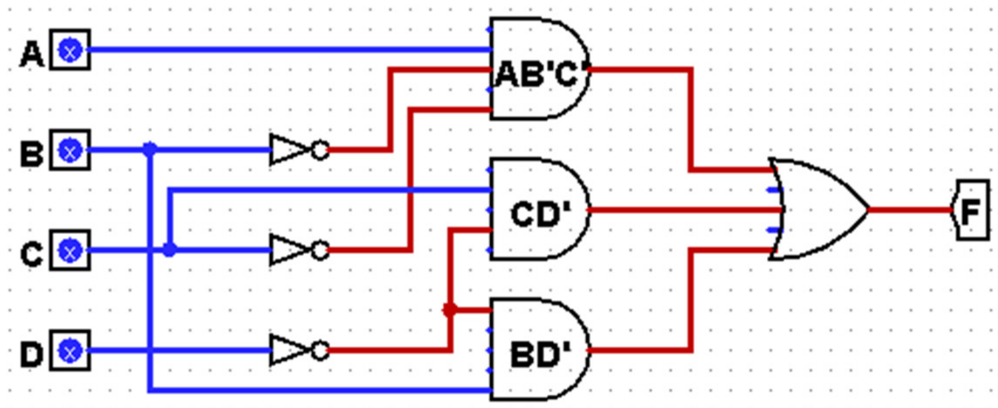
It can be integrated as logical digital circuit that looks like:



However by using the Theorems, the expression was reduced:

$$F = AB'C' + CD' + BD'$$

Which make the circuit change to:



- It is very important and useful to reduce the logic to its minimum expression
- The less number of variables in the expression translates to:
  - Less number of wires for connections
  - Less number of inputs to the gates
  - Less bits of information to compute
  - More power friendly to the overall digital circuit