Week 0011
Boolean algebra and Logic gates

CCS1310

Computer Systems Architecture

Dr. K.Dimopoulos

OVERVIEW OF THE LECTURE

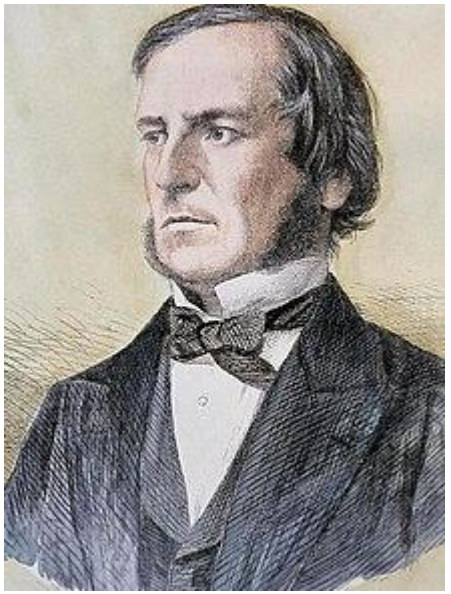
- Boolean algebra
- Boolean Functions
- Logic gates
- Logic gate families
- K-maps

Rules and Law of

BOOLEAN ALGEBRA

BOOLEAN ALGEBRA

- Boolean Algebra was invented by **George Boole** and described in his first book known as **The Mathematical Analysis of Logic** in the year **1847**. Further, he made several laws which were described by him in his second book known as **An investigation of the Laws of Thought** in the year **1854**.
- As the word Boolean is prefixed with the word ‘bool’ which implies a Boolean value which could either be **true** or **false**. In this case it is referred to as 0 and 1.
- Boolean algebra is how computers think!



THE 4 MAIN BOOLEAN OPERATORS

- NOT: Given a Boolean A, NOT(A) is the opposite of A. Usually we use a bar on top of the Boolean, or a prime after it. A' is read as "not A"
- AND: Given 2 Booleans (A and B), A AND B is true if **both** A and B are true. Usually we use a dot "." to show the operation. A.B is read as "A and B"
- OR: Given 2 Booleans (A and B), A OR B is true if **either or both** A or B are true. Usually we use a cross "+" to show the operation. A+B is read as "A or B"
- XOR: Given 2 Booleans (A and B), A XOR B is true if **either but not both** A or B are true. Usually we use a cross in a circle ⊕ to show the operation. A ⊕ B is read as "A xor B"

RULES OF BOOLEAN ALGEBRA

- **Null rule.** ADDing with 1 gives a 0. ORing with 0 gives 1.
  - A + 1 = 1 **and also** A . 0 = 0
- **Identity rule.** When 0 is added i.e., ‘OR’ and when 0 is ‘OR’ with a variable it gives the variable back.
  - A + 0 = A **and also** A . 1 = A
- **Complement rule.** Whenever ‘AND’ or ‘OR’ is used with the negation of the variable and the variable itself it gives 0 and 1 respectively.
  - A + A' = 1 **and also** A . A' = 0
- **Involution rule.** Any variable when double negated gives the variable back.
  - (A')' = A

LAWS of BOOLEAN ALGEBRA

- Commutative law:
  - A + B = B + A **and also** A . B = B . A
- Associative law:
  - (A + B) + C = A + (B + C) **and also** (A . B) . C = A . (B . C)
- Idempotence law:
  - A + A = A **and also** A . A = A
- Distributive law:
  - A (B + C) = A.B + A.C **and also** A + (B.C) = (A + B)(A + C)
- Redundance law:
  - A + A.B = A **and also** A(A + B) = A
- De Morgan's law
  - (A + B)' = (A'B') **and also** (AB)' = (A' + B')

BOOLEAN FUNCTIONS

WHAT ARE BOOLEAN FUNCTIONS

- In mathematics, a function is an expression of variables. The function can be evaluated, if we are given values for the variables. E.g.
  - f(x,y) = 3x+y, then for x=2 and y=3, f(2,3) = 3.2+3 = 9
- A **Boolean function** is a mathematical expression consisting of **Boolean** variables **combined using the Boolean algebra** operators: logical addition (OR), multiplication (AND) and negation (NOT) is a Boolean function.
  - f(A, B, C) = (A+B')+C.(A+B) , then
  - if A=true, B and C are false
  - f(A,B,C) = (true+false')+false.(true+false) = (true)+false.true = true

IMPORTANT CONCEPTS

- **Literal:** A logic variable or its complement
  - e.g. A, B, C’
- **Product term:** An expression where literals are combined by the logical AND operator
  - e.g. A.B.C’
- **Sum term:** An expression where literals are combined by the logical OR operator
  - e.g. A’+B+C’
- **Normal term:** A (product or sum) term without repeated variables
  - e.g. A+B.C
- **Sum of Products (SoP):** A sum of product terms
  - e.g. A+A’.B+A.C
- **Product of Sums (PoS):** A product of sum terms
  - e.g. (A+B).(A’+C).(A+B’+C’)

EXPANSION THEOREM

- The *Shannon expansion* or *decomposition* theorem, also known as *Boole's expansion* theorem is an identity which allow the expansion of any logic function to broken down in parts. There are 2 variations:
  1. f(A, B, C, ...Z) = A. f(1, B, C, ...Z) + A'. f(0, B, C, ...Z)
  2. f(A, B, C, ...Z) = (A+ f(0,B,C,...Z)) . (A'+ f(1,B,C,...Z))
  3. f(A, B, C, ...Z) = A. f(1, B, C, ...Z) $\oplus$ A'. f(0, B, C, ...Z)

LOGIC GATES

COMPUTER HARDWARE BASICS

- Many complex operations can be created using simple Boolean logic
- These simple logical operations are used to perform arithmetic and logical operations in the CPU
- Bit-wise logic operations are used to construct more complex mathematical expressions
- But, how do we perform logic operations with a machine?
- Switches!

COMPUTER HARDWARE BASICS

- Remember, the fundamental part of the digital computer is the switch
- A switch simply lets current flow completely or not at all (binary)
  - Relay: mechanical switch
  - Vacuum tubes: electronic switch
  - Transistors: electronic switch
  - ICs: use transistors on a circuit board
- Speed of computer determined by how fast a switch "switches"!

DIGITAL LOGIC

- Combinatorial Logic:
  - Logic in which the results of an operation depend only on the present inputs to the operation
  - Example: arithmetic operations

- Sequential Logic:
  - Digital logic that is dependent on the previous state of an operation
  - Example: counter

COMBINATORIAL CIRCUITS

- A Combinatorial Circuit is an interconnected set of gates whose output at any time is a function only of the input at that time

- The three most common methods of describing the behavior of a combinatorial circuit are:
  - Truth tables
  - Boolean Algebraic Expressions
  - Logic Diagrams

TRUTH TABLES

- They are at a higher level of abstraction than algebraic expressions or logic diagrams
- Specify what the combinatorial circuit does and not how it does it
- List the output for every possible combination of input values

A	B	F
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

A	B	C	F
FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE
TRUE	TRUE	TRUE	FALSE

For n inputs there are 2^n entries in the truth table

DE MORGAN'S THEOREM

How can we show that: $(X + Y)' = X' \cdot Y'$

x	y	x+y	(x+y)′	x′	y′	x′ . y′
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

First calculate the Left Hand Term

Then calculate the Right Hand Term

LOGIC GATES

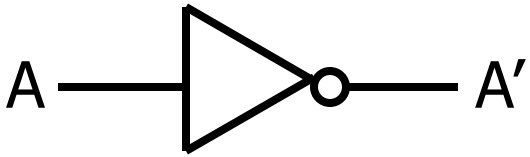
- Boolean logic in the computer is implemented by using electronic circuits called gates
- Gates are constructed from transistor switches and other electronic components, formed into integrated circuits
- A gate produces an output signal that is a simple operation on its input signal
  - This means that different inputs produce different outputs
  - Different gates produces different output patterns

NOT LOGIC GATE

NOT Truth Table

A	A’
0	1
1	0

NOT Gate Symbol



NOT Boolean Symbol

A’ but also $\overline{A}$

Also called the **Inverter**. It changes LOW to HIGH and HIGH to LOW.
Can you think where the inverter is very useful?

AND LOGIC GATE

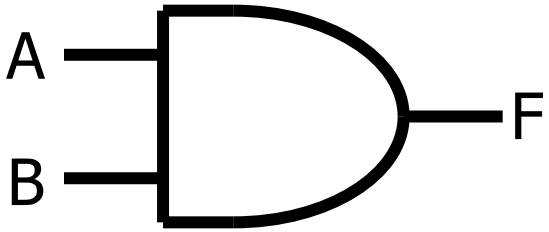
AND Truth Table

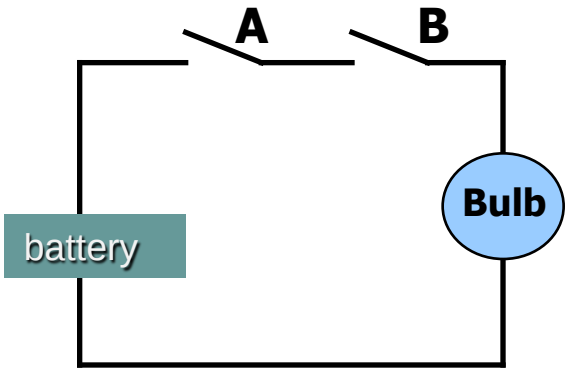
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

AND Boolean Symbol

A●B

AND Gate Symbol





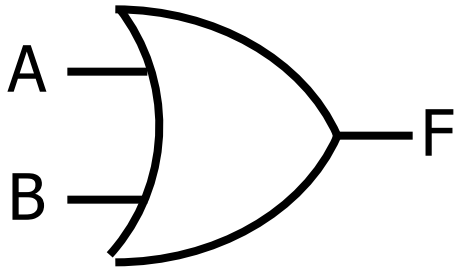
Both A and B switches must be ON for bulb to light up.

OR LOGIC GATE

OR Truth Table

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

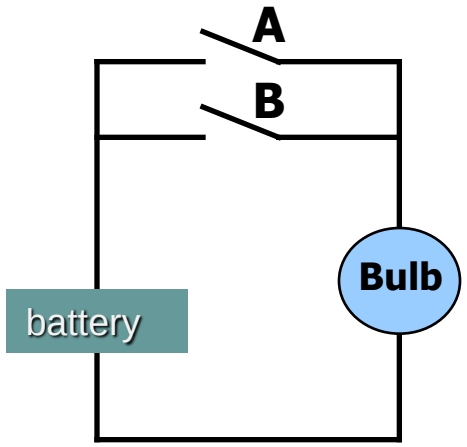
OR Gate Symbol



OR Boolean Symbol

A+B

Either A or B switches must be ON for bulb to light

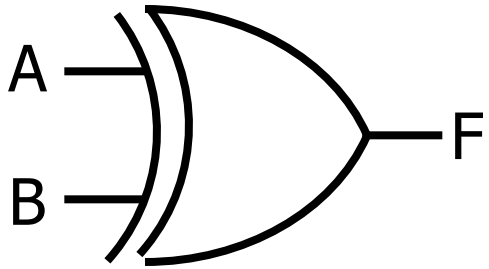


XOR LOGIC GATE

XOR Truth Table

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate Symbol



XOR Boolean Symbol

$A \oplus B$

XOR is a very important function that is used extensively in cryptography

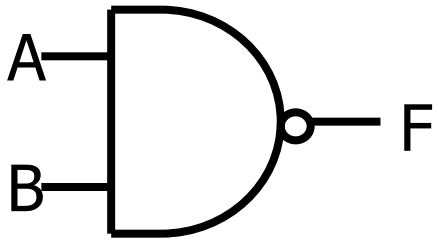
NAND AND NOR GATES

NAND and NOR gates are very important in digital circuits. They are considered universal gates, as any logic equation may be implemented using NAND or NOR gates only. They are also:

- Simplest and cheapest to fabricate
- Fastest operating speed
- Lowest power dissipation

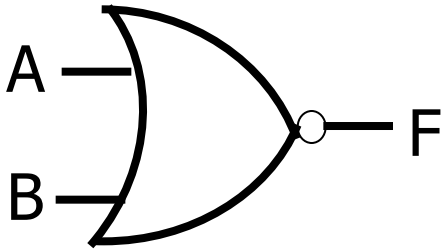
NAND Truth Table - Gate Symbol

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

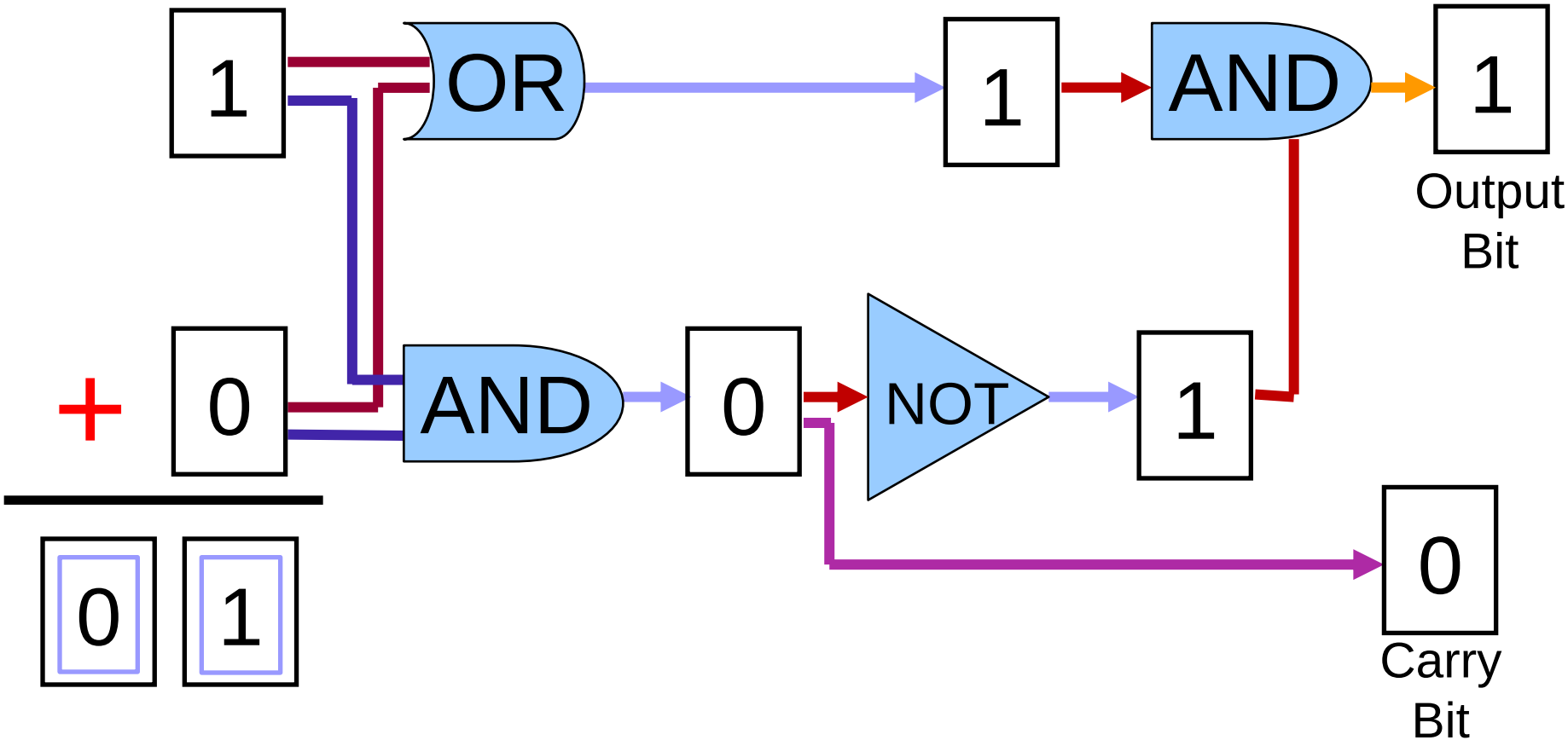


NOR Truth Table - Gate Symbol

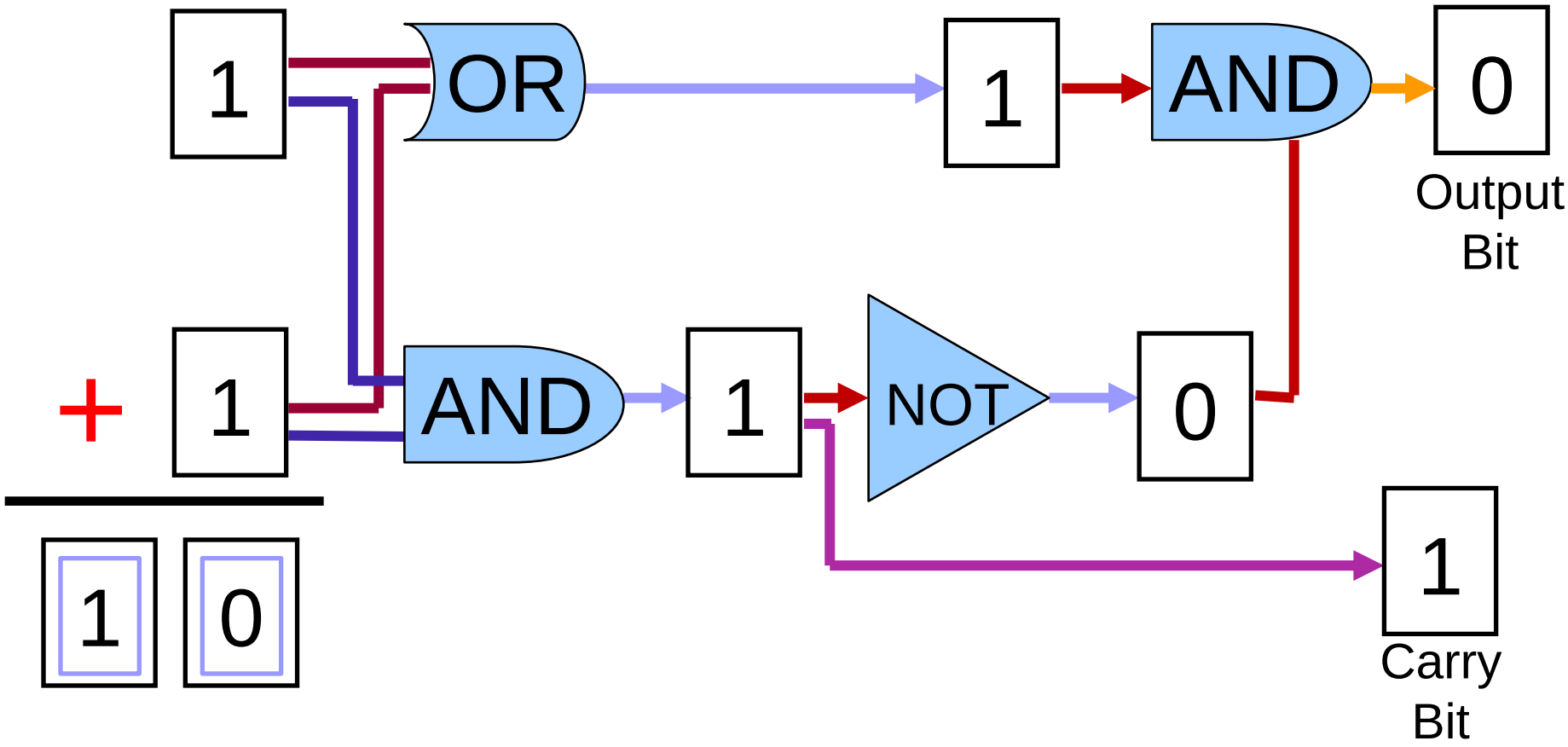
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



HOW CAN BOOLEAN CIRCUITS ADD?



HOW CAN BOOLEAN CIRCUITS ADD?



SUMMARY

Computers are built from integrated circuits. Integrated circuits are made up of transistors, resistors, capacitors. Transistors can act as amplifiers or switches.

ON and OFF positions of transistor switches represent 1's and 0's of binary digital circuits.

Digital circuits are used to perform arithmetic, to control the movement of data, to compare values for decision making.