Introduction to Digital Logic

CS 64: Computer Organization and Design Logic

Lecture #11

Winter 2020

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Administrative

- Lab 6 out today
  - Due by next week **Thursday**
    - Extra time to finish it (it's challenging)

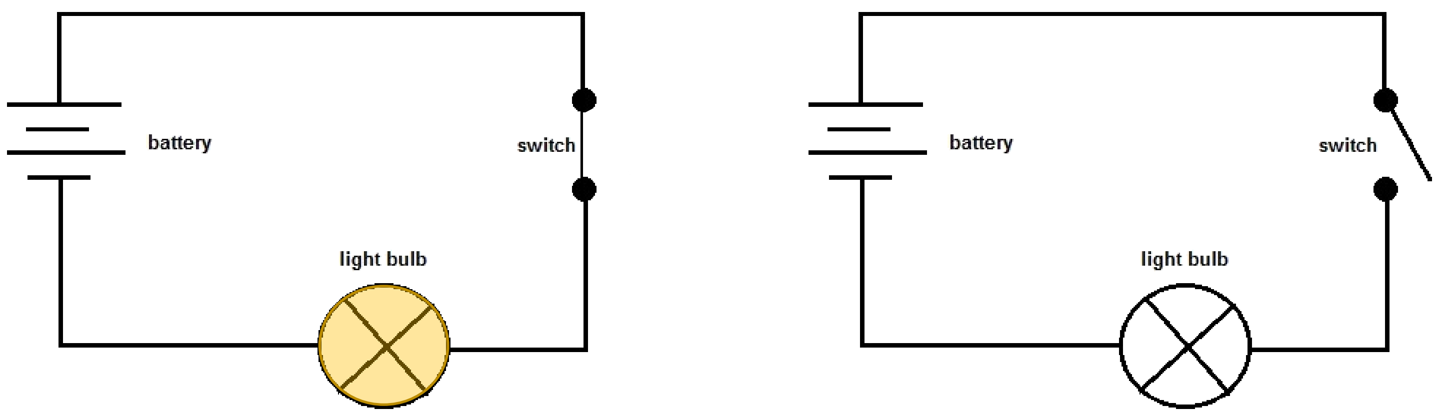
- Midterm Exam grades will be posted by the weekend

Lecture Outline

- Intro to Binary (Digital) Logic Gates
- Truth Table Construction
- Logic Functions and their Simplifications
- The Laws of Binary Logic

Digital i.e. Binary Logic

- Electronic circuits when used in computers are a series of switches
- 2 possible states: either ON (1) and OFF (0)



- **Perfect for binary logic representation!**

Basic Building Blocks of Digital Logic

- Same as the bitwise operators:

NOT

AND

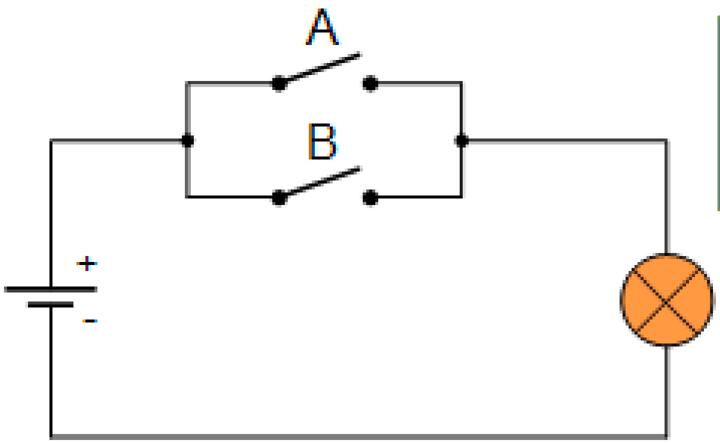
OR

XOR

etc...

- We often refer to these as "**logic gates**" in digital design

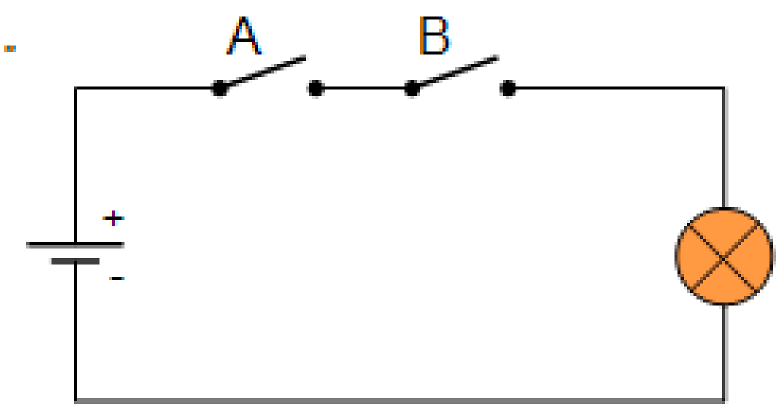
Electronic Circuit Logic Equivalents



OR

Lamp - ON = "1"
Lamp - OFF = "0"

Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"

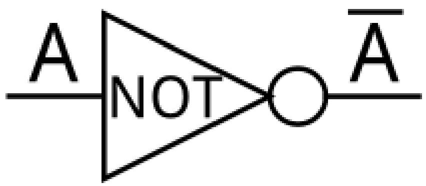


AND

Lamp - ON = "1"
Lamp - OFF = "0"

Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"

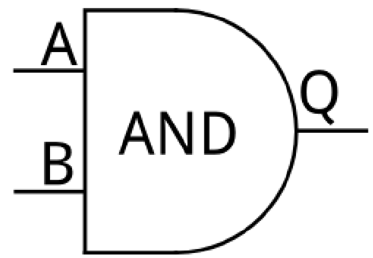
Graphical Symbols and Truth Tables
NOT



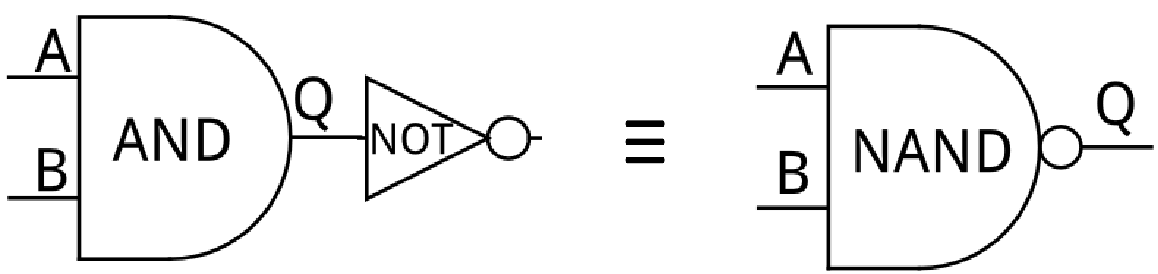
A	$\overline{A}$ or !A
0	1
1	0

Graphical Symbols and Truth Tables
AND and *NAND*

Practice Drawing the Symbol!



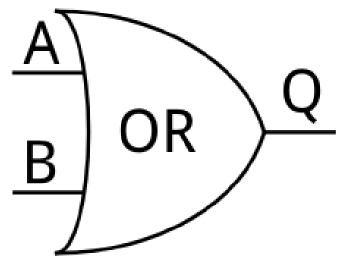
A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1



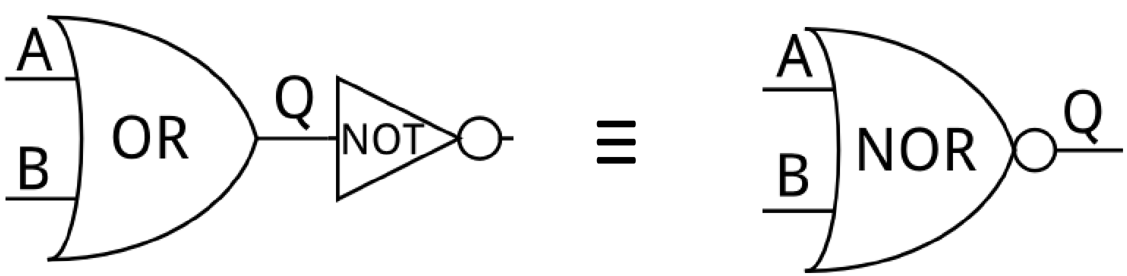
A	B	$\overline{A . B}$ or !(A.B)
0	0	1
0	1	1
1	0	1
1	1	0

Graphical Symbols and Truth Tables
OR and *NOR*

Practice Drawing the Symbol!



A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

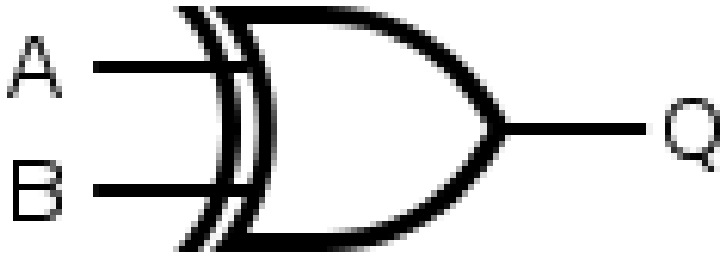


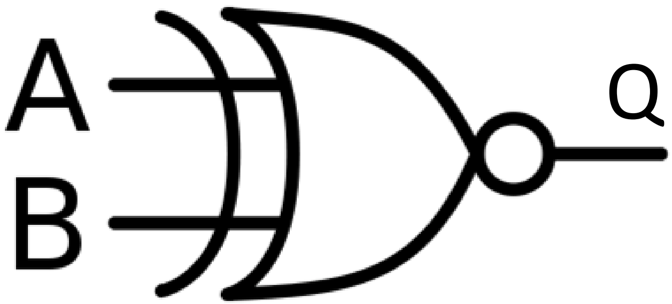
A	B	$\overline{A + B}$ or !(A + B)
0	0	1
0	1	0
1	0	0
1	1	0

Graphical Symbols and Truth Tables
XOR and XNOR

Practice Drawing the Symbol!

XOR





XNOR

A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Constructing Truth Tables

- T.Ts can be applied to ANY digital circuit
- They show ALL possible inputs with ALL possible outputs
- Number of entries in the T.T.

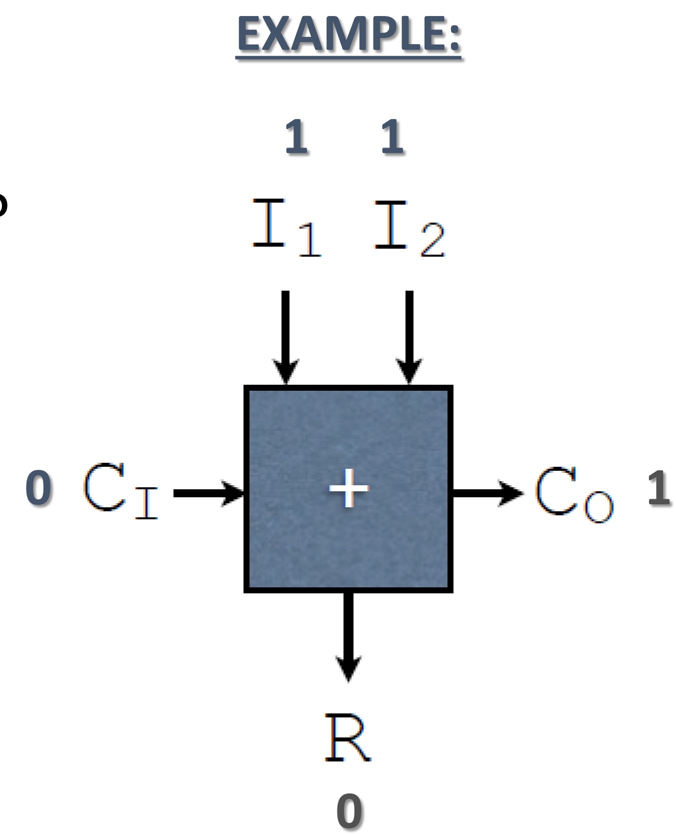
  = $\mathbf{2^N}$, where N is the number of **inputs**

Example: Constructing the T.T of a 1-bit Adder

- Recall the 1-bit adder:

- **3 inputs**: I_1 and I_2 and C_I
  - Input1, Input2, and Carry-In
  - How many entries in the T.T. is that?

- **2 outputs**: R and C_O
  - Result, and Carry-Out
  - You can have multiple outputs: **each** will still depend on *some combination* of the inputs



Example: Constructing the T.T of a 1-bit Adder

T.T Construction Time!

Example: Constructing the T.T of a 1-bit Adder

Note the order of the inputs!!!

	INPUTS			*OUTPUTS*	
#	**I1**	**I2**	**CI**	**CO**	**R**
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Logic Functions

- An **output function F** can be seen as
  a *combination* of 1 or more inputs
- Example: F = A . B + C (all single bits)

- This is called **combinatorial logic**

Equivalent in C/C++:

```
boolean f (boolean a, boolean b, boolean c)
{
    return ( (a & b) | c );
}
```

OR and AND as Sum and Product

- Logic functions are often expressed with basic logic building blocks, like ORs and ANDs and NOTs, etc…

- OR is sometimes referred to as "logical sum" or "logical union"
  - Partly why it's symbolized as "+"
  - BUT IT'S **NOT** THE SAME AS NUMERICAL ADDITION!!!!!!

- AND as "logical product" or "logical disjunction"
  - Partly why it's symbolized as "."
  - BUT IT'S **NOT** THE SAME AS NUMERICAL MULTIPLICATION!!!!!!

Example

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

- **A XOR B** takes the value "1" (i.e. is TRUE) ***if and only if***
  - A = 0, B = 1 i.e. **!A.B** is TRUE, ***<u>or</u>***
  - A = 1, B = 0 i.e. **A.!B** is TRUE
- In other words, **A XOR B** is TRUE **iff** (if and only if) **A!B + !AB** is TRUE

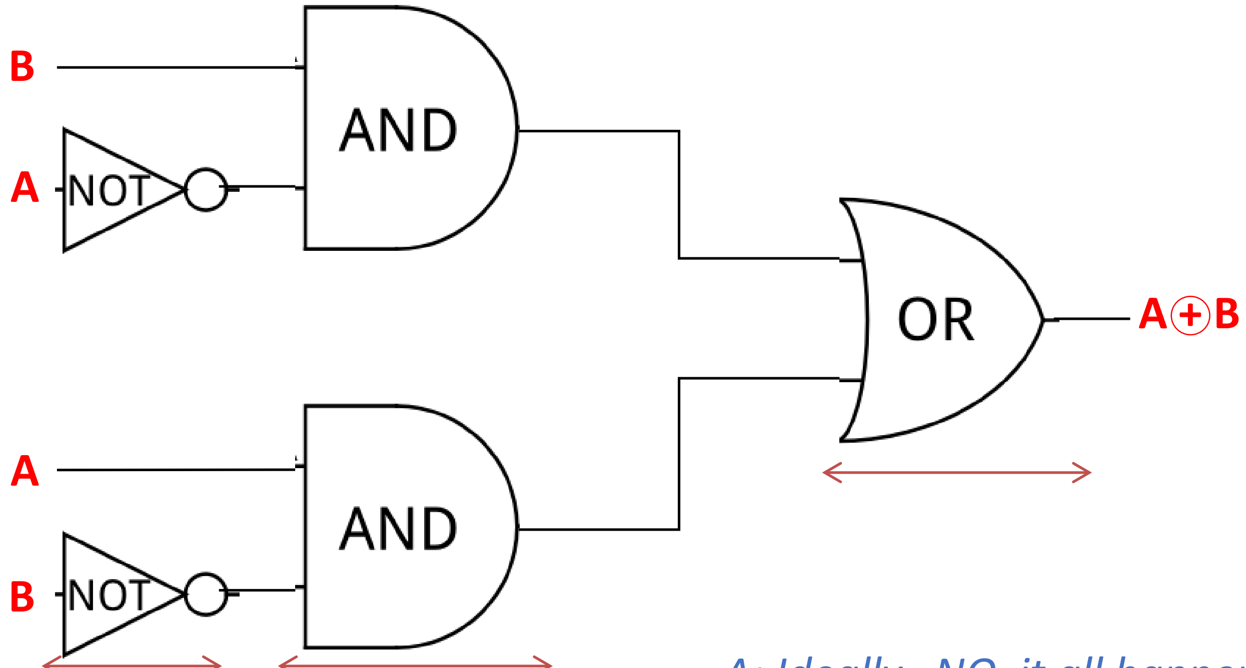
A $\oplus$ B = !A.B + A.!B

Which can also be written as: $\overline{A}.B + A.\overline{B}$

Representing the Circuit Graphically

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

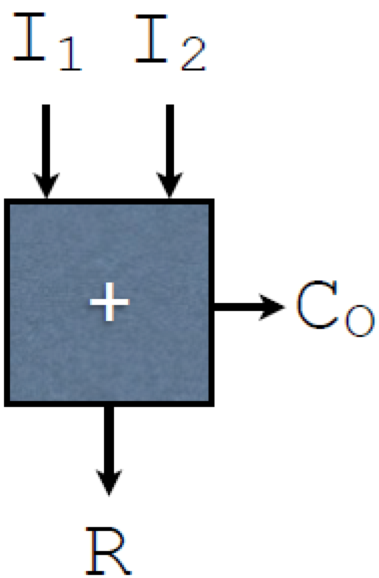
A⊕B = !A.B + A.!B



Q: Does it take any time for a electronic signal to go thru 3 "layers" of logic gates?

A: Ideally, NO, it all happens simultaneously.
*In reality, OF COURSE it takes time (it's called **latency**)*

What is The Logical Function for The **Half Adder**?



	INPUTS		OUTPUTS	
#	I1	I2	CO	R
0	0	0	0	0
1	0	1	0	1
2	1	0	0	1
3	1	1	1	0

Half Adder
1-bit adder that does not have a Carry-In (Ci) bit.

This logic block has only 2 1-bit inputs and 2 1-bit outputs

Our attempt to describe the outputs as functions of the inputs:

$$CO = I_1 \cdot I_2$$
$$R = I_1 \oplus I_2$$

What is The Logical Function for A **Full** 1-bit adder?

	INPUTS			***OUTPUTS***	
#	**I1**	**I2**	**CI**	**CO**	**R**
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Ans.: CO = !I1.I2.CI + I1.!I2.CI + I1.I2.!CI + I1.I2.CI

R = !I1.!I2.CI + !I1.I2.!CI + I1.!I2.!CI + I1.I2.CI

Minimization of Binary Logic

- Why?
  - It's MUCH easier to read and understand…
  - Saves memory (software) and/or physical space (hardware)
  - Runs faster / performs better
    - Why?... remember ***latency***?

- For example, when we do the T.T. for (see demo on board):

X = A.B + A.!B + B.!A, we find that it is the same as

A + B

(saved ourselves a bunch of logic gates!)

Using T.Ts vs. Using Logic Rules

- In an effort to simplify a logic function, we don't always have to use T.Ts – we can use *logic rules* instead

Example: What are the following logic outcomes?

A . A	A
A + A	A
A . 1	A
A + 1	1
A . 0	0
A + 0	A

Using T.Ts vs. Using Logic Rules

- Binary Logic works in **Associative** ways
  - (A.B).C *is the same as* A.(B.C)
  - (A+B)+C *is the same as* A+(B+C)

- It also works in **Distributive** ways
  - (A + B).C *is the same as:* **A.C + B.C**
  - (A + B).(A + C) *is the same as:*

A.A + A.C + B.A + B.C

= A + A.C + A.B + B.C

= A + B.C

More Examples of Minimization
a.k.a Simplification

- Simplify: R = A.B + !A.B

Let's verify it with a truth-table

= (A + !A).B

= B

Note: often, the AND dot symbol (.) is omitted, but understood to be there (like with multiplication dot symbol)

- Simplify: R = !A**BCD** + A**BCD** + !A**B!CD** + A**B!CD**

= BCD(A + !A) + !AB!CD + AB!CD

= BCD + B!CD(!A + A)

= B**C**D + B**!C**D

= BD(C + !C)

= BD

Let's verify it with a truth-table

More *Simplification* Exercises

- Simplify: R = !A!BC + !A!B!C + !ABC + !AB!C + A!BC
  = !A!B(C + !C) + !AB(C + !C) + A!BC
  = !A!B + !AB + A!BC
  = !A (!B + B) + A!BC
  = !A + A!BC

You can verify it with a truth-table

- Reformulate using **only** AND and NOT logic:
  R = !AC + !BC
  = C (!A + !B)
  = C. !(A.B) ← *De Morgan's Law*

Important: Laws of Binary Logic

Circuit Equivalence - each law has 2 forms that are duals of each other.

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

More Simplification Examples

Simplify the Boolean expression:

- **(A+B+C).(D+E)' + (A+B+C).(D+E)**

Simplify the Boolean expression and write it out on a truth table as proof

- **X.Z + Z.(X'+ X.Y)**

Use DeMorgan’s Theorm to re-write the expression below using at least one OR operation

- **NOT(X + Y.Z)**

Scaling Up Simplification

- When we get to *more* than 3 variables, it becomes challenging to use truth tables

- We can instead use ***Karnaugh Maps*** to make it immediately apparent as to what can be simplified

Your To-Dos

- Start Lab 6 on Thursday

</LECTURE>