

Introduction to Digital Logic

CS 64: Computer Organization and Design Logic
Lecture #11

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Re: Midterm Exam #1
- Re: Midterm Exam #2
 - Next Thursday!
 - Everything from **lectures 7 thru 12**

Administrative

- Lab# 6 (next week) are exercises that do not require a computer
 - You are *excused* from going to Monday lab next week, but T.As will be there to help, if you need it
 - **Material we cover today and on Tuesday will be necessary to finish Lab# 6**
- Lab# 6 will still be due end of day Friday
- Lab #7 will be different...!

Administrative

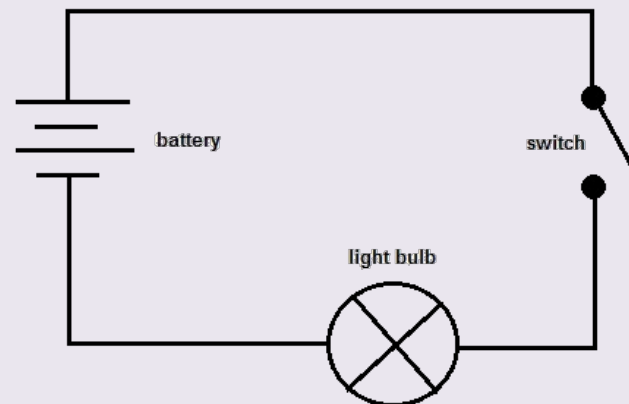
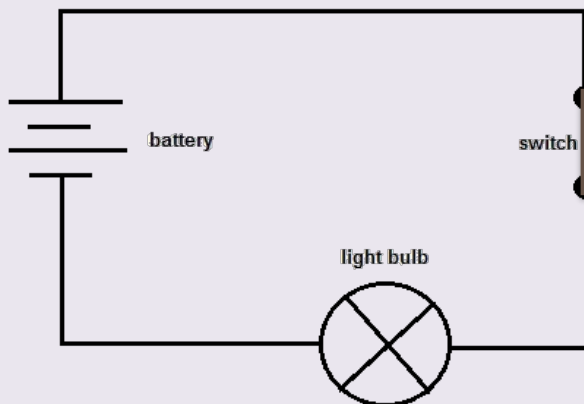
- Mid-quarter evaluations for T.As and for Prof.
 - Links on Piazza
 - Optional to do, but very appreciated by us all!
 - Deadline is Monday

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:

AND

OR

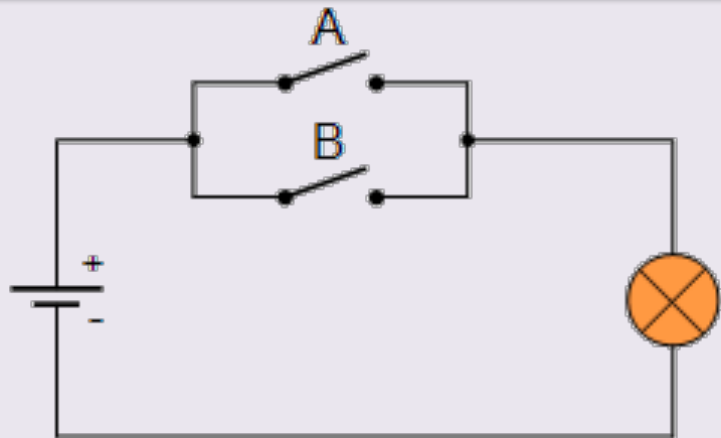
XOR

NOT

etc...

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

Electronic Circuit Logic Equivalents

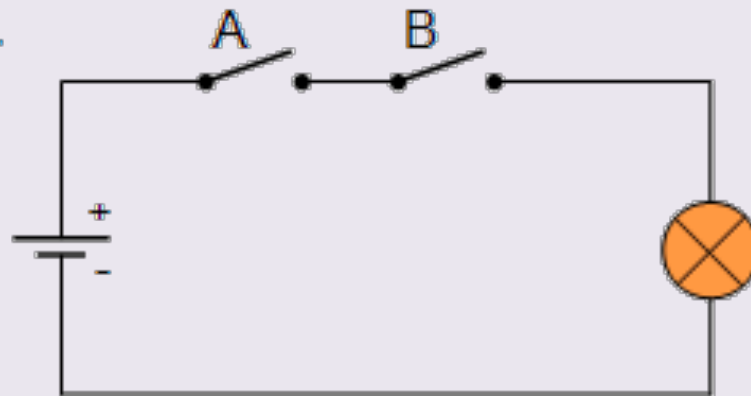


OR

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

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

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



AND

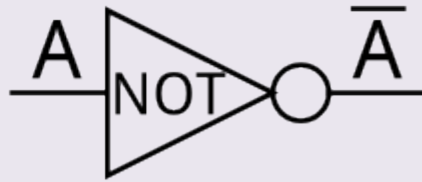
Lamp - ON = "1"
Lamp - OFF = "0"

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

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

Graphical Symbols and Truth Tables

NOT

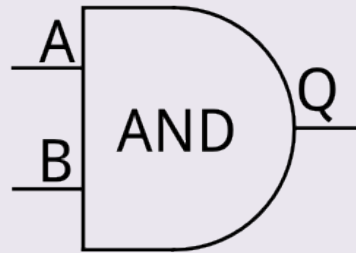


A	\bar{A} or !A
0	1
1	0

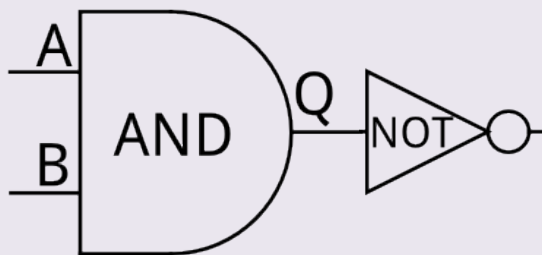
Graphical Symbols and Truth Tables

AND and NAND

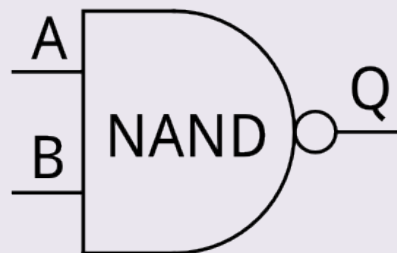
Practice Drawing
the Symbol!



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



≡

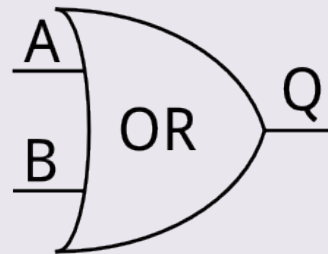


A	B	$\overline{A . B}$ or $!(A.B)$
0	0	1
0	1	1
1	0	1
1	1	0

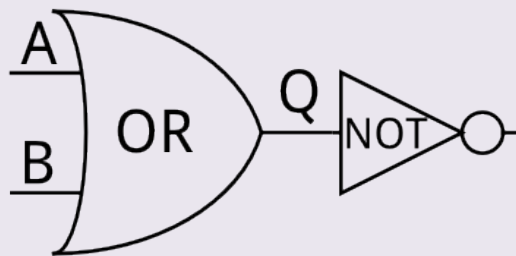
Graphical Symbols and Truth Tables

OR and NOR

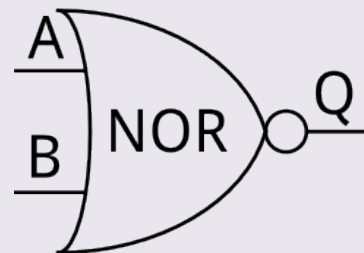
Practice Drawing
the Symbol!



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



≡

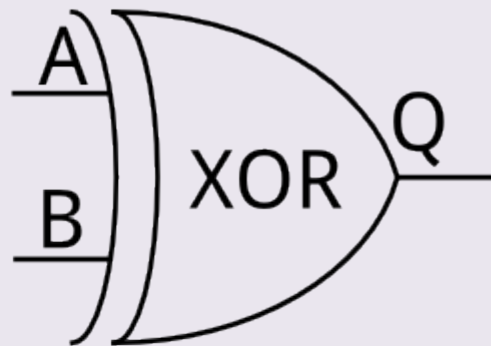


A	B	$\overline{A + B}$ or $\neg(A + B)$
0	0	1
0	1	0
1	0	0
1	1	0

Graphical Symbols and Truth Tables

XOR

Practice Drawing
the Symbol!



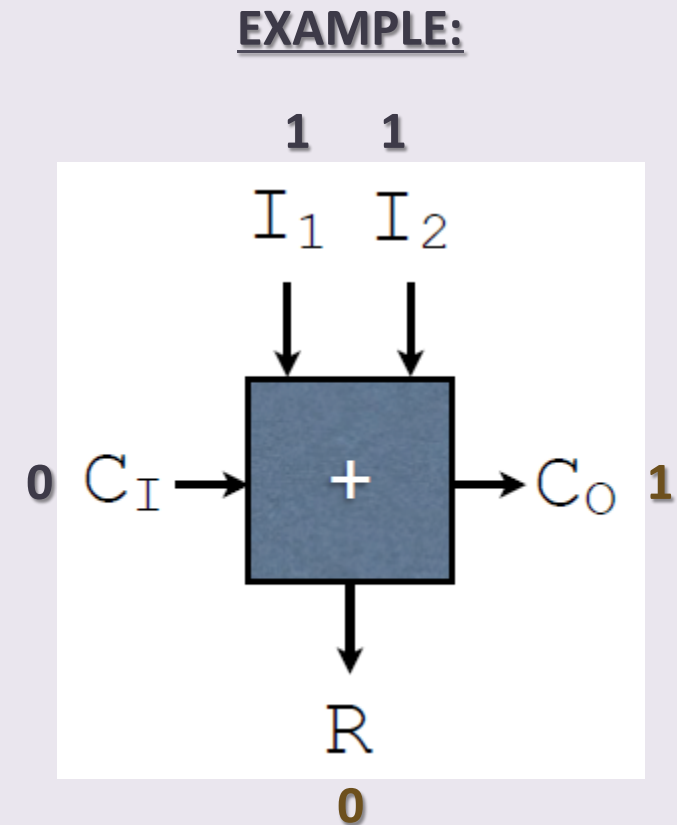
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

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.
= 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

#	<i>INPUTS</i>			<i>OUTPUTS</i>	
	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:

$$F = A \cdot B + C \quad (\text{all single bits})$$

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. $\neg A.B$ is TRUE, or
 - $A = 1, B = 0$ i.e. $A.\neg B$ is TRUE
- In other words, **A XOR B** is TRUE **iff** (if and only if) $A\neg B + \neg AB$ is TRUE

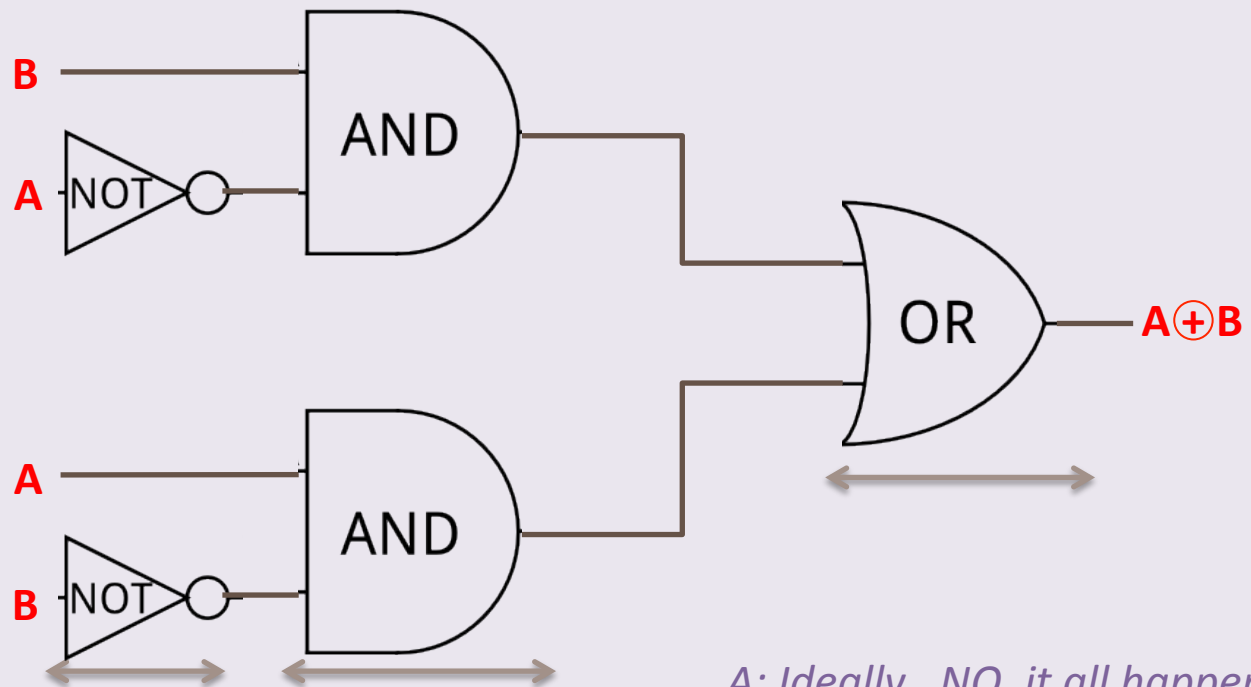
$$A \oplus B = \neg A.B + A.\neg B$$

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

Representing the Circuit Graphically

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

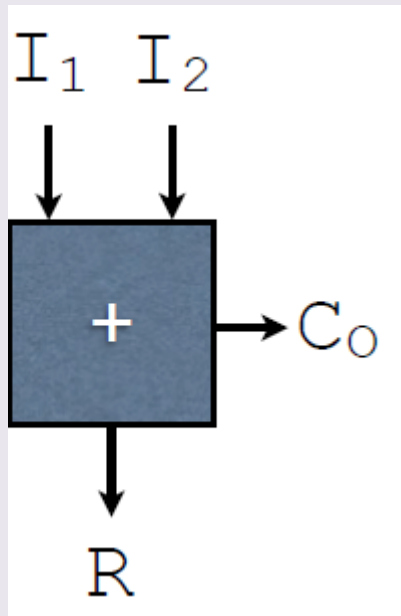
$$A \oplus B = !A.B + A.!B$$



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 (C_i) 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:

$$\begin{aligned} CO &= I_1 \cdot I_2 \\ R &= I_1 \oplus I_2 \end{aligned}$$

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
$$\mathbf{A + B}$$

(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?

$$A \cdot A \quad A$$

$$A + A \quad A$$

$$A \cdot 1 \quad A$$

$$A + 1 \quad 1$$

$$A \cdot 0 \quad 0$$

$$A + 0 \quad A$$

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:
$$\begin{aligned} R &= A.B + !A.B \\ &= (A + !A).B \\ &= B \end{aligned}$$

Let's verify it with a truth-table

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

- Simplify:
$$\begin{aligned} R &= !ABCD + ABCD + !AB!CD + AB!CD \\ &= BCD(A + !A) + !AB!CD + AB!CD \\ &= BCD + B!CD(!A + A) \\ &= BCD + B!CD \\ &= BD(C + !C) \\ &= BD \end{aligned}$$

Let's verify it with a truth-table

More *Simplification* Exercises

- Simplify:
$$\begin{aligned} R &= !A!BC + !A!B!C + !ABC + !AB!C + A!BC \\ &= !A(C + !C) + !AB(C + !C) + A!BC \\ &= !A + !AB + A!BC \\ &= !A + A!BC \end{aligned}$$

Let's verify it with a truth-table

- Reformulate using **only** AND and NOT logic:

$$\begin{aligned} R &= !AC + !BC \\ &= C (!A + !B) \\ &= C. !(A.B) \end{aligned}$$

← *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	$A\bar{A} = 0$	$A + \bar{A} = 1$
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	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Digital Circuit Design Process

CAN THIS PROCESS BE REVERSED?

Function
definition

adder

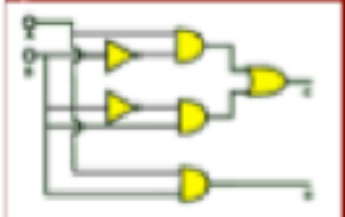
Truth
table

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Boolean
expression

$$\text{Sum} = (A\bar{B}) + (\bar{A}B)$$
$$\text{Carry} = AB$$

Logic
block



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

- $XZ + Z(X' + XY)$

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

- $\text{NOT}(X + YZ)$

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

Example of a K-Map

0
1
2
3

A	B	f(A,B)
0	0	a
0	1	b
1	0	c
1	1	d

B \ A	0	1
0	a	c
1	b	d

B \ A	0	1
0	0	2
1	1	3

A	B	f(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

B \ A	0	1
0	0	1
1	1	1

More on K-Maps Next Week!

YOUR TO-DOs

- Finish Lab #5 by tomorrow!
- Optional: Do the online mid-term evaluations
 - See announcement on Piazza for links
- Next Time: More *K-Maps* and *Combinatorial Logic*

</LECTURE>