

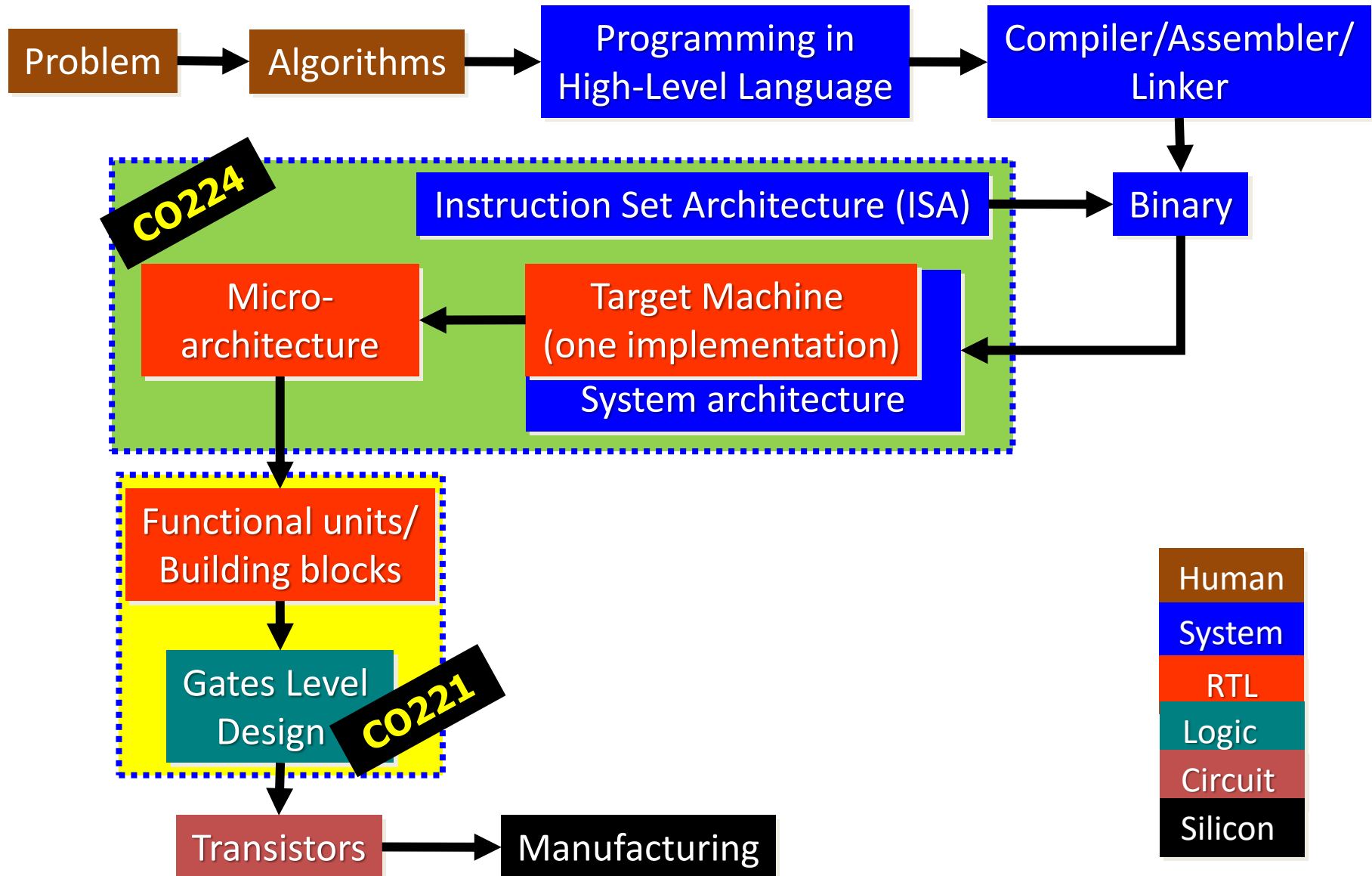
C0221: Digital Design

Swarnalatha Radhakrishnan, PhD

Department of Computer Engineering

University of Peradeniya

The BIG Picture



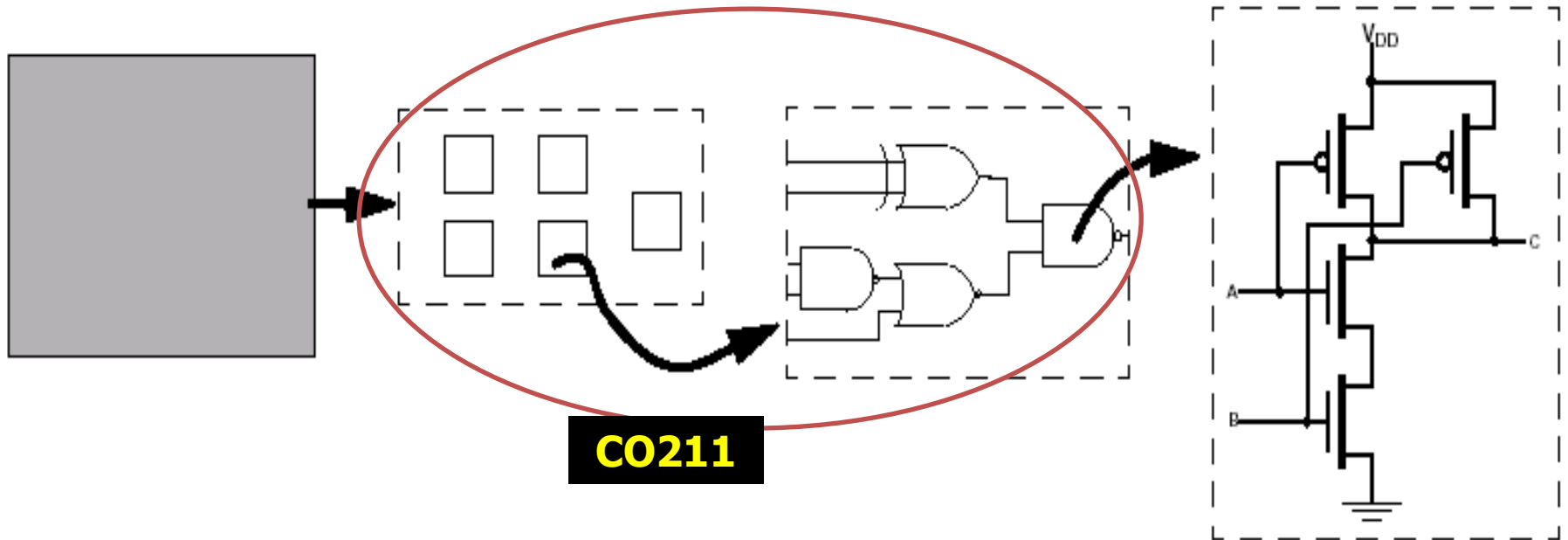
Zoom-in a System Component

SYSTEM

MODULES

GATES AND
FLIP-FLOPS

TRANSISTORS

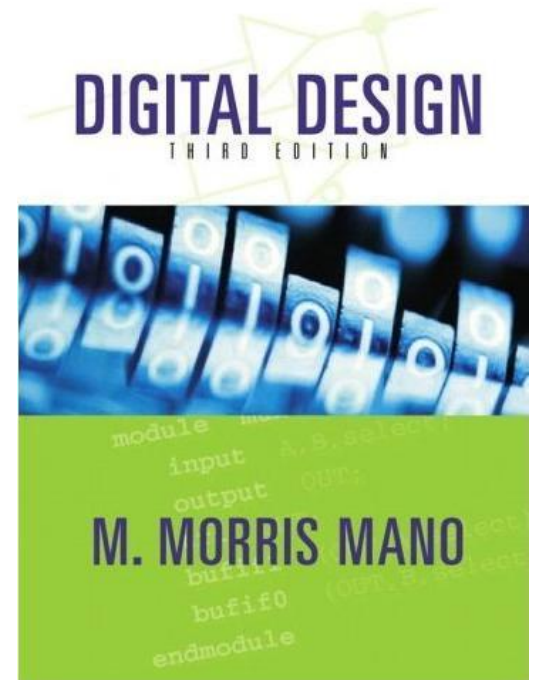


CO221: Assessments

- Continuous Assessment
 - Practical/Lab Work 30
 - Assignments (homework) 10
 - Mid-semester 20
- End Semester Exam 40

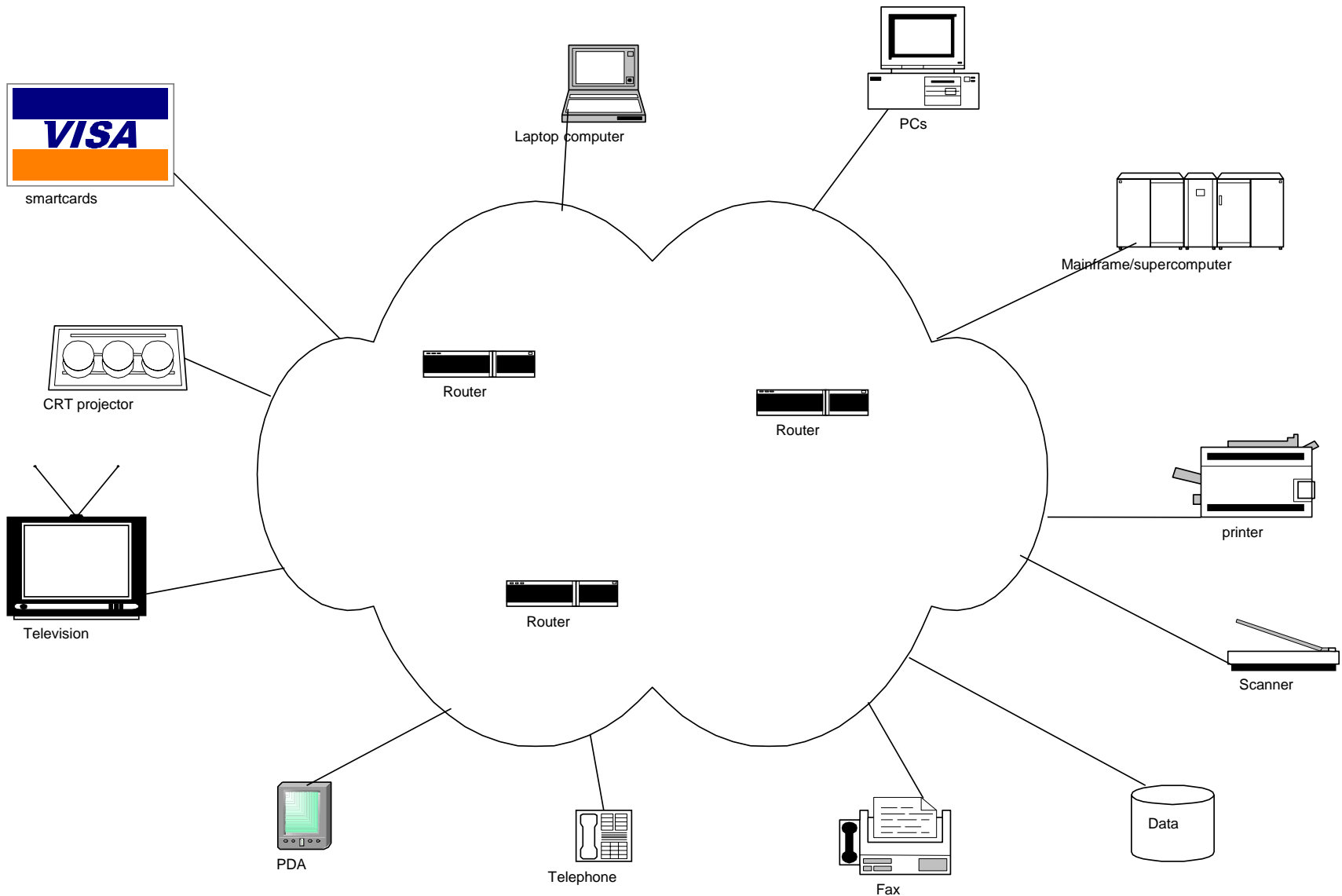
CO221: Text Book

- Digital Design – Fourth Edition
– By M. Morris Mano



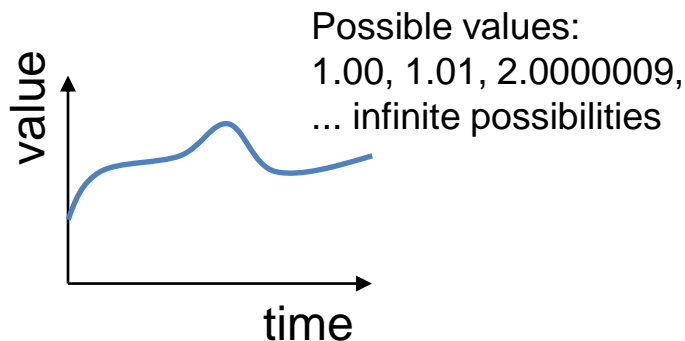
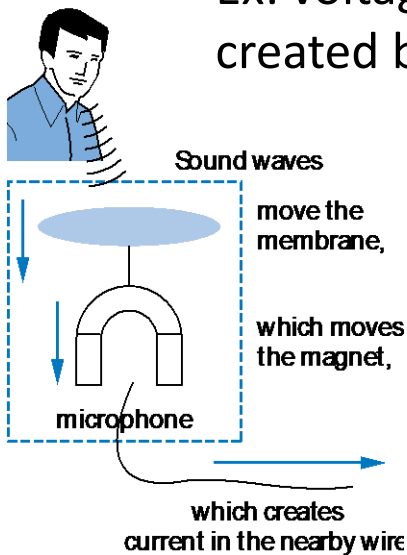
- Low priced edition available
- e-copy available in the CMS

The Digital World

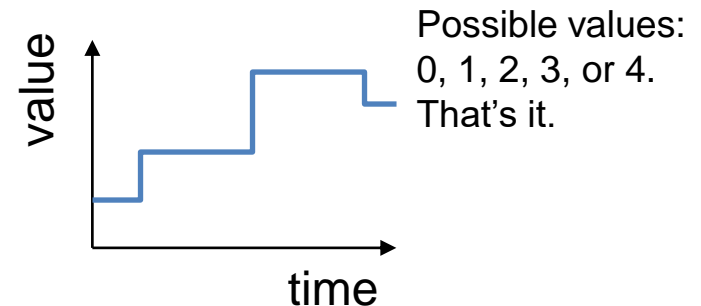
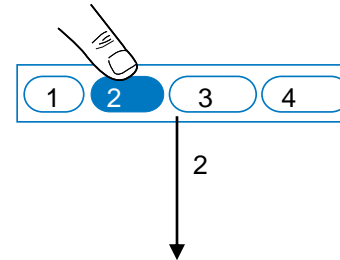


What Does “digital” Mean?

- Analogue signal
 - Infinite possible values
 - Ex: voltage on a wire created by microphone



- Digital signal
 - Finite possible values
 - Ex: button pressed on a keypad



Benefits of Digitization

- Analogue signal (e.g., audio) may lose quality
 - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/trn.
 - “Sample” voltage at particular rate, save sample using bit encoding
 - Voltage levels still not kept perfectly
 - But we can distinguish 0s from 1s
- Digitized audio can be compressed
 - e.g., MP3s
 - A CD can hold about 20 songs uncompressed, but about 200 compressed
- Compression also done on digitized pictures (jpeg), movies (mpeg), and more
- Digitization has many other benefits too, which you will see later

Binary: Digital Signals with 2 Values

- **Binary** digital signal -- only two possible values
 - Typically represented as **0** and **1**
 - One binary digit is a **bit**
 - **We'll only consider binary digital signals**
 - Binary is popular because
 - Transistors, the basic digital electric component, operate using two voltages (more later)
 - Storing/transmitting one of two values is easier than three or more

Implementing Digital Systems

1. Programming Microprocessors

- Microprocessors a common choice to implement a digital system
- Easy to program
- Cheap (as low as \$1)
- Available now

2. Designing Digital Circuits

- With microprocessors so easy, cheap, and available, why design a digital circuit?
- Microprocessor may be too slow
- Or too big, power hungry, or costly
- **When Microprocessors Aren't Good Enough**

Implementing Digital Systems (2)

- Sample digital camera task execution times (in seconds) on a microprocessor versus a digital circuit:

Task	Microprocessor	Custom Digital Circuit
Read	5	0.1
Compress	8	0.5
Store	1	0.8

Topics Covered in CO221

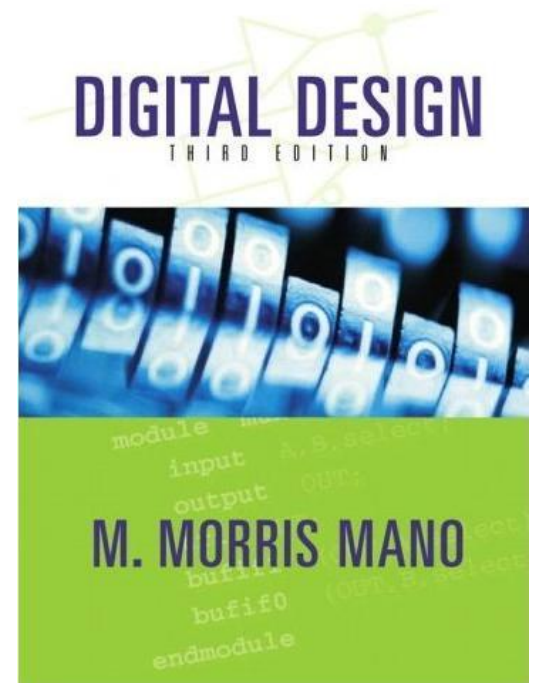
- **Introduction to Digital Logic:**
Digital signals, Digital Logic, Computers and Digital Systems , Purpose and role of digital logic in computer engineering
- **Number Systems and Digital logic:**
Binary number system, , Number Base Conversions,Â Representation of Negative Numbers, Binary arithmetic, Levels of Description of Logic Networks, Basic logic gates, Positive and negative logic

- **Combinational logic circuits:**

Boolean algebra, Boolean laws and theorems, Sum-of-products and Product-of-sums methods, Simplifications of Boolean expressions, Truth tables, Karnaugh Maps, Quine Mc-clusky method, Donâ€™t care combinations, Elimination of timing Hazards

Summary

- The BIG picture (where this course fits in)
- CO221 – Administrative matters
- Analogue vs. Digital Signals
- Why Digital Signal?
- Implementing Digital Systems



Digital Design 3e, Morris Mano
Chapter 1 – Binary Systems

NUMBER SYSTEMS & DIGITAL LOGIC

Decimal Numbers

Uses 10 digits [0, 1, 2, . . . 9]

Positional Number Notation

Weight of digit determined by its position.

Example:

$$\begin{aligned} 246 &= 200 + 40 + 6 \\ &= 2 \times 10^2 + 4 \times 10^1 + 6 \times 10^0 \end{aligned}$$

In general:

$$N = \sum N_i \times 10^i \quad \text{where } N_i \in [0, 1, 2, \dots, 9],$$

where N_i are the weights

Note: decimal fractions occur when i is negative

$$0.35 = 3 \times 10^{-1} + 5 \times 10^{-2}$$

Binary Numbers

Uses 2 digits [0 & 1]

Positional Number Notation

Example:

$$\begin{aligned} 111001_2 &= 100000_2 + 10000_2 + 1000_2 + 000_2 + 00_2 + 1_2 \\ &= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 57_{10} \end{aligned}$$

In general:

$$N = \sum N_i \times 2^i \quad \text{where } N_i \in [0, 1].$$

Note: binary fractions occur when i is negative

$$0.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 0.5_{10} + 0.25_{10} = 0.75_{10}$$

Hexadecimal (Hex) Numbers

Uses 16 digits [0, 1, 2, . . . 9, A, B, C, D, E, F]

Positional Number Notation

Note: A - F represent the decimal values of 10 - 15, respectively.

Example:

$$\begin{aligned} 89AB_{16} &= 8000_{16} + 900_{16} + A0_{16} + B_{16} \\ &= 8 \times 16^3 + 9 \times 16^2 + A \times 16^1 + B \times 16^0 \\ &= 8 \times 4096 + 9 \times 256 + 10 \times 16 + 11 \times 1 = 35243_{10} \end{aligned}$$

In general:

$$N = \sum N_i \times 16^i \quad \text{where } N_i \in [0, 1, 2, \dots, 9, A, B, C, D, E, F].$$

Note: hexadecimal fractions occur when i is negative

$$0.9A_{16} = 9 \times 16^{-1} + 10 \times 16^{-2} = 0.5625_{10} + 0.0390625_{10} = 0.6015625_{10}$$

Exercise

- The solution to the quadratic equation $x^2 - 11x + 22 = 0$ is $x=3$ and $x=6$. What is **the base** of the numbers used here?

Exercise

- Determine the base of the numbers in each cases for the following operations to be correct:
 - (a) $14/2 = 5$
 - (b) $54/4 = 13$
 - (c) $24+17 = 40$

Conversion Among Bases

1. In general, with positional number notation and the known decimal weights for each position in any arbitrary base, it is the **easiest to convert other bases to decimal**.
2. This was demonstrated in each previous example where the decimal value was found using the equation for base B:

$$\sum N_i \times B^i \quad \text{where } N_i \in [0, 1, 2, \dots, B-1]$$

and substituting the equivalent decimal weight for the digits N_i and the decimal value of B^i .

Hexadecimal \Leftrightarrow Binary

Example: convert $10011000111001101_2 \Leftrightarrow$ hexadecimal

Group by 4s, starting at the right \Leftrightarrow expand into 4 bit groups

$1\ 0011\ 0001\ 1100\ 1101_2 \Leftrightarrow 1\ 3\ 1\ C\ D_{16}$

Hex	Binary	Decimal	Hex	Binary	Decimal
0	0000	0	1	0001	1
2	0010	2	3	0011	3
4	0100	4	5	0101	5
6	0110	6	7	0111	7
8	1000	8	9	1001	9
A	1010	10	B	1011	11
C	1100	12	D	1101	13
E	1110	14	F	1111	15

Break

- Programmers Confuse **Halloween** (occurs on October 31) and **Christmas** (December 25). Do you know why?
 - Because $31_{\text{OCT}} = 25_{\text{DEC}}$
- If only DEAD people understand hexadecimal, how many people understand hexadecimal?
 - $\text{DEAD}_{\text{HEX}} = 57005_{\text{DEC}}$

Decimal \Rightarrow Binary

Successive Division: an easy way to convert Decimal to Binary

Based on the remainders after dividing the decimal number by higher powers of 2.

If the decimal number is even, the corresponding binary number will end in a 0, and if it is odd, the binary number will end in a 1, and so on.

Example:

$57 / 2 = 28$, remainder = **1** (binary number will end with 1)

$28 / 2 = 14$, remainder = **0**

$14 / 2 = 7$, remainder = **0**

$7 / 2 = 3$, remainder = **1**

$3 / 2 = 1$, remainder = **1**

$1 / 2 =$ **0**, remainder = **1** (binary number will start with 1)

Therefore, collecting the remainders, $57_{10} = 111001_2$

$$((((0 \times 2 + 1) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2 + 1 = 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 57$$

(check: $32 + 16 + 8 + 0 + 0 + 1 = 57$)

Decimal \Rightarrow Hex

Successive Division: an easy way to convert Decimal to Hexadecimal

Based on the remainders after dividing the decimal number by higher powers of 16.

Example:

$35243 / 16 = 2202$, remainder = 11 \rightarrow B (hex number will end with B)
 $2202 / 16 = 137$, remainder = 10 \rightarrow A
 $137 / 16 = 8$, remainder = 9
 $8 / 16 = \underline{0}$, remainder = 8 (hex number will start with 8)

Therefore, collecting the remainders, $35243_{10} = 89AB_{16}$

(check: $8 \times 4096 + 9 \times 256 + 10 \times 16 + 11 = 35243$)

Hex \Rightarrow Decimal

Multiplication: an easy way to convert Hexadecimal to Decimal

Multiply the hexadecimal weights by powers of 16.

Example:

$$\begin{aligned} \text{BA89} &\rightarrow \text{B} \times 16^3 + \text{A} \times 16^2 + 8 \times 16 + 9 \\ &\quad \quad \quad \text{(CAUTION: mixed bases)} \\ &= 11 \times 4096 + 10 \times 256 + 8 \times 16 + 9 \\ &= 45056 + 2560 + 128 + 9 \\ &= 47753 \end{aligned}$$

We can check by converting back to hex:

$$\begin{aligned} 47753 / 16 &= 2984 + \text{remainder, } 9 \\ 2984 / 16 &= 186 + \text{remainder, } 8 \\ 186 / 16 &= 11 + \text{remainder, } 10 \rightarrow \text{A} \\ 11 / 16 &= \underline{0} + \text{remainder, } 11 \rightarrow \text{B} \\ \therefore 47753_{10} &\rightarrow \text{BA89}_{16} \end{aligned}$$

Binary \Rightarrow Decimal

Multiplication: an easy way to convert Binary to Decimal

We can multiply the binary weights by powers of 2.

However, sometimes it is just as easy to convert the binary number to hexadecimal first and then into decimal.

Compare:

$$\begin{aligned} 11010110_2 &\rightarrow 2^7 + 2^6 + 2^4 + 2^2 + 2 \text{ (exploiting the binary weights)} \\ &= 128 + 64 + 16 + 4 + 2 \\ &= 214_{10} \end{aligned}$$

$$11010110_2 \rightarrow D6_{16} \rightarrow 13 \times 16 + 6 = 208 + 6 = 214_{10}$$

Exercise

- Convert the decimal number 345 to binary in two ways :
 - Convert directly to binary
 - Convert first to hexadecimal, then from hexadecimal to binary.
- Which method is faster ?

Exercise

- Do the following conversion problems :
 - (a) Convert decimal 34.4375 to binary.
 - (b) Calculate the binary equivalent of $1/3$ out to 8 places.
 - (c) Then convert from binary to decimal. How close is the result to $1/3$?
 - (d) Convert the binary result in (b) into hexadecimal. Then convert the result to decimal. Is the answer the same ?

Answer (a)

(a)

34.4375

34

0.4375

$$34:2=17 \quad r=0$$

$$17:2=8 \quad r=1$$

$$8:2=4 \quad r=0$$

$$4:2=2 \quad r=0$$

$$2:2=1 \quad r=0$$

$$0.4375*2=0.875 \quad r=0$$

$$0.875*2=1.75 \quad r=1$$

$$0.75*2=1.5 \quad r=1$$

$$0.5*2=1.0 \quad r=1$$

$$0*2=0 \quad r=0$$

$$34=(100010)_2$$

$$0.4375=(0.01110)_2$$

$$34.4375=(100010.01110)_2$$

Wednesday's or Friday's Lecture

- **We will test a final year project**
 - An interactive lecture delivery system
 - All of you can participate as long as
 - You have a Bluetooth + java enabled phone **AND**
 - You like to participate
 - **I am requesting all of you to participate and give feedback**
 - What do you have to do?
 - Your Bluetooth ID along with your registration number
 - Download and install a java apps that I will share
 - Use it during my lecture

Binary Arithmetic

- Computers have circuits that do binary arithmetic.
- You already know the rules for decimal addition and subtraction (how to handle sums, carries, differences, and borrows).
- Analogously, we will develop the rules for binary addition and subtraction.

Decimal Addition

Refresher

$$\begin{array}{r} \boxed{\begin{array}{r} 95_{10} \\ + 16_{10} \end{array}} \Rightarrow \begin{array}{r} 9 \times 10^1 + 5 \times 10^0 \\ + 1 \times 10^1 + 6 \times 10^0 \\ \hline 10 \times 10^1 + 11 \times 10^0 \end{array} \\ \downarrow \quad \swarrow \quad \searrow \quad \downarrow \\ 111_{10} = 1 \times 10^2 + (0+1) \times 10^1 + 1 \times 10^0 \end{array}$$

Summary

11 ← Column carries

$$\begin{array}{r} 95_{10} \\ + 16_{10} \\ \hline 111_{10} \end{array}$$

Binary Addition

This table calculates the sum for pairs of binary numbers

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ with a carry of } 1$$

Also known as the Half Adder Table

Binary Addition with Carry

This table shows all the possible sums for binary numbers with carries

carry		addend		augend		sum	
0	+	0	+	0	=	0	
0	+	0	+	1	=	1	
0	+	1	+	0	=	1	
0	+	1	+	1	=	0	with a carry of 1
1	+	0	+	0	=	1	
1	+	0	+	1	=	0	with a carry of 1
1	+	1	+	0	=	0	with a carry of 1
1	+	1	+	1	=	1	with a carry of 1

Also known as the Full Adder Table

Binary Addition

Similar to the decimal case

Example: Add 5 and 3 in binary

(carries)

$$\begin{array}{r} 1 1 1_2 = 5_{10} \\ + 0 1 1_2 = 3_{10} \\ \hline 1 1 0 0_2 = 8_{10} \end{array}$$

Decimal Subtraction

Refresher

$$\begin{array}{r} 9 \quad 15_{10} \\ - 1 \quad 6_{10} \\ \hline 7 \quad 9_{10} \end{array}$$

$$\begin{aligned} 95 &= 9 \times 10^1 + 5 \times 10^0 = 9 \times 10^1 + 15 \times 10^0 \\ -16 &= -1 \times 10^1 + -6 \times 10^0 = -1 \times 10^1 + -6 \times 10^0 \\ \text{borrow} &= -1 \times 10^1 \\ \hline &7 \times 10^1 + 9 \times 10^0 \end{aligned}$$

Note: borrows are shown as explicit subtractions.

Binary Subtraction

This table calculates the difference for pairs of binary numbers

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ with a borrow of 1}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Also known as the Half Subtractor Table

Binary Subtraction with Borrow

This shows all the possible differences for binary numbers with borrows

minuend subtrahend borrow difference

$$0 - 0 - 0 = 0$$

$$0 - 0 - 1 = 1 \quad \text{with a borrow of 1}$$

$$0 - 1 - 0 = 1 \quad \text{with a borrow of 1}$$

$$0 - 1 - 1 = 0 \quad \text{with a borrow of 1}$$

$$1 - 0 - 0 = 1$$

$$1 - 0 - 1 = 0$$

$$1 - 1 - 0 = 0$$

$$1 - 1 - 1 = 1 \quad \text{with a borrow of 1}$$

Also known as the Full Subtractor Table

Binary Subtraction

Similar to the decimal case

Example: Subtract 3 from 5 in binary

$$\begin{array}{r} 101_2 = 5_{10} \\ - 11_2 = 3_{10} \\ \hline 010_2 = 2_{10} \end{array}$$

(borrows)

Break!

- There are only 10 types of people in the world: those who understand binary, and those who don't.
- There are only 10 types of people in the world: those who understand binary, those who don't, and those who understand Gray code.
- There are three kinds of people in the world: those who can count, and those who can't.

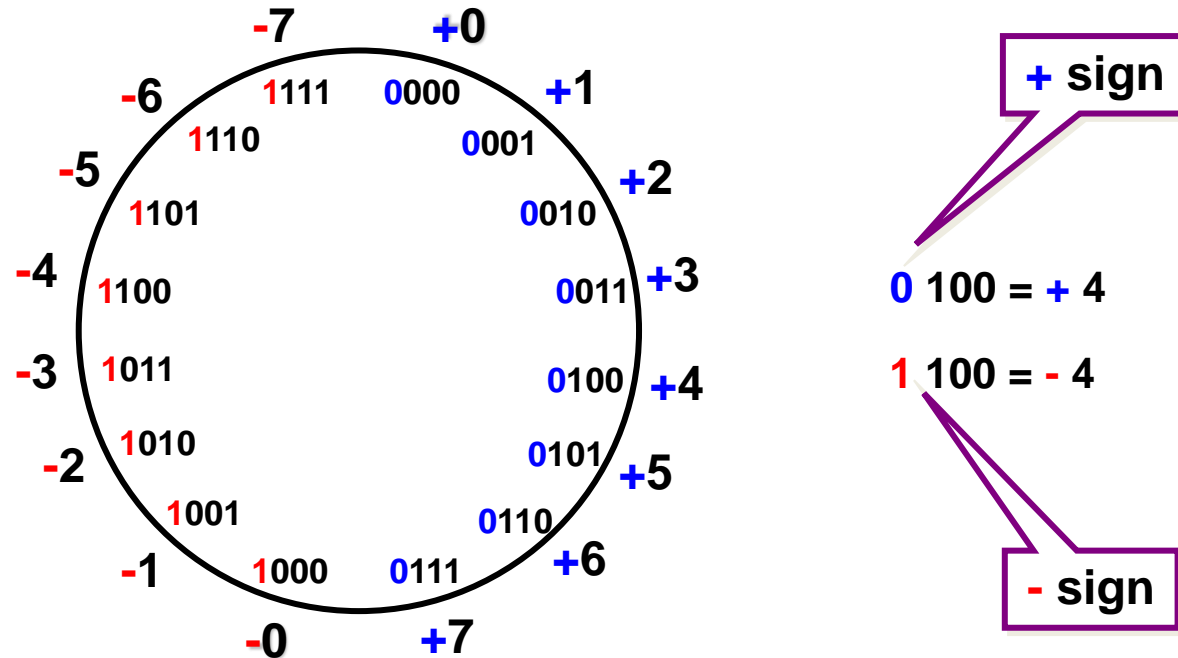
Number Systems

- **Representation of Integers**

- Positive numbers are same in most systems
- Major differences are in how negative numbers are represented
- Three major schemes:
 - (1) sign & magnitude (2) one's complement (3) two's complement
- If we assume a 4 bit machine word
 - 16 different values can be represented
 - roughly half are positive, half are negative

(1) Sign & Magnitude Representation

4 bit example



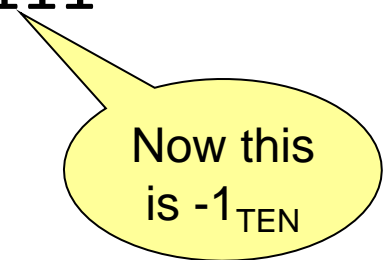
- Sign equals the High order bit: 0 = positive or zero (non-negative)
1 = negative
- Magnitude equals the three low order bits: 0 = 000 thru 7 = 111
- The number range = ± 7 for 4 bit numbers; for n bits, $\pm 2^{n-1} - 1$
- Two Representations for 0 (redundant & problematic)

(2) One's Complement Representation

- Example:
 - $+9_{\text{ten}} \rightarrow 00001001_{\text{two}}$
 - $-9_{\text{ten}} \rightarrow 11110110_{\text{two}}$
- Positive numbers have leading zeros and negative numbers have leading ones
- Shortcomings of one's complement
 - Complicated arithmetic circuit
 - Again two zeros
 - $00000000 \rightarrow +0$
 - $11111111 \rightarrow -0$

(3) Two's Complement Representation

- The **most common representation** for signed binary integers
- The idea came from subtracting a large unsigned number from a small one
 - We would try to borrow from the string of leading zeros and therefore the result would be a string of leading ones
 - $4 - 5 \rightarrow 00000100 - 00000101 = 11111111$



Now this
is -1_{TEN}

(3) Two's complement – Contd.

- Example:

$$+1_{\text{ten}} = 0000\ 0001_{\text{two}}$$

$$\begin{aligned} -1_{\text{ten}} &= \text{inverse}(1_{\text{ten}}) + 1_{\text{two}} \\ &= 1111\ 1110_{\text{two}} + 1_{\text{two}} \\ &= 1111\ 1111_{\text{two}} \end{aligned}$$

$$\begin{aligned} -8_{\text{ten}} &= \text{inverse}(8_{\text{TEN}}) + 1_{\text{two}} \\ &= \text{inverse}(0000\ 1000_{\text{two}}) + 1_{\text{two}} \\ &= 1111\ 0111_{\text{two}} + 1_{\text{two}} \\ &= 1111\ 1000_{\text{two}} \end{aligned}$$

Signed Number System - Summary

- All the 4-bit numbers in the different signed number systems are given
- Positive numbers are the same in all three representations
- Signed magnitude and one's complement have two ways of representing 0. This makes things more complicated
- Two's complement has asymmetric ranges; there is one more negative number than positive number. Here, you can represent -8 but not +8
- However, two's complement is preferred because it has only one 0, and its addition algorithm is the simplest

Decimal	S.M.	1's comp.	2's comp.
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

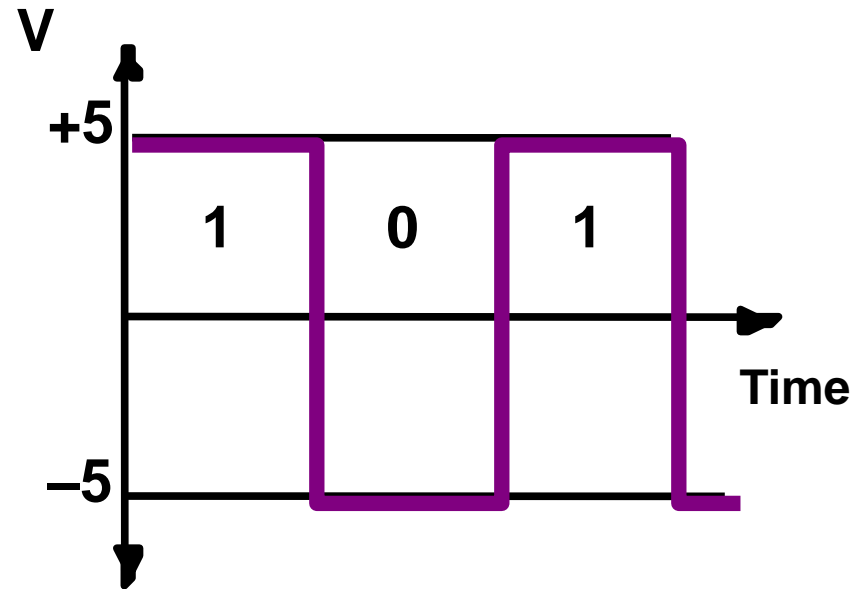
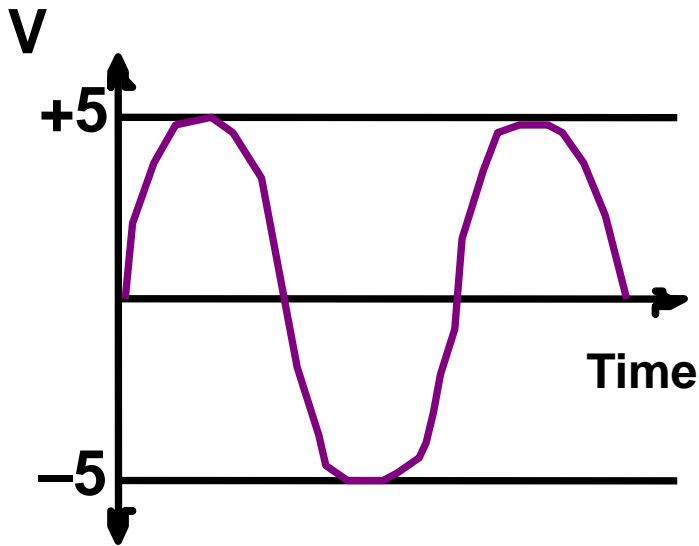
Binary Codes

- **Binary Coded Decimals (BCD)**
 - decimal numbers are encoded with each digit is represented by its own binary sequence
 - Check http://en.wikipedia.org/wiki/Binary-coded_decimal
- **Gray Code**
 - two successive values differ in only one bit
 - Produces quantities that are continuous
 - Check http://en.wikipedia.org/wiki/Gray_code
- **ASCII Character Code**
 - American Standard Code for Information Interchange
 - character-encoding scheme based on the ordering of the English alphabet
 - Check <http://en.wikipedia.org/wiki/ASCII>

Error Detecting Code

- Using parity bit
 - Usually added to a data transmission or storage to verify the correctness of data
 - Example:
 - Data without parity bit -> 0100 100
 - Data with parity bit -> 0010 0100 (even parity)
 - Data with parity bit -> 1010 0100 (odd parity)
 - Can we correct errors using this parity bit?

Analogue vs. Binary



- **Analog:** values vary over a broad range continuously
- **Digital:** only discrete values

Analogue vs. Binary

- Analog devices process signal that can assume any value across a continuous range and produce results that are also in continuous form.
- Digital devices process signals that take on only two discrete values such as 0 and 1 and produces output that can be represented by 0 and 1.

Analogue vs. Digital Systems

Analogue systems:

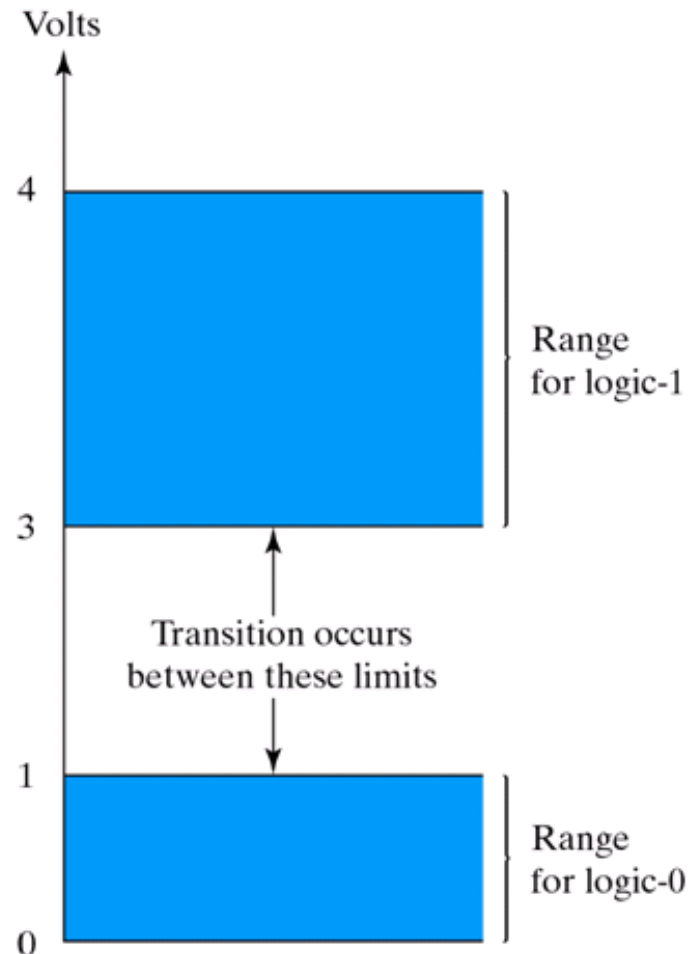
- Limited precision, errors accumulate
- Interface circuits (i.e., sensors & actuators) are often analogue

Digital systems:

- More accurate and reliable
- Readily available as self-contained, easy to cascade building blocks
- Computers use digital circuits internally

Binary Logic

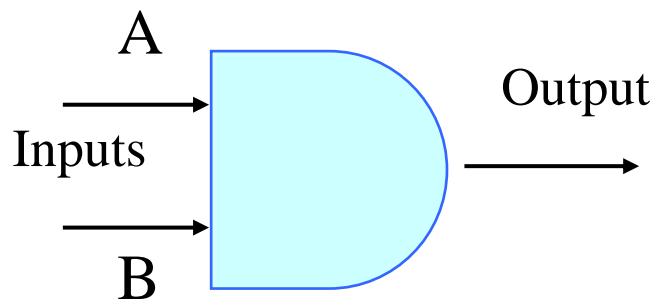
- Logic with **yes (1)** and **no (0)**
- Binary variables and logical operations
- Typical range of signals shown here



Logic Operators and Gates (1)

- AND

- AND means that both conditions must be *true* in order for the conclusion to be *true*
 - **Consider:** If the *car is fueled* AND the *engine works*, then the *engine will start*
- We can build an electrical device that performs the logical AND operation on voltage equivalents of logic values



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

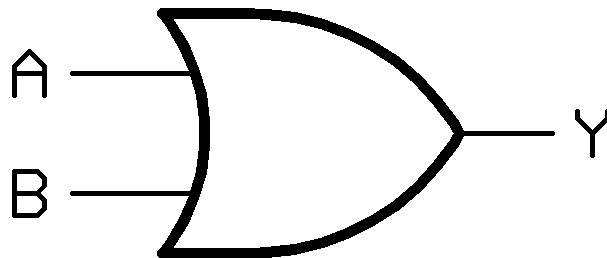
Logic Operators and Gates (2)

- OR

- Another basic operator is the OR

- **Consider:** If I have cash OR a credit card, then I can pay the bills

- OR works such that the output is true, if either of the two inputs is true or both are true



A	B	Y - Output
0	0	0
0	1	1
1	0	1
1	1	1

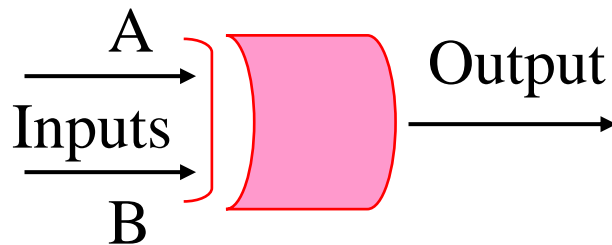
Logic Operators and Gates (3)

- XOR (Exclusive OR)

- **Consider:** I would like to have a tea OR a coffee.
I usually catch a bus OR walk.

these are XOR!!

- XOR works such that the output is true, if both inputs are of different value



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

- Note that when we say OR in day to day life as in the example, we mean XOR and not OR.

Logic Gates - Summary

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal



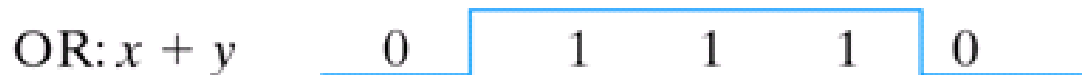
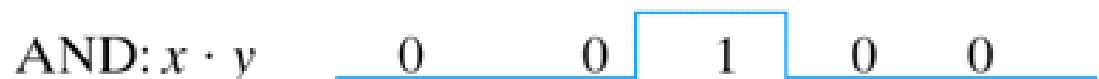
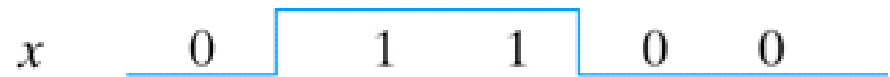
(a) Two-input AND gate



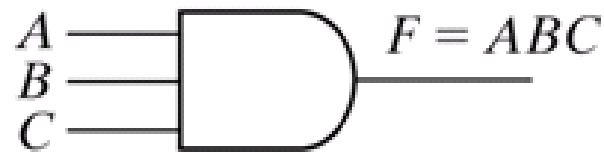
(b) Two-input OR gate



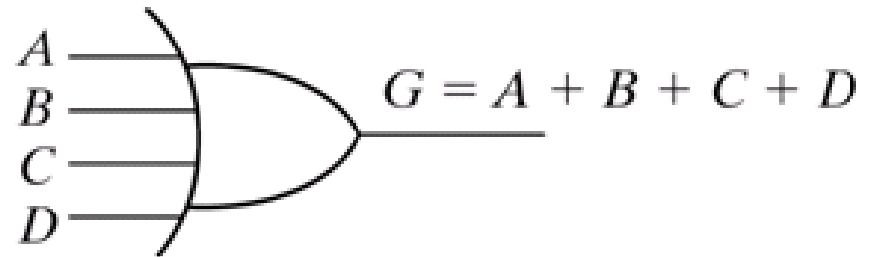
(c) NOT gate or inverter



Gates with Multiple Inputs



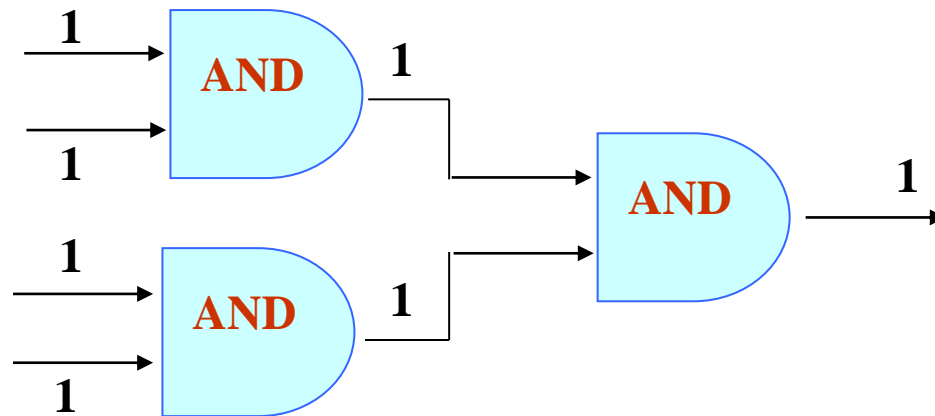
(a) Three-input AND gate



(b) Four-input OR gate

Apps: Digital Combination Lock

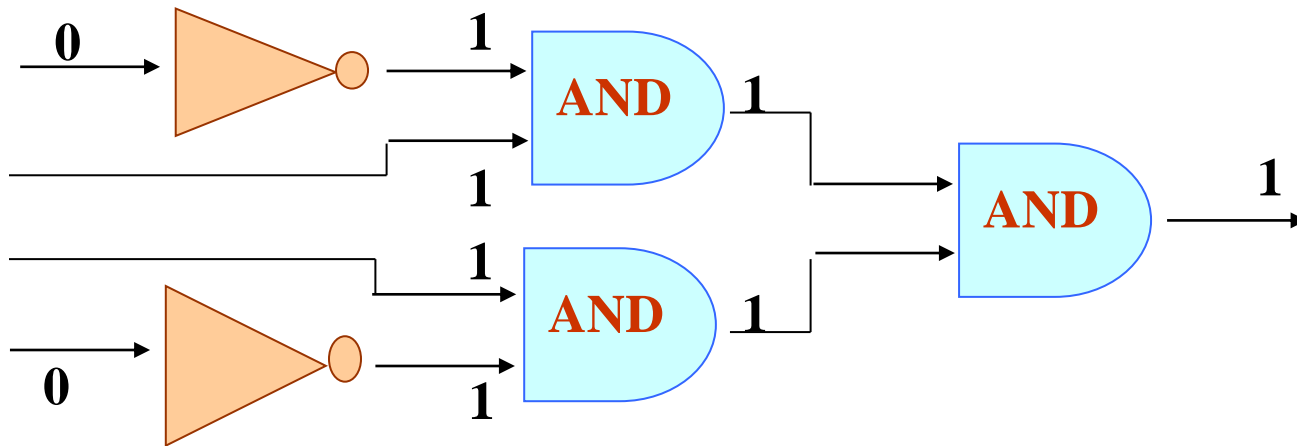
- Using 3 two-input AND gates, we could build a combination lock that requires a four-digit code, specifically: 1 1 1 1



- The number of inputs could be increased by using more and more AND gates

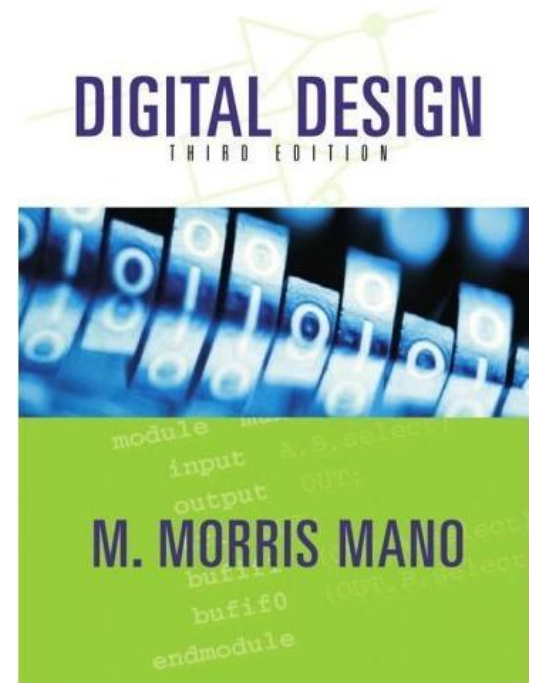
Apps: Digital Combination Lock (2)

- We could build a combination lock that only uses the AND gate, but that would be of little use since everyone would know our combination, namely 1 1 1 1
- To build a more interesting combination lock, we will utilize the NOT (inverter) gate
 - Let's build a combination lock whose input (key code) combination is **0 1 1 0**



Homework

- Try the rest of the exercises from Chapter 01 of the textbook.
 - I have already posted the questions
 - We will discuss the answers in the forum



Digital Design 3e, Morris Mano
Chapter 2 – Boolean Algebra and Logic Gates

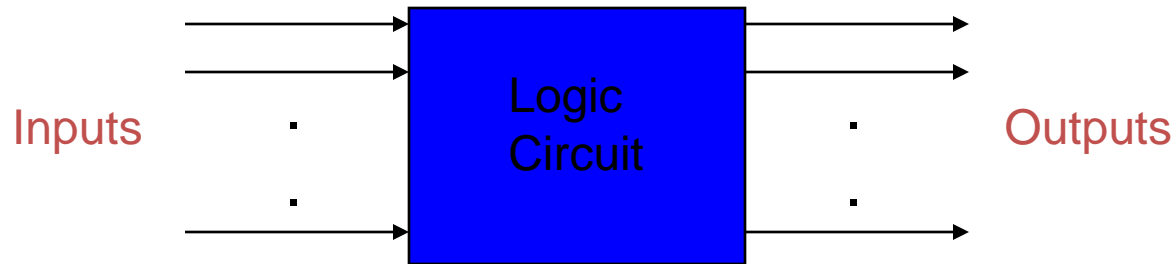
BOOLEAN ALGEBRA

Overview

- Logic functions with 1's and 0's
 - Building digital circuitry
- Truth tables
- Logic symbols and waveforms
- Boolean algebra
- Properties of Boolean Algebra
 - Reducing functions
 - Transforming functions

Digital Systems

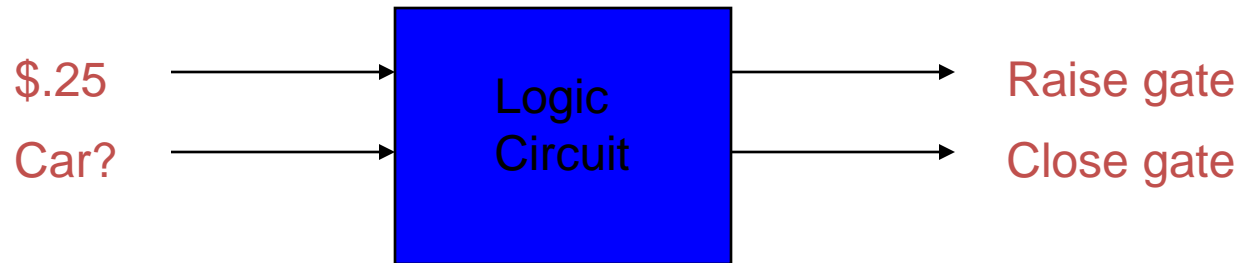
- Analysis problem:



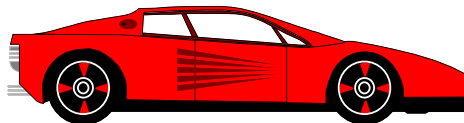
- Determine binary outputs for each combination of inputs
- Design problem: given a task, develop a circuit that accomplishes the task
 - Many possible implementation
 - Try to develop “best” circuit based on some criterion (size, power, performance, etc.)

Toll Booth Controller

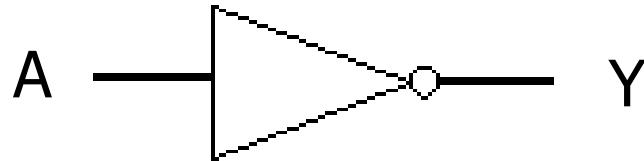
- Consider the design of a toll booth controller
- Inputs: quarter, car sensor
- Outputs: gate lift signal, gate close signal



- If driver pitches in quarter, raise gate.
- When car has cleared gate, close gate.



Describing Circuit Functionality: Inverter



Symbol

Truth Table

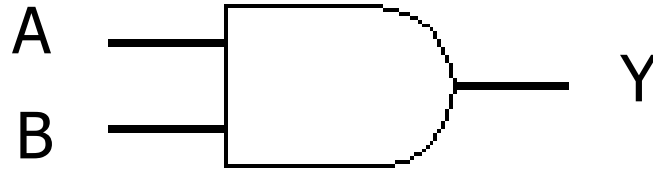
A	Y
0	1
1	0

Input

Output

- Basic logic functions have symbols.
- The same functionality can be represented with truth tables.
 - Truth table completely specifies outputs for all input combinations.
- The above circuit is an inverter.

The AND Gate

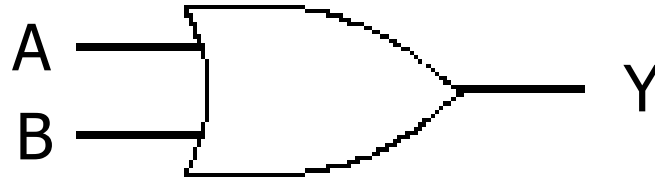


- This is an AND gate.
- So, if the two inputs signals are asserted (high) the output will also be asserted. Otherwise, the output will be deasserted (low).

Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

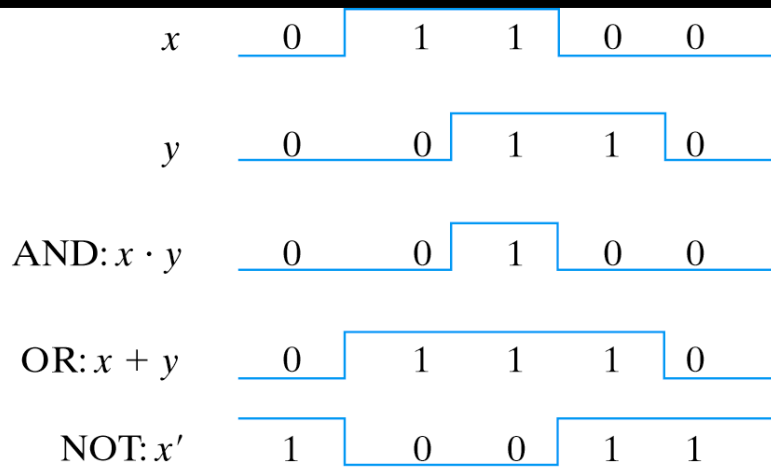
The OR Gate



- This is an OR gate.
- So, if either of the two input signals are asserted, or both of them are, the output will be asserted.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Describing Circuit Functionality: Waveforms



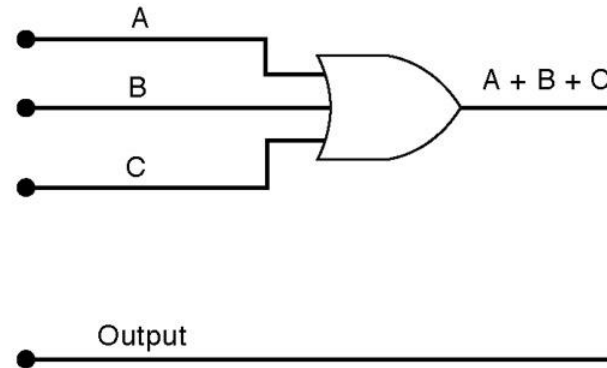
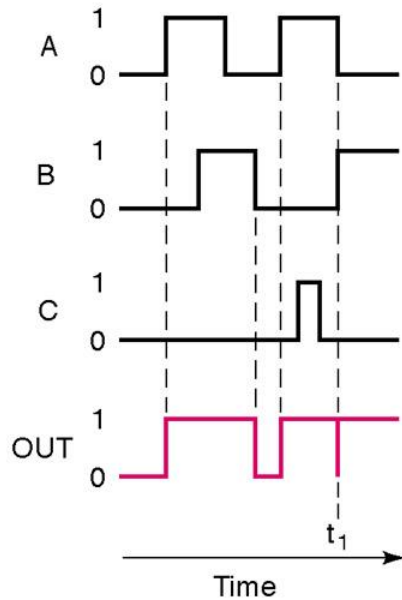
AND Gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

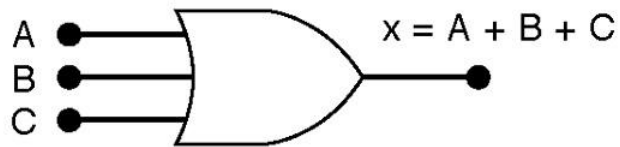
Fig. 1-5 Input-output signals for gates

- Waveforms provide another approach for representing functionality.
- Values are either high (logic 1) or low (logic 0).
- Can you create a truth table from the waveforms?

Consider three-input gates



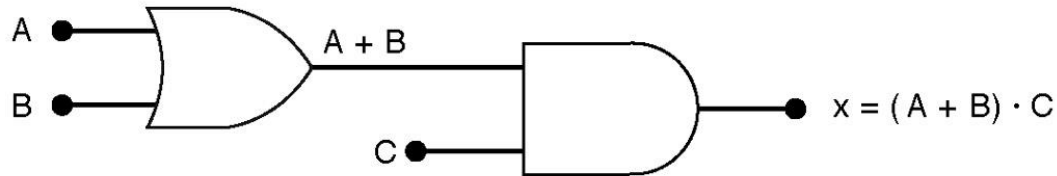
3 Input OR Gate



0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Ordering Boolean Functions

- How to interpret $A \bullet B + C$?
 - Is it $A \bullet B$ ORed with C ?
 - Is it A ANDed with $B + C$?
- Order of precedence for Boolean algebra:
AND before OR.
- Note that parentheses are needed here :



Boolean Algebra

- A *Boolean algebra* is defined as a closed algebraic system containing a set K or two or more elements and the two operators, $.$ and $+$.
- Useful for identifying and *minimizing* circuit functionality
- Identity elements
 - $a + 0 = a$
 - $a . 1 = a$
- 0 is the identity element for the $+$ operation.
- 1 is the identity element for the $.$ operation.

Commutativity and Associativity of the Operators

- The Commutative Property:

For every a and b in K ,

$$- a + b = b + a$$

$$- a \cdot b = b \cdot a$$

- The Associative Property:

For every a , b , and c in K ,

$$- a + (b + c) = (a + b) + c$$

$$- a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

Distributivity of the Operators and Complements

- The Distributive Property:

For every a , b , and c in K ,

$$- a + (b \cdot c) = (a + b) \cdot (a + c)$$

$$- a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

- The Existence of the Complement:

For every a in K there exists a unique element called a' (*complement of a*) such that,

$$- a + a' = 1$$

$$- a \cdot a' = 0$$

- To simplify notation, the \cdot operator is frequently omitted. When two elements are written next to each other, the AND (\cdot) operator is implied...

$$- a + b \cdot c = (a + b) \cdot (a + c)$$

$$- a + bc = (a + b)(a + c)$$

Duality

- The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all + operators with . operators, all . operators with + operators, all ones with zeros, and all zeros with ones.
- Form the dual of the expression
$$a + (bc) = (a + b)(a + c)$$
- Following the replacement rules...

Involu tion

- This theorem states:

$$a'' = a$$

- Remember that $aa' = 0$ and $a+a'=1$.
 - Therefore, a' is the complement of a and a is also the complement of a' .
 - As the complement of a' is unique, it follows that $a''=a$.
- Taking the double inverse of a value will give the initial value.

Absorption

- This theorem states:

$$a + ab = a$$

$$a(a+b) = a$$

- To prove the first half of this theorem:

$$a + ab = a \cdot 1 + ab$$

$$= a (1 + b)$$

$$= a (b + 1)$$

$$= a (1)$$

$$a + ab = a$$

DeMorgan's Theorem

- A key theorem in simplifying Boolean algebra expression is DeMorgan's Theorem. It states:

$$(a + b)' = a'b'$$

$$(ab)' = a' + b'$$

- Complete the expression $a(b + z(x + a'))$ and simplify.

$$\begin{aligned} &= a' + (b + z(x + a'))' \\ &= a' + b'(z(x + a'))' \\ &= a' + b'(z' + (x + a'))' \\ &= a' + b'(z' + x'a'') \\ &= a' + b'(z' + x'a) \end{aligned}$$

Summary

- Basic logic functions can be made from AND, OR, and NOT (invert) functions
- The behavior of digital circuits can be represented with waveforms, truth tables, or symbols
- Primitive gates can be combined to form larger circuits
- Boolean algebra defines how binary variables can be combined
- Rules for associativity, commutativity, and distribution are similar to algebra