# CS250: FOUNDATION OF COMPUTER SYSTEMS
# [BINARY REPRESENTATIONS]

**The Mighty Bits**
And then there were two

Stringing up in sequences
to form streams
traveling far and wide

Nope … not the
dot and dash siblings
from Morse code

These are the binary siblings: 0 and 1
Powered by logic and Boolean algebra
Undergirding circuits, memory,
networks, storage, and displays …

SHRIDEEP PALLICKARA
Computer Science
Colorado State University



1

## Frequently asked questions from the previous class survey

- Where can I find what you *will* be covering? And the order of coverage?
  - **https://www.cs.colostate.edu/~cs250/**
- How will the assignments be graded?
- ENIAC (Electronic Numerical Integrator and Computer)
- What is the difference between the midterm(s) and coding exam(s)?
- What if I miss lectures? Will you announce quizzes?
- How do I get over the fear of coding?
- Lots of questions about IP v4/v6, TCP/IP, Gates, NAND, GPUs, and quantum computing



2

## Why so long in the truck delivery problem

- 50!
  - 3041409320171337804361260816606476884437764156896051200000000000000 [65 digits]
- 70!
  - 11978571669969891796072783721689098736458938142546425857555362864628009582789845319680000000000000000 [101 digits]
- 100!
  - 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000 [158 digits]



3

## Coding Exam Scenario: Class average is 3.75/5 (75%) → add +0.25 to reach 4.00/5 (80%)

- Rule: Only students strictly 40% and higher (≥ 2/5) are included in
  - The average used for curving and
  - Receiving the curve

| Student | Raw score (out of 5) | % | Above 40% (>2.0)? | Included in class avg? | Curve added | Adjusted score |
|---|---|---|---|---|---|---|
| A | 4.5 | 90% | Yes | Yes | 0.25 | 4.75 |
| B | 3.75 | 75% | Yes | Yes | 0.25 | 4 |
| C | 2.1 | 42% | Yes | Yes | 0.25 | 2.35 |
| D | 1.75 | 35% | **No** | **No** | 0 | 1.75 |
| E | 1.5 | 30% | **No** | **No** | 0 | 1.5 |

If your first reaction is '*but what if…*', congratulations! You are among your people. Read the rule twice and keep coding.



4

# How the top-2 out of 3 for coding exams work

- Top-2 happens only if you score ≥ 2/5 on *each* of the three exams
- Otherwise, the Coding Exams portion = (Exam1 + Exam2 + Exam3) out of 15
  - i.e., no "*drop the lowest and rescale*" benefit)

| Student | Exam 1 (/5) | Exam 2 (/5) | Exam 3 (/5) | All three ≥ 2? | Rule applied | Computation | Coding Exams portion (/15) |
|---|---|---|---|---|---|---|---|
| A | 4 | 4 | 1 | **No** (one < 2) | **No scaling** | 4 + 4 + 1 | **9/15** |
| B | 4 | 4 | 2 | **Yes** | **Top-2 scaled** | (4 + 4)/10 = 80% → 0.80×15 | **12/15** |
| C | 2 | 2 | 2 | **Yes** | **Top-2 scaled** | (2 + 2)/10 = 40% → 0.40×15 | **6/15** |
| D | 5 | 3 | 1.5 | **No** (one < 2) | **No scaling** | 5 + 3 + 1.5 | **9.5/15** |
| E | 5 | 3 | 2 | **Yes** | **Top-2 scaled** | (5 + 3)/10 = 80% → 0.80×15 | **12/15** |

If your first reaction is '*but what if…*', congratulations! You are among your people. Read the rule twice and keep coding.





# Topics covered in this lecture

- Digital representations with signals and binary codes
- The clocked CPU
- Binary Representations
  - Properties of binary numbers

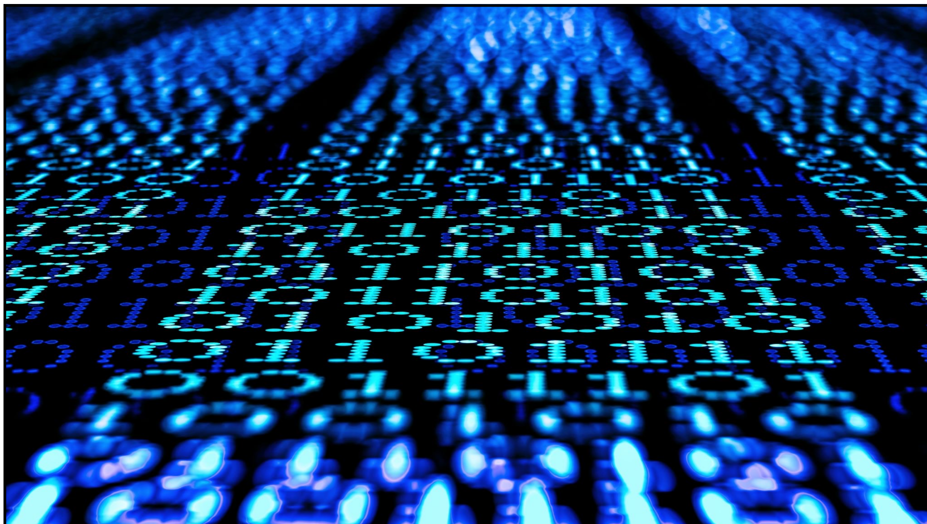
## Coding and why it's important

- Primary skill as computer science professionals
  - Gateway to understanding algorithms and software engineering
- Analogy with musicians
  - What kind of music would someone produce if they worked on their craft (and honing their skills) only for 2-3 hours per week?
  - As budding computer scientists, you should be coding 20+ hours/week on assignments, research (if you are doing this), or personal projects
- For courses
  - Get started early on assignments
  - Procrastination adversely impacts retention



## Submissions so far … [281 and counting]

| Name | Status | Total Submissions | Last Submission Status | Last Submission Date |
|---|---|---|---|---|
| 01 - Hello World | | 179 | | 1/22/2026, 11:26:32 AM |
| 02 - Types, Operators | | 65 | | 1/22/2026, 11:43:53 AM |
| 03 - IO, Random | | 22 | | 1/22/2026, 12:05:18 AM |
| 04 - Strings | | 10 | | 1/22/2026, 11:10:05 AM |
| 05 - Conditionals | | 5 | | 1/22/2026, 10:01:24 AM |
| 06 - Arrays | | – | – | – |
| 07 - Loops | | – | – | – |
| 08 - Exceptions | | – | – | – |
| 09 - Methods | | – | – | – |
| 10 - Classes | | – | – | – |
| Working with numbering systems, bitwise operations, and common binary operations | | – | – | – |



# DIGITAL REPRESENTATION WITH SIGNALS AND BINARY CODES

COMPUTER SCIENCE DEPARTMENT

COLORADO STATE UNIVERSITY

9

## Signals

- To be processable, data must be represented as **signals** in the machine or as *measurable disturbances* in the structure of storage media
- There is no information without representation
- Arithmetic operations such as add and subtract must be represented as **rules for transforming signals**

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | BINARY REPRESENTATIONS | L2.10

10

## Using decimal digits?

- One early way to represent a decimal digit was a ring of 10 dual-triode vacuum tubes simulating a 10-position wheel
  - Very **expensive**!
- Proposals to represent decimal digits with 10 distinct voltages were dismissed because of the **complexity** of the circuits



## Engineers quickly settled on using **binary** codes to represent numbers

- Binary-coded arithmetic used *many fewer* components than decimal-coded arithmetic
- Also, circuits to distinguish two voltage values were **much more reliable** than circuits to distinguish more than two values
- Also, storage and display could easily be built from available two-state technology
  - Magnetic cores, flip-flop circuits, or phosphor patches on a cathode-ray screen

## The decision to **abandon decimal arithmetic** and use binary codes for everything in the computer

- Led to very simple, much more reliable circuits and storage
- The term "**bit**" came into standard use as shorthand for "binary digit"
- Today no one can think about contemporary computers without thinking about binary representations





## Keep in mind that internally the computer does not process numbers and symbols

- Computer circuits deal only with voltages, currents, switches, and malleable materials
- Cannot overemphasize the importance of physical forms in computers such as signals in circuits or magnetic patches on disks
  - **Without these physical effects we could not build a computer**
- The patterns of zeroes and ones are **abstractions** invented by the designers to describe what their circuits do

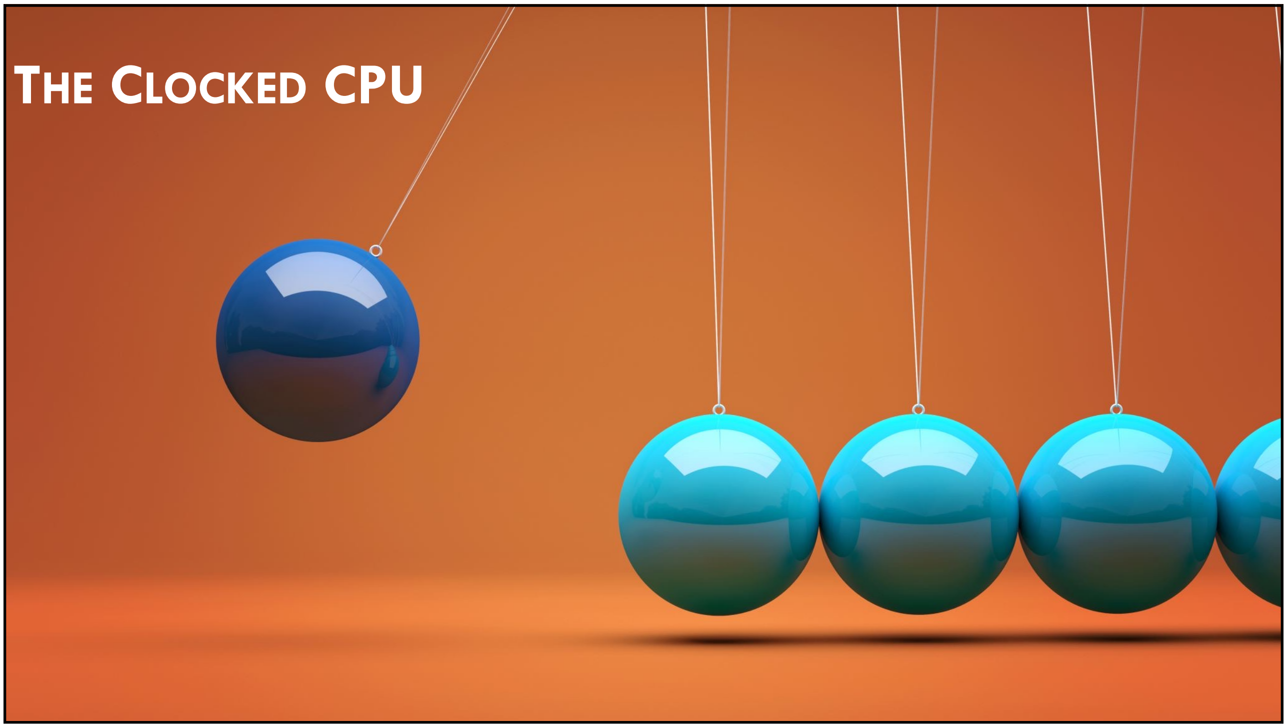
## Because not every binary code is a valid description of a circuit, symbol, or number

- Designers invented syntax rules that distinguished valid codes from invalid ones
- Although the machine cannot understand what patterns mean
  - **It can** *distinguish* allowable patterns from others by applying the syntax rules



## To summarize: Binary isn't minimalism It's error tolerance with a paycheck

- A signal is a disturbance you can reliably measure (voltage, current, charge, light intensity, magnetization)
- Real hardware is noisy!
  - temperature, manufacturing variation, interference, aging all jiggle measurement
- With two levels, you can design big "safety margins" (thresholds):
  - "Anything below X is 0; anything above Y is 1"
  - That gap is your **noise tolerance**
- With 10 levels, the *thresholds get crowded*: the same noise that's harmless in binary now pushes symbols across boundaries
  - Causing errors and forcing much more complex circuitry!




17

## The Clocked CPU Cycle for Basic Computational Steps [1/2]

- The physical structure of computers consists of
  - Registers, which store bit patterns
  - Logic circuits, which compute ***functions*** of the data in the registers
- It *takes time* for these logic circuits to propagate signals from their input registers to their output registers



18

## The Clocked CPU Cycle for Basic Computational Steps [2/2]

- If new inputs are provided before the circuits *settle*?
  - The outputs are likely to be misinterpreted by subsequent circuits
- Engineers solved this problem by adding **clocks** to computers
  - At each clock tick the output of a logic circuit is stored in its registers
- The interval between ticks is long enough to guarantee that the **circuit is completely settled** before its output is stored
  - Computers of the von Neumann architecture cannot function without a clock
- The clock is the computer's metronome: without it, the band 'improvises' into nonsense



## Computers are rated by their **clock speeds** [1/2]

- For e.g., a "3.8 GHz processor"?
  - Is one whose clock ticks 3.8 billion times a second
- Existence of clocks gives a **precise physical interpretation** to the "algorithmic steps" in the digital realm

## Computers are rated by their **clock speeds** [2/2]

- Every algorithmic step must be completed *before* the next step is attempted
- The machine supports this by guaranteeing each instruction will be correctly finished *before* the next instruction is attempted
- Clocks are essential to support our notion of computational steps and guarantee that the computer performs them reliably



## Control Flow [1/2]

- From the time of Babbage and Lovelace, programmers have realized that the machine must be able to decide **which instructions are next**
- Instructions *do not always* follow a linear sequence
- In the von Neumann architecture, the address of the next instruction is stored in a CPU register called the **program counter** (PC)
  - Updated after each instruction
  - The *default* is to execute the next instruction in sequence (PC set to PC+1)



## Control Flow [2/2]

- One common deviation from linearity is to **branch** to another instruction at a different memory location, say **X**
- The decision to branch is governed by a condition **C** (such as "is **A** equal to **B**?")
  - The jump from one part of the program to another part is implemented by an instruction that says "if **C** then set PC to **X**"
- if-then-else construct in programming languages


25

## **Loops**: Small Programs Making Big Computations

- If all our programs were nothing more than decision trees of instruction sequences each selected by if-then-else?
  - They could never generate computations longer than the number of instructions in the program
- The loop allows us to design computations that are *much longer* than the size of the program
  - A loop is a sequence of instructions that are repeated over and over until a **stopping condition** is satisfied



26

## A common programming error is a **faulty stopping condition** that does not exit the loop

- That behavior is called an "**infinite loop**"
- Alan Turing proved that there is no algorithm for inspecting a program to determine if any of its loops is infinite
  - This makes debugging a challenging problem that cannot be automated
- Some programs are built on purpose to loop forever: e.g., web servers
  - The service process waits at a homing position for an incoming request
    - Executes code to fulfill the request and returns to its homing position



27

## BINARY REPRESENTATIONS

And that is also the way the human mind works—by the compounding of old ideas into new structures that become new ideas that can themselves be used in compounds, and round and round endlessly, growing ever more remote from the basic earthbound imagery that is each language's soil.

Douglas R. Hofstadter, I Am a Strange Loop

28

## What Is Language?

- Language is a convenient shortcut
- It allows us to communicate complex concepts without having to demonstrate them
- Every language whether written, spoken, or expressed in a series of gestures or by banging two rocks together
  - Is **meaning encoded as a set of symbols**



## Written language is a **sequence** of symbols

- We form words by placing symbols in a particular order
- For example, in English we can form the word `yum` by placing three symbols (that is, letters) in order from left to right as follows: `y u m`
- Many possible symbols and combinations.
  - For example, there are 26 basic symbols (A–Z) in English: if we ignore things like upper- and lowercase, punctuation, ligatures, and so on

## Three components frame the technology of written language, including computer language

- The containers that hold symbols
- The symbols that are allowed in the containers
- The ordering of the containers



## Bit

- The term **bit** is an awkward marriage between binary and digit
- Awkward because binary is a word for something with two parts
  - But digit is a word for one of the 10 symbols (0–9) that make up our everyday number system
- A bit is binary and can hold only one of two symbols
  - Kind of like the dot and dash from Morse code

## Morse code uses just two symbols to represent complex information

- dot and dash
- Represent complex information by **stringing** those symbols together in different combinations
- The letter **A** is dot-dash; **B** is dash-dot-dot-dot; **C** is dash-dot-dash-dot, and so on
- The order of the symbols is important just like in a human language:
  - dash-dot means **N**, not **A**
- E.g.: "Hello" in Morse code is **.... . . -.. . -.. ---**

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | BINARY REPRESENTATIONS | L2.33

33


34

## Bits and Bytes

- A single bit cannot convey much information
  - It's either off or on, 0 or 1
- We need a **sequence** of bits to represent anything more complex
- To make these sequences of bits easier to manage, computers group bits together in sets of eight, called **bytes**
  - A set of 4-bits is referred to as a nibble



35

## Bits and Bytes

| | |
|---|---|
| 1 | That's a bit |
| 0 | That is also a bit |
| 11001110 | That's a byte, or 8 bits |
| 00111000 | That's also a byte! |
| 10100101 | Yet another byte |
| 0011100010100101 | That's two bytes, or 16 bits |



36

## The decimal numbering system

- We typically write numbers using something called decimal place-value notation
- **Place-value notation** (or positional notation) means that each position in a written number represents a different order of magnitude
- Decimal, or base 10, means that the orders of magnitude are factors of 10
  - and each place can have one of ten different symbols, 0 through 9



## The decimal numbering system: Example [1/2]

| 2 | 7 | 5 |
|---|---|---|
| Hundreds place | Tens place | Ones place |

- Why is the rightmost place the ones place? And why is the next place the tens place, and so on?

## The decimal numbering system: Example [2/2]

- It's because we are working in decimal, or base 10, and therefore **each place is a power of ten**; in other words, 10 multiplied by itself a certain number of times

| 2 | 7 | 5 |
|---|---|---|
| $10^2$ | $10^1$ | $10^0$ |
| 10 x 10 | 10 | 1 |





## If we needed to represent a number larger than 999 in decimal?

- We'd add **another place to the left**, the thousands place, and its weight would be equal to $10^3$ ($10 \times 10 \times 10$), which is 1,000
- This pattern continues so that we can represent any large whole number by *adding more places as needed*

## Binary is still a place-value system

- So, the fundamental mechanics are the same as decimal
- But there are a couple of changes
  - First, *each place* represents a **power of 2**, rather than a power of 10
  - Second, each place can only have **one of two symbols**, rather than ten
    - Those two symbols are **0** and **1**

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | BINARY REPRESENTATIONS | L2.41

41

## Some useful powers of 2 to remember

- These powers occur frequently in computing
- Good idea to memorize these values!

| n | $2^n$ |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1,024 |
| 11 | 2,048 |
| 12 | 4,096 |
| 13 | 8,192 |
| 14 | 16,384 |
| 15 | 32,768 |
| 16 | 65,536 |

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | BINARY REPRESENT

42

## Consider the binary number: 101 (or 0b101)

- That may look like one hundred and one to you, but when dealing in binary, this is actually a representation of five!

| 1 | 0 | 1 |
|---|---|---|
| Fours place | Twos place | Ones place |
| $2^2$ | $2^1$ | $2^0$ |
| 2x2 = 4 | 2 | 1 |



## Just like in decimal, each place has a weight equal to the base raised to various powers

- Since we are in base 2, the rightmost place is 2 raised to 0; $\mathbf{2^0}=1$
- The next place is 2 raised to 1; $\mathbf{2^1}=2$ and …
- The next place is 2 raised to 2; $\mathbf{2^2}=4$
- Also, just like in decimal, to get the total value:
  1. Multiply the symbol in each place by the **place-value weight** and
  2. Sum the results
     - (4 x 1) + (2 x 0) + (1 x 1) = 5

| 1 | 0 | 1 |
|---|---|---|
| Fours place | Twos place | Ones place |
| $2^2$ | $2^1$ | $2^0$ |
| 2x2 = 4 | 2 | 1 |

## Let's look at a few more examples [1/4]

- What is the number: 0b1010
  - $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - $8 + 0 + 2 + 0 = 10$
- How about: 0b1111
  - $1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - $8 + 4 + 2 + 1 = 15$



## Let's look at a few more examples [2/4]

- What is the number: 0b00001010
  - $0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - $0 + 0 + 0 + 0 + 8 + 0 + 2 + 0 = 10$
- How about: 0b00001111
  - $0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - $0 + 0 + 0 + 0 + 8 + 4 + 2 + 1 = 15$

## Let's look at a few more examples [3/4]

- What is the number: 0b10101010
  - $1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - $128 + 0 + 32 + 0 + 8 + 0 + 2 + 0 = 170$
- How about: 0b11111111
  - $1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$



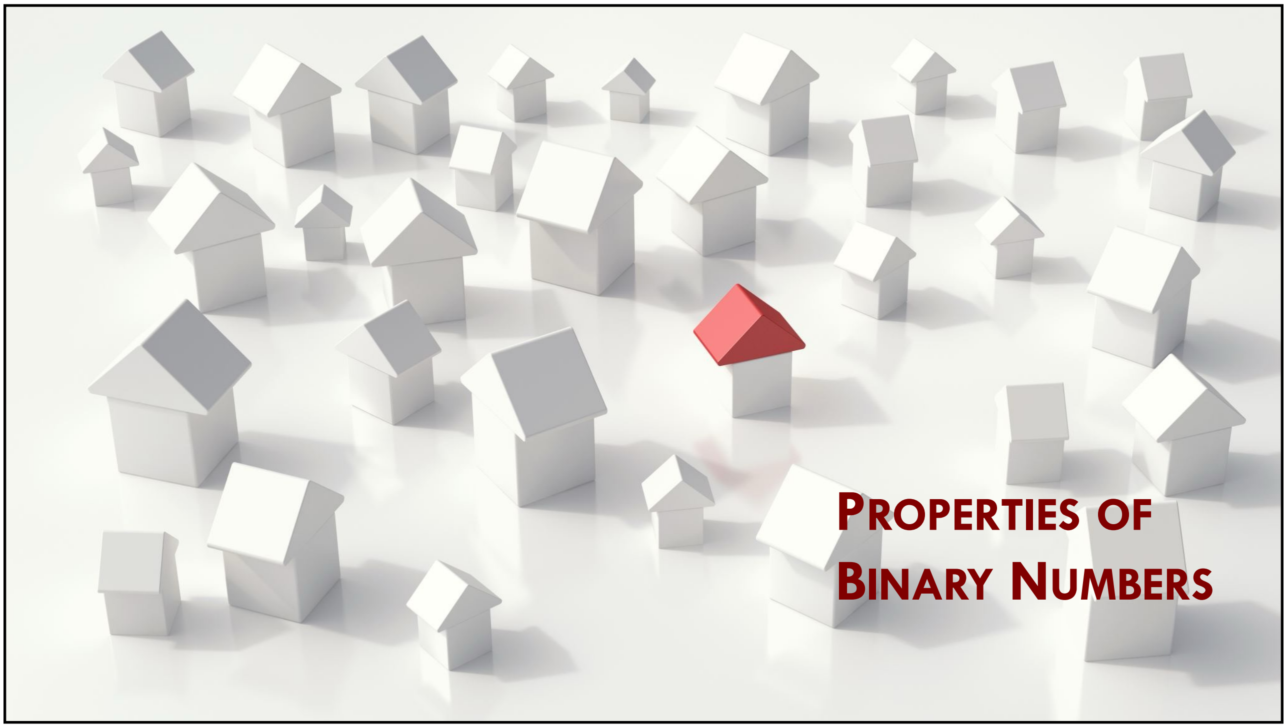
## Let's look at a few more examples [4/4]

- How about 0b1001110100100?
  - The answer is 5028

| $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

$= 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

$= 5028$




49

## Properties of binary numbers [1/8]

- If the least significant bit (position 0) of a binary integer value contains 1, the number is an odd number
  - E.g.: 0b0011?
    - 3
  - E.g.: 0b10101011= 171; 0b11110001 = 241
- If the least significant bit contains 0, then the number is even
  - E.g.: 0b11110000?
    - 240
  - E.g.: 0b10101010= 170; 0b11110010 = 242

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA, COMPUTER SCIENCE DEPARTMENT | BINARY REPRESENTATIONS | L2.50

50

## Properties of binary numbers [2/8]

- If the least significant ***n*** bits of a binary number all contain 0, then the number is **evenly divisible by $2^n$**
  - E.g.: 0b11110000 = 240; 4 LSB bits are 0
    - So, 240 is divisible by $2^4$=16
      - 240/16 = 15
  - E.g.: 0b11000000 = 192; 6 LSB bits are 0
    - So, 192 is divisible by $2^6$= 64
      - 192/64 = 3



## Properties of binary numbers [3/8]

- If a binary value contains all 1s from bit **position** 0 up to (but not including) bit position *n*, and all other bits are 0, then that value is equal to $2^n - 1$
  - Nota Bene: In computer science we count from 0
  - For e.g.; 0b00000111 positions 0-2 have 1's; all other bits starting at 3 are 0
    - Value = $2^3 - 1 = 7$
  - For e.g.; 0b01111111 positions 0-6 have 1's; all other bits starting at 7 are 0
    - Value = $2^7 - 1 = 128 - 1 = 127$

## Properties of binary numbers [4/8]

- Shifting all the bits in a number **to the left** by one position **multiplies** the binary value by 2
  - E.g.: 0b00000111 (value = 7)
    - Shift to the left (<<): 0b000001110 (value = $2^3 + 2^2 + 2^1 + 0 = 14$)
  - E.g.: 0b01010111 (value = 87)
    - Shift to the left (<<): 0b010101110
      - Value = $2^7 + 0 + 2^5 + 0 + 2^3 + 2^2 + 2^1 + 0 = 128 + 32 + 8 + 4 + 2 = 174$



## Properties of binary numbers [5/8]

- Shifting all the bits of an unsigned binary number **to the right** by one position effectively **divides** that number by 2
  - This does not apply to signed integer values
  - **Odd** numbers are *rounded down*
- E.g.: 0b01010110 (value = 86)
  - Shift to the right (>>): 0b0101011~~0~~ = 0b0101011 = 43
- E.g.: 0b01010111 (value = 87)
  - Shift to the right (>>): 0b0101011~~1~~ = 0b0101011 = 43

## Properties of binary numbers [6/8]

- Multiplying two *n*-bit binary values together may require *as many as* $2 \times n$ bits to hold the result
  - Biggest n-digit decimal number is $10^n - 1$ (e.g., for $n = 3$, it's 999). Worst case: $(10^n - 1)^2 < 10^{2n}$. For e.g.: 999×999=998001 → 6 digits = 2 x 3
  - Binary version is the same story with base 2: max n-bit value is $2^n - 1$. And, $(2^n - 1)^2 < 2^{2n}$ so you may need up to $2n$ bits
- Adding or subtracting two *n*-bit binary values never requires more than $n + 1$ bits to hold the result





## Properties of binary numbers [7/8]

- *Incrementing* (adding 1 to) the largest unsigned binary value for a given number of bits always produces a value of 0
  - 0b11111111 = 0b00000000 (the last carryover of 1 overflows)
- *Decrementing* (subtracting 1 from) 0 always produces the largest unsigned binary value for a given number of bits

## Number of unique combinations in a byte?

- Another way to think about this question is how many unique combinations of 0s and 1s can we make with our 8 bits?
- Let's first illustrate this with 4-bits



57

## 16 unique combinations of 0s and 1s in a 4-bit number, ranging in decimal value from 0 to 15

- We could determine the largest possible number that 4 bits can represent by setting all the bits to one, giving us 0b1111
  - That is 15 in decimal
- if we add 1 to account for representing 0, then we come to our total of 16

| Binary | Decimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |



58

## Properties of binary numbers [8/8]

- In general, for n bits
  - The total number of unique combinations: $2^n$
  - The largest possible number is $2^n - 1$



## The contents of this slide-set are based on the following references


- Peter J. Denning and Matti Tedre. *Computational Thinking*. Essential Knowledge series. The MIT Press. ISBN-10:ISBN-13 0262536560/ 978-0262536561. 2019. [Chapter 2]
- Matthew Justice. *How Computers Really Work: A Hands-On Guide to the Inner Workings of the Machine*. ISBN-10/ISBN-13 : 1718500661/ 978-1718500662. No Starch Press. 2020. [Chapter 1]
- Jonathan E. Steinhart. *The Secret Life of Programs: Understand Computers -- Craft Better Code*. ISBN-10/ ISBN-13 : 1593279701/ 978-1593279707. No Starch Press. [Chapter 1]
- Randall Hyde. Write Great Code, Volume 1, 2nd Edition: Understanding the Machine 2nd Edition. ASIN: B07VSC1K8Z. No Starch Press. 2020. [Chapter 2]