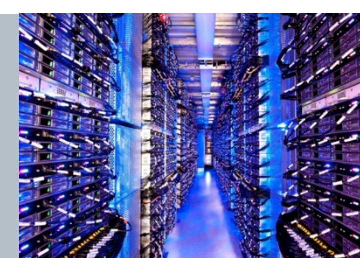
# Introduction to Computer Science


William Hsu
Advanced Computation Laboratory
Department of Computer Science and Engineering
Department of Environmental Biology and Fisheries Science
National Taiwan Ocean University

# Chapter 1: Data Storage

*If the average MP3 encoding for mobile is around 1MB per minute, and the average song lasts about four minutes, then a petabyte of songs would last over 2,000 years playing continuously.*

- 1.1 Bits and Their Storage
- 1.2 Main Memory
- 1.3 Mass Storage
- 1.4 Representing Information as Bit Patterns
- 1.5 The Binary System
- 1.6 Storing Integers
- 1.7 Storing Fractions
- 1.8 Data and Programming
- 1.9 Data Compression
- 1.10 Communications Errors

# Bits and Bit Patterns

- **Bit:** Binary Digit (0 or 1)
- Bit Patterns are used to represent information
  - Numbers
  - Text characters
  - Images
  - Sound
  - And others

# Boolean Operations

- **Boolean Operation:** An operation that manipulates one or more true/false values.
- Specific operations
  - AND
  - OR
  - XOR (exclusive or)
  - NOT

# The Possible Input and Output Values of Boolean Operations AND, OR, and XOR (exclusive or)

**The AND operation**

| Operation | Input 1 | Input 2 | Output |
|---|---|---|---|
| AND | 0 | 0 | 0 |
| AND | 0 | 1 | 0 |
| AND | 1 | 0 | 0 |
| AND | 1 | 1 | 1 |

**The OR operation**

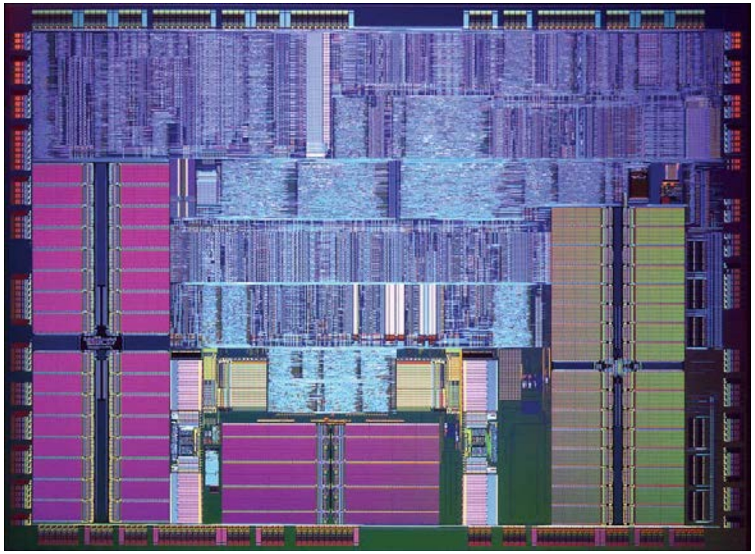
| Operation | Input 1 | Input 2 | Output |
|---|---|---|---|
| OR | 0 | 0 | 0 |
| OR | 0 | 1 | 1 |
| OR | 1 | 0 | 1 |
| OR | 1 | 1 | 1 |

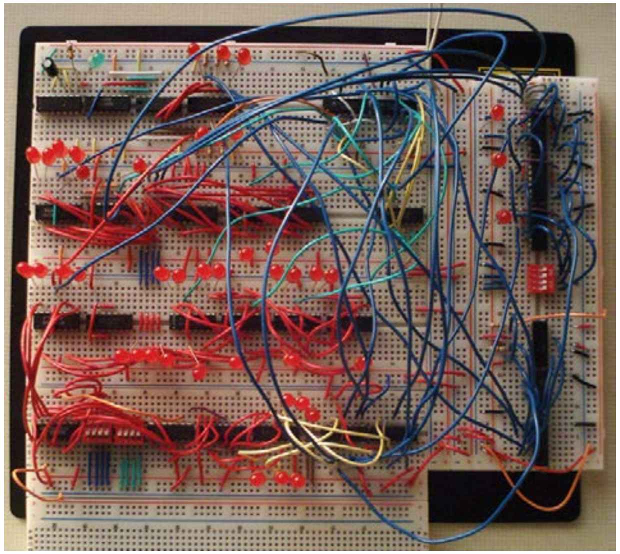
**The XOR operation**

| Operation | Input 1 | Input 2 | Output |
|---|---|---|---|
| XOR | 0 | 0 | 0 |
| XOR | 0 | 1 | 1 |
| XOR | 1 | 0 | 1 |
| XOR | 1 | 1 | 0 |

# Gates

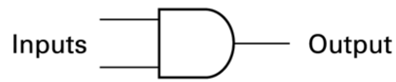
- **Gate:** A device that computes a Boolean operation.
  - Often implemented as (small) electronic circuits.
  - Provide the building blocks from which computers are constructed.
  - VLSI (Very Large Scale Integration).

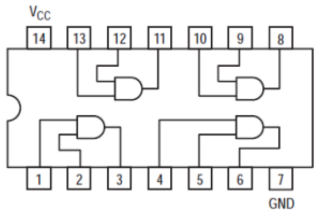




# A Pictorial Representation of AND, OR, XOR, and NOT Gates as Well as Their Input and Output Values

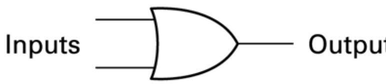
**AND**



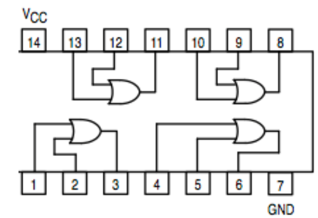




| Inputs | Output |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

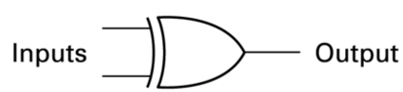

**OR**




| Inputs | Output |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 1 |

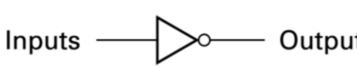



**XOR**




| Inputs | Output |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |

**NOT**




| Inputs | Output |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Basic Characteristics of Digital ICs

- Digital ICs are a collection of resistors, diodes and transistor fabricated on a single piece of semiconductor material called a substrate, which is commonly referred to as a chip.
- The chip is enclosed in a package.
- Dual-in-line package (DIP)

# IC Families

- TTL Family: bipolar digital ICs
- CMOS Family: unipolar digital ICs
- TTL and CMOS dominate the field of SSI and MSI devices.

# Integrated circuits (IC)

| Complexity | Number of Gates |
|---|---|
| Small-scale integration(SSI) | <12 |
| Medium-scale integration(MSI) | 12 to 99 |
| Large-scale integration(LSI) | 100 to 9999 |
| Very large-scale integration(VLSI) | 10,000 to 99,999 |
| Ultra large-scale integration(ULSI) | 100,000 to 999,999 |
| Giga-scale integration (GSI) | 1,000,000 or more |

# TTL Family

| TTL Series | Prefix | Example IC |
|---|---|---|
| Standard TTL | 74 | 7404 (hex inverter) |
| Schottky TTL | 74S | 74S04 |
| Low-power Schottky TTL | 74LS | 74LS04 |
| Advanced Schottky TTL | 74AS | 74AS04 |
| Advanced low-power Schottky TTL | 74ALS | 74ALS04 |

# CMOS Family

| CMOS Series | Prefix | Example IC |
|---|---|---|
| Metal-gate CMOS | 40 | 4001 |
| Metal-gate, pin-compatible with TTL | 74C | 74C02 |
| Silicon-gate, pin-compatible with TTL, high-speed | 74HC | 74HC02 |
| Silicon-gate, high-speed, pin-compatible and electrically compatible with TTL | 74HCT | 74HCT02 |
| Advanced-performance CMOS, not pin or electrically compatible with TTL | 74AC | 74AC02 |
| Advanced-performance CMOS, not pin but electrically compatible with TTL | 74ACT | 74ACT02 |

# Power and Ground

- To use digital IC, it is necessary to make proper connection to the IC pins.
- Power: labeled $V_{cc}$ for the TTL circuit, labeled $V_{DD}$ for CMOS circuit.
- Ground

# Logic-level Voltage Ranges

› For TTL devices, $V_{CC}$ is normally 5V.

› For CMOS circuits, $V_{DD}$ can range from 3 ~ 18V.

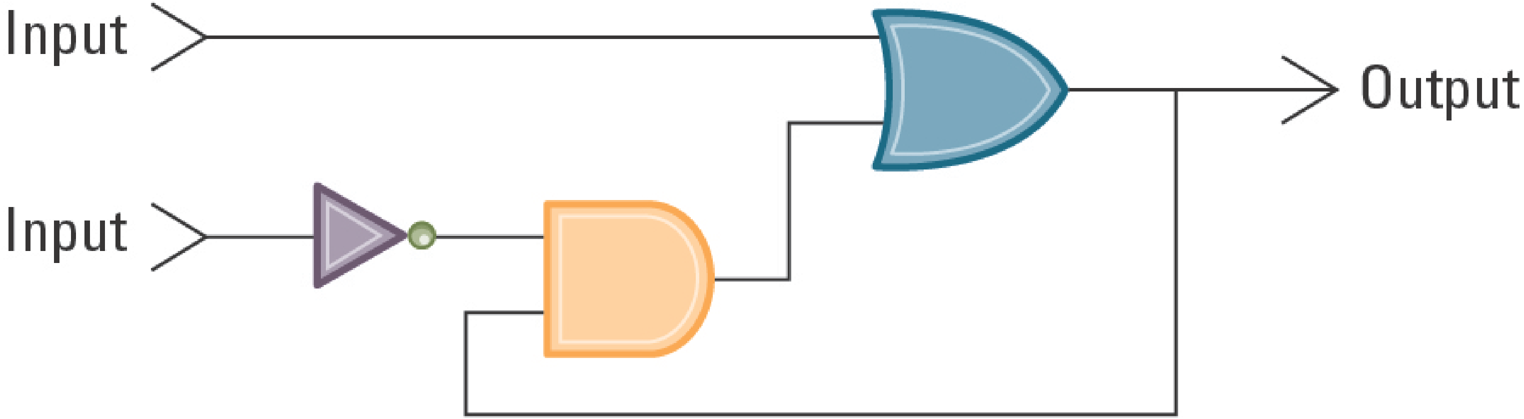
› For TTL
  – logic 0 : 0 ~ 0.8V
  – logic 1: 2 ~ 5V

› For CMOS
  – logic 0 : 0 ~ 1.5V
  – logic 1: 3.5 ~5V

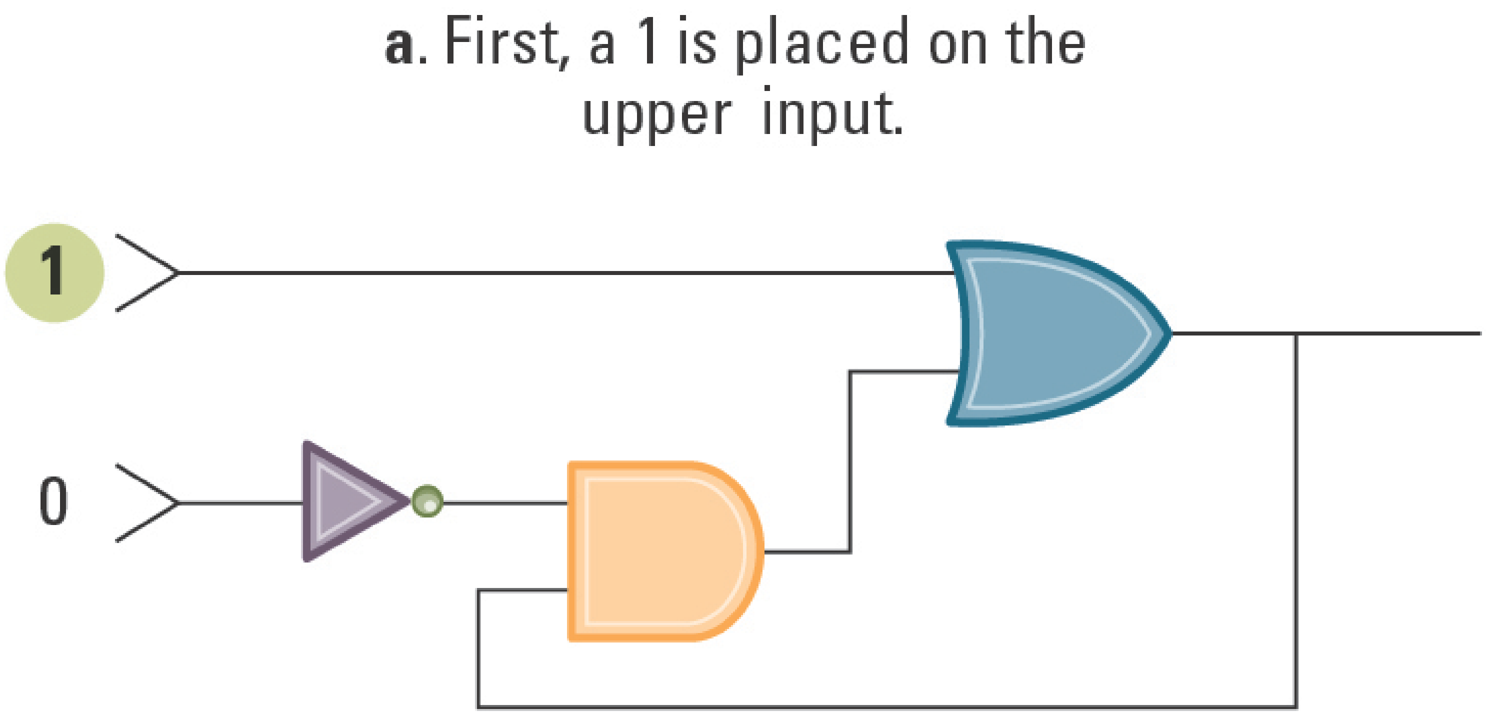
# Flip-flops

- **Flip-flop:** A circuit built from gates that can store one bit.
  - One input line is used to set its stored value to 1
  - One input line is used to set its stored value to 0
  - While both input lines are 0, the most recently stored value is preserved

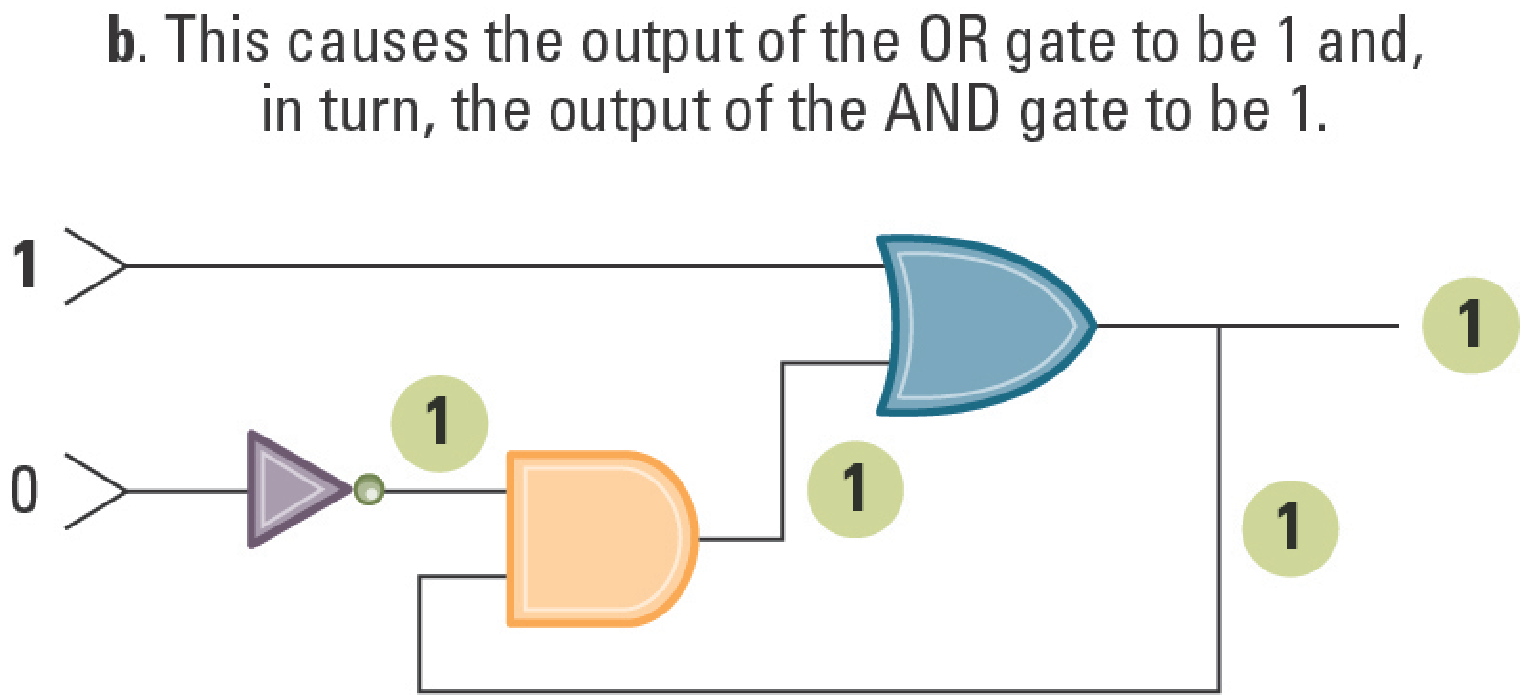
# A Simple Flip-flop Circuit



# Setting the Output of a Flip-flop to 1

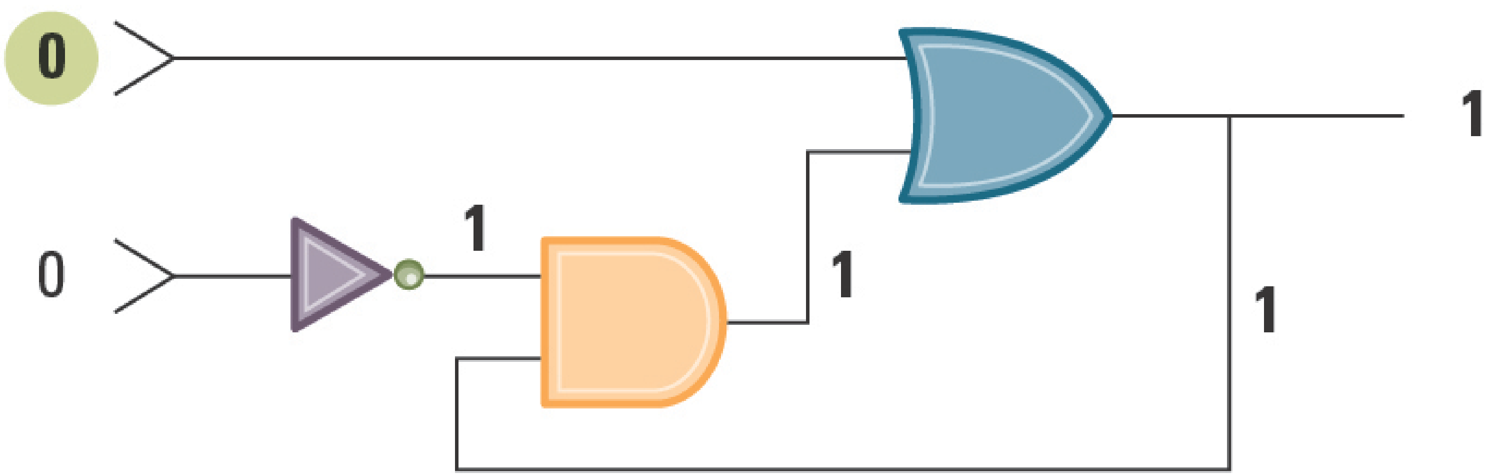


# Setting the Output of a Flip-flop to 1

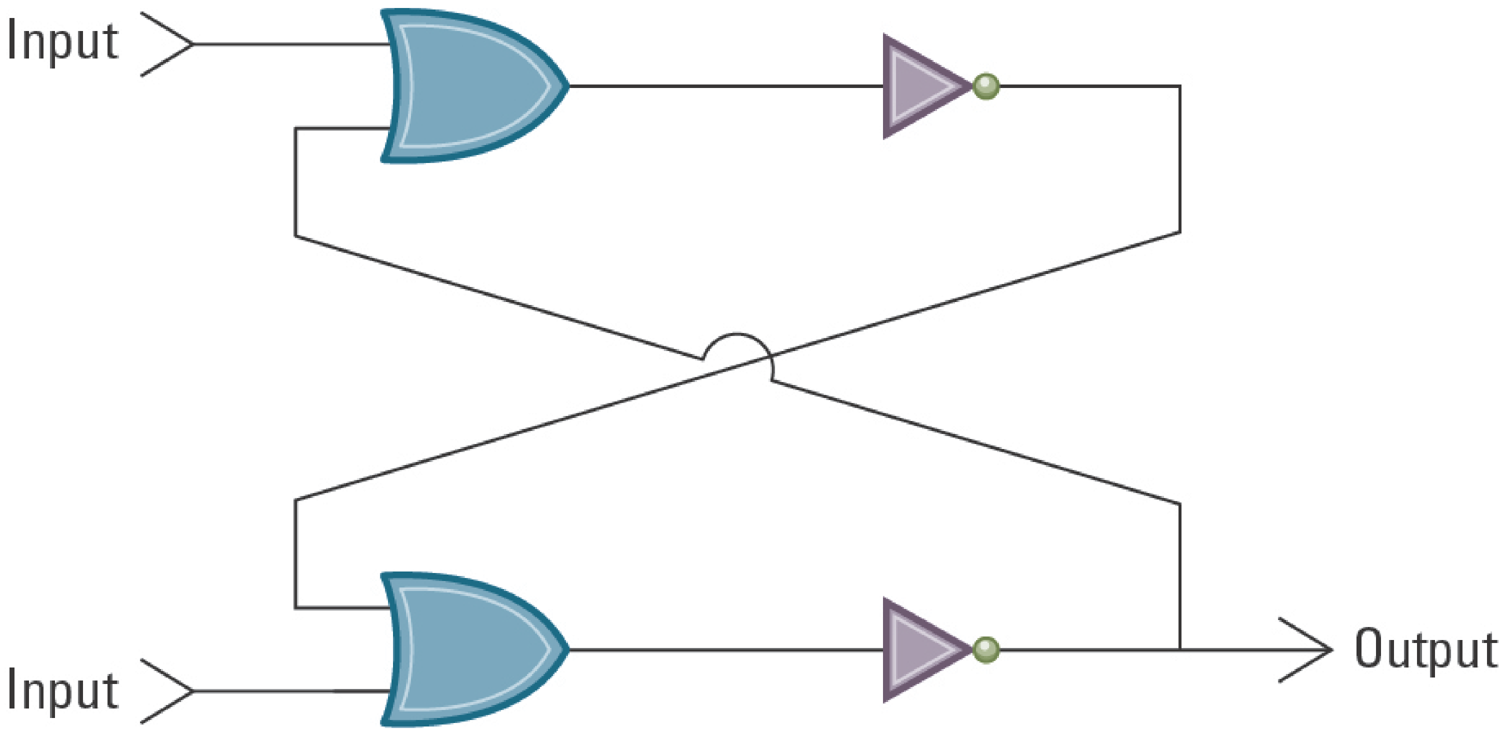


# Setting the Output of a Flip-flop to 1

**c.** Finally, the 1 from the AND gate keeps the OR gate from changing after the upper input returns to 0.

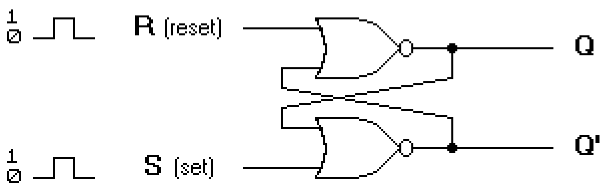


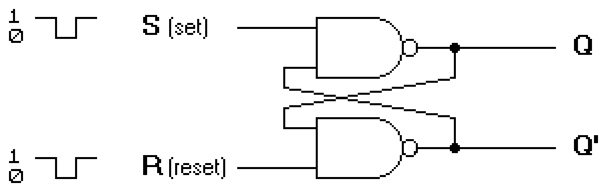
# Another way of constructing a Flip-flop



# Types of Flip-flops

› RS (SR) flip-flop

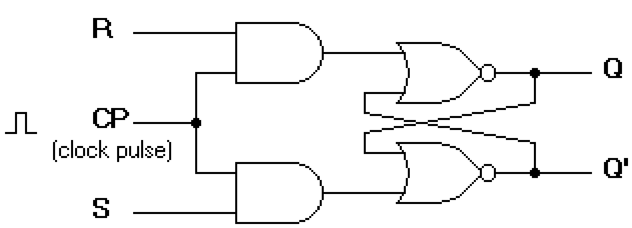



| S | R | Q | Q' | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | (after S=1, R=0) |
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 0 | 1 | (after S=0, R=1) |
| 1 | 1 | 0 | 0 | |

| S | R | Q | Q' | |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | (after S=1, R=0) |
| 0 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 0 | (after S=0, R=1) |
| 0 | 0 | 1 | 1 | |

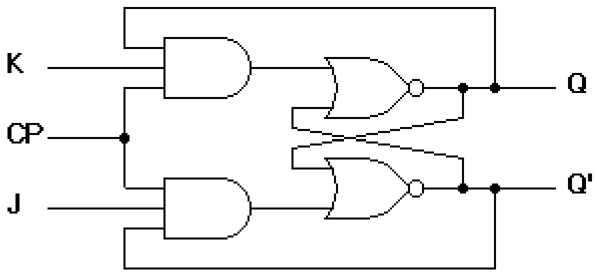
› Clocked RS (SR) flip-flop




| Q | S | R | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | indeterminate |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | indeterminate |

# Types of Flip-flops

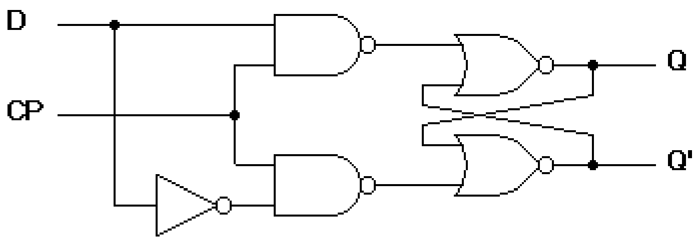
› JK flip-flop




| Q | J | K | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

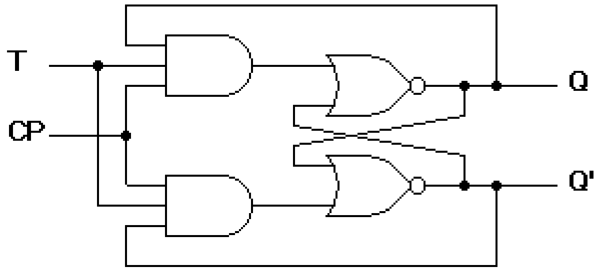
# Types of Flip-flops

- D flip-flop




| Q | D | Q(t+1) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

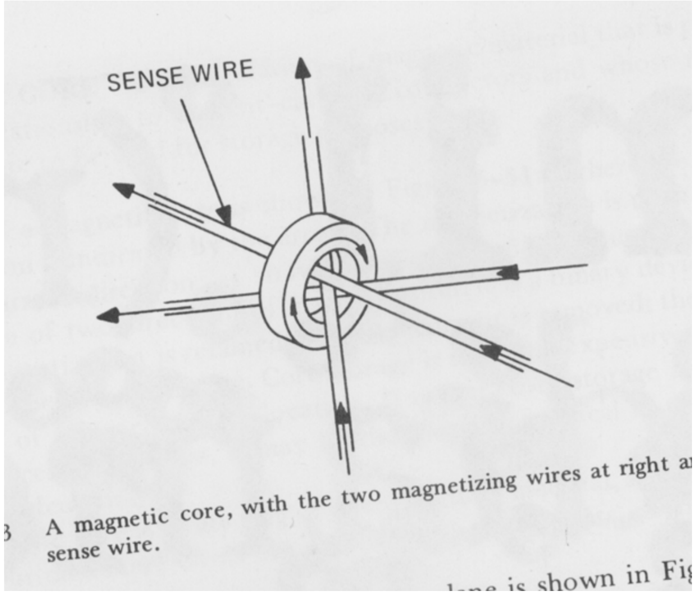
- T flip-flop



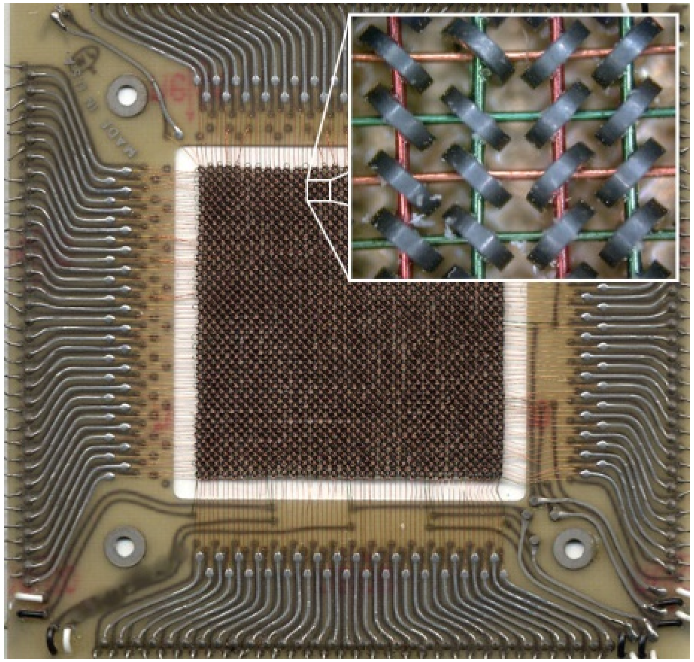

| Q | T | Q(t+1) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# How Was it Done in the Past?

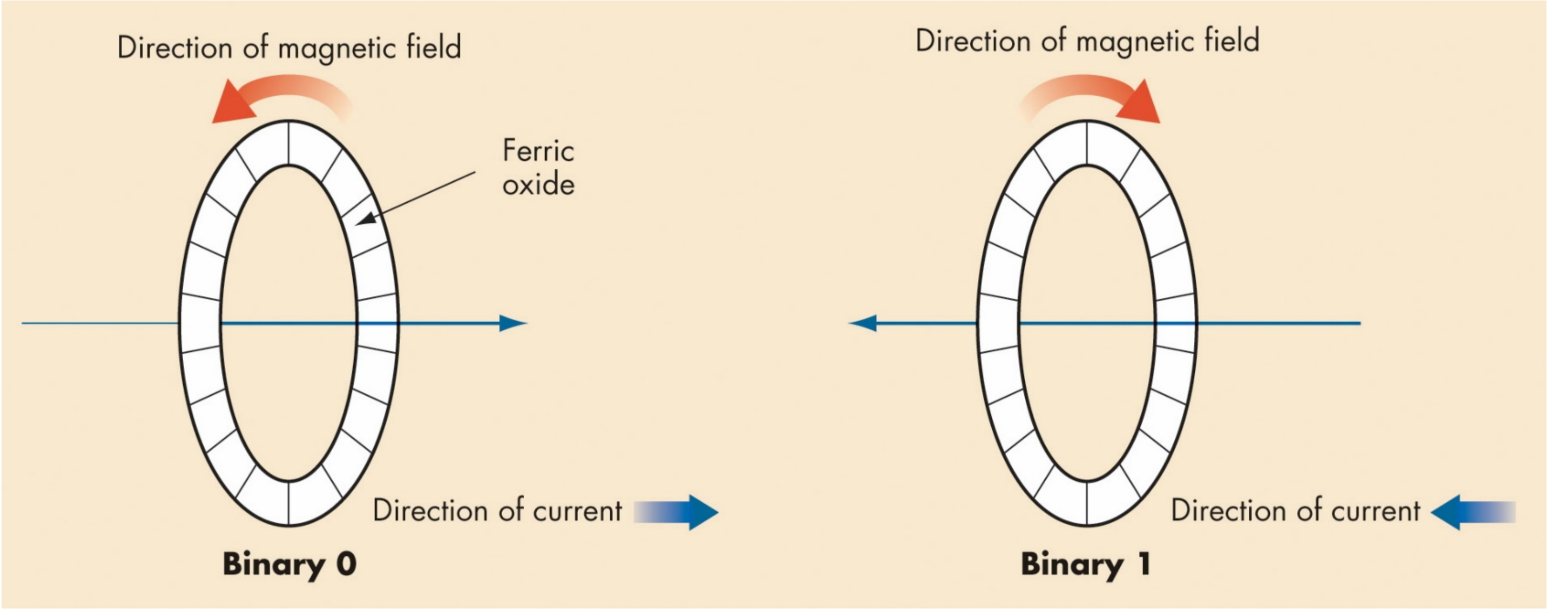
› Magnetic cores.



A magnetic core, with the two magnetizing wires at right a
sense wire.



# Using Magnetic Cores to Represent Binary Values



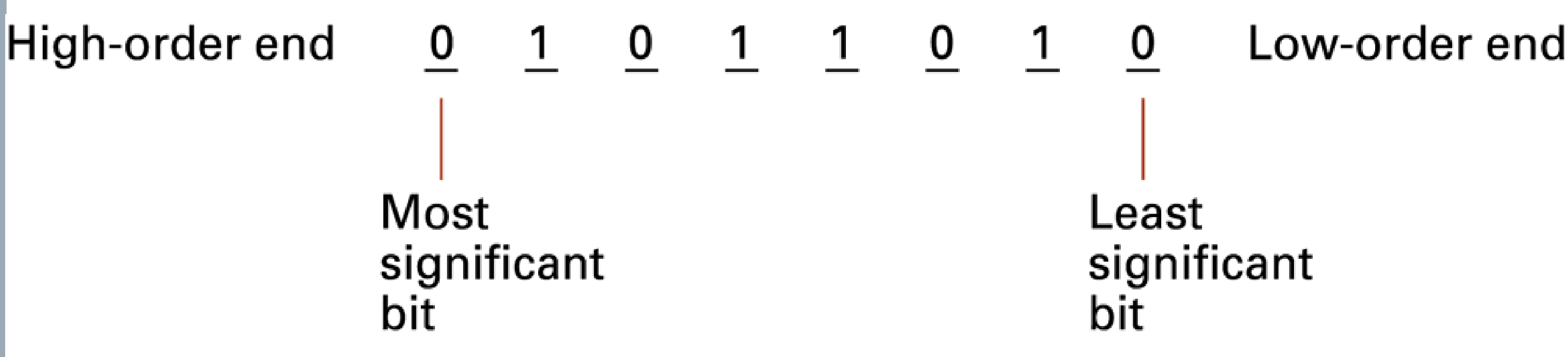
# Hexadecimal Notation

- **Hexadecimal notation:** A shorthand notation for long bit patterns
  - Divides a pattern into groups of four bits each
  - Represents each group by a single symbol
- Example: 10100011 becomes A3

| Bit pattern | Hexadecimal representation |
|---|---|
| 0000 | 0x0 |
| 0001 | 0x1 |
| 0010 | 0x2 |
| 0011 | 0x3 |
| 0100 | 0x4 |
| 0101 | 0x5 |
| 0110 | 0x6 |
| 0111 | 0x7 |
| 1000 | 0x8 |
| 1001 | 0x9 |
| 1010 | 0xA |
| 1011 | 0xB |
| 1100 | 0xC |
| 1101 | 0xD |
| 1110 | 0xE |
| 1111 | 0xF |

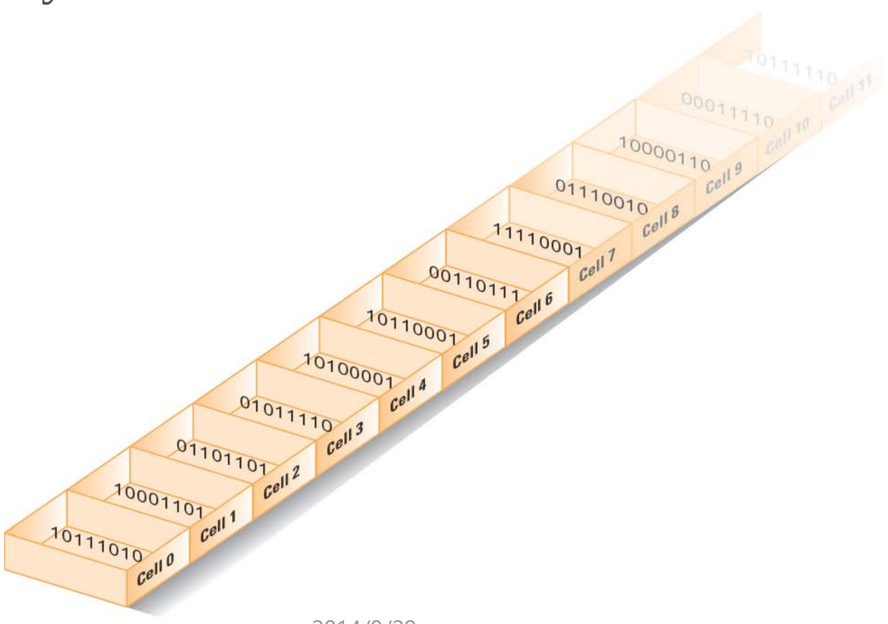
# Main Memory Cells

- **Cell:** A unit of main memory (typically 8 bits which is one **byte**)
  - **Most significant bit (MSB):** the bit at the left (high-order) end of the conceptual row of bits in a memory cell.
  - **Least significant bit (LSB):** the bit at the right (low-order) end of the conceptual row of bits in a memory cell.

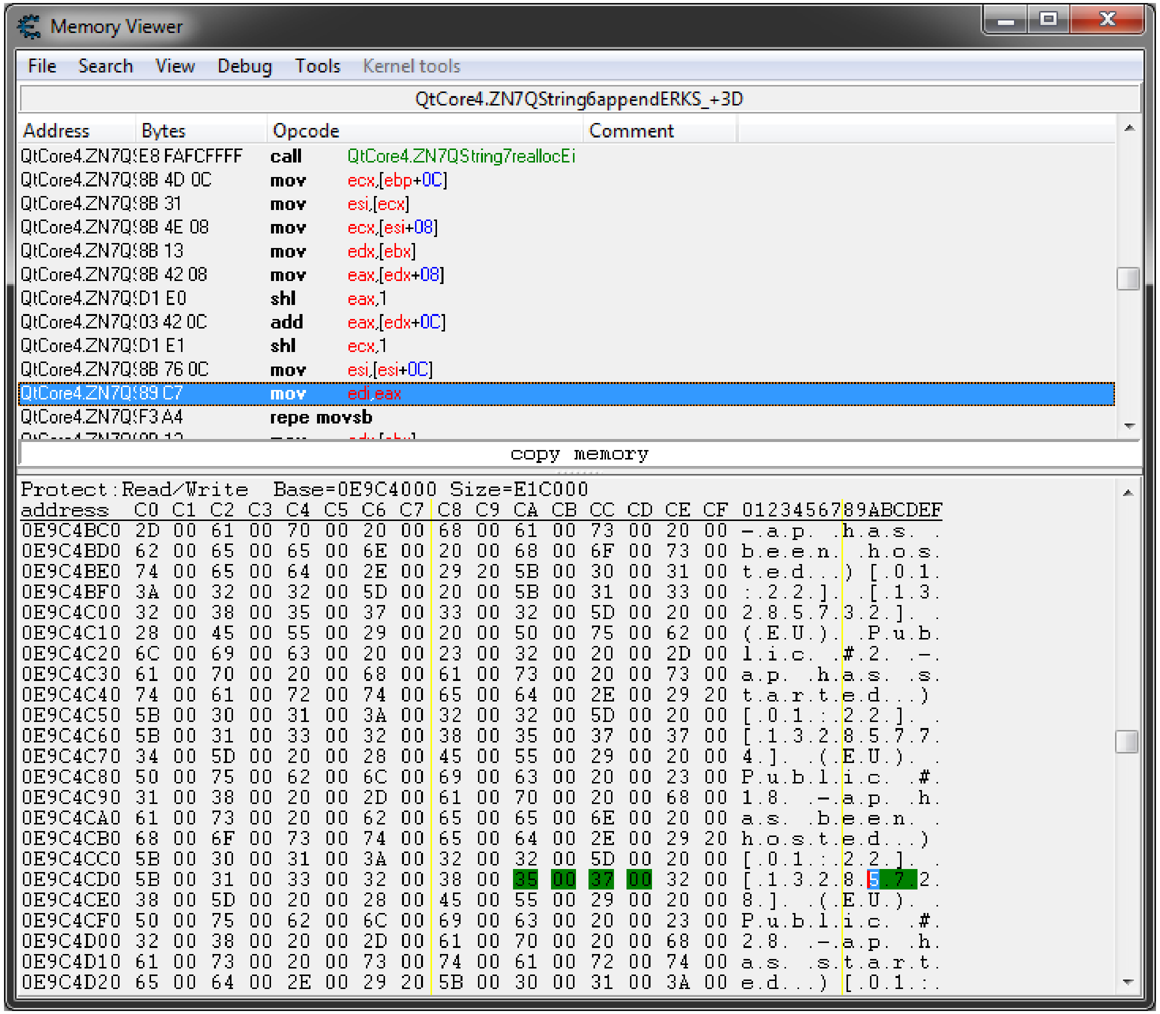


# Main Memory Addresses

› **Address:** A "name" that uniquely identifies one cell in the computer's main memory
  – The names are actually numbers.
  – These numbers are assigned consecutively starting at zero.
  – Numbering the cells in this manner associates an order with the memory cells.

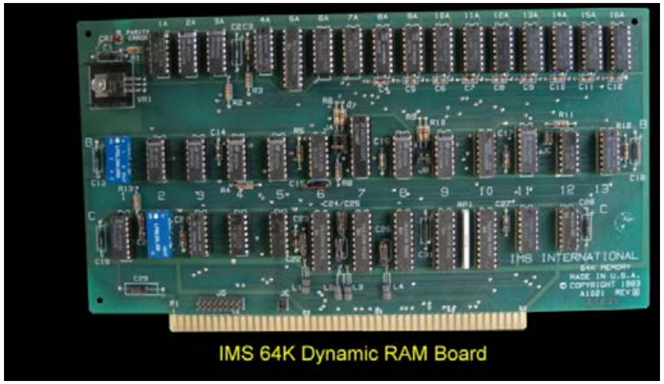


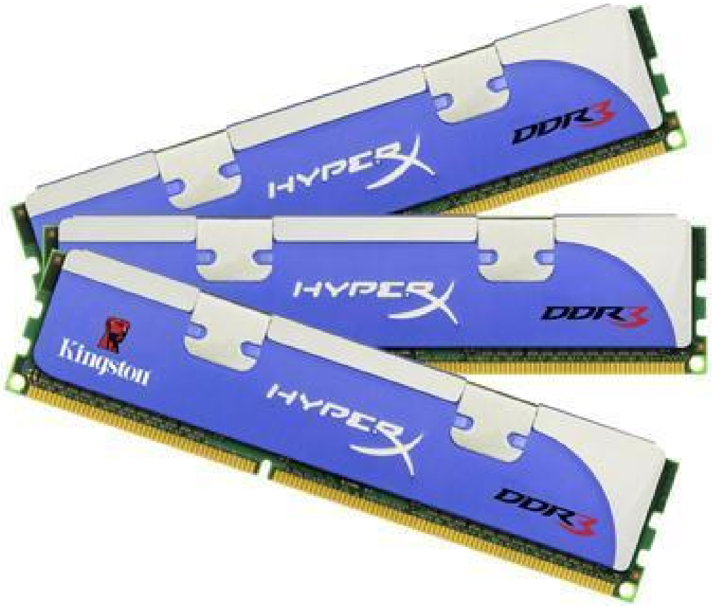
# Cheat engine memory table



# Memory Terminology

- **Random Access Memory (RAM):** Memory in which individual cells can be easily accessed in any order.
- **Dynamic Memory (DRAM):** RAM composed of volatile memory.



# Memory Terminology

- **Read-only memory** (**ROM**): refers to memory chips storing permanent data and instructions.
  - Used in firmwares.
- A PROM (programmable read-only memory) chip is a blank ROM chip that can be written to permanently
  - EPROM: Erased by ultraviolet rays.
  - EEPROM: Erased electronically.

# Measuring Memory Capacity

- **Kilobyte**: $2^{10}$bytes = 1024 bytes
  - Example: 3 KB = 3 * 1024 bytes
  - $\approx 10^3$ bytes
- **Megabyte**: $2^{20}$ bytes = 1,048,576 bytes
  - Example: 3 MB = 3 * 1,048,576 bytes
  - $\approx 10^6$ bytes
- **Gigabyte**: $2^{30}$ bytes = 1,073,741,824 bytes
  - Example: 3 GB = 3 * 1,073,741,824 bytes
  - $\approx 10^9$ bytes
- **Terabyte**, **Petabyte**
  - $\approx 10^{12}$ bytes, $\approx 10^{15}$ bytes
  - 1 PB ~ 220,000 DVDs (4.7G)

# Measuring Memory Capacity

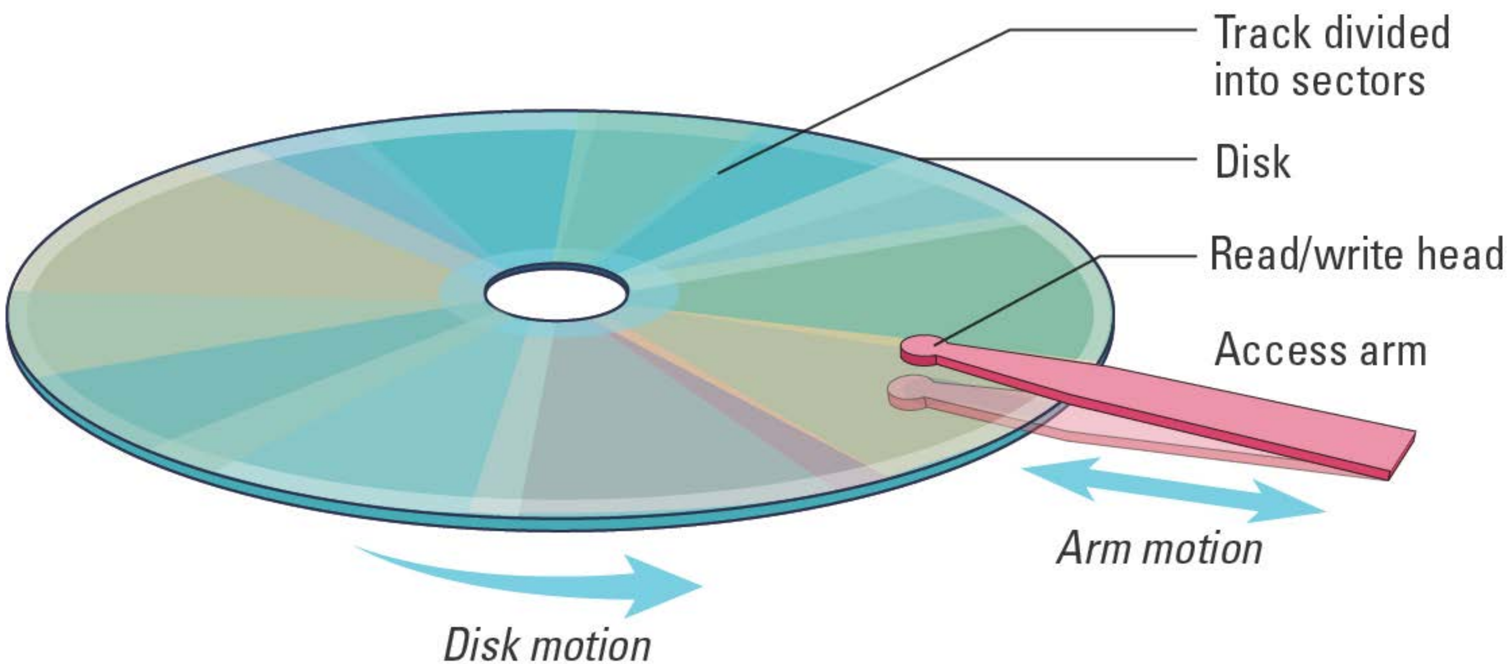
**Storage Terms**

| Storage Term | Approximate Number of Bytes | Exact Number of Bytes |
|---|---|---|
| Kilobyte (KB) | 1 thousand | $2^{10}$ or 1,024 |
| Megabyte (MB) | 1 million | $2^{20}$ or 1,048,576 |
| Gigabyte (GB) | 1 billion | $2^{30}$ or 1,073,741,824 |
| Terabyte (TB) | 1 trillion | $2^{40}$ or 1,099,511,627,776 |
| Petabyte (PB) | 1 quadrillion | $2^{50}$ or 1,125,899,906,842,624 |
| Exabyte (EB) | 1 quintillion | $2^{60}$ or 1,152,921,504,606,846,976 |
| Zettabyte (ZB) | 1 sextillion | $2^{70}$ or 1,180,591,620,717,411,303,424 |
| Yottabyte (YB) | 1 septillion | $2^{80}$ or 1,208,925,819,614,629,174,706,176 |

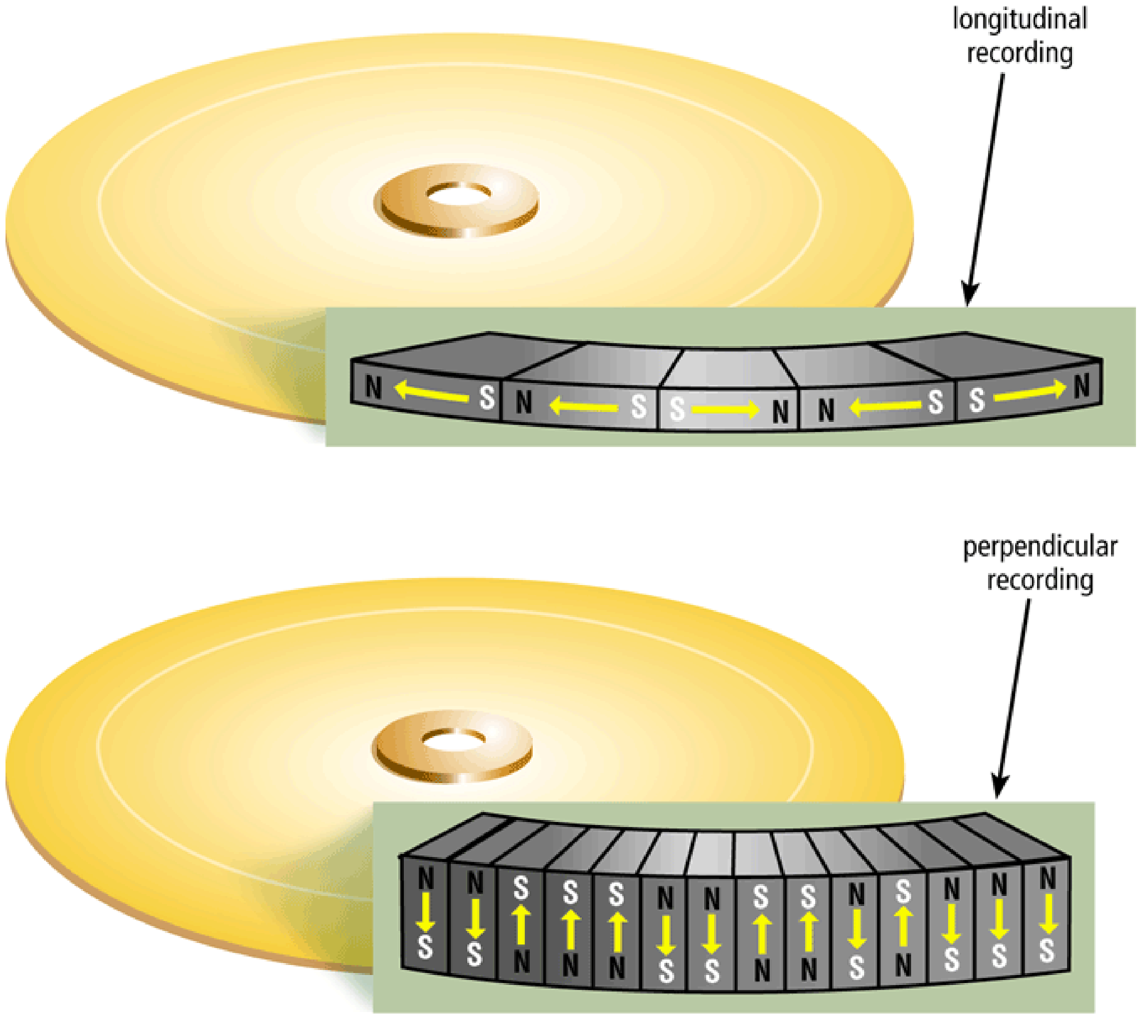
# Mass Storage

- Additional devices:
  - Magnetic disks (HD), Magnetic tape
  - CDs, DVDs
  - Flash drives, Solid-state disks
- Advantages over main memory.
  - Less volatility
  - Larger storage capacities
  - Low cost
  - In many cases can be removed

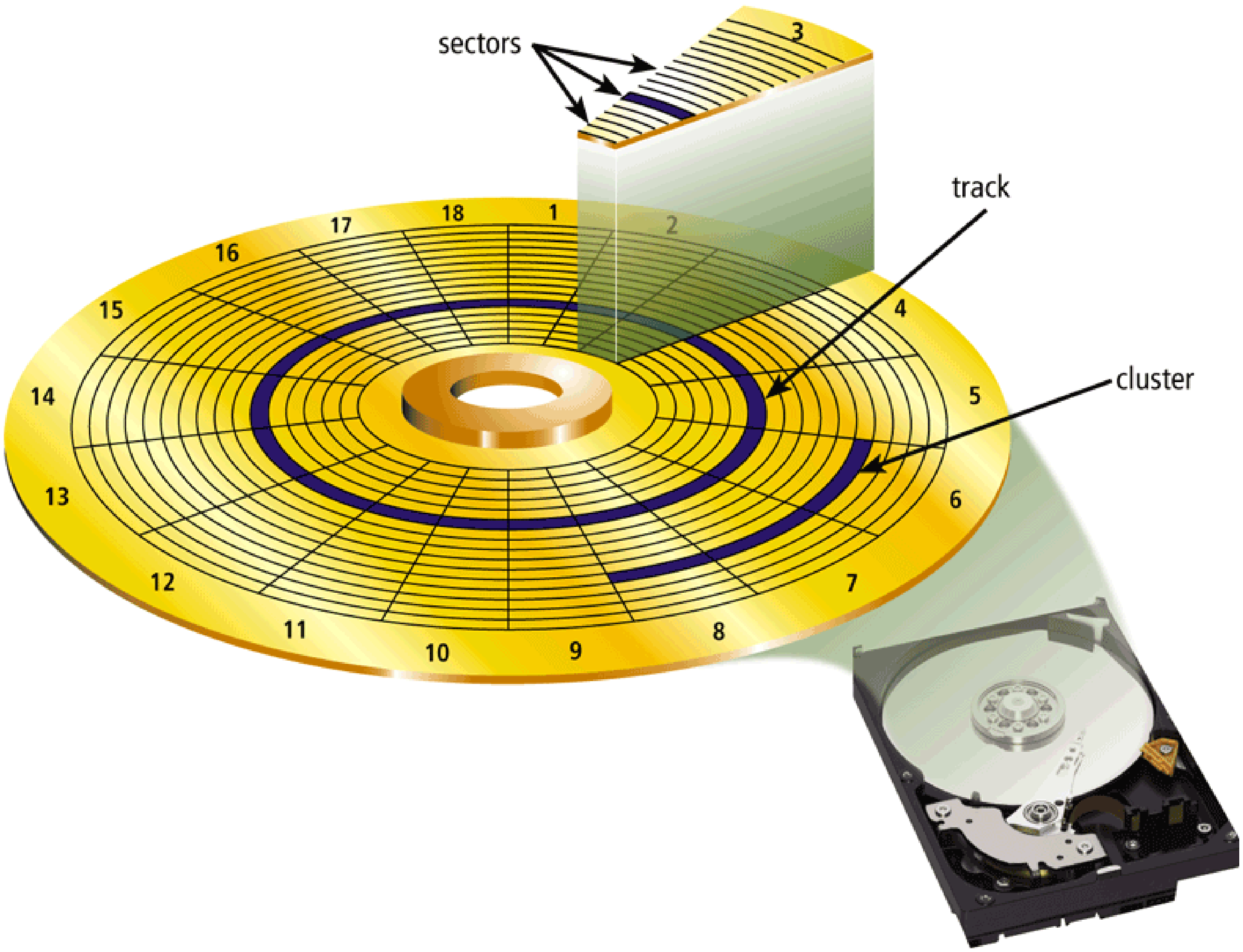
# A Magnetic Disk Storage System



# Hard Disk Storage



# Characteristics of Hard Drives

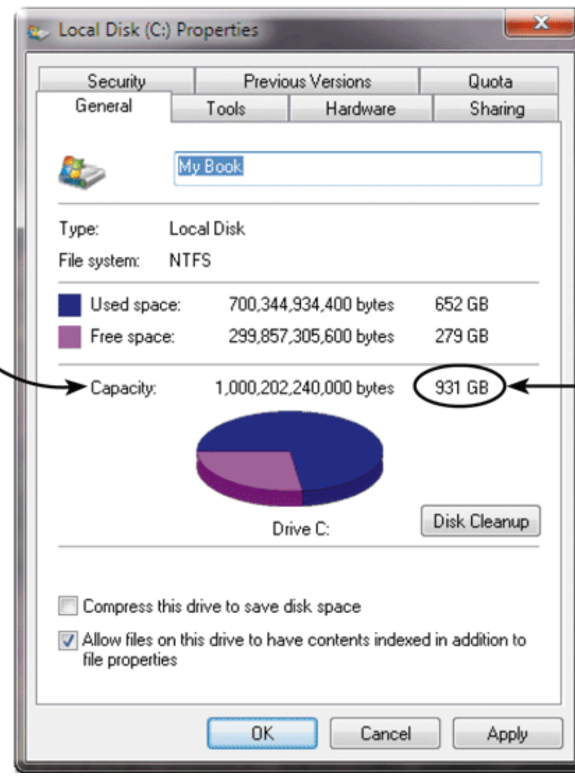


# Question: Why does Advertised Size differ from Actual Size?

| Sample Hard Disk Characteristics | |
|---|---|
| Advertised capacity | 1 TB |
| Platters | 4 |
| Read/write heads | 8 |
| Cylinders | 16,383 |
| Bytes per sector | 512 |
| Sectors per track | 63 |
| Sectors per drive | 1,953,525,168 |
| Revolutions per minute | 7,200 |
| Transfer rate | 300 MBps |
| Access time | 8.5 ms |

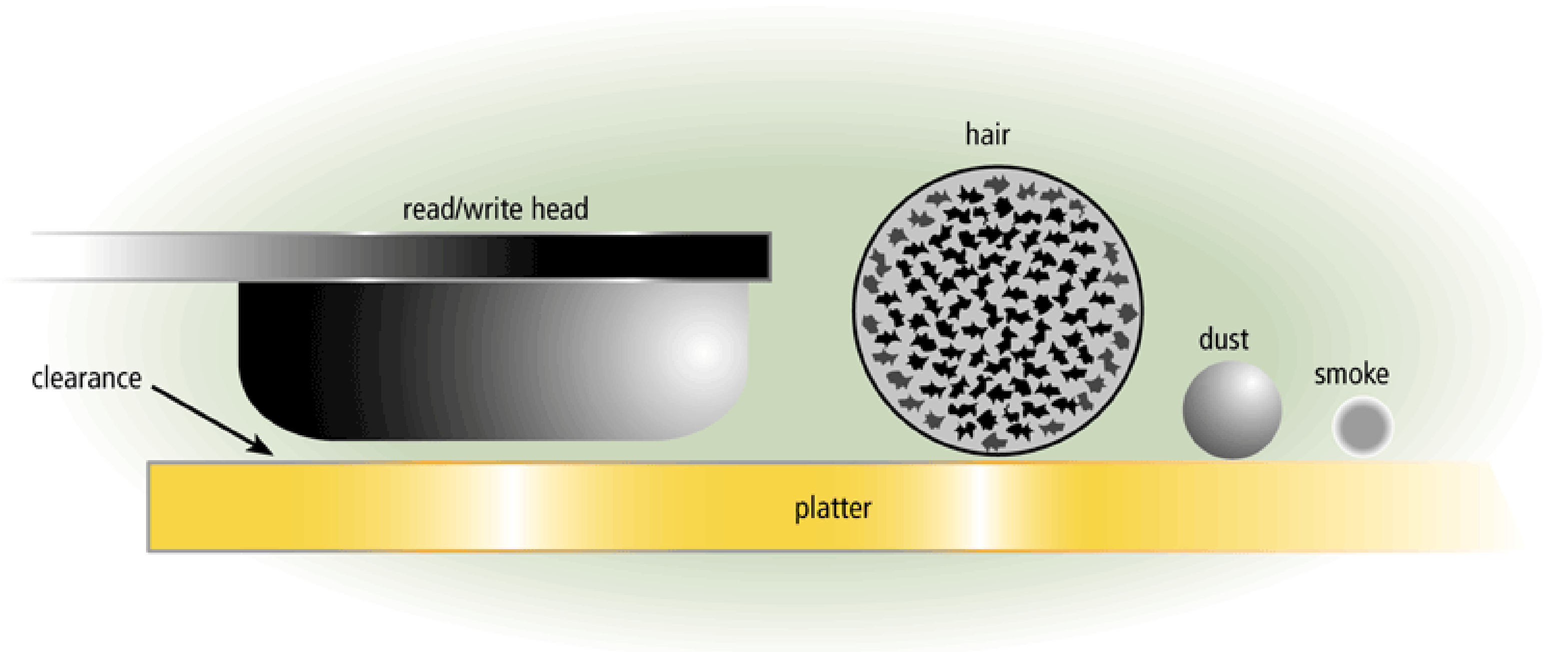
**1 TB disk can store any of the following:**

- 500,000,000 pages of text
- 285,000 digital photos
- 250,000 songs
- 120 hours of digital video

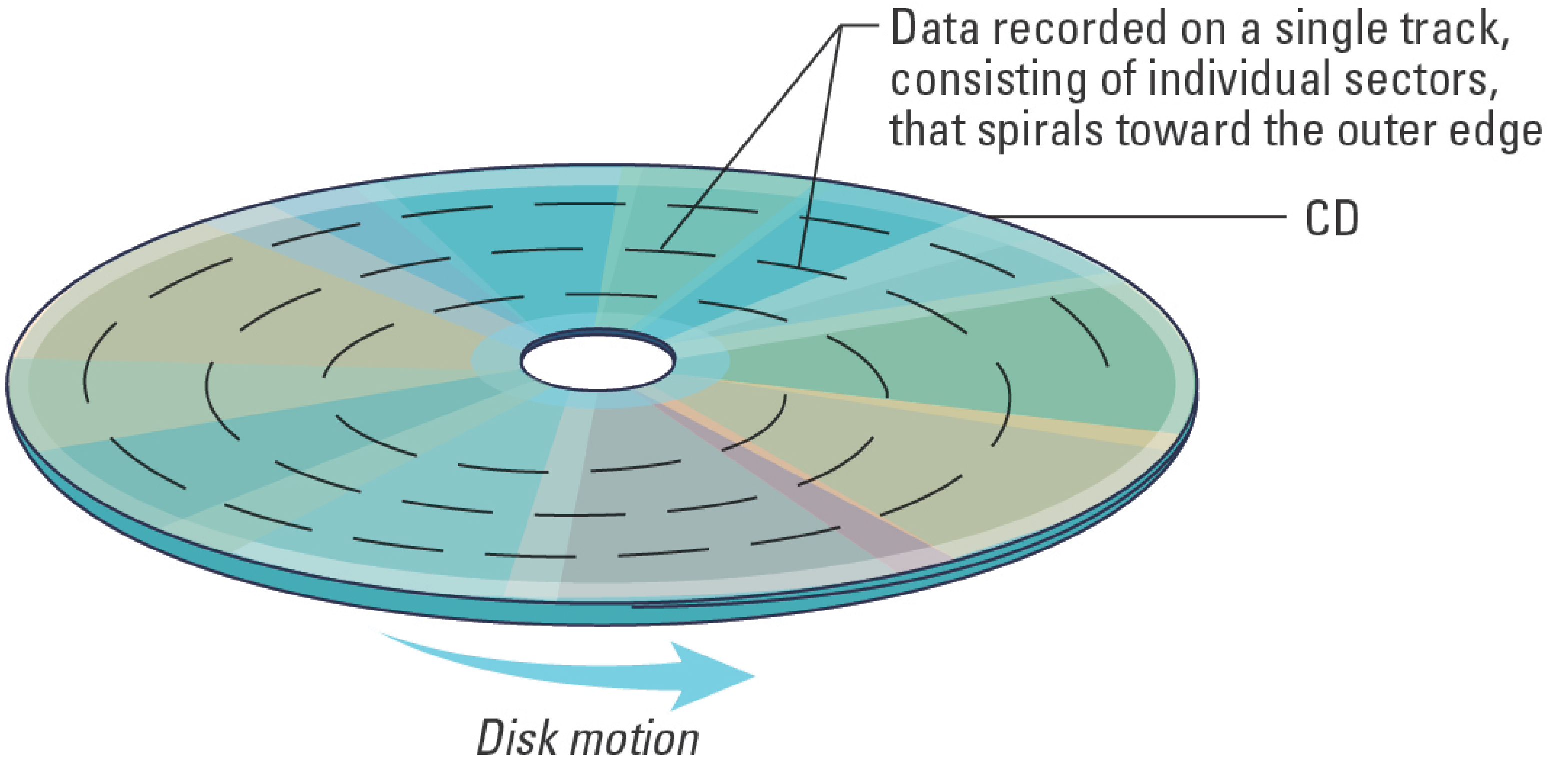



actual disk capacity

# Read/Write Head of Hard Drive



# CD Storage


Data recorded on a single track, consisting of individual sectors, that spirals toward the outer edge
CD
Disk motion

# Flash Drives

› **Flash Memory** – circuits that traps electrons in tiny silicon dioxide chambers.

› Repeated erasing slowly damages the media.

› Mass storage of choice for:

  – Digital cameras.


› **SD Cards** provide GBs of storage


# Other Types of Media

| Media Life Expectancies* (when using high-quality media) | | |
|---|---|---|
| **Media Type** | **Guaranteed Life Expectancy** | **Potential Life Expectancy** |
| Magnetic disks | 3 to 5 years | 20 to 30 years |
| Optical discs | 5 to 10 years | 50 to 100 years |
| Solid state drives | 50 years | 140 years |
| Microfilm | 100 years | 500 years |

* according to manufacturers of the media

# Representing Text

- **Each character (letter, punctuation, etc.) is assigned a unique bit pattern.**
  - **ASCII**: Uses patterns of 7-bits to represent most symbols used in written English text.
    - EASCII: Extended-ASCII uses 8-bits.
  - ISO developed a number of 8 bit extensions to ASCII, each designed to accommodate a major language group.
  - **Unicode**: Uses patterns up to 21-bits to represent the symbols used in languages world wide, 16-bits for world's commonly used languages.

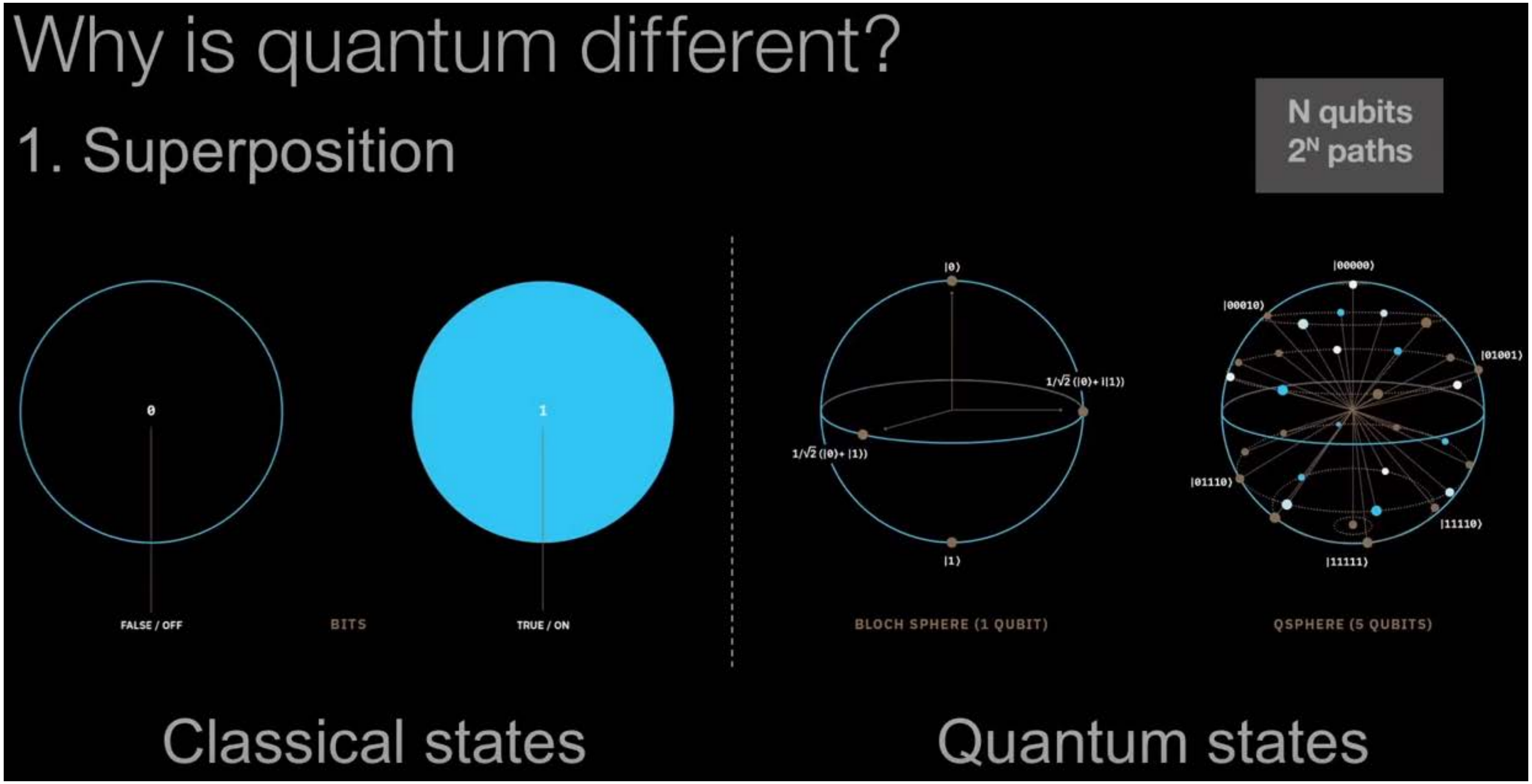
# The Message "Hello." in ASCII or UTF-8 Encoding

| | | | | | |
|---|---|---|---|---|---|
| 01001000 | 01100101 | 01101100 | 01101100 | 01101111 | 00101110 |
| H | e | l | l | o | . |

# Representing Numeric Values

- **Binary notation**: Uses bits to represent a number in base two.
  - All numeric values in a computer are stored in sequences of 0s and 1s
  - Counting from 0 to 8:
    - 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000
- Limitations of computer representations of numeric values.
  - Overflow: occurs when a value is too big to be represented
  - Truncation: occurs when a value cannot be represented accurately.

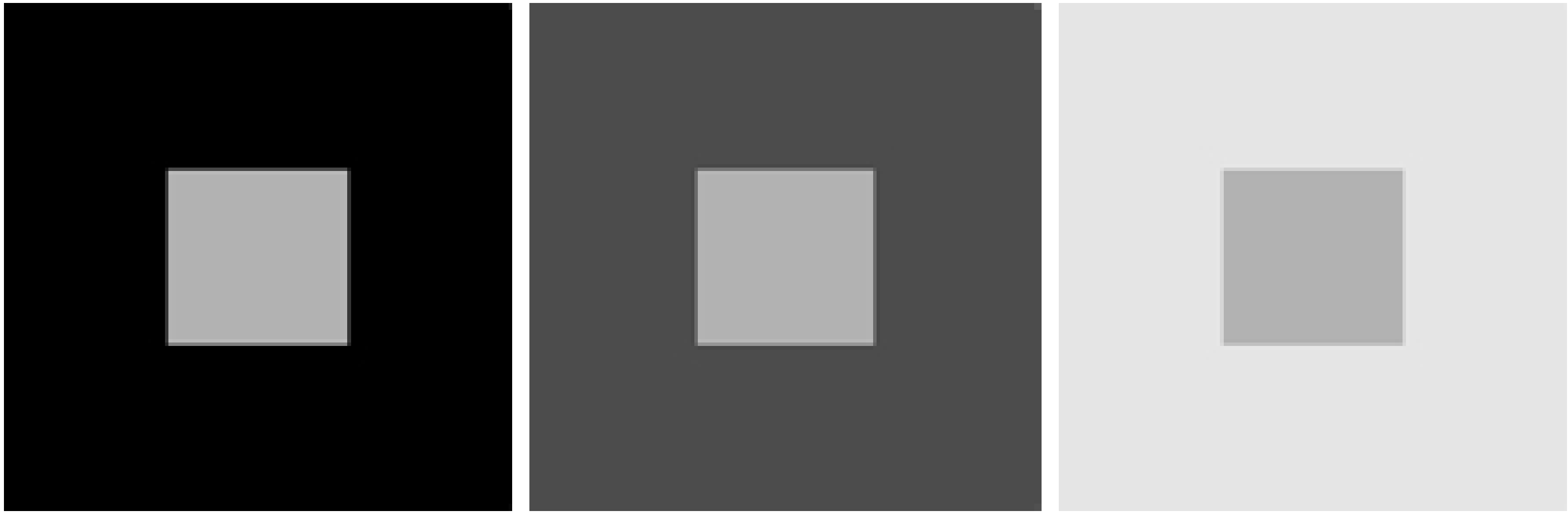
# Future: Quantum digits?



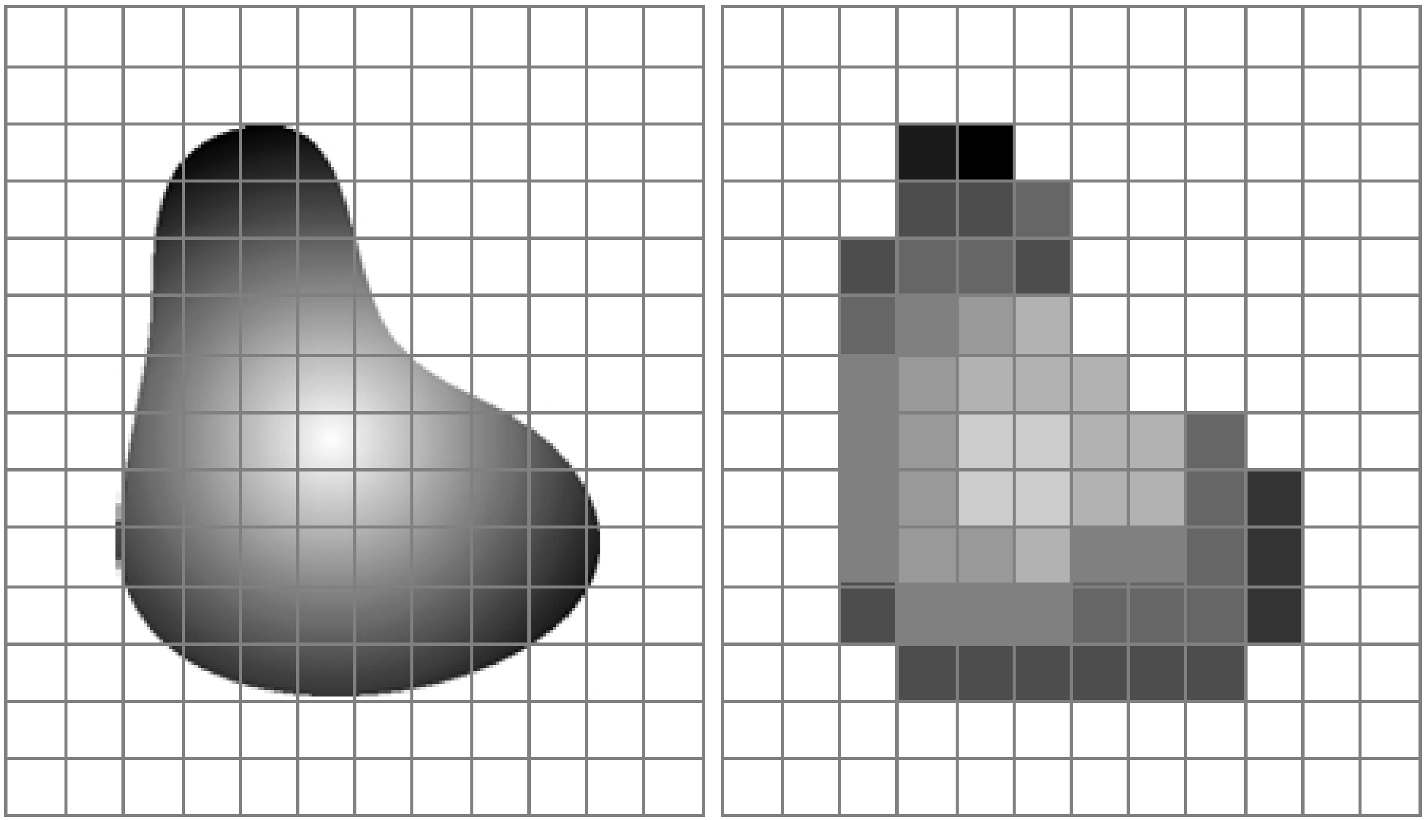
# Representing Images

- Bit map techniques
  - Pixel: short for “picture element”.
  - RGB: Red, Green, and Blue components
    - How about CYMK, HSI?
  - Luminance and chrominance
  - Problems with scaling up images
- Vector techniques
  - Represent images with geometric structures
  - Scalable
  - TrueType and PostScript

# Contrast and vision



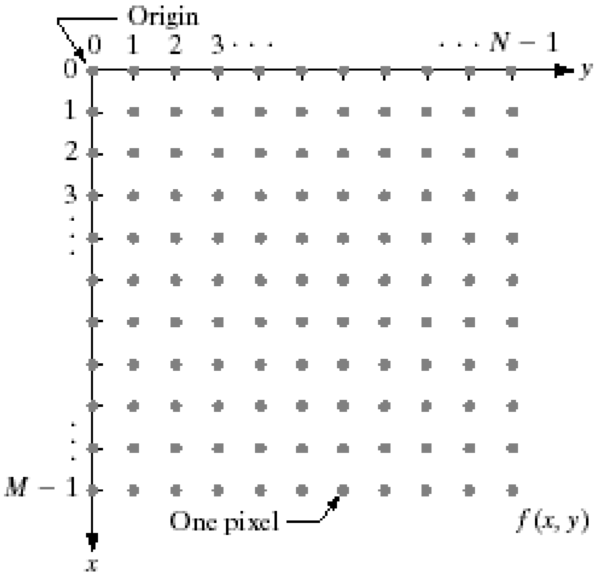
# Quantizing



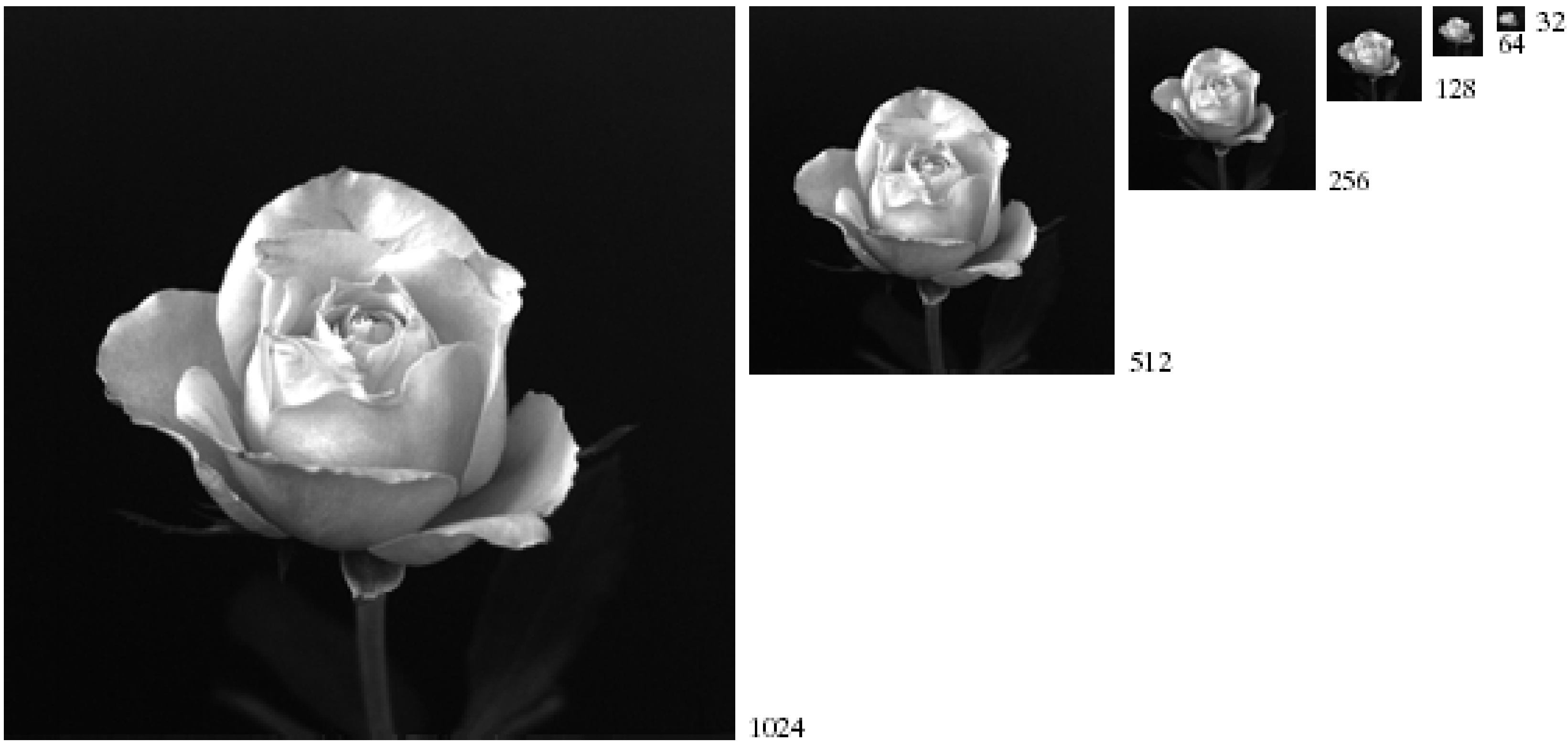
# Representing digital images

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}.$$

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix}.$$



# Spatial resolution

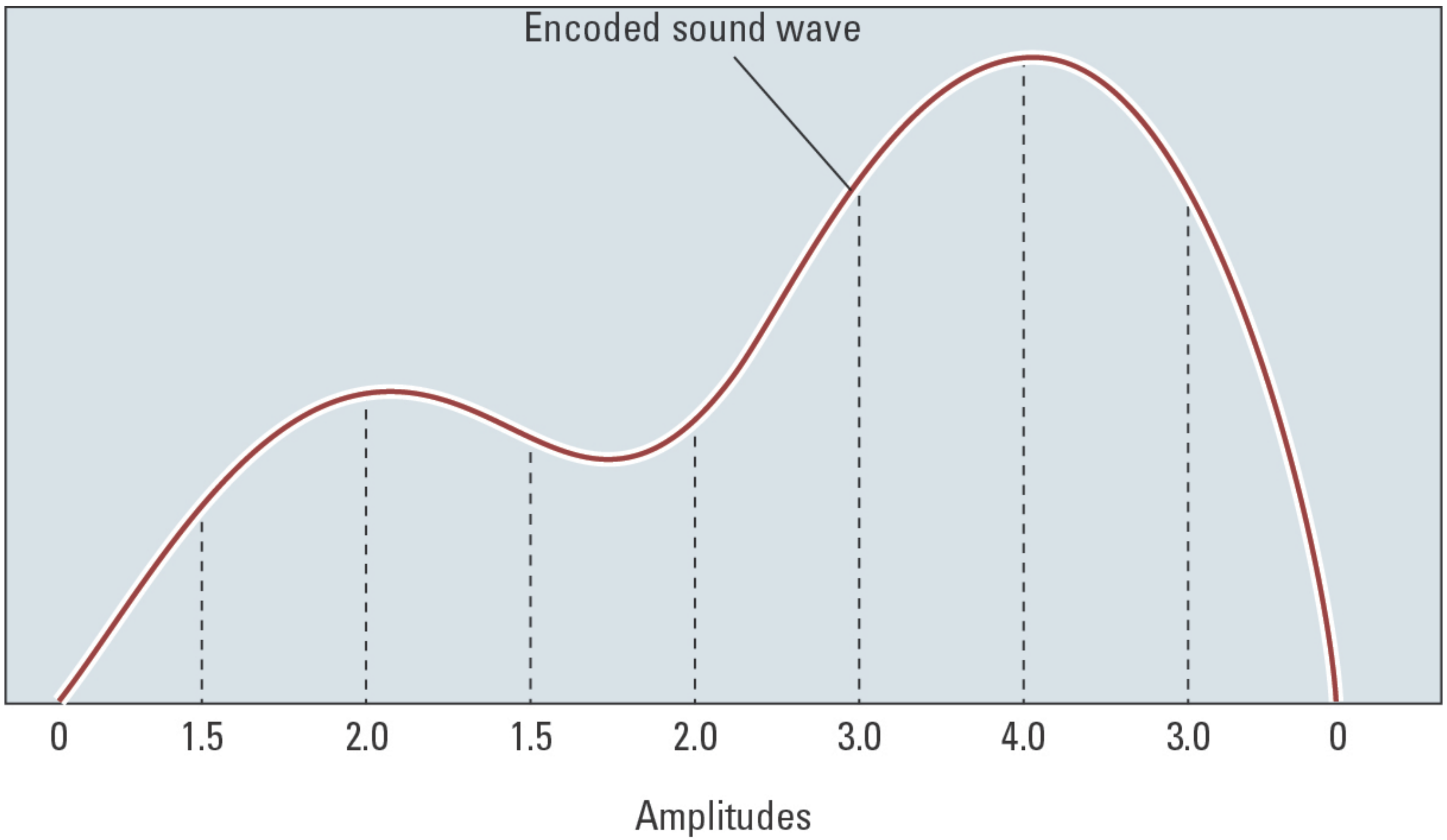


# Subsample and resample

# Representing Sound

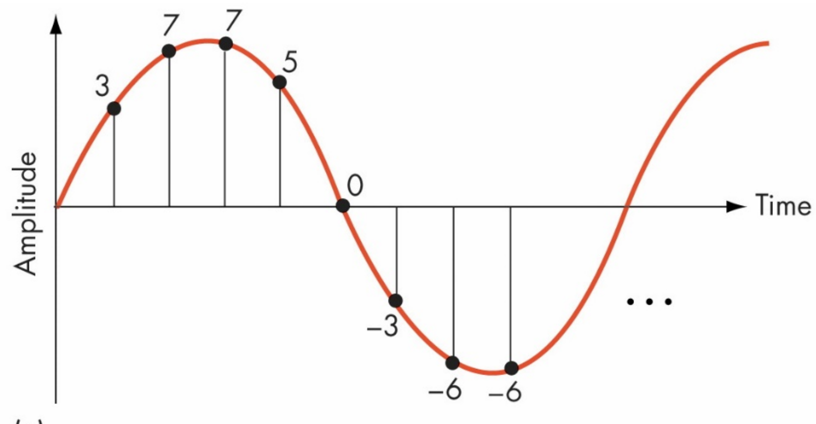
- Sampling techniques
  - Used for high quality recordings
  - Records actual audio
    - Long-distance telephone: 8000 samples/sec
    - CD sound: 44,100 samples/sec
  - Nyquist–Shannon sampling theorem
- MIDI
  - Used in music synthesizers
  - Records “musical score”: Encodes which instrument, note, and duration

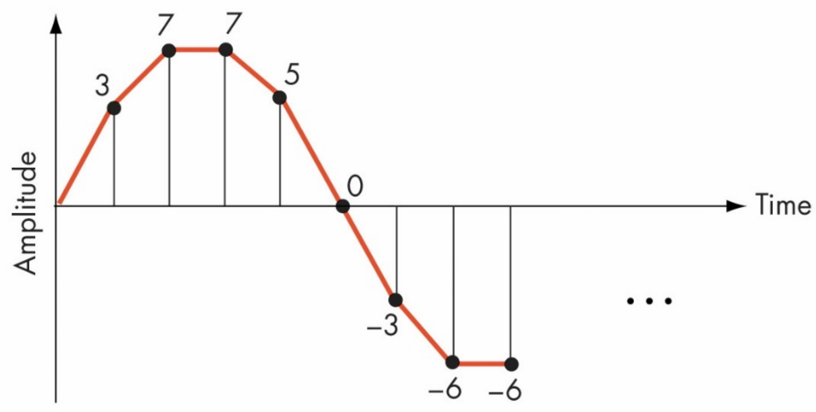
# The Sound Wave Represented by the Sequence 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0



# Representing Sound

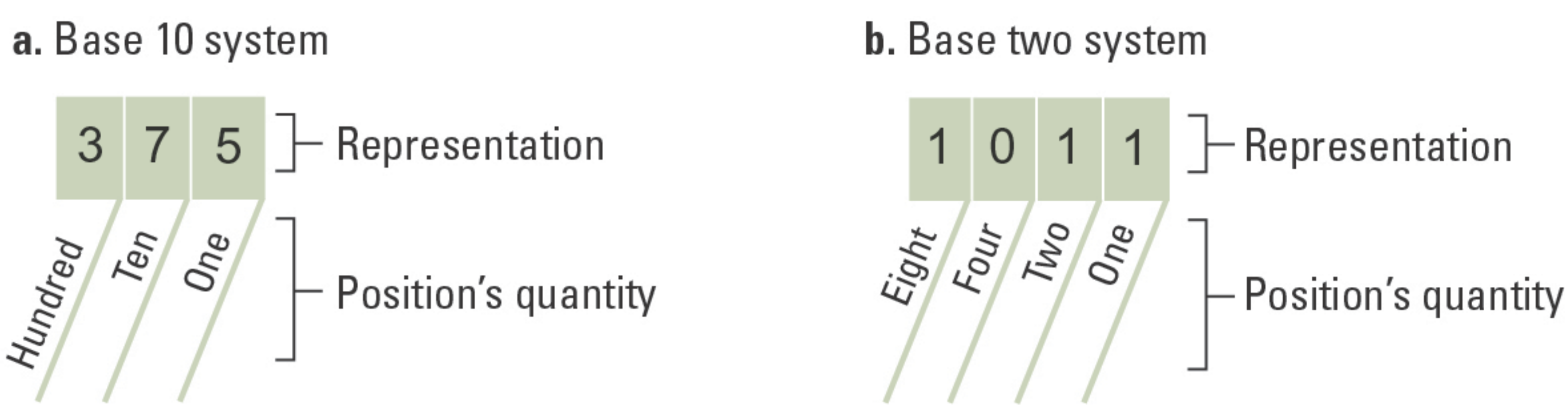
- Sampling a analog waveform
  - [3,7,7,5,0,-3,-6,-6…]
- Reconstructed from digital data.



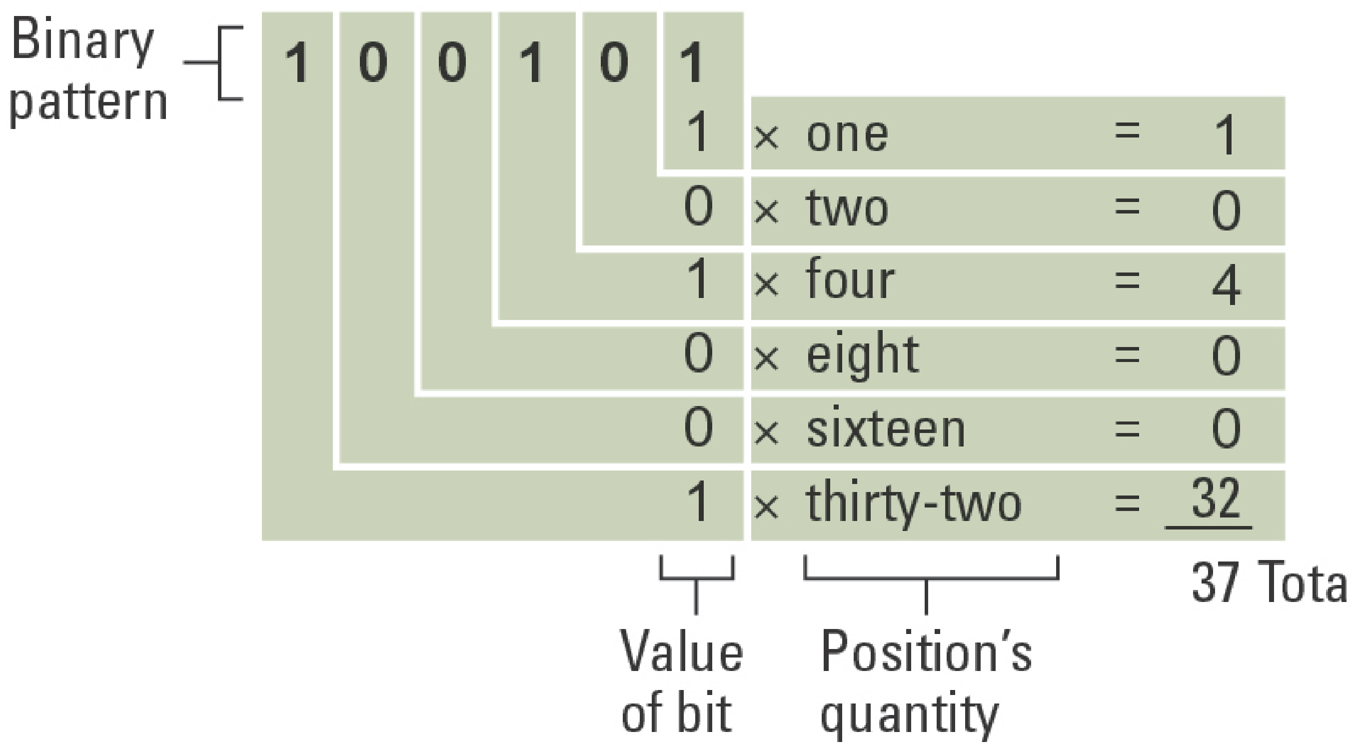



# The Binary System

- The traditional decimal system is based on powers of ten.
- The binary system is based on powers of two.
- Given $k$ bits, the largest unsigned integer is $2^k - 1$.



# Decoding the Binary Representation 100101



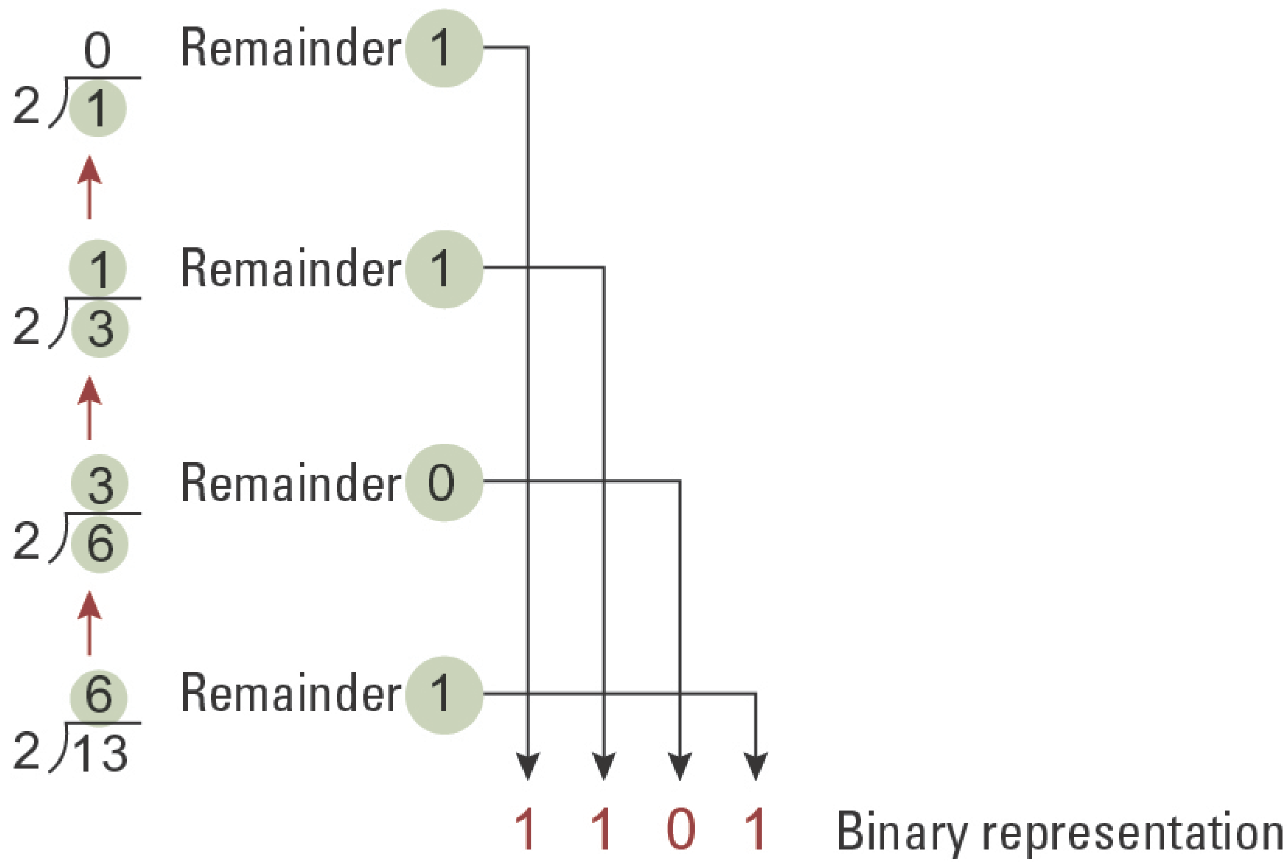
# An Algorithm for Finding the Binary Representation of a Positive Integer

**Step 1.** Divide the value by two and record the remainder.

**Step 2.** As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.

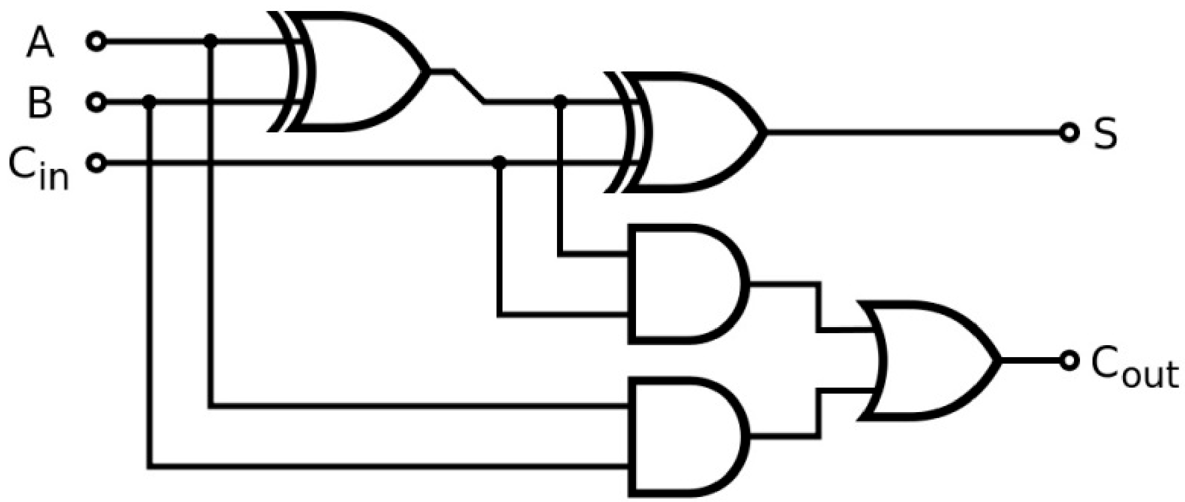
**Step 3.** Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

# Obtaining the Binary Representation of 13

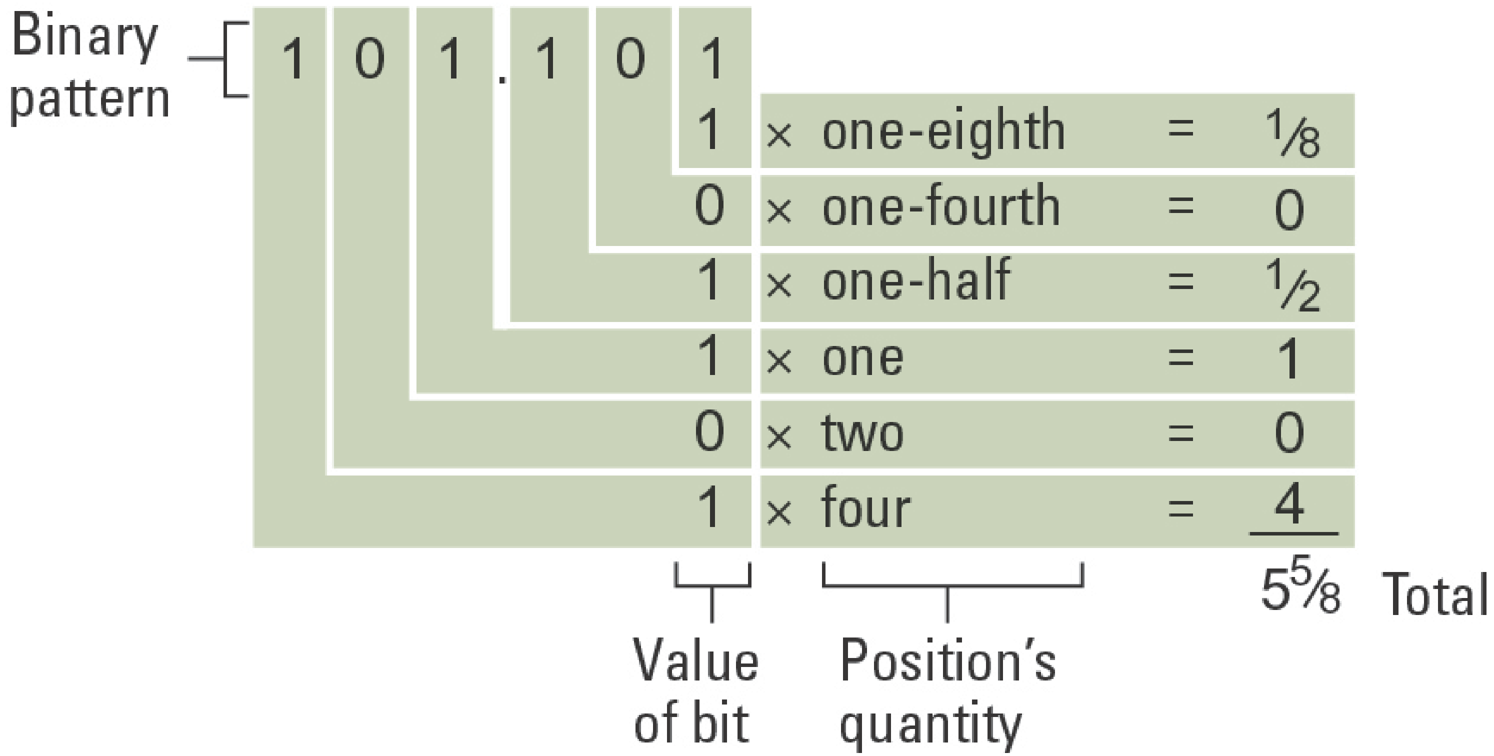


# The Binary Addition Facts

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$



# Decoding the Binary Representation 101.101



# Storing Integers

- **Two's complement notation:** The most popular means of representing integer values.
- **Excess notation:** Another means of representing integer values.
- Both can suffer from overflow errors.
  - What is underflow?

# 2’s Complement Notation Systems

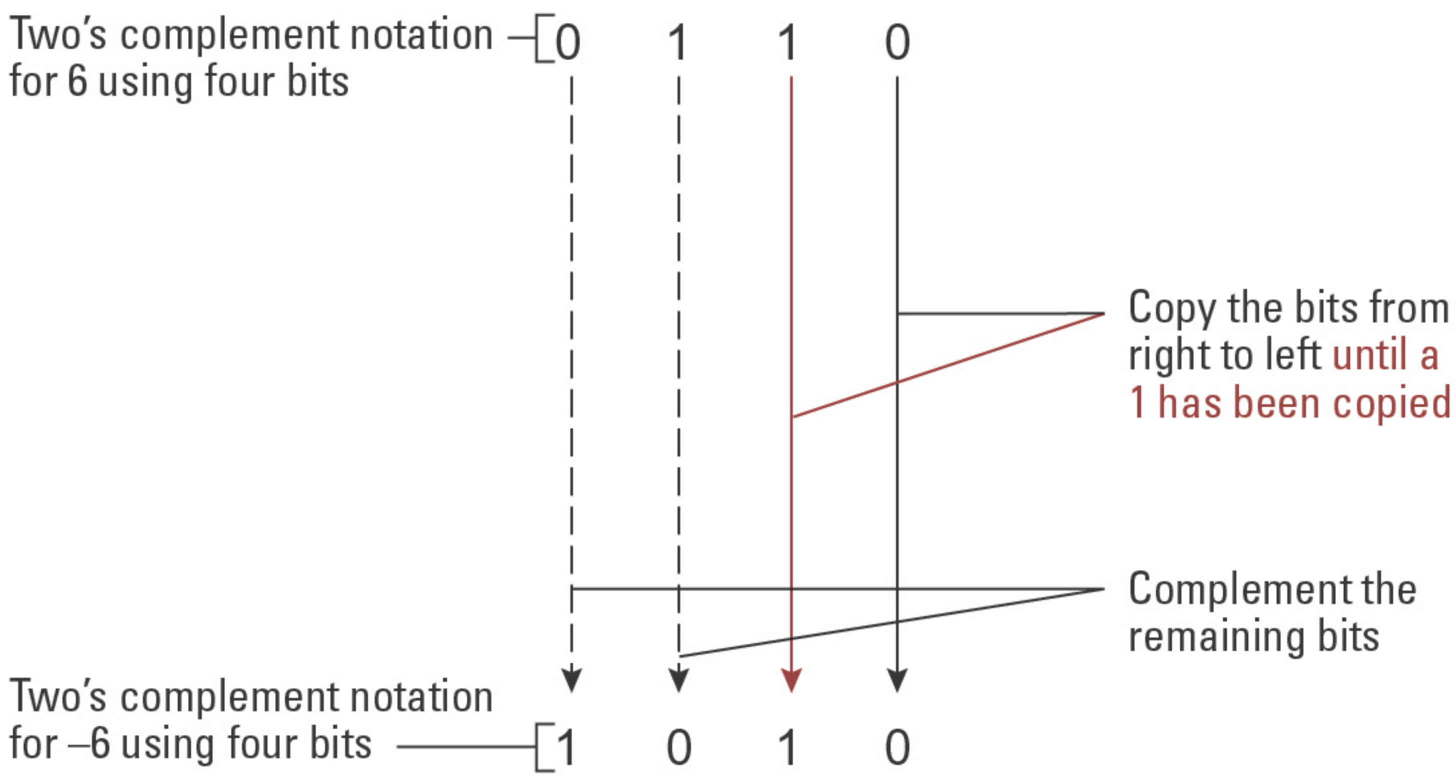
**a. Using patterns of length three**

| Bit pattern | Value represented |
|---|---|
| 011 | 3 |
| 010 | 2 |
| 001 | 1 |
| 000 | 0 |
| 111 | –1 |
| 110 | –2 |
| 101 | –3 |
| 100 | –4 |

**b. Using patterns of length four**

| Bit pattern | Value represented |
|---|---|
| 0111 | 7 |
| 0110 | 6 |
| 0101 | 5 |
| 0100 | 4 |
| 0011 | 3 |
| 0010 | 2 |
| 0001 | 1 |
| 0000 | 0 |
| 1111 | –1 |
| 1110 | –2 |
| 1101 | –3 |
| 1100 | –4 |
| 1011 | –5 |
| 1010 | –6 |
| 1001 | –7 |
| 1000 | –8 |

# Coding the Value -6 in 2's Complement Notation Using 4 Bits



# Addition Problems Converted to 2's Complement Notation

| Problem in base 10 | Problem in two's complement | Answer in base 10 |
|---|---|---|
| 3 + 2 | 0011 + 0010 = 0101 | 5 |
| –3 + –2 | 1101 + 1110 = 1011 | –5 |
| 7 + –5 | 0111 + 1011 = 0010 | 2 |

# The Problem of Overflow

- There is a limit to the size of the values that can be represented in any system
- Overflow
  - occurs when a computation produces a value that falls outside the range of values that can be represented in the machine
  - If the resulting sign bit is incorrect, an overflow has occurred
  - 16 bit systems have been upgraded to 32 bit systems
- What is underflow?

# An Excess Eight Conversion Table

| Bit pattern | Value represented |
|---|---|
| 1111 | 7 |
| 1110 | 6 |
| 1101 | 5 |
| 1100 | 4 |
| 1011 | 3 |
| 1010 | 2 |
| 1001 | 1 |
| 1000 | 0 |
| 0111 | −1 |
| 0110 | −2 |
| 0101 | −3 |
| 0100 | −4 |
| 0011 | −5 |
| 0010 | −6 |
| 0001 | −7 |
| 0000 | −8 |

- Why use excess representation?
  - Machine comparison.
  - IEEE 754 floating point.

# An Excess Notation System Using Bit Patterns of Length Three

| Bit pattern | Value represented |
|---|---|
| 111 | 3 |
| 110 | 2 |
| 101 | 1 |
| 100 | 0 |
| 011 | –1 |
| 010 | –2 |
| 001 | –3 |
| 000 | –4 |

# Binary to Decimal Conversion

$$23.47 = 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$$

$$\begin{aligned} 10.01_2 &= 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 1 \times 2 + 0 \times 1 + 0 \times \frac{1}{2} + 1 \times \frac{1}{4} \\ &= 2 + 0.25 = 2.25 \end{aligned}$$

# Caution

- Finite decimal digits ≠ finite binary digits
- Example:

  $0.1_{10} \rightarrow 0.2 \rightarrow 0.4 \rightarrow 0.8 \rightarrow 1.6 \rightarrow 1.2 \rightarrow 0.4 \rightarrow 0.8 \rightarrow 1.6 \rightarrow 1.2 \rightarrow$
  $0.4 \ldots$

  $0.1_{10} = 0.00011001100110011\ldots_2$
  $= 0.00011_2$ (infinite repeating binary)

  The more bits, the binary rep gets closer to $0.1_{10}$

# Scientific Notation

- Decimal:

  $-123{,}000{,}000{,}000{,}000 \rightarrow -1.23 \times 10^{14}$

  $0.000\ 000\ 000\ 000\ 000\ 123 \rightarrow +1.23 \times 10^{-16}$

- Binary:

  $110\ 1100\ 0000\ 0000 \rightarrow 1.1011 \times 2^{14}$

  $-0.0000\ 0000\ 0000\ 0001\ 1011 \rightarrow -1.1101 \times 2^{-16}$

# Decimal to Binary Conversion

- Write number as sum of powers of 2

$$0.8125 = 0.5 + 0.25 + 0.0625$$
$$= 2^{-1} + 2^{-2} + 2^{-4}$$
$$= 0.1101_2$$

- Algorithm: Repeatedly multiply fraction by two until fraction becomes zero.

$0.8125 \rightarrow 1.625$

$0.625 \rightarrow 1.25$

$0.25 \rightarrow 0.5$

$0.5 \rightarrow 1.0$

# Floating Point Representation

- Three pieces:
  - sign
  - exponent
  - mantissa (significand)

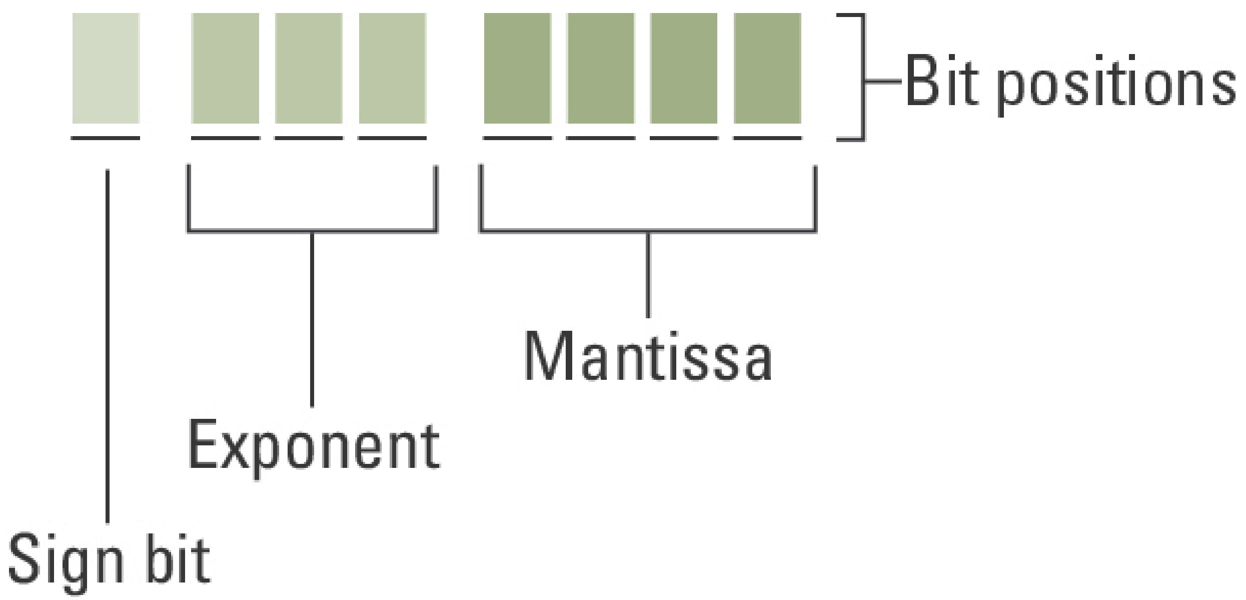
- Format:

| sign | exponent | mantissa |
|---|---|---|

  - Fixed-size representation (32-bit, 64-bit)
  - 1 sign bit
  - more exponent bits → greater range
  - more significand bits → greater accuracy

# Storing Fractions (Textbook)

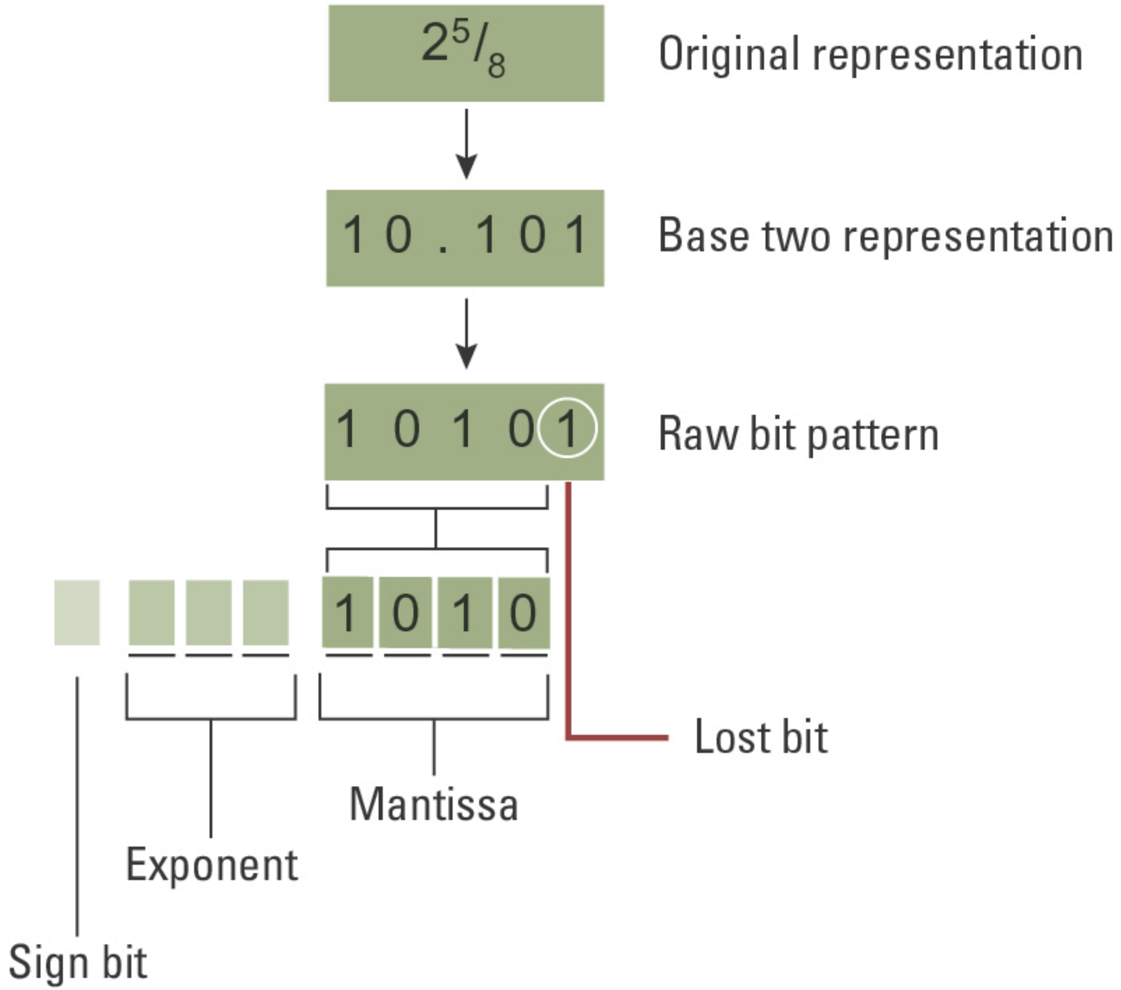
- **Floating-point Notation:** Consists of a sign bit, a mantissa field, and an exponent field.
- Related topics include
  - Normalized form.
  - Truncation errors.



# Decoding floating points

- Suppose we have the pattern 01101011.
  - Sign bit 0: Positive
  - Exponent: 110
  - Mantissa: 1011
- The exponent is represented in Excess-3 format
  - $110_2 = 2_{10}$
  - This means moving the radix in our solution to the right by 2 bits.
- Mantissa is 1011, moved by 2 bits is 10.11
  - The solution is $10.11_2 = 2\frac{3}{4}$
- What does 00111100 represent?

# Encoding the Value $2\frac{5}{8}$ (Truncation error)



# IEEE 754 floating point standards

› Single precision (32-bit) format

| 1 | 8 | 23 |
|---|---|---|
| S | E | F |

› Normalized rule: number represented is

$$(-1)^S \times 1.F \times 2^{E-127}, \quad E\ (\neq\ 00 \ldots 0\ or\ 11 \ldots 1)$$

› Example: $+101101.101 \rightarrow\ +1.01101101 \times 2^5$

| | | |
|---|---|---|
| 0 | 1000 0100 | 0110 1101 0000 0000 0000 000 |

# Features of IEEE 754 Format

- Sign: 1 → negative, 0 → non-negative
- Significand:
  - Normalized number: always a 1 left of binary point (except when E is 0 or 255)
  - Do not waste a bit on this 1 → "hidden 1"
- Exponent:
  - Not two's-complement representation
  - Unsigned interpretation minus bias

# Example: 0.75

- $0.75_{10} = 0.11_2 = 1.1 \times 2^{-1}$
  - $1.1 = 1.F \rightarrow F = 1$ (padded with 0s)
- $E - 127 = -1 \rightarrow E = 127 - 1 = 126 = 01111110_2$
- $S = 0$

0 01111110 10000000000000000000000

# Example: 0.75

- $0.75_{10} = 0.11_2 = 1.1 \times 2^{-1}$
  - $1.1 = 1.F \rightarrow F = 1$ (padded with 0s)
- $E - 127 = -1 \rightarrow E = 127 - 1 = 126 = 01111110_2$
- $S = 0$
- 00111111010000000000000000000000 = 0x3F400000

# Special-case Numbers

- Problem:
  - hidden 1 prevents representation of 0
- Solution:
  - make exceptions to the rule

- Bit patterns reserved for unusual numbers:
  - E = 00…0
  - E = 11…1

# Special-case Numbers

- Zeroes:

| | | | |
|---|---|---|---|
| 0 | 00…0 | 00…0 | $\longrightarrow$ +0 |
| 1 | 00…0 | 00…0 | $\longrightarrow$ -0 |

- Infinities:

| | | | |
|---|---|---|---|
| 0 | 11…1 | 00…0 | $\longrightarrow +\infty$ |
| 1 | 11…1 | 00…0 | $\longrightarrow -\infty$ |

# Data and Programing

- A ***programming language*** is a computer system created to allow humans to precisely express algorithms using a higher level of abstraction.

# Getting Started with Python

- ***Python:*** a popular programming language for applications, scientific computation, and as an introductory language for students.
- Freely available from [www.python.org](www.python.org)
- Also available at the computing resources cluster.
- Python is an *interpreted language*
  - Typing:

```
print('Hello, World!')
```

  - Results in:

```
Hello, World!
```

# Variables

- **Variables:** name values for later use
- Analogous to mathematic variables in algebra

```
s = 'Hello, World!'
print(s)

my_integer = 5
my_floating_point = 26.2
my_Boolean = True
my_string = 'characters'
my_integer = 0xFF
```

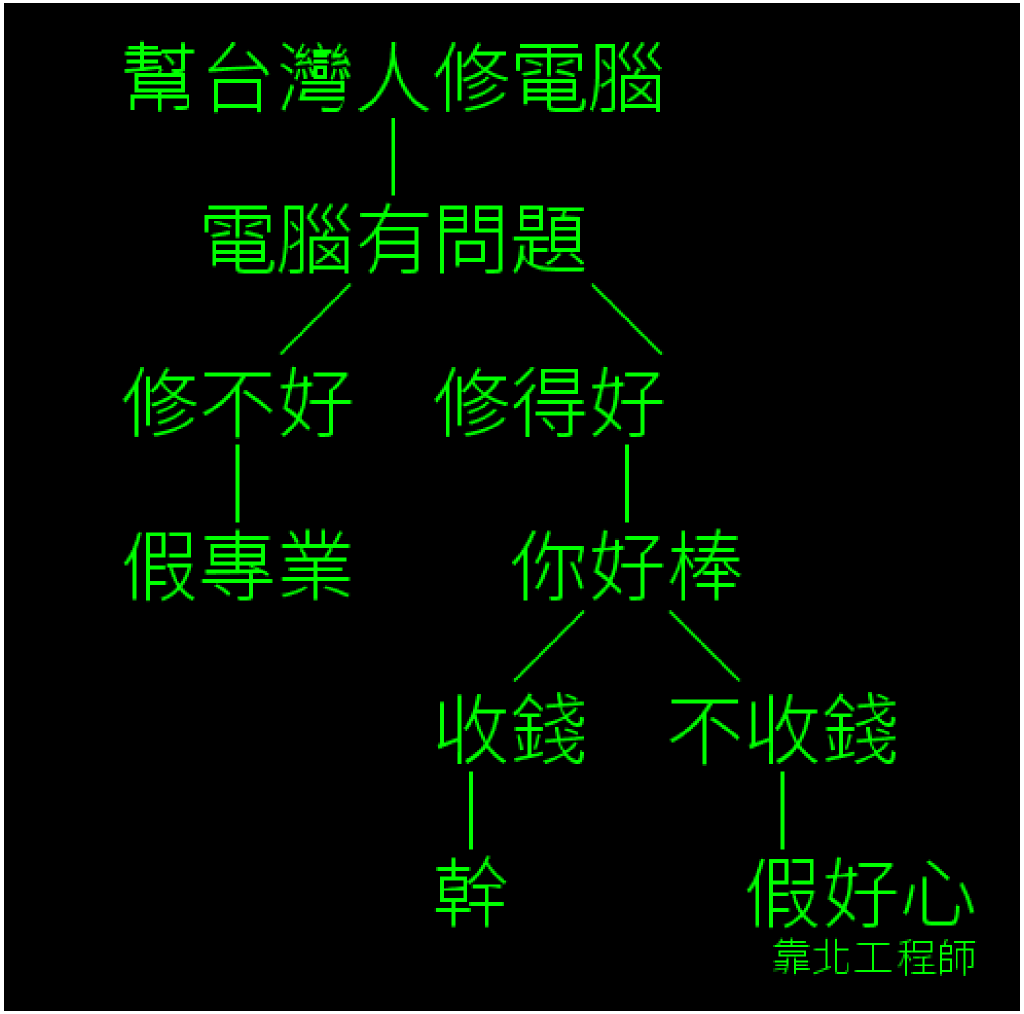
# Operators and Expressions

```
print(3 + 4)      # Prints 7
print(5 – 6)      # Prints -1
print(7 * 8)      # Prints 56
print(45 / 4)     # Prints 11.25
print(2 ** 10)    # Prints 1024

s = 'hello' + 'world'
s = s * 4
print(s)
```

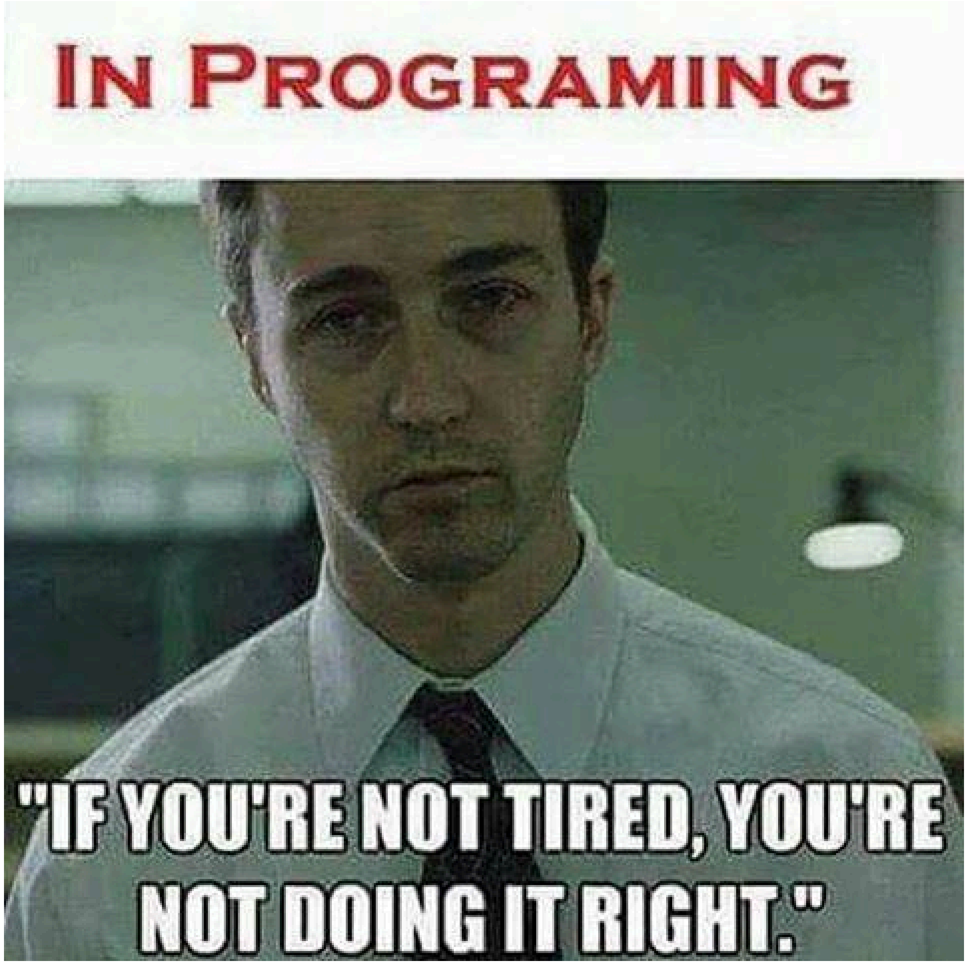
# Currency Conversion

```
# A converter for currency exchange.

USD_to_GBP = 0.66   # Today's exchange rate
GBP_sign = '\u00A3' # Unicode value for £
dollars = 1000      # Number dollars to
convert

# Conversion calculations
pounds = dollars * USD_to_GBP

# Printing the results
print('Today, $' + str(dollars))
print('converts to ' + GBP_sign + str(pounds))
```

# PC repair man



# Tired



# Debugging

- *Syntax errors*

```
print(5 +)
SyntaxError: invalid syntax

pront(5)
NameError: name 'pront' is not defined
```

- *Semantic errors*
  - Incorrect expressions like:

```
total_pay = 40 + extra_hours * pay_rate
```

- *Runtime errors*
  - Unintentional divide by zero.

# Data Compression

- Lossy versus lossless
- Run-length encoding
- Frequency-dependent encoding

  (Huffman codes)
- Relative encoding
- Dictionary encoding (Includes adaptive dictionary encoding such as LZW encoding.)
  - Lempel-Ziv-Welsh.

# LZW Encoding

- Encode message:

`xyx xyx xyx xyx`

- Steps:
  - `xyx xyx xyx xyx`
  - `xyx xyx xyx xyx`
  - `xyx xyx xyx xyx`
  - `xyx_xyx xyx xyx`
  - `xyx_xyx xyx xyx`
  - `xyx_xyx_xyx xyx`
  - …
  - `xyx_xyx_xyx_xyx`
- Final value: 121343434
  - 15 bytes → 9 bytes

| Key | Word |
|---|---|
| 1 | x |
| 2 | y |
| 3 | [space] |
| 4 | xyx |

# Compressing Images

› GIF: Good for cartoons

› JPEG: Good for photographs

› TIFF: Good for image archiving

# Compressing Audio and Video

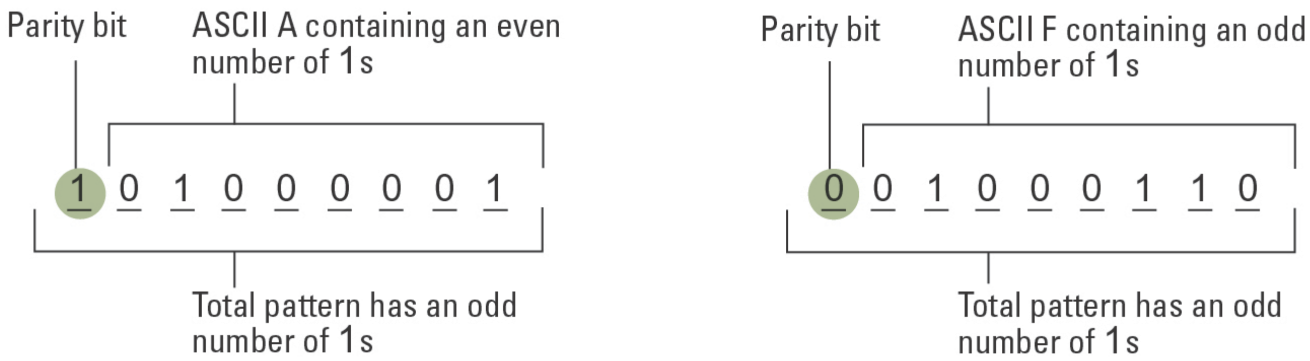
- MPEG
  - High definition television broadcast
  - Video conferencing
- MP3: Mpeg Layer 3
  - Temporal masking
  - Frequency masking

# Communication Errors

- Goal:  To reduce errors and increase the reliability of computing equipment
- Parity bits (even versus odd)
- Checkbytes
- Error correcting codes
  - Hamming distance

# Parity bits

› The ASCII codes for the letters A and F adjusted for odd parity:



# An Error-correcting Code

| Symbol | Code |
|---|---|
| A | 000000 |
| B | 001111 |
| C | 010011 |
| D | 011100 |
| E | 100110 |
| F | 101001 |
| G | 110101 |
| H | 111010 |

# Decoding the pattern 010100

| Character | Code | Pattern received | Distance between received pattern and code | |
|---|---|---|---|---|
| A | 0 0 0 0 0 0 | 0 **1** 0 **1** 0 0 | 2 | |
| B | 0 0 1 1 1 1 | 0 **1** **0** 1 **0** **0** | 4 | |
| C | 0 1 0 0 1 1 | 0 1 0 **1** **0** **0** | 3 | |
| D | 0 1 1 1 0 0 | 0 1 **0** 1 0 0 | **1** | Smallest distance |
| E | 1 0 0 1 1 0 | **0** **1** 0 1 **0** 0 | 3 | |
| F | 1 0 1 0 0 1 | **0** **1** **0** **1** 0 **0** | 5 | |
| G | 1 1 0 1 0 1 | **0** 1 0 1 0 **0** | 2 | |
| H | 1 1 1 0 1 0 | **0** 1 **0** **1** **0** 0 | 4 | |

# Supplementary: Parity and Raid 5

- RAID (redundant array of independent disks):
  - Data storage virtualization technology that combines multiple physical disk drive components into a single logical unit for the purposes of data redundancy, performance improvement, or both.
  - JBOD (*just a bunch of disks*) concatenate disks or RAID sets.
  - RAID 0 consists of striping, without mirroring or parity
  - RAID 1 consists of data mirroring, without parity or striping.
  - RAID 5 consists of block-level striping with distributed parity.
  - RAID 6 consists of block-level striping with double distributed parity.

# How does RAID 5 work

- By the XOR (exclusive OR) operation.
- Suppose we have 3 drives, the data stored in the drive is
  - 101
  - 010
  - 011
- We use the 4th drive to store parity
  - XOR(101,010,011) = XOR(XOR(101,010),011)
  - 100
- Suppose the 2nd drive is broken, then the new drive to reconstruct can be calculated by
  - XOR(101,011,100) = 010