香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

# Introduction to Computer Science: Programming Methodology

# Lecture 1 Introduction

**Prof. Yunming XIAO**

**School of Data Science**

# Outline

- Tuesday
  - Introduction & logistics
  - Overview of modern computers
  - Questionnaire (1)
- Thursday
  - Preliminary knowledge for computer programming

- Boilers for next week
  - Python Basics

# Outline

- Tuesday
  - **Introduction & logistics**
  - Overview of modern computers
  - Questionnaire (1)
- Thursday
  - Preliminary knowledge for computer programming

- Boilers for next week
  - Python Basics

# About me

- Background
  - Education: B.Eng. BUPT (2019) & Ph.D. Northwestern U (2024)
  - Work Experience
    - Intern: ByteDance (2019), Bell Labs (2021), HPE Labs (2022), Google (2023)
    - Research Fellow (2024-2025), University of Michigan—Ann Arbor
    - Assistant Professor (2025.7-*Present*), SDS, CUHK-Shenzhen
  - Research: computer systems, networks, and security

- Contact
  - Email: yunmingxiao@cuhk.edu.cn
  - Office: Zhi Xin Building 403a
  - Office Hour: 5pm-6pm Tuesday (except holidays)

# Useful materials

- Personal website:
  - https://yunmingxiao.github.io
- Course website: https://yunmingxiao.github.io/csc1001-25fall/schedule.html
- Blackboard: https://bb.cuhk.edu.cn
- Online resources/books:
  - W3School Python Tutorial (https://www.w3schools.com/python/)
  - A Practical Introduction to Python Programming, by Brian Heinold
  - How to think like a Computer Scientist, by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers

# About this course

- This course is a required course for all SDS students, and we welcome students from other schools

- Need to synchronize between other 5 parallel sessions (schedules, assignments, exams)

# Learning objectives

- This course will introduce the key programming concepts using Python language as examples

- Students will learn basic elements of modern computer systems, key programming concepts, problem solving, and basic algorithm design

# A message for freshmen:

- University courses are very different from what you might have been familiar with in your high school
  - Languages
  - Assignments
  - Exams

- In your future university life, there are no more head teachers (班主任)
  - No one is watching you to finish the assignments
  - It is the right time to start being mature and independent
  - Make best use of tutorials (starting next week)
  - Check your emails often
  - Assignments and important course announcements will be sent out via emails

# Assessment

| Activity | Grade |
| --- | --- |
| Assignments × 4 | 10% × 4 |
| Mid-term quiz | 20% |
| Final exam | 40% |

# Course components

| Activity | Hours/week |
| --- | --- |
| **Lecture** | **90min × 2** |
| **Tutorial** | **60min × 1** |

# Indicative teaching plans

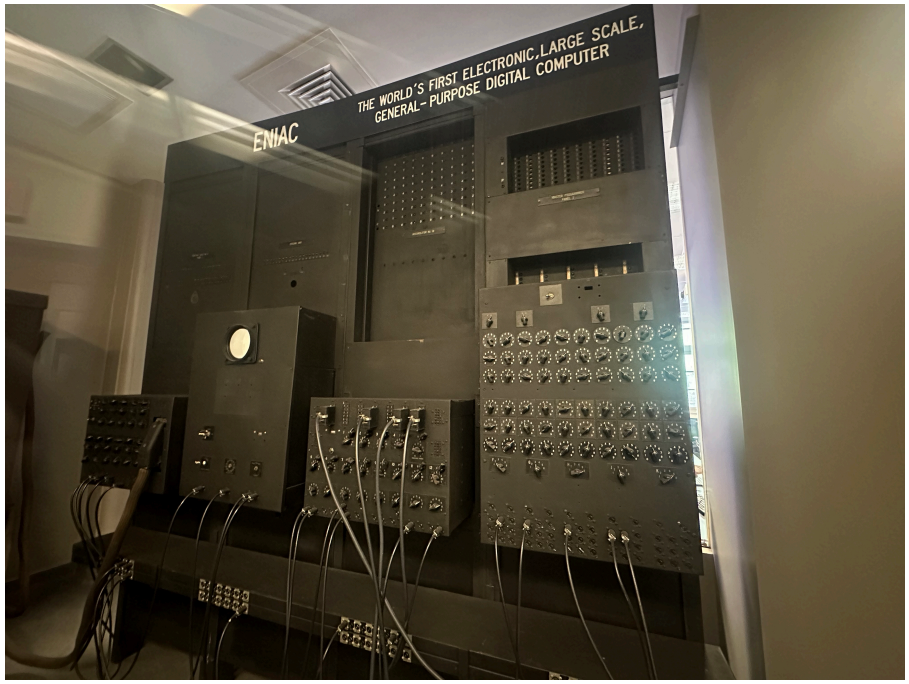
| Week | Content/ topic/ activity |
|---|---|
| 1 | Introduction to modern computers<br>Preliminary knowledge for computer programming |
| 2 | Basic introduction to Python language<br>Data types and operators in Python language<br>Input/output |
| 3 | Flow control and loop |
| 4 | Function |
| 5 & 6 | Basic data structures |
| 7 | Introduction to object-oriented programming, part I |
| 8 | Review for mid-term quiz |
| 9 | Introduction to object-oriented programming, part II |
| 10 | Data Structure, part I |
| 11 | Data Structure, part II |
| 12 | Introduction to algorithm design, part I |
| 13 | Introduction to algorithm design, part II |
| 14 | Review for final exam |

# Outline

- Tuesday
  - Introduction & logistics
  - **Overview of modern computers**
  - Questionnaire (1)
- Thursday
  - Preliminary knowledge for computer programming

- Boilers for next week
  - Python Basics

# Modern computers




ENIAC @ University of Pennsylvania

# Why learning programming?

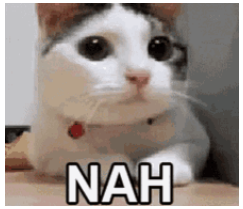
- Computers are built to help people solve problems


- ~~Computers do not understand what we say~~

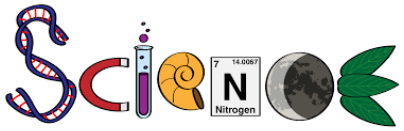
- ~~We need to communicate with computers using their languages (computer programming language)~~

ChatGPT

deepseek



# Why learning programming?

- Computers are built to help people solve problems 


- Computers might seem to understand what we say, but they still process everything using code under the hood

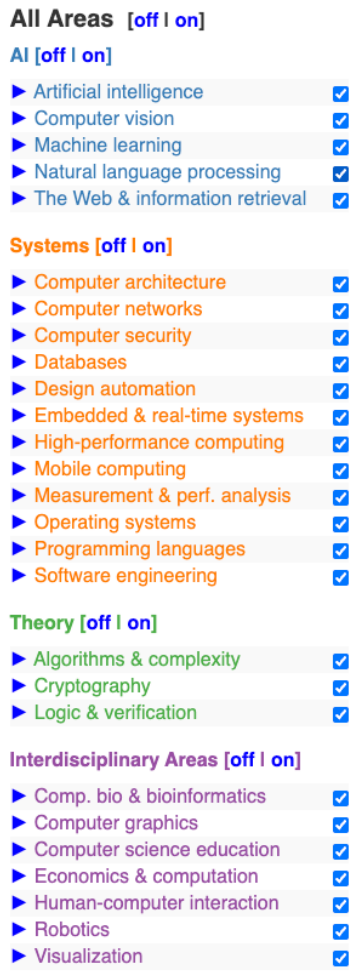
- Understanding programming is key to solving problems
- Programming is a way of thinking

Alan Turing has proved that a computer, in theory, can compute anything that is computable

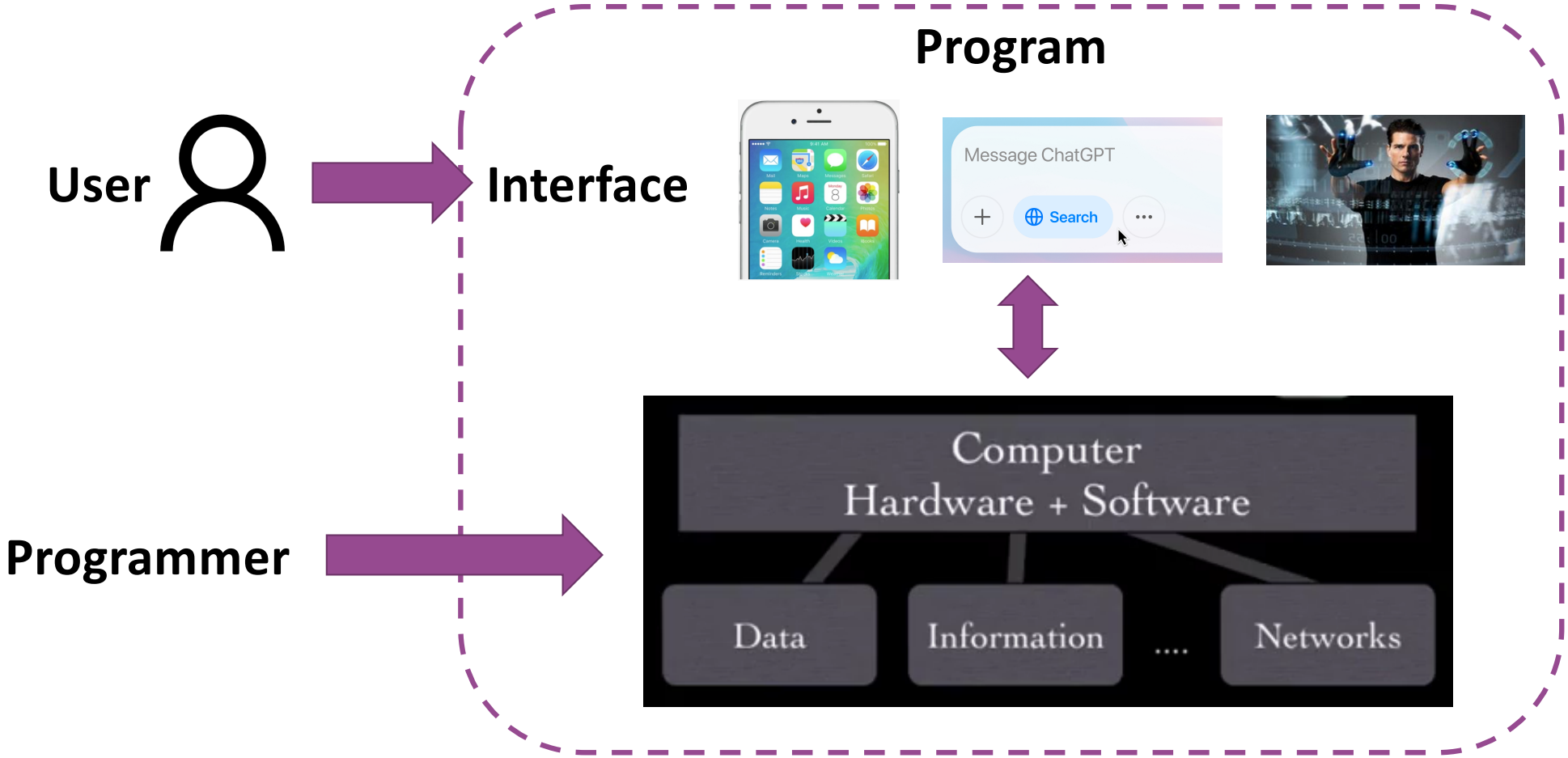
The success of AI imply that human intelligence is computable?

# Why computer science?

- AI, Data Science, Cybersecurity, HCI, etc., all stem from CS
- Without understanding algorithms, systems, and theory, people risk becoming tool users rather than innovators
- Programming and CS aren't just skills but a way of (computational) thinking
- CS evolves: many "hot" fields today may look very different in 10 years, but the core principles of CS remain
- With a CS foundation, you can branch into any area



# Digital world



# Programmer

- Professional programmer writes computer programs and develops software
- A junior programmer gets a salary of 30k+ RMB in an INTERNET company like Tencent
- A programmer can earn up to 500k – 1m USD in Google!!
- Software and Internet are huge industries

Google

ByteDance

Alibaba Group
阿里巴巴集团

Tencent 腾讯

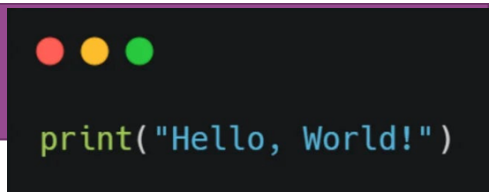
NVIDIA

……

# Why being a programmer?

- Even if you are NOT in the IT industry, programming is pervasive in your life,
  - Electrical/electronic engineer – control program
  - Economist – mathematical modeling
  - Salesman – analyzing sales data
  - …

# What is code, program, and software?

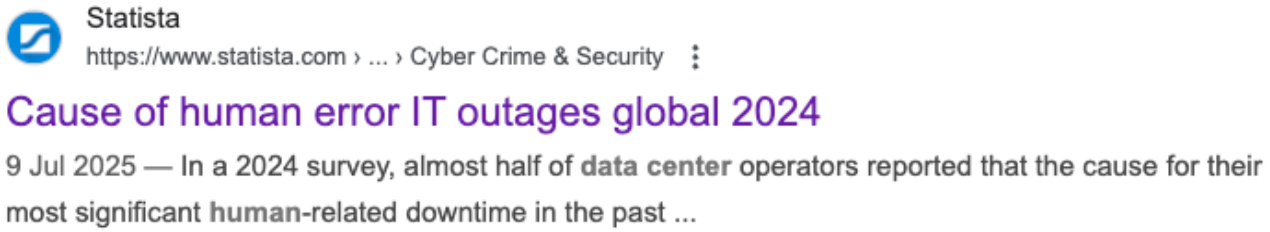



- Code (sentence): text written by humans in a programming language (e.g., Python, Java) to tell the computer what to do
- Program (recipe of one dish): a set of code that performs a specific task at runtime (code with purpose)
- Software (cookbook): a collection of programs and data

- Software is compiled into instructions and executed by computers
- It is a little piece of our intelligence in the computer
- Intelligence that is re-usable

# Computers are good at following instructions

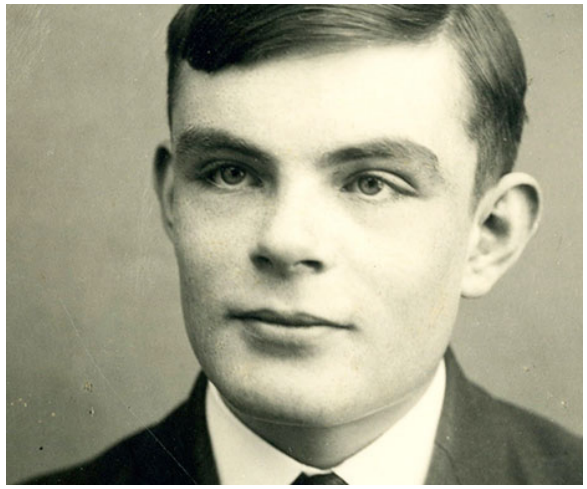
- Humans can easily make mistakes when following a set of instructions
- On the contrary, computers (almost) do not make mistakes, regardless of they are given 10 or 10 billion instructions!!



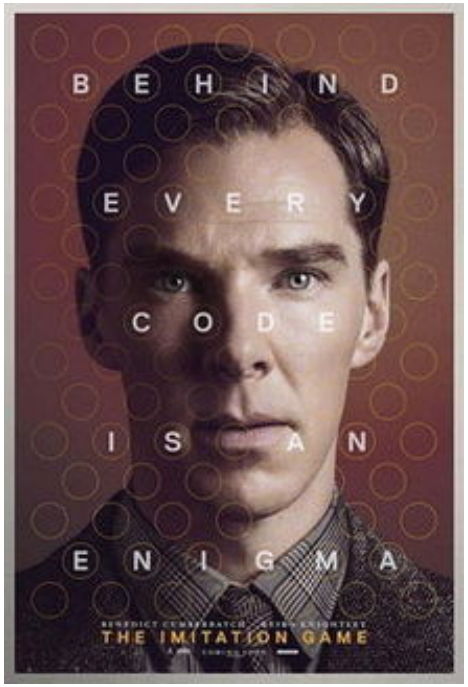

"Humans are the weakest link in any security system"
– *The Art of Deception,* by Kevin Mitnick

# Theoretical foundation of CS

- The theoretical foundation of computer science are built by Alan Turing (which course?)
- Father of theoretical computer science and artificial intelligence
- Computability theory and Turing test
- ACM Turing Award is the highest honour in computer science



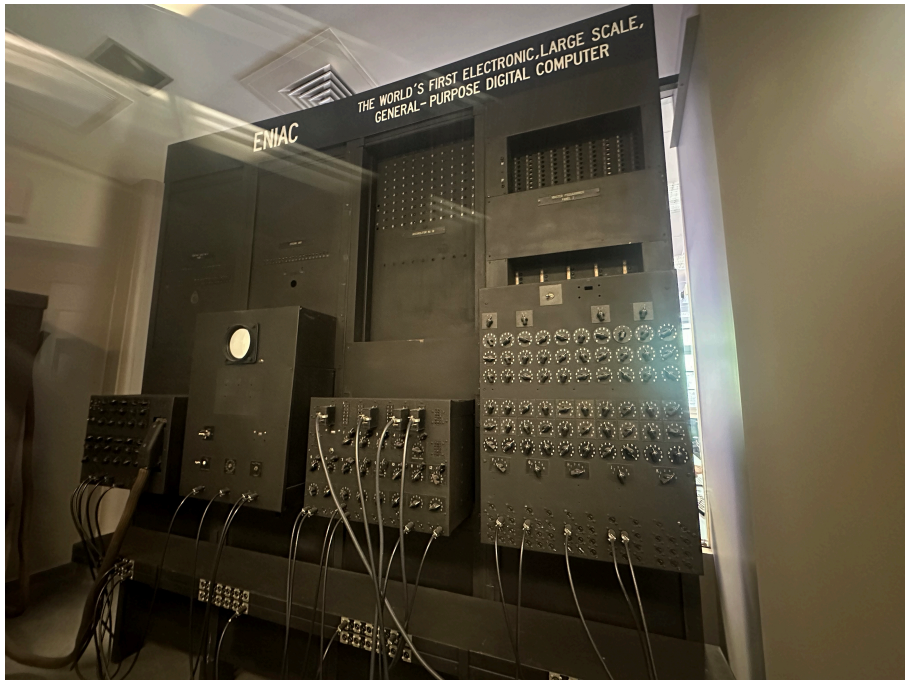
# A movie about Turing




模仿遊戲

*The Imitation Game*

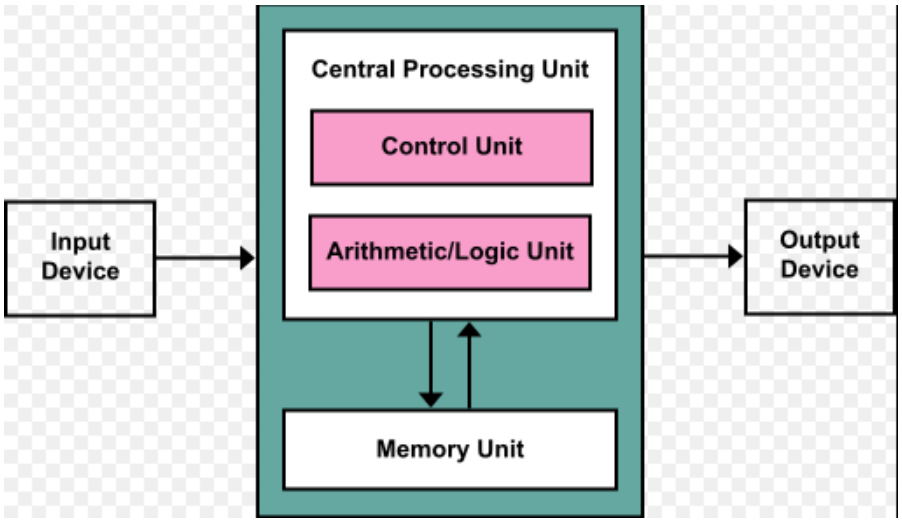
# What is the common in them?




ENIAC @ University of Pennsylvania


# Von Neumann architecture

- The modern computer architecture is proposed by John Von Neumann
- It is one possibility to build computer defined by Turing (aka Turing machine)




# A side note

## John von Neumann Theory Prize

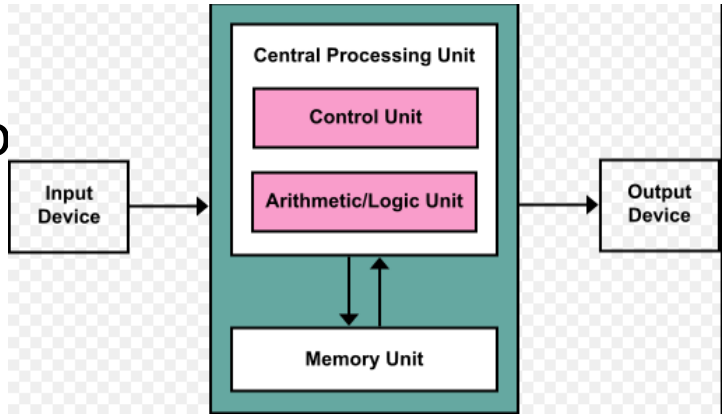
SHARE:


**2024 Winner(s)**

- **Jim Dai**, Cornell University

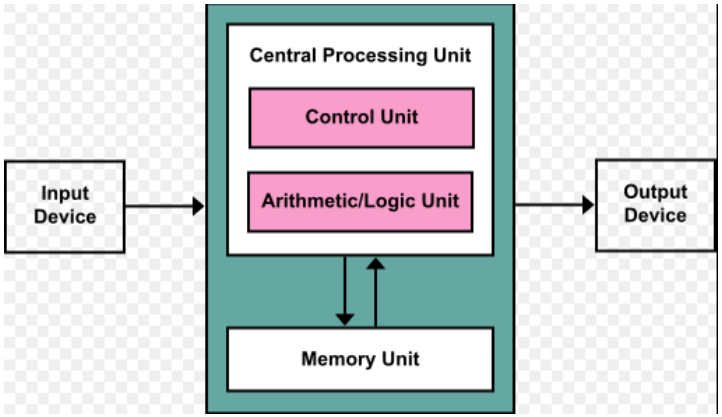
# Key hardware components in a computer

- Central processing unit (CPU): program execution
- Memory unit: store instructions and data
- Input device: take inputs from users o other devices
- Output device: output information to users or other devices

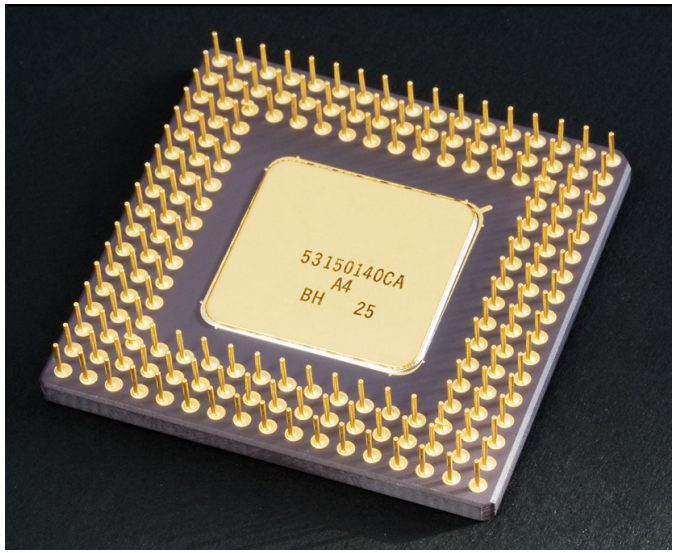


# Central Processing Unit (CPU)

- A processor contains two units, a control unit (CU) and an arithmetic/logic unit (ALU)
- CU is used to fetch commands from the memory
- ALU contains the electric circuits which can execute commands



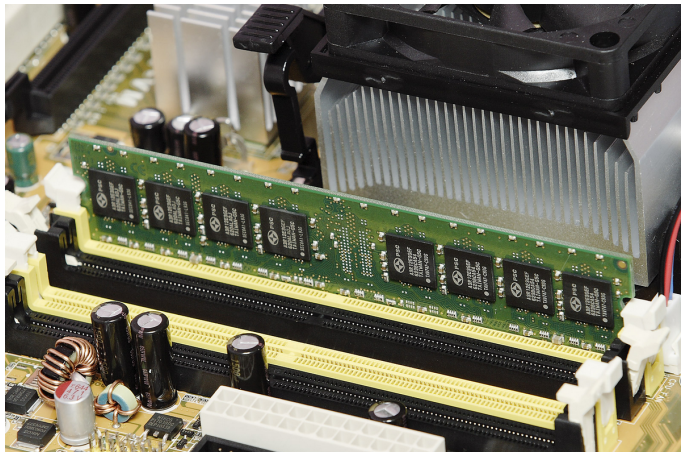
# Central Processing Unit (CPU)


- Processor manufacturer: Intel, AMD, ARM, etc

# Memory/Storage

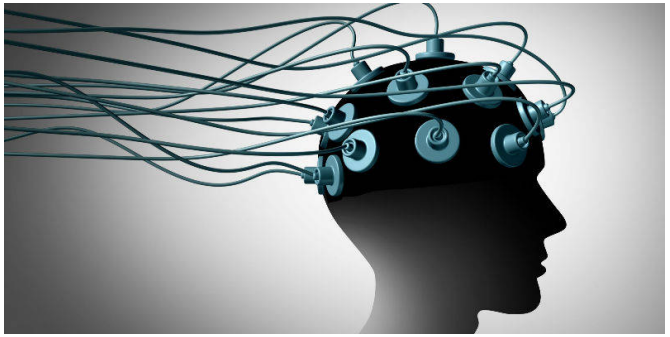
- High speed cache
- RAM
- ROM
- Flash
- Hard disk



# Input/output devices

- Input devices: mouse, keyboard, panel, touch screen, audio input, mind reading, etc
- Output devices: screen, audio output, etc
- Research field
  - human-machine interaction (HCI)

# Any other input devices?

# Any other output devices?

VR

Holographic projection

# How the hard disk works


http://v.youku.com/v_show/id_XNjA4NzMxNDk2.html?from=s1.8-1-1.2

# What can a computer actually understand?

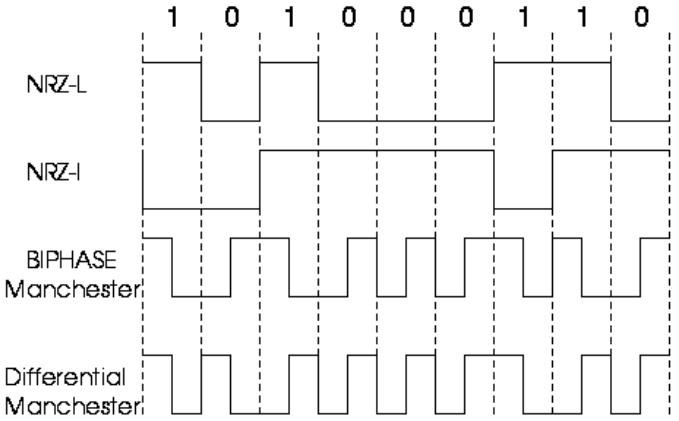
- The computers used nowadays can understand only binary number (i.e., 0 and 1)
- Computers use voltage levels to represent 0 and 1
- NRZL and NRZI coding
- The instructions expressed in binary code is called machine language

0 0 0 1 numerical value $2^0$

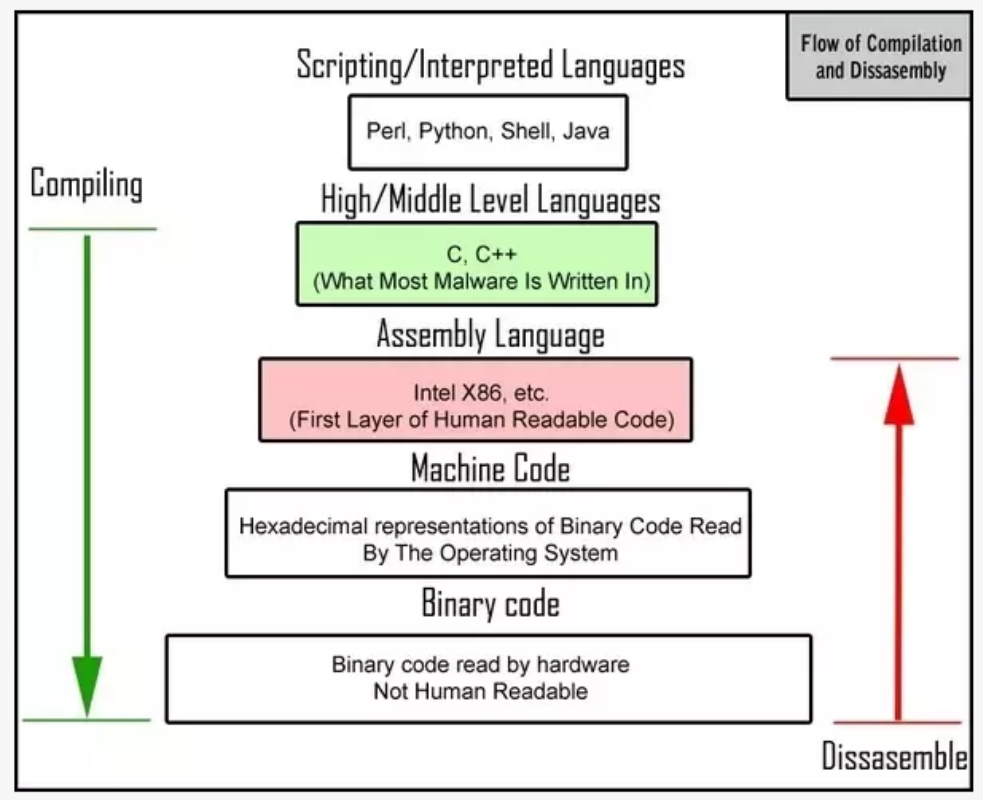
0 0 1 0 numerical value $2^1$

0 1 0 0 numerical value $2^2$

1 0 0 0 numerical value $2^3$



# Programing language




https://www.quora.com/I-am-an-11th-grader-I-find-it-quite-difficult-to-write-C++-code-especially-when-the-only-way-to-practice-is-to-solve-maths-problems-Should-I-keep-learning-C++-or-drop-it-for-C

# Low-level language – Assembly language

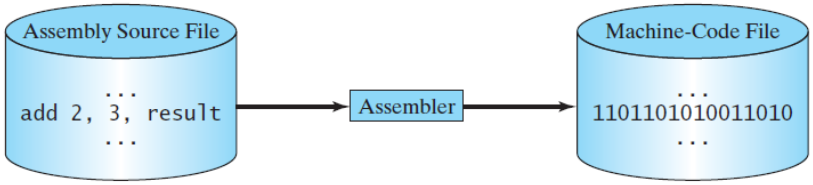
- An assembly language is a low-level programming language, in which there is a very strong (generally one-to-one) correspondence between the language and machine code instructions.

- Each assembly language is specific to a particular computer architecture

- Assembly language is converted into executable machine code by a utility program referred to as an assembler

```
                        ****************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input
C01E 8D F0      INHEX   BSR     INCH        GET A CHAR
C020 81 30              CMP A   #'0         ZERO
C022 2B 11              BMI     HEXERR      NOT HEX
C024 81 39              CMP A   #'9         NINE
C026 2F 0A              BLE     HEXRTS      GOOD HEX
C028 81 41              CMP A   #'A
C02A 2B 09              BMI     HEXERR      NOT HEX
C02C 81 46              CMP A   #'F
C02E 2E 05              BGT     HEXERR
C030 80 07              SUB A   #7          FIX A-F
C032 84 0F      HEXRTS  AND A   #$0F        CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF   HEXERR  JMP     CTRL        RETURN TO CONTROL LOOP
```



# C language (1969 - 1973)

- C was developed by Dennis Ritchie between 1969 and 1973 at Bell Labs
- One of the early high-level programming language
- Somewhere between assembly and other high-level languages
- Provide powerful functionalities for low level memory manipulations
- Have the highest efficiency within high level languages
- Very widely used in low level applications, such as operating systems, embedded programming, super computers, etc

# C++ language (1979)

- C++ was developed by Bjarne Stroustrup at Bell Labs since 1979
- Inherent major features of C
- An object oriented programming language, supporting code reuse
- High efficiency and powerful in low level memory manipulation
- Still platform dependent

# Java language (1995)

- Java was developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995
- A new generation of general-purpose object oriented programming language
- Platform independent, “write once, run anywhere” (WORA)
- Java is one of the most popular programming languages currently in use

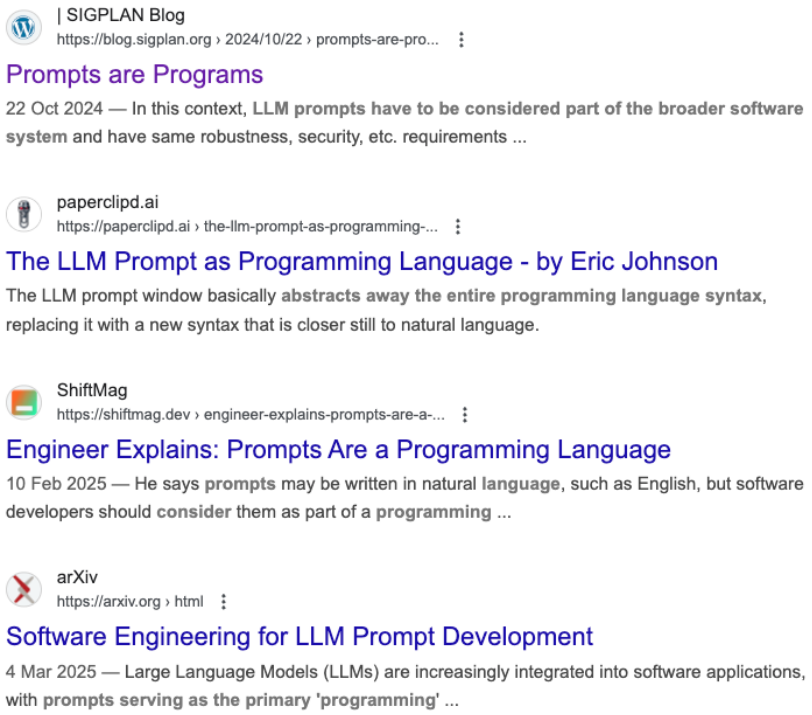
# Python language (1991)

- Developed by Guido van Rossum in 1989, and formally released in 1991
- An open source, object-oriented programming language
- Powerful libraries
- Powerful interfaces to integrate other programming languages (C/C++, Java, and many other languages)
- In AI research, people mainly use Python

# AI model prompt? (2022)

- Whether AI model prompt is programming language is debatable

- Primary issues:
  - Lack of formal grammar
  - Non-deterministic Execution
  - Low Abstraction and Formal Verification
  - ……



# Language efficiency v.s. development efficiency

- High-level languages cannot be executed directly
- High-level languages must be converted into low-level languages first
- Lower-level languages have higher language efficiency (they are faster to run on a computer)
- Higher-level languages have higher development efficiency (it is easier to write programs in these languages)

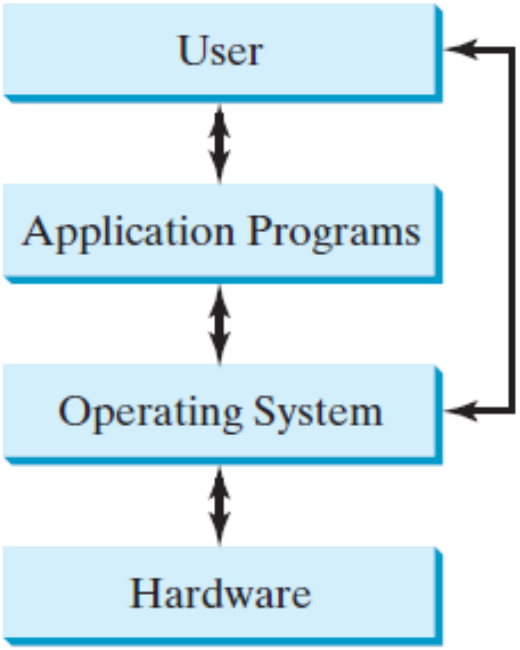
# Operating system

- The operating system (OS) is a low-level program, which provides all basic services for managing and controlling a computer's activities

- Applications are programs which are built based upon an OS

- Main functions of an OS:
  - ✓ Controlling and monitoring system activities
  - ✓ Allocating and assigning system resources
  - ✓ Scheduling operations

- Popular OS: Windows, Mac OS, Linux, iOS, Android…



# Outline

- Tuesday
  - Introduction & logistics
  - Overview of modern computers
  - **Questionnaire (1)**
- Thursday
  - Preliminary knowledge for computer programming

- Boilers for next week
  - Python Basics

# Please take a moment to complete the questionnaire

https://forms.gle/BAcFZv3tkQpycXyZ6 TODO

Break

# Outline

- Tuesday
  - Introduction & logistics
  - Overview of modern computers
  - **Questionnaire (1)**
- Thursday
  - Preliminary knowledge for computer programming

- Boilers for next week
  - Python Basics

# Slogan for Python


Life is short, use Python!

# Review of last lecture

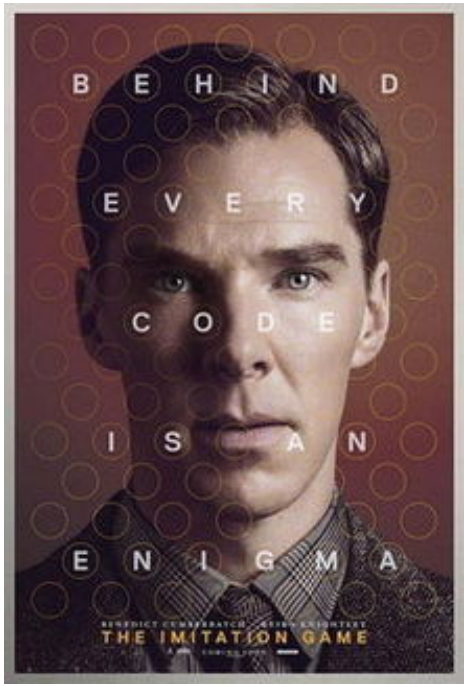
- Why programming and CS
- Von Neumann Architecture
- CPU and memory
- Input/output devices
- Programming language
- Operating system

# Outline

- Tuesday
  - Introduction & logistics
  - Overview of modern computers
  - Questionnaire (1)
- Thursday
  - **Preliminary knowledge for computer programming**

- Boilers for next week
  - Python Basics

# How do you like the movie?




模仿遊戲

*The Imitation Game*

# Data representation and conversion

- We use positional notation (进位记数法) to represent or encode numbers in a computer
- Data are stored essentially as binary numbers in a computer
- In practice, we usually represent data using either binary (二进制), decimal (十进制), octal (八进制) or hexadecimal (十六进制) number systems
- We may need to convert data between different number systems

# The basic idea of positional notation

- Each positional number system contains two elements, a base (基数) and a set of symbols
- Using the decimal system (十进制系统) as an example, its base is 10, and the symbols are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- When a number “hits” 9, the next number will not be a different symbol, but a “1” followed by a “0” (逢十进一)

# Decimal number system

- In the decimal number system, the base is 10, the symbols include 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Every number can be decomposed into the sum of a series of numbers, each is represented by a positional value times a weight

- $N = a_n \times 10^n + a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2} \ \ldots\ldots + a_0 \times 10^0 + a_{-1} \times 10^{-1} + a_{-2} \times 10^{-2} \ \ldots$

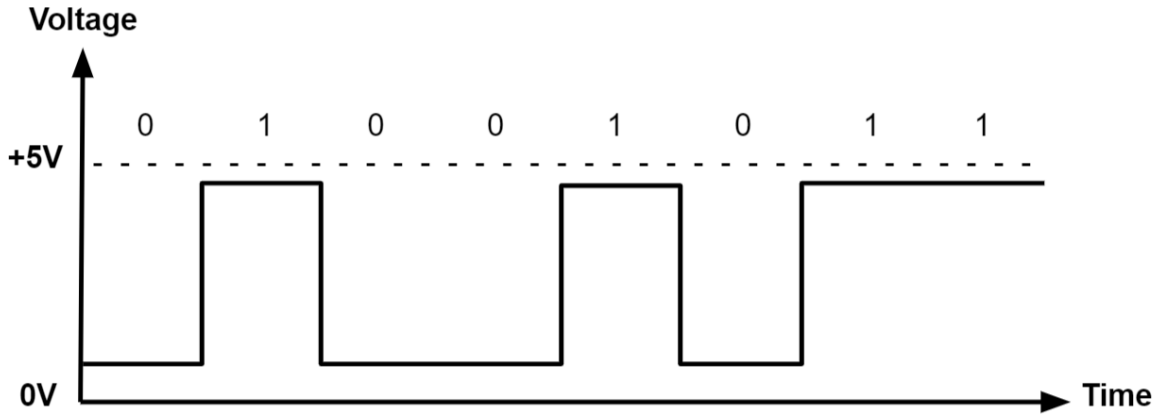
- $a_n$ is the positional value (ranging from 0 to 9), while $10^n$ represents the weight

# Binary number system

- In the binary system, the base is 2, we use only two symbols 0 and 1
- "10" is used when we hit 2 (逢二进一)
- $N = a_n \times 2^n + a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} \ \ldots\ldots + a_0 \times 2^0 + a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} \ldots$
- $a_n$ is the positional value (ranging from 0 to 1), while $2^n$ represents the weight

# Why use binary number?

- Easy to implement physically
- Simple calculation rules
- Easy to combine arithmetic and logic operations



# Hexadecimal number system

- In the hexadecimal system, the base is 16, we use 16 symbols {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f}
- "10" is used when we hit 16 (逢十六进一)
- $N = a_n \times 16^n + a_{n-1} \times 16^{n-1} + a_{n-2} \times 16^{n-2} \ \ldots\ldots + a_0 \times 16^0 + a_{-1} \times 16^{-1} + a_{-2} \times 16^{-2} \ldots$
- $a_n$ is the positional value (ranging from 0 to 15), while $16^n$ represents the weight

# Octal number system

# Converting binary number into decimal number

Example $(1101.01)_2$

$$=(1\times2^3+1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2})_{10}$$

$$=(13.25)_{10}$$

Practice $(10110.11)_2=(?)_{10}$

# Converting binary number into decimal number

Answer

$(10110.11)$

$=(1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2})_{10}=(22.75)_{10}$.

# Converting octal number into decimal number

Example $(24.67)_8=(2\times8^1+4\times8^0+6\times8^{-1}+7\times8^{-2})_{10}$

$=(20.859375)_{10}$

Practice $(35.7)_8=(?)_{10}$

# Converting octal number into decimal number

Answer

$$(35.7)_8=(3\times8^1+5\times8^0+7\times8^{-1})_{10}$$
$$=(29.875)_{10}$$

# Converting hexadecimal -> decimal

Example $(2AB.C)_{16}$

$=(2\times16^{2}+10\times16^{1}+11\times16^{0}+12\times16^{-1})_{10}$

$=(683.75)_{10}$

Practice $(A7D.E)_{16}=(?)_{10}$

# Converting hexadecimal -> decimal

Answer

$$(A7D.E)_{16}=(10\times16^{2}+7\times16^{1}+13\times16^{0}+14\times16^{-1})_{10}$$
$$=(2685.875)_{10}$$

# Converting other number system into decimal system

- Other number system can also be converted into decimal system in a similar way

- We just need to change the corresponding base

# Tests: converting into decimal system

- $(110110)\_2 = (?)\_10$
- $(101011.11)\_2 = (?)\_10$
- $(120)\_8 = (?)\_10$
- $(34.01)\_8 = (?)\_10$
- $(\mathrm{BCA})\_16 = (?)\_10$
- $(\mathrm{E05.C})\_8 = (?)\_10$

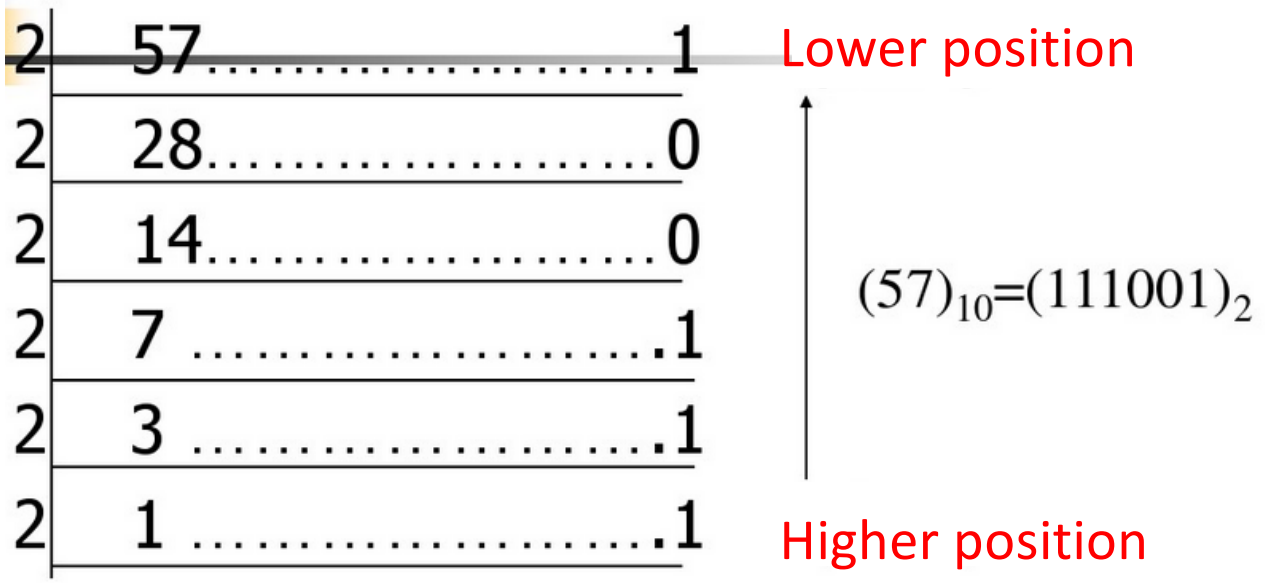
# Tests: converting into decimal system

- $(110110)\_2 = (\mathbf{118})\_10$
- $(101011.11)\_2 = (\mathbf{43.75})\_10$
- $(120)\_8 = (\mathbf{80})\_10$
- $(34.01)\_8 = (\mathbf{28.015625})\_10$
- $(\mathrm{BCA})\_16 = (\mathbf{3018})\_10$
- $(\mathrm{E05.C})\_8 = (\mathbf{3589.75})\_10$

https://www.rapidtables.com/convert/number/hex-to-decimal.html

# Converting decimal integer into binary integer

Example: $(57)_{10} = (?)_2$



# Converting decimal fraction into binary fraction

Example: $(0.875)_{10} = (?)_2$

Higher position

$0.875 \times 2 = \mathbf{1}.75$ Integer part: **1**

$0.75 \times 2 = \mathbf{1}.5$ Integer part: **1**

$0.5 \times 2 = \mathbf{1}$ Integer part: **1**



Lower position

Answer: $(0.875)_{10} = (0.111)_2$

Practice: $(0.6875)_{10} = (?)_2$

# Converting decimal fraction into binary fraction

Answer:

$0.6875 \times 2 = \mathbf{1}.375$ Integer part: **1**

$0.375 \times 2 = \mathbf{0}.75$ Integer part: **0**

$0.75 \times 2 = \mathbf{1}.5$ Integer part: **1**

$0.5 \times 2 = \mathbf{1}$ Integer part: **1**

Higher position



Lower position

So, $(0.6875)_{10} = (0.1011)_2$

# Converting decimal fraction into binary fraction

- For a decimal number that has both integer and fractional parts
- Convert the integer and fractional parts separately
- Example: $(215.3125)_{10} = (?)_2$

# Converting decimal fraction into binary fraction

Answer:

$$(215)_{10} = (11010111)_2$$
$$(0.3125)_{10} = (0.0101)_2$$
$$(215.3125)_{10} = (11010111.0101)_2$$

# The one-to-one relationship between binary and octal numbers

There is a “one-to-one” relationship between three digits binary number and one-digit octal number

$$(0)_8 = (000)_2$$
$$(1)_8 = (001)_2$$
$$(2)_8 = (010)_2$$
$$(3)_8 = (011)_2$$
$$(4)_8 = (100)_2$$
$$(5)_8 = (101)_2$$
$$(6)_8 = (110)_2$$
$$(7)_8 = (111)_2$$

# Converting octal number into binary number

- Convert each octal digit into binary number of three digits
- Keep the digit order unchanged
- Example: $(0.754)_8 = (\underline{000}.\underline{111}\ \underline{101}\ \underline{100})_2$
  $= (0.1111011)_2$
- Practice: $(16.327)_8 = (?)_2$

# Converting octal number into binary number

Answer:

$$(16.327)_8$$
$$= (\underline{001\ 110}.\ \underline{011\ 010\ 111})_2$$
$$= (1110.011010111)_2$$

# Converting hexadecimal number into binary number

- Convert each hexadecimal digit into binary number of four digits
- Keep the digit order unchanged

- Example: $(4C.2E)_{16} = (\underline{0100}\ \underline{1100}.\underline{0010}\ \underline{1110})_2$
  $= (1001100.0010111)_2$

- Practice: $(AD.7F)_{16} = (?)_2$

# Converting hexadecimal number into binary number

Answer:

$$(AD.7F)_{16}$$
$$= (\underline{1010}\ \underline{1101}.\underline{0111}\ \underline{1111})_2$$
$$= (10101101.01111111)_2$$

# Converting binary number into octal number

- Starting from lower positions, convert every three digits of the integer part into an octal digit
- When there is not enough higher positions in the integer part, fill with 0
- Starting from higher positions, convert every three digits of the fractional part into an octal digit
- When there is not enough lower positions in the fractional part, fill with 0
- Keep the digit order unchanged

# Converting binary number into octal number

Example:

$(0.10111)_2 = (\underline{000}.\ \underline{101}\ \underline{110})_2 = (0.56)_8$

$(11101.01)_2 = (\underline{011}\ \underline{101}.\ \underline{010})_2 = (35.2)_8$

Practice:

$(1101101.011)_2$

# Converting binary number into octal number

Answer:

$$(1101101.011)_2 = (\underline{001}\ \underline{101}\ \underline{101}.\ \underline{011})_2$$
$$= (155.3)_8$$

# Converting binary number into hexadecimal number

- Starting from lower positions, convert every four digits of the integer part into an octal digit
- When there is not enough higher positions in the integer part, fill with 0
- Starting from higher positions, convert every three digits of the fractional part into an octal digit
- When there is not enough lower positions in the fractional part, fill with 0
- Keep the digit order unchanged

# Converting binary number into hexadecimal number

Example:

$$(11101.01)_2 = (\underline{0001}\ \underline{1101}.\ \underline{0100})_2$$
$$= (1D.4)_{16}$$

# The units of information (data)

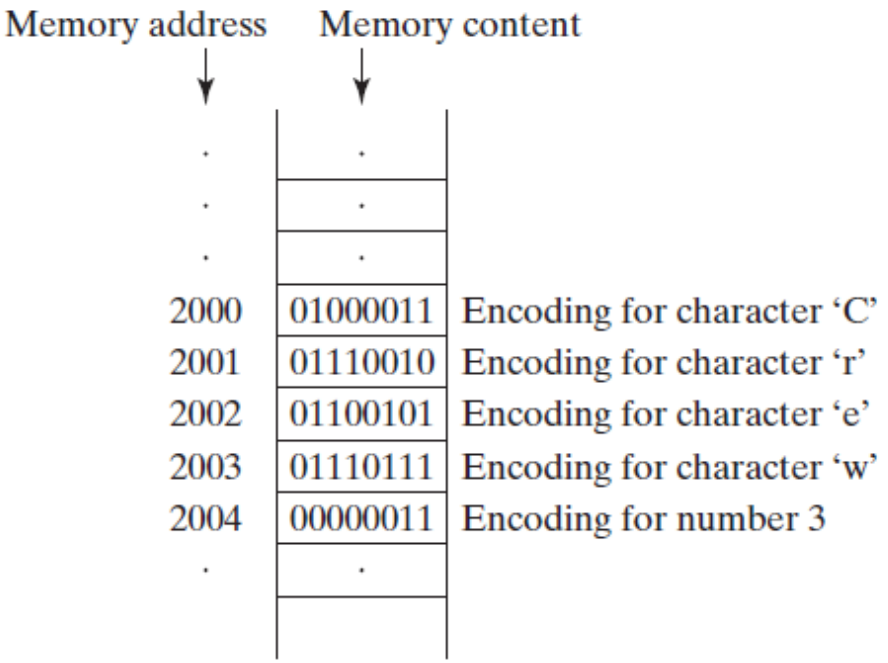
- Bit: a binary digit which takes either 0 or 1
- Bit is the smallest information unit in computer programming
- Byte: 1 byte = 8 bits, every English character is represented by 1 byte
- Kilobyte (KB): 1 KB = 2^10 B = 1024 B
- Megabyte (MB): 1MB = 2^20 B = 1024 KB
- Gigabyte (GB): 1GB = 2^30 B = 1024 MB
- Terabyte (TB): 1TB = 2^40 B = 1024 GB

Note:
In computation, 2^10 is often used to represent 1K because computers operate in binary.

However, in storage and networking, 1K = 1000 (based on the metric system, SI units).

# Memory and addressing

- A computer's memory consists of an ordered sequence of bytes for storing data
- Every location in the memory has a unique address
- The key difference between high- and low-level programming languages is whether programmer has to deal with memory addressing directly



Thanks