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

SCHOOL OF
DATA SCIENCE
數據科學學院

Introduction to Computer Science: Programming Methodology

Lecture 1 Introduction

Tongxin Li

School of Data Science

About this course

- CSC1001 introduces programming with Python. No prior programming experience is required.
- Fall 2026 teaching runs from 7 September to 18 December.
- Tutorials begin next week.
- Lecture materials, tutorial details, and announcements are available through the course website and Blackboard.
- This course is a required course for all SDS students.

(For students from other schools, please directly submit your applications to add)

- Course materials synced between other 5 sessions (Lecture slides, assignments, exams are the same)

About me

Background

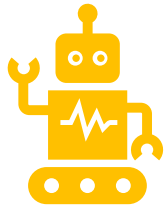
Education: CUHK, Caltech

Working Experience:

- Amazon Web Services (2020, 2021), Applied Scientist Intern
- Assistant Professor (2022-present): SDS, CUHK-SZ

Contact

Email: litongxin@cuhk.edu.cn



Research

General areas: Machine Learning, Control, Sustainability

Topics: control & optimization, online algorithms, reinforcement learning

Learning Objectives

- This course introduces the basics of computer programming using Python

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

Learning at university

- Plan regular practice time and make use of tutorials.
- Check Blackboard and university email for announcements.
- AI tools are allowed on assignments. You must understand and explain your solution.
- Complete assignment problems independently. Collaboration with other students is not allowed.
- Submit one PDF through Blackboard by the stated deadline.

Key Topics

- Introduction to modern computers
- Preliminary knowledge for computer programming
- Basic introduction to Python language
- Data types and operators in Python language
- Input/output
- Flow control and loop
- Function
- List
- Basic data structure
- Introduction to algorithm design
- Introduction to object oriented programming

Assessment

Assignments Best 3 of 4	**15%**
Midterm exam	**35%**
Final exam	**50%**

Midterm Exam: Friday 23 October 2026 6.30 pm to 8.30 pm

Exam venue to be announced on Blackboard

Course materials and exam rules

- Lecture notes and course information
- tongxin.me/CSC1001-2026fall/
- Announcements and submissions
- bb.cuhk.edu.cn
- Python reference
- docs.python.org/3
- Exams are in person and closed book. One A4 sheet is allowed, handwritten or printed. No other materials or electronic aids.

Lecture times and course support

Section	Monday and Wednesday
L02	10.30 am to 11.50 am
L03	1.30 pm to 2.50 pm

Teaching B Building 103

Fall 2026 teaching plan

Week	Dates and topic
1	7 and 9 Sep Introduction
2	14 and 16 Sep Python basics
3	21 and 23 Sep Flow control
4	28 and 30 Sep Functions
5	5 and 7 Oct National Day holiday, no lectures
6	12 and 14 Oct Lists
7	19 and 21 Oct Midterm review
8	26 and 28 Oct Classes and objects
9	2 and 4 Nov Inheritance
10	9 and 11 Nov Data structures and complexity
11	16 and 18 Nov Recursion, stacks, and queues
12	23 and 25 Nov Linked lists
13	30 Nov and 2 Dec Trees
14	7 and 9 Dec Final review

Course website and contacts

- Course website
- tongxin.me/CSC1001-2026fall/
- Instructor Tongxin Li
- litongxin@cuhk.edu.cn
- Leading TA Yu Mao
- 118010224@link.cuhk.edu.cn
- Full teaching team and tutorial updates appear on the website and Blackboard.

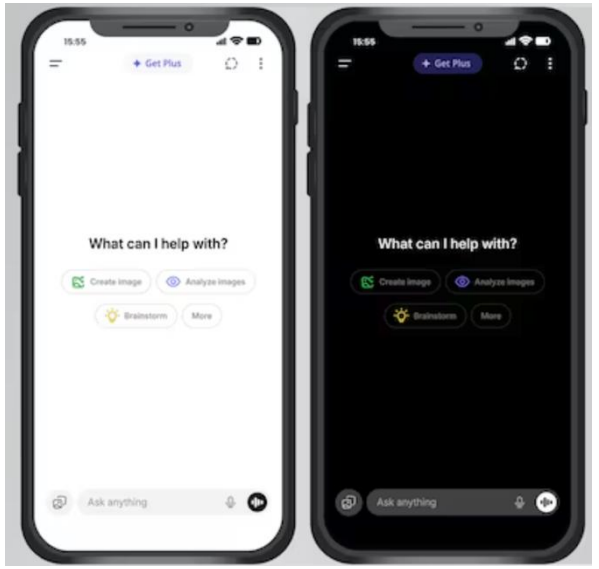
Why learn programming?

- Programming turns a problem into precise instructions.
- AI can translate natural language into candidate code.
- Python lets us read those instructions and test whether they solve the intended problem.

User → **Interface**

Enters a question or
chooses an action

An app, a website,
or a conversation



Program and computer ← **Programmer**

Code processes data
and produces results

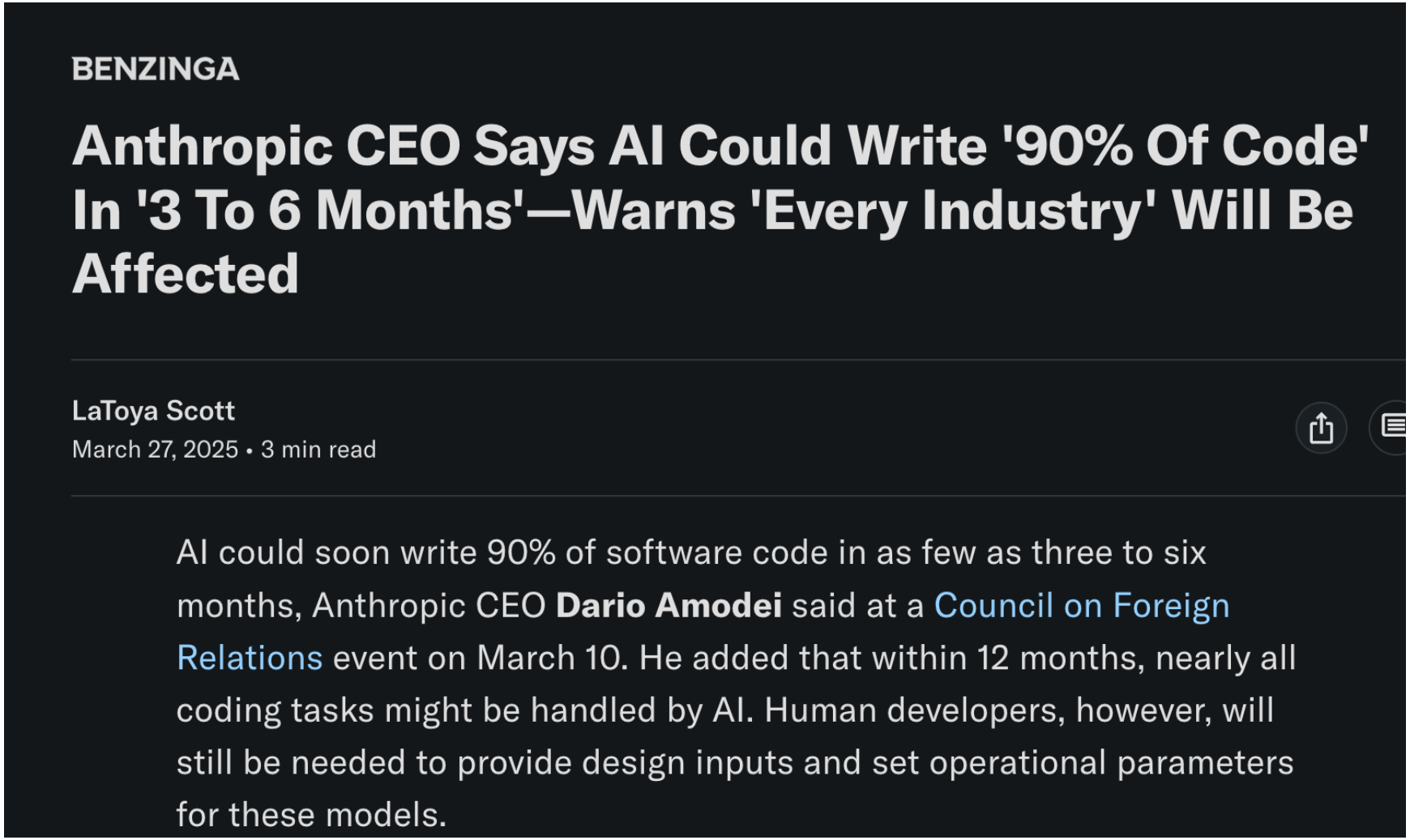
Defines behavior
and checks results

- Programmers build software that helps users solve problems.

What programmers do

- Translate needs into precise and testable instructions
- Build, review, and maintain software
- Work across research, engineering, business, and public services

AI coding forecasts



BENZINGA

Anthropic CEO Says AI Could Write '90% Of Code' In '3 To 6 Months'—Warns 'Every Industry' Will Be Affected

LaToya Scott
March 27, 2025 • 3 min read

AI could soon write 90% of software code in as few as three to six months, Anthropic CEO **Dario Amodei** said at a Council on Foreign Relations event on March 10. He added that within 12 months, nearly all coding tasks might be handled by AI. Human developers, however, will still be needed to provide design inputs and set operational parameters for these models.

Benzinga 27 March 2025 A prediction made in 2025, not a measured result.

AI coding in industry

TECH

Satya Nadella says as much as 30% of Microsoft code is written by AI

PUBLISHED TUE, APR 29 2025•9:33 PM EDT | UPDATED TUE, APR 29 2025•9:58 PM EDT

Jordan Novet
@JORDANNOVET

Jonathan Vanian
@IN/JONATHAN-VANIAN-B704432/

SHARE

KEY POINTS

- Microsoft CEO Satya Nadella on Tuesday said that as much as 30% of the company's code is now written by artificial intelligence.
- Nadella made the comments during a conversation before a live audience with Meta CEO Mark Zuckerberg at the social media company's LlamaCon AI developer event.

CNBC 29 April 2025 A reported company claim with its own scope.

Coding agents in 2026

GitHub Copilot coding agent

GitHub update 26 February 2026

- Agents can implement a delegated task and open a pull request for review.
- New checks include AI code review and scanning for security issues.
- The developer reviews the proposed changes before accepting them.

Source GitHub Blog What’s new with GitHub Copilot coding agent

What Should We Do?

Describe the task precisely

Predict what the code will do

Compare the result with a test

Why learn Python when AI can write code?

- **Aim Higher:** AI develops code. You develop the AI.
- **Get Hired:** Prove your problem-solving skills in technical interviews.
- **Work Smarter:** Need knowledge to prompt and debug for more effective vibe coding.

Why be 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? Software? Program?

- A sequence of instructions
- Computers take the instructions and execute them
- It is a little piece of our intelligence in the computer
- Intelligence which is re-usable

Computers are good at following instructions

- Computers execute instructions precisely and at great scale
- Results can still be wrong because specifications, code, data, models, and hardware can all fail
- Programming includes checking assumptions and testing the result

Computers

Are they computers ？

calculator

router

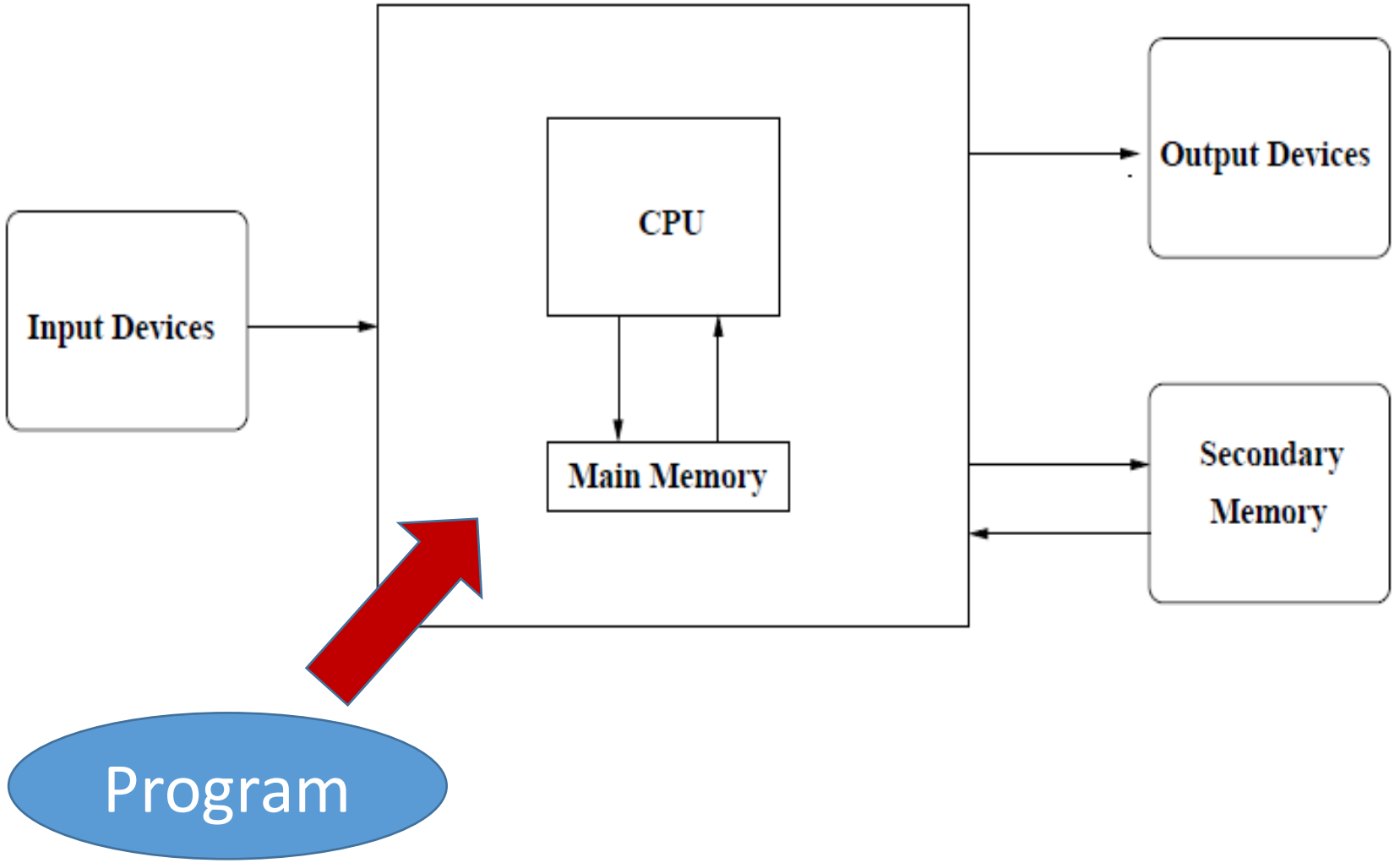
robot

smartwatch

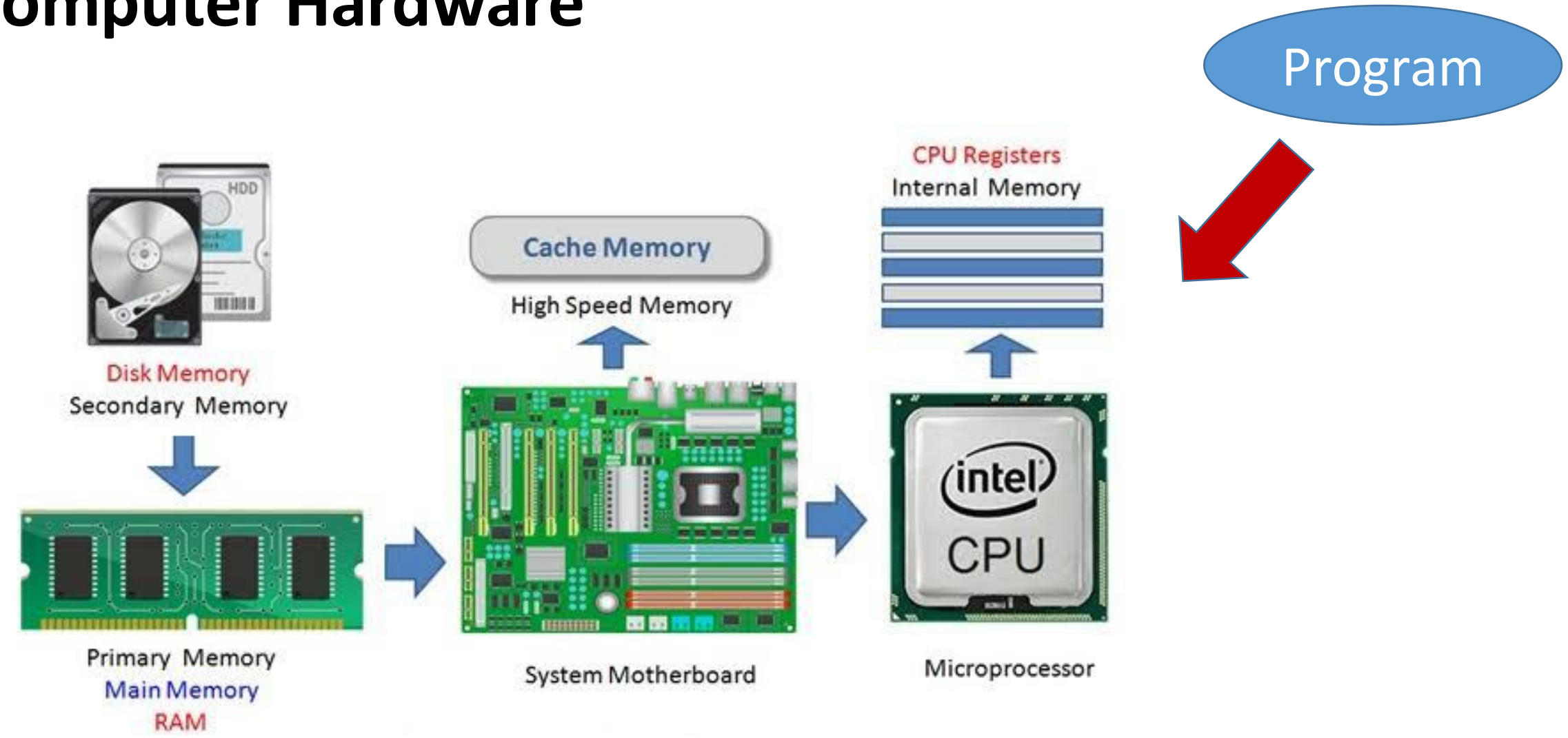
Smart TV

Smartphone

Computer Hardware

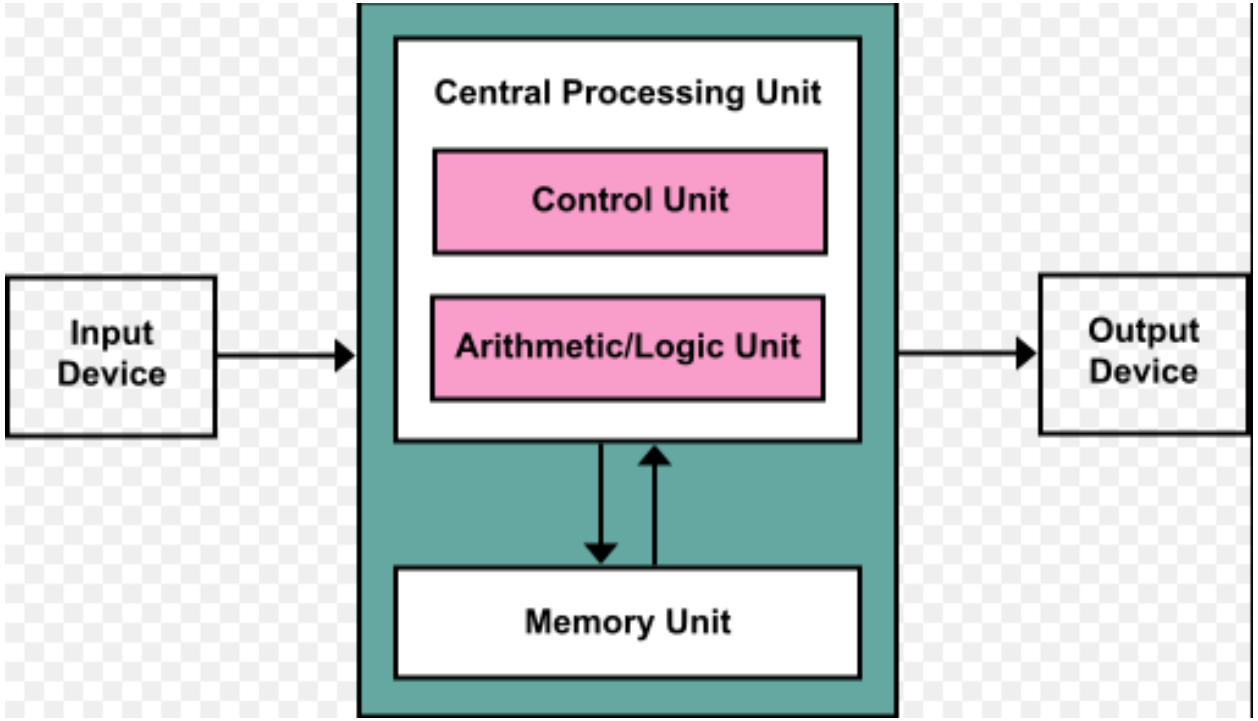


Computer Hardware



Von Neumann Architecture

- The modern computer architecture is proposed by John Von Neumann



Key components in a computer

- Central processing unit (CPU) executes instructions and coordinates computation
- Input devices provide data or commands
- Output devices communicate results
- Main memory is fast and temporary
- Secondary storage is larger and persistent

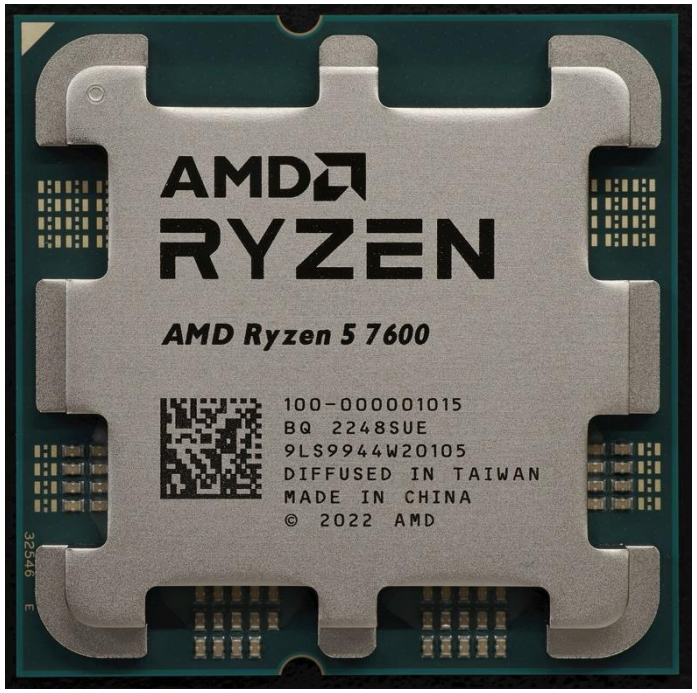
Central Processing Unit

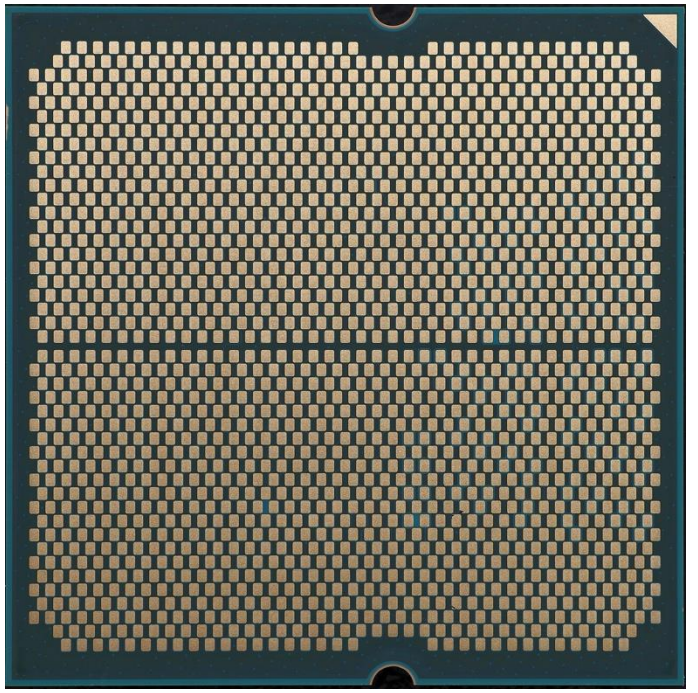
A processor contains two units, a control unit (CU) and an arithmetic/logic unit (ALU)

CU is used to fetch commands from the memory

ALU contains the electric circuits which can execute commands

Central Processing Unit





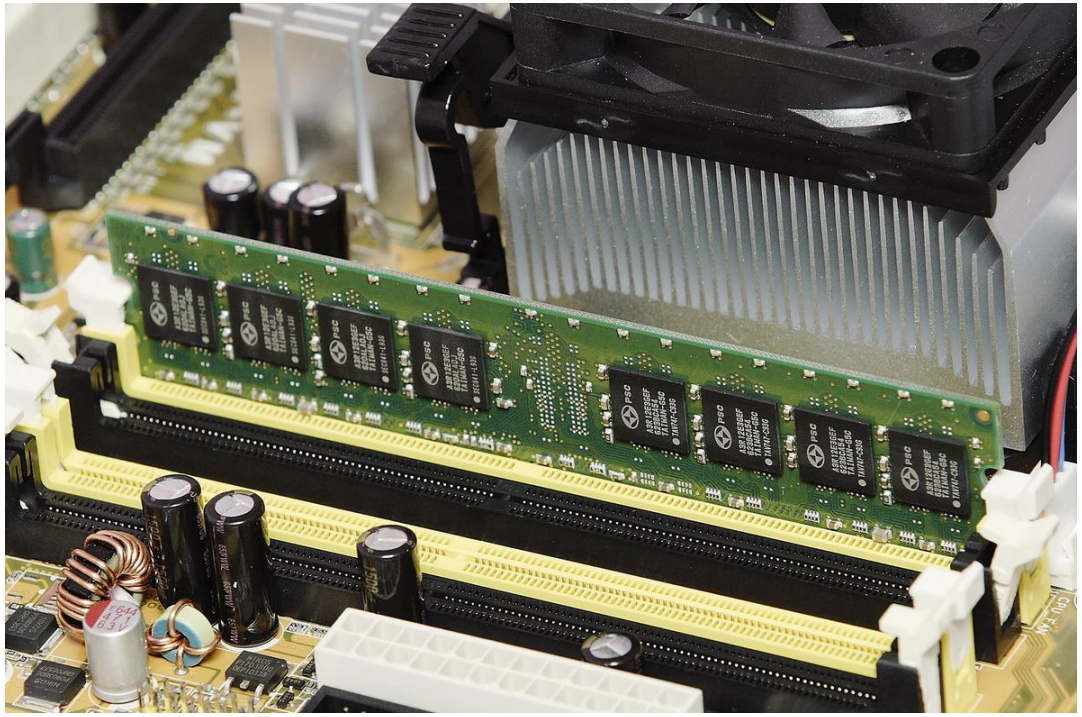
A modern processor package and its electrical contacts

- Processor vendors include Intel and AMD. Architecture families include x86 and Arm.

Photos by Rainer Knäpper Free Art License Sources in notes

Memory/Storage

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



Memory and storage

Storage type	Typical use	Without power
CPU registers	Values used immediately	Data lost
CPU cache L1, L2, L3	Recently used data and instructions	Data lost
Main memory RAM	Running programs and their data	Data lost
Secondary storage SSD, HDD	Files and installed programs	Data retained
Backup and archive For example, tape	Copies kept for recovery or long term storage	Data retained

Moving from registers to secondary storage trades speed for capacity.

Volatile memory needs power. Backups can use disks or tape.

Input/output devices

- Input devices include a keyboard, touchscreen, microphone, and camera.
- Output devices include a display and speakers.

One device can support several kinds of input

Any other input devices?

Which input could replace the keyboard?

Think of one situation where typing is inconvenient.

Any other input devices?

Camera

Images for a QR code reader

Motion sensor

Orientation for a phone

Microphone

Audio for speech recognition

Touch sensor

Contact and pressure

Any other output devices?

How could a computer give feedback without a screen?

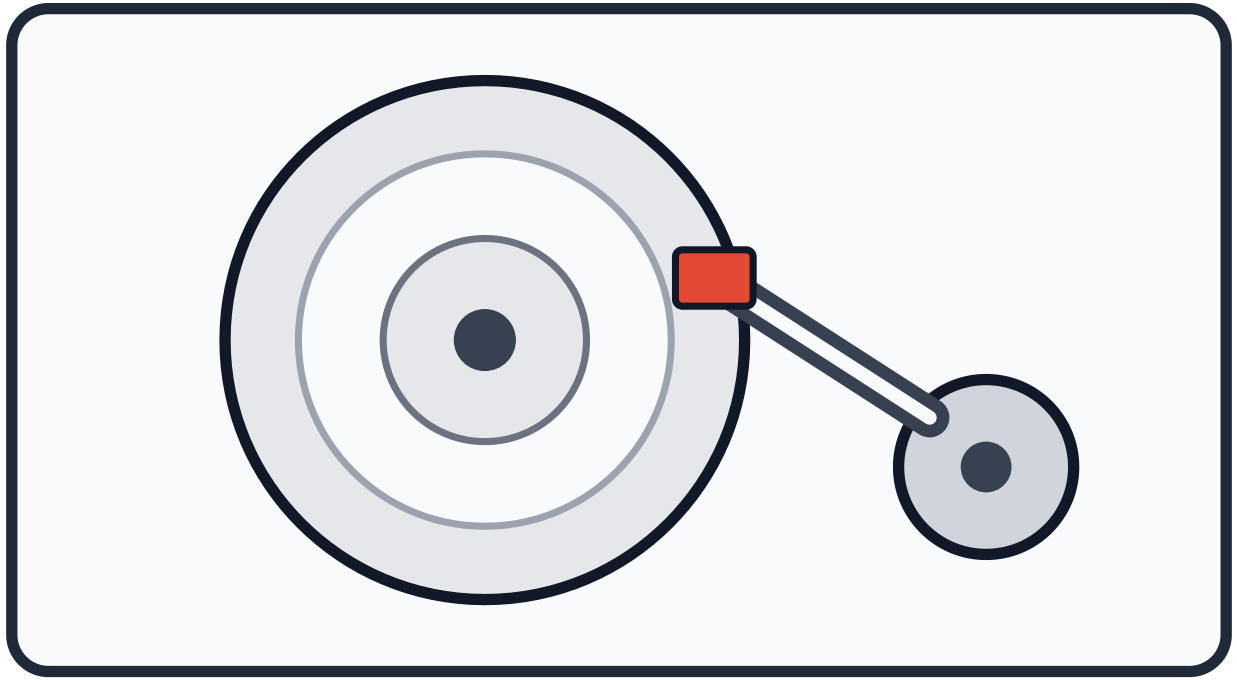
Think about sound, vibration, or physical movement.

Any other output devices?

Headset displays

Spatial audio

How the hard disk works



Static view of a hard disk
Platter • track • sector • read or write head

Programming languages

Python source

Human readable instructions such as print (2 + 3)

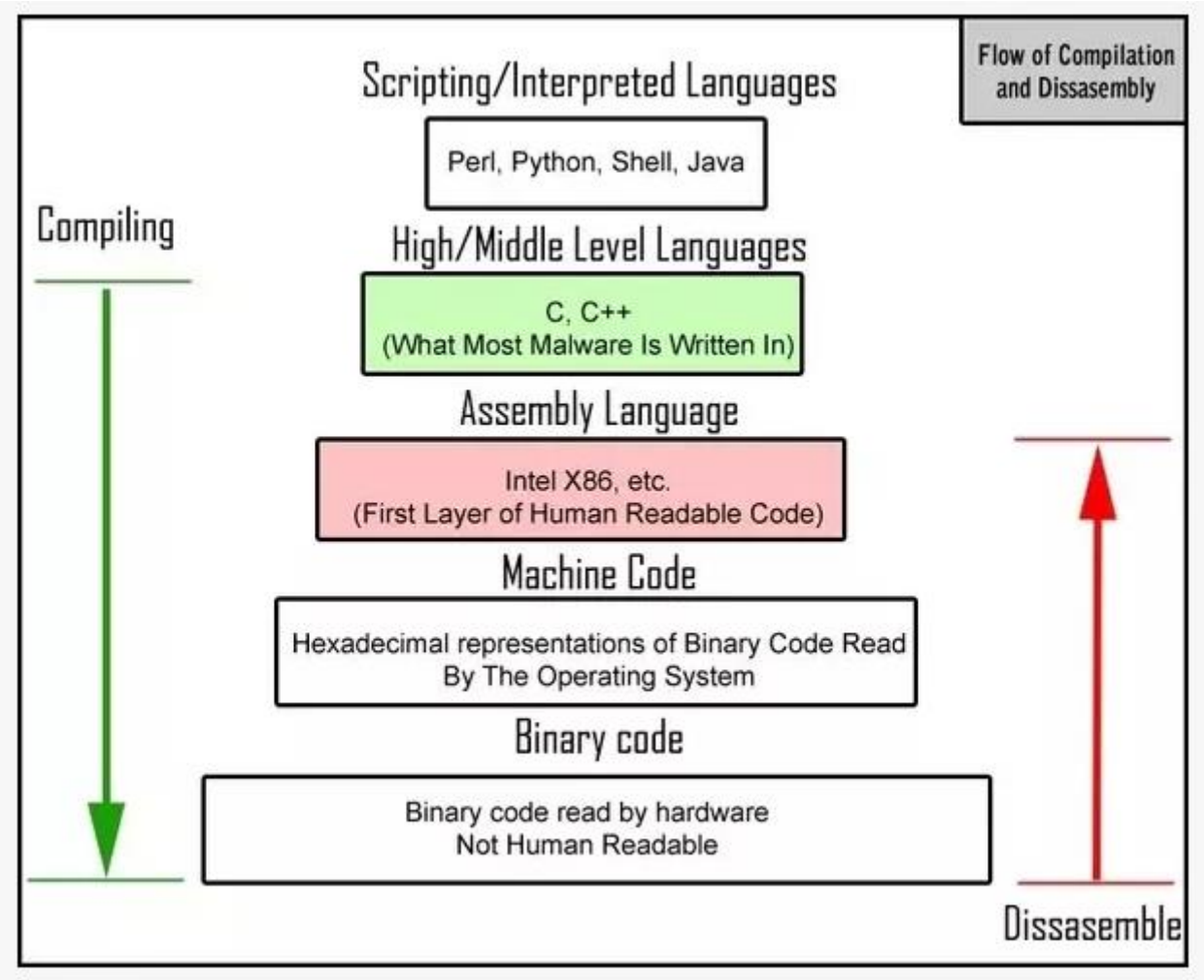
Python runtime

Executes the program
CPython uses bytecode internally

Computer hardware

The CPU executes machine instructions
The program produces the output 5

Programming languages



Low level language – Assembly Language

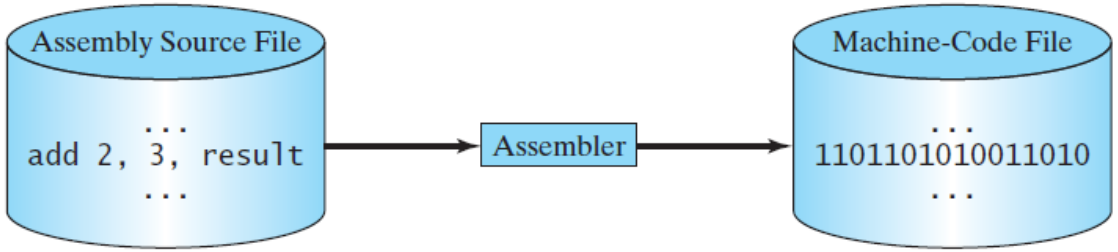
- An assembly language is a low-level programming language, in which there is a very strong (generally one-to-one) correspondence between the language and machine code instructions.

- 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
```



Runtime performance and development productivity

- High level programs are translated or interpreted (to low level programs) for execution
- Higher level languages often improve development speed and maintainability, often with a cost of runtime performance
- Runtime performance depends on the algorithm, implementation, compiler or interpreter, and hardware

Operating Systems

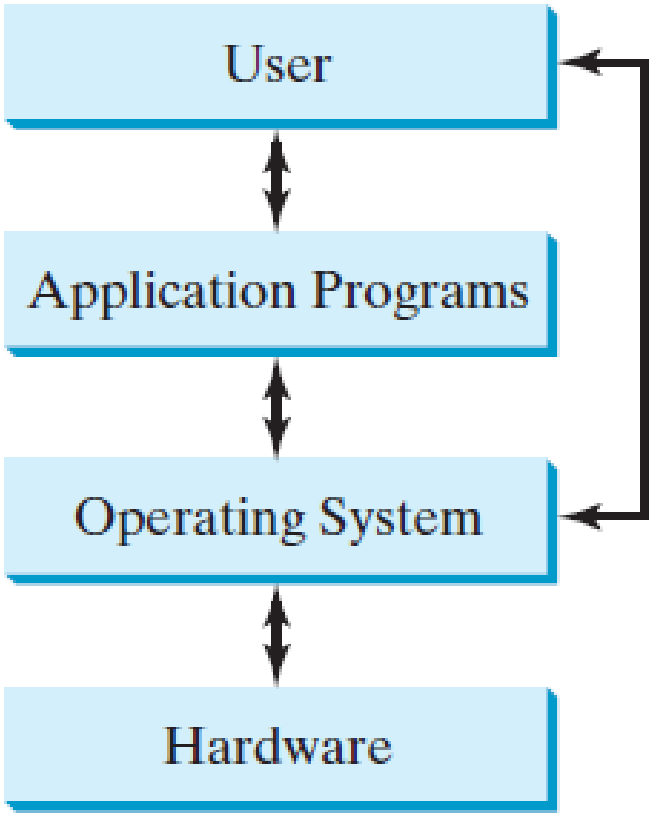
- The operating system (OS) is a low level program, which provides all basic services for managing and controlling a computer's activities

- 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…



C Language (1969 - 1973)

- C was developed by Dennis Ritchie between 1969 and 1973 at AT&T 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 could be 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

Popular Java Software?

Popular Java Software?

Most games use C++

Python (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.

Popular Python Software?

Popular Python Software?

Do they use Python 100%?

Python libraries for data and AI

Arrays and data tables	**NumPy and pandas**
Machine learning	**scikit learn**
Neural networks	**PyTorch**

Libraries build on the same Python fundamentals we learn here.

Python (1991)

- **Python is evolving …**

- The best way to keep track of the updates is to learn by really using Python after taking lectures

From https://www.python.org/doc/versions/

Python Documentation by Version

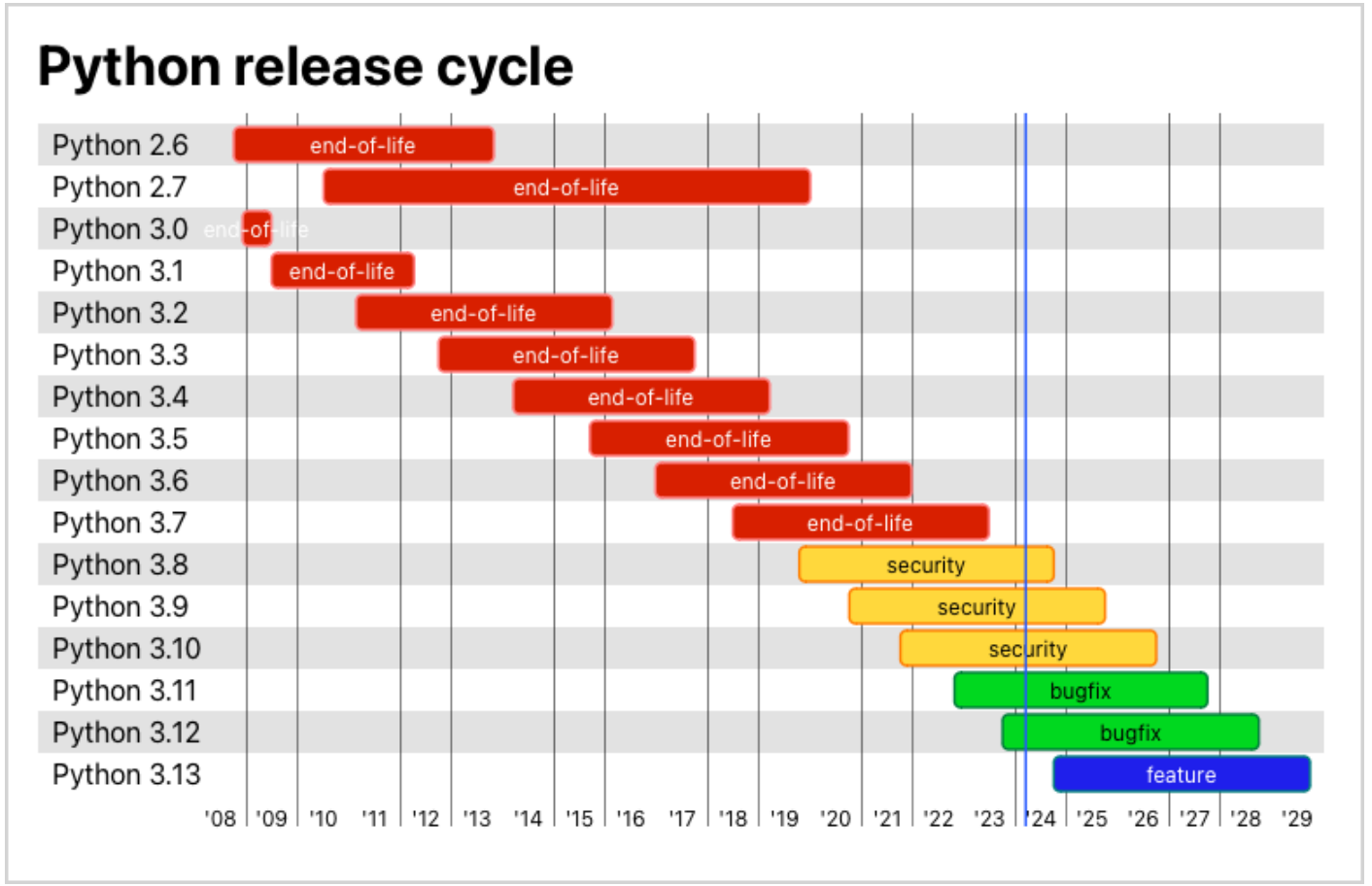
Some previous versions of the documentation remain available online. Use the list below to select a version to view.

For unreleased (in development) documentation, see In Development Versions.

- Python 3.12.5, documentation released on 6 August 2024.
- Python 3.12.4, documentation released on 6 June 2024.
- Python 3.12.3, documentation released on 9 April 2024.
- Python 3.12.2, documentation released on 6 February 2024.
- Python 3.12.1, documentation released on 8 December 2023.
- Python 3.12.0, documentation released on 2 October 2023.
- Python 3.11.9, documentation released on 2 April 2024.
- Python 3.11.8, documentation released on 6 February 2024.
- Python 3.11.7, documentation released on 4 December 2023.
- Python 3.11.6, documentation released on 2 October 2023.
- Python 3.11.5, documentation released on 24 August 2023.
- Python 3.11.4, documentation released on 6 June 2023.
- Python 3.11.3, documentation released on 5 April 2023.
- Python 3.11.2, documentation released on 8 February 2023.
- Python 3.11.1, documentation released on 6 December 2022.
- Python 3.11.0, documentation released on 24 October 2022.
- Python 3.10.14, documentation released on 19 March 2024.
- Python 3.10.13, documentation released on 24 August 2023.
- Python 3.10.12, documentation released on 6 June 2023.
- Python 3.10.11, documentation released on 5 April 2023.
- Python 3.10.10, documentation released on 8 February 2023.
- Python 3.10.9, documentation released on 6 December 2022.
- Python 3.10.8, documentation released on 8 October 2022.

Python (1991)

- **Python is evolving …**
- **The goal of this course:**
  - ~~Keep track of recent updates~~ ❌
  - Provide you a comprehensive knowledge base of programming via Python ✓
  - Our students have very diverse backgrounds …



Python 3 for this course

- Use the Python 3 environment specified by the course
- Verify the interpreter before every assignment
- Use docs.python.org/3 for current language and library documentation

Python 3 for this course

- Python 3 is the language for our code examples.
- The core ideas remain useful as languages change.
- Read and test generated code using the same fundamentals.

Course focus

Types and control flow
Functions and data structures
Classes and algorithms

What can a computer actually understand?

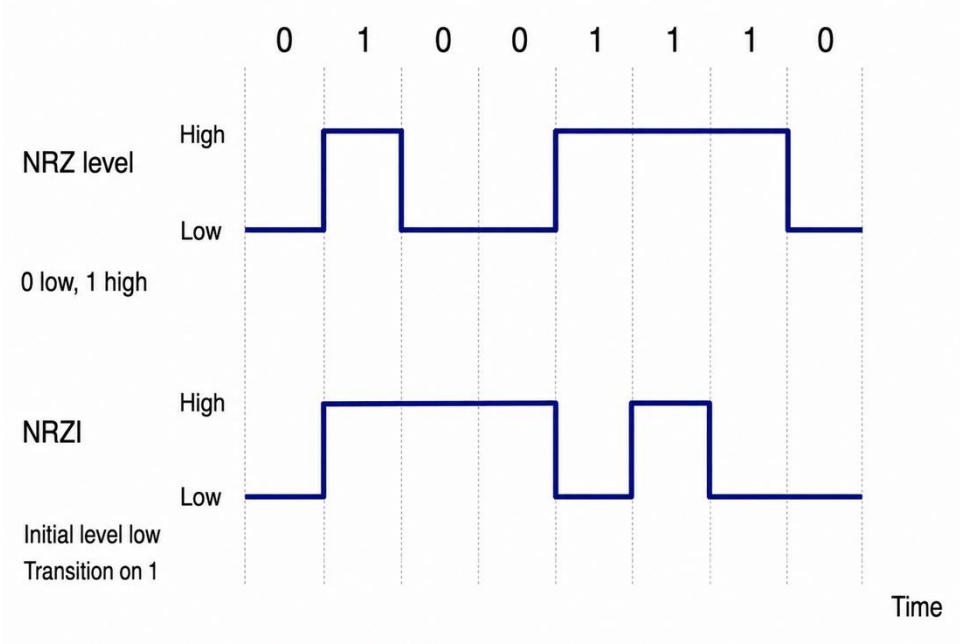
- The computers used nowadays can understand only binary number (i.e. 0 and 1)
- Computers use voltage levels to represent 0 and 1
- NRZL and NRZI coding
- The instructions expressed in binary code is called machine language

0 0 0 1 numerical value 2^0

0 0 1 0 numerical value 2^1

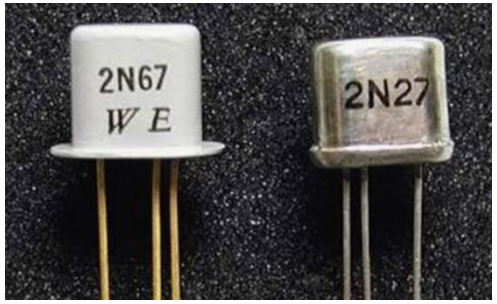
0 1 0 0 numerical value 2^2

1 0 0 0 numerical value 2^3

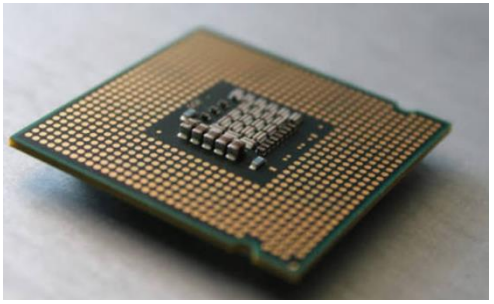


Numbers inside a chip

- Transistors form logic gates that work with binary states 0 and 1.
- Registers store bits. Adders in the ALU follow binary carry rules.



Transistors



CPU with Registers

Numbers inside a chip

- Transistors form logic gates that work with binary states 0 and 1.
- Registers store bits. Adders in the ALU follow binary carry rules.

Example of an unsigned value in an 8 bit register

Place value	128	64	**32**	16	**8**	4	**2**	1
Stored bit	0	0	**1**	0	**1**	0	**1**	0

$00101010_2 = 32 + 8 + 2 = 42_{10} = 2A_{16}$

These rules let us interpret stored data and check calculations.
Conversion expresses the same value in a different number system.

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
- $N = a_n \times 10^n + a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2}$ …… $+ a_0 \times 10^0 + a_{-1} \times 10^{-1}$ - 10^{-2} …
- 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)
- $N = a_n \times 2^n + a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} \ \ldots\ldots + a_0 \times 2^0 + \ a_{-1} \times 2^{-1} + 2^{-2} \ldots$
- a_n is the positional value (ranging from 0 to 1), while 2^n represents the weight

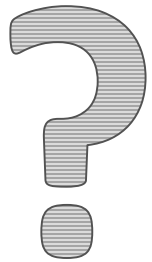
Why use binary number?

- Easy to implement physically
- Simple calculation rules
- Easy to combine arithmetic and logic operations
- Against noise (for analog signal)

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)
- $N = a_n \times 16^n + a_{n-1} \times 16^{n-1} + a_{n-2} \times 16^{n-2}$ …… $+ a_0 \times 16^0 + a_{-1} \times 16^{-1}$ - 16^{-2} …
- 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 $(24.67)_8=(2 \times 8^1+ 4 \times 8^0+6 \times 8^{-1}+7 \times 8^{-2})_{10}$

$=(20.859375)_{10}$

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 number into decimal number

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 number into decimal number

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

Some tests: converting into **decimal** system

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

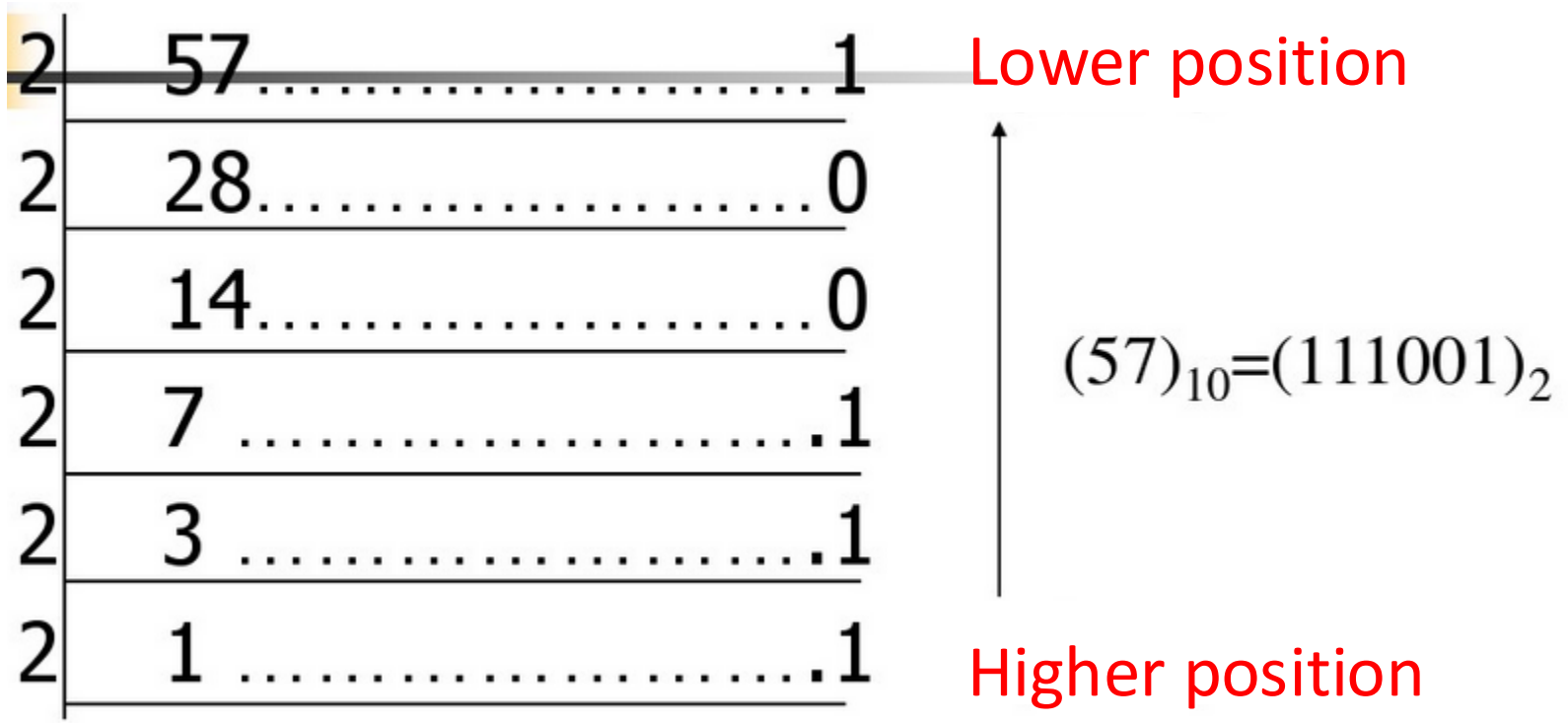
Some tests: converting into **decimal** system

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

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

How to convert fractions to binary?

STEP 1: Take a decimal fraction and start multiplying by two the decimal part.

STEP 2: Every time the result is smaller than 1 , add a 0 to the binary representation. If the result is greater or equal to 1 , add a 1 to the binary representation and subtract 1 from the multiplication result.

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 number into binary number

- 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 number into binary number

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” (一一对应) relationship between three digits binary number and one digit octal number

$$
\begin{aligned}
(0)_8 &= (000)_2 \\
(1)_8 &= (001)_2 \\
(2)_8 &= (010)_2 \\
(3)_8 &= (011)_2 \\
(4)_8 &= (100)_2 \\
(5)_8 &= (101)_2 \\
(6)_8 &= (110)_2 \\
(7)_8 &= (111)_2
\end{aligned}
$$

Converting octal number into binary number

- Convert each octal digit into binary number of three digits
- Keep the digit order unchanged
- Example: $(0.754)_8 = (?)_2$

$$(0.754)_8 = (\underline{000}.\underline{111}\ \underline{101}\ \underline{100})_2$$
$$= (0.1111011)_2$$

- Practice: $(16.327)_8 = (?)_2$

Converting octal number into binary number

Answer:

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

Converting hexadecimal number into binary number

- Convert each hexadecimal digit into binary number of four digits

- Keep the digit order unchanged

- Example: $(4C.2E)_{16} = (?)_2$

$$
\begin{aligned}
&(4C.2E)_{16} \\
&= (\underline{0100}\ \underline{1100}.\underline{0010}\ \underline{1110})_2 \\
&= (1001100.0010111)_2
\end{aligned}
$$

- 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 binary digits of the integer part into one hexadecimal digit

When higher positions are missing, fill with 0

Starting from higher positions, convert every four binary digits of the fractional part into one hexadecimal digit

When lower positions are missing, fill with 0

Keep the digit order unchanged

Converting binary number into hexadecimal number

Example:

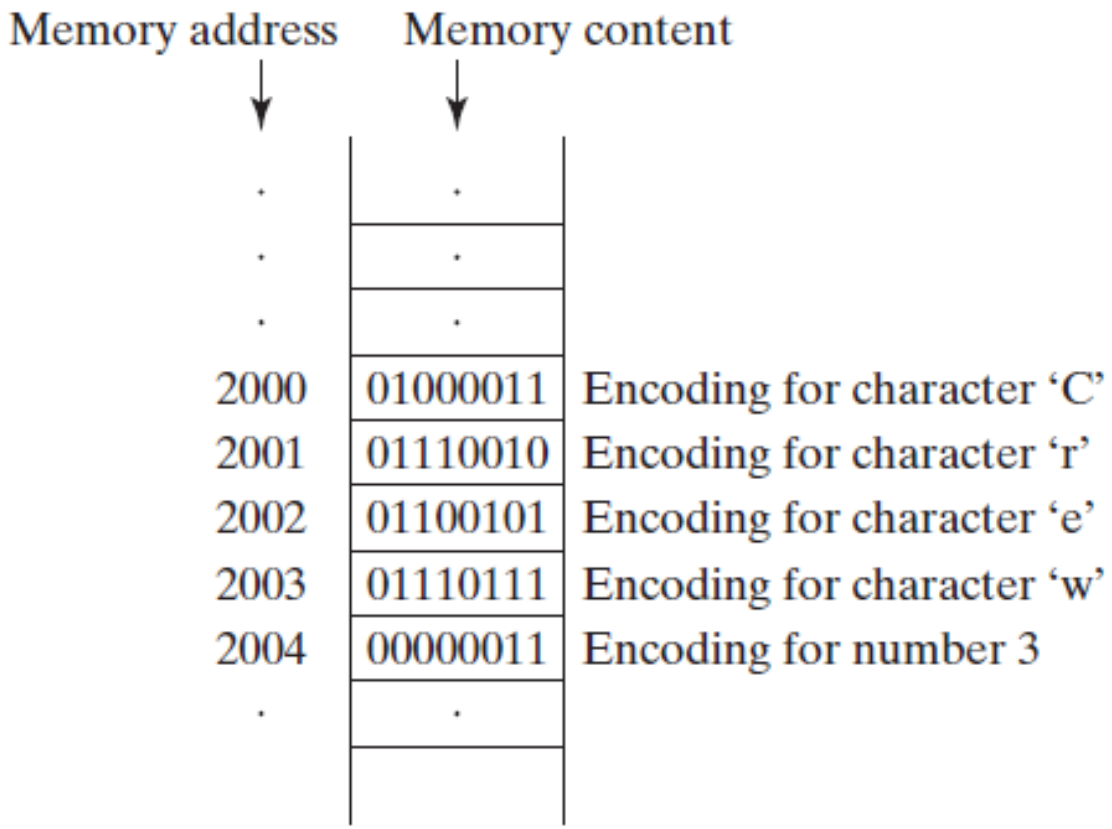
$$(11101.01)_2 = (\underline{0001}\ \underline{1101}.\ \underline{0100})_2$$
$$= (1D.4)_{16}$$

The units of information (data)

- Bit a binary digit that stores 0 or 1
- Byte 8 bits
- Character an abstract symbol. An encoding such as UTF 8 maps characters to one or more bytes
- KiB 2^10 bytes
- MiB 2^20 bytes
- GiB 2^30 bytes
- TiB 2^40 bytes

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 needs to deal with memory addressing directly



Practice

- $(135.8125)_{10} = (10000111.1101)_{2}$
- $(1314.205)_{8} = (1\ 011\ 001\ 100.010\ 000\ 101)_{2}$
- $(0101010000.0010110011)_{2} = (520.1314)_{8}$
- $(0101010000.0010110011)_{2} = (150.2CC)_{16}$

Thanks