

CS2311 Computer Programming

LT01: Introduction

Computer Science, City University of Hong Kong
Semester B 2022-23

About the Course

- Lecture Instructor:
 - Dr. Junqiao QIU, Assist. Prof. at CS, YEUNG Y7708
 - Email: junqiqiu@cityu.edu.hk
- Lab Instructor:

• Dr. Junqiao QIU	1000-1050 Wednesday
• Runze MAO	1000-1050 Monday (runzemaos2-c@my.cityu.edu.hk)
• Wentao DONG	1200-1250 Monday (wentdong-c@my.cityu.edu.hk)
• Rui LIAN	1300-1350 Monday (rui.lian@my.cityu.edu.hk)
- Post your questions on Canvas for *quick response*
 - Canvas => Discussions

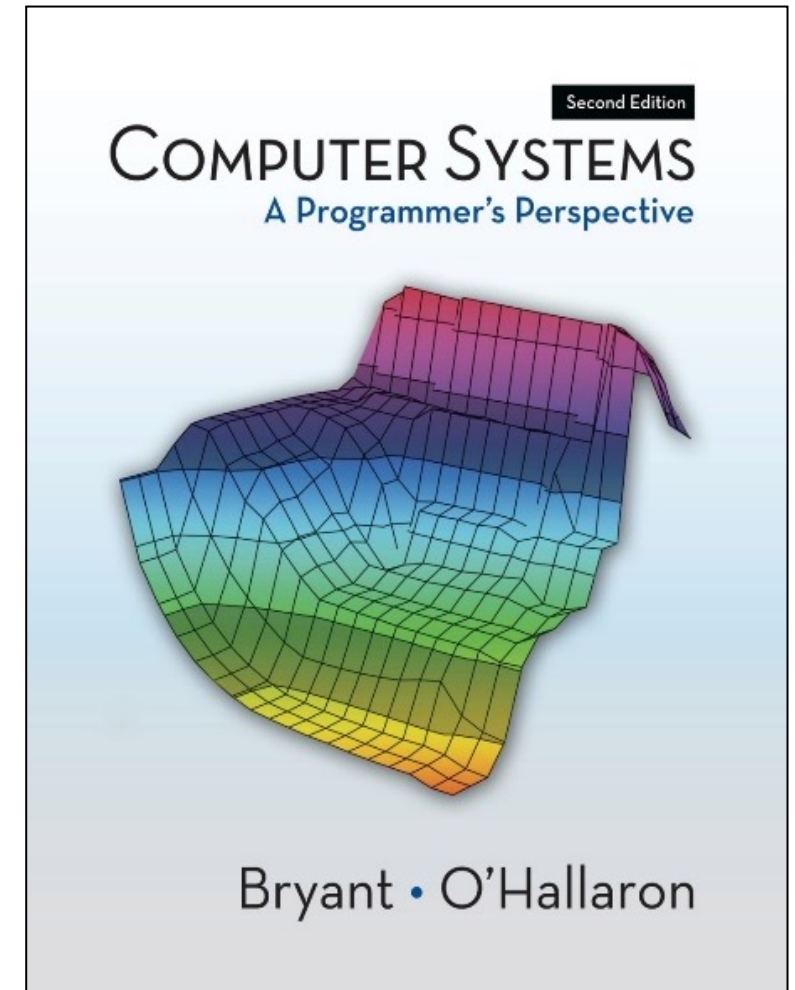
About the Course

- Teaching pattern
 - Lectures
 - Explain the terminologies, concepts, methodologies, ...
 - Face-to-face at 15:00 ~ 17:50, Friday, YEUNG LT-9
 - Labs
 - Each lecture will have a corresponding Lab (4 sessions on Monday/Wednesday)
 - *No lab on wk12 Wed and wk13 Mon (due to public holiday)*
 - Analyzing and solving example problems
 - Hands-on programming practice
- Office Hours
 - 14:00 - 15:00, Friday, YEUNG Y7708 or Schedule an appointment

About the Course

- Resources

- Canvas-based course website
 - Teaching materials are **all** in Canvas
 - It is your own responsibility to check Canvas and University emails **regularly** for updates
- Textbook (NIL)
- Reference books
 - *Computer Systems: A Programmer's Perspective*, by Randal E. Bryant and David R. O'Hallaron, Prentice Hall, 2011
- Microsoft Visual Studio 2019 (Windows)
 - Develop environment for compiling & debugging
- PASS (program assignment assessment system)
 - Program testing and submission



About the Course

- Assessment
 - Coursework (40%)
 - Assignments (15%)
 - Start **as early as possible** and submit **on time**
 - Debugging takes **much longer** time than you expect, make a good plan
 - Midterm quiz (15%)
 - **10 Mar. 2023 (week 8)**
 - Lab exercises (10%)
 - **Assess 10 lab problems**
 - **Deadline is 24 hours after your lab session (except the last one)**
 - **Bonus (3%)**
 - Final exam (60%)

About the Course

- Assessment
 - To **pass** the course you must obtain at least **30%** of the final exam marks (No. 1 reason to fail this course)

Student	Coursework	Exam	Final Mark	Grade
1	94.3	95.5	95.14	A+
2	33.8	34	34	D
3	86.8	26.5	44.59	F

About the Course

- Key to success

Just Do It

But, do it yourself

About the Course

- “*Do it yourself*” means
 - Discuss the problems with any other people
 - Study materials on the internet
 - Refer to any books
- But, the **details** and **write-ups** must be **entirely** your own work

University requirement on academic honesty.

- Violations of academic honesty are regarded as serious offences in the University. Acts such as plagiarism (and fabrication of research findings) can lead to disciplinary action. Most commonly the penalty is **failure in a course**, but in the most serious cases expulsion from the University and debarment from re-admission may occur.

About the Course

- Plagiarism
 - **Punishment** ranges from **warning** to course **failure**
 - May cause you be forced out of CityU
 - Can be **automatically detected** by PASS system
- How to prevent
 - In plagiarism cases, both the **giver** and **copier** get punishments
 - Protect your code
- As instructors
 - We have the responsibility to report academic dishonesty cases so as not to compromise the quality of education
 - We take suspected plagiarism cases very seriously.

Outline for Today

- What's a computer
- Programming languages
- Basics of computer programming

Decimal vs Binary

10^2 10^1 10^0

1 0 3

Decimal vs Binary

10^2	10^1	10^0		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	3		0	1	1	0	0	1	1	1
				$0 + 64 + 32 + 0 + 0 + 4 + 2 + 1$							
				$= 103$							

Decimal to Binary

- what's the binary expression of $(30)_{10}$?

11110

divide the given number by 2 continuously
till we get the quotient as 0.

Hexadecimal

$$\begin{array}{ccccccccc} & 16^3 & 16^2 & 16^1 & 16^0 & & (257)_{10} \\ 0x & 0 & 1 & 0 & 1 & = & ? \end{array}$$

Hexadecimal

16^3

16^2

16^1

16^0

0x 0 1 0 1 = ?

0x 0 0 a f = ?

$(175)_{10}$

Hexadecimal

- what's the hex expression of **4387**?

0x1123

Conversion steps:

- Divide the number by 16.
- Get the integer quotient for the next iteration.
- Get the remainder for the hex digit.
- Repeat the steps until the quotient is equal to 0.

Logic Gate

- Logic gate to a computer is like a brick to a skyscraper
- It's a circuit that implements a Boolean function
 - Typically implemented using *diodes* or *transistors*



A AND B

V_IN		V_OUT
A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1



A XOR B

V_IN		V_OUT
A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0



Arithmetic Circuits

- You can build a simple arithmetic circuits using logic gates
- E.g., a 1-bit adder

$$0 + 0 = 00$$

$$1 + 0 = 01$$

$$0 + 1 = 01$$

$$1 + 1 = 10$$

Arithmetic Circuits

- You can build a simple arithmetic circuits using logic gates
- E.g., a 1-bit adder

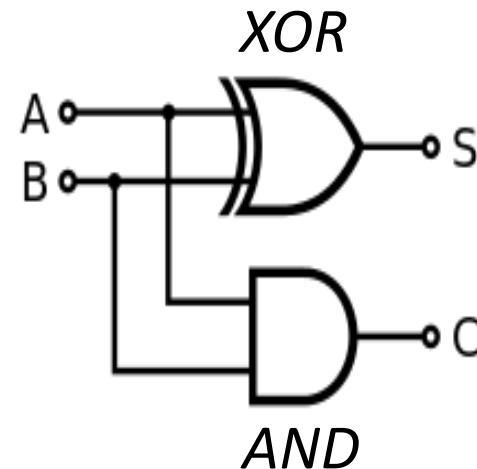
$$0 + 0 = 00$$

$$1 + 0 = 01$$

$$0 + 1 = 01$$

$$1 + 1 = 10$$

Input		Output	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



Arithmetic Circuits

- You can build a simple arithmetic circuits using logic gates
- E.g., a 1-bit adder

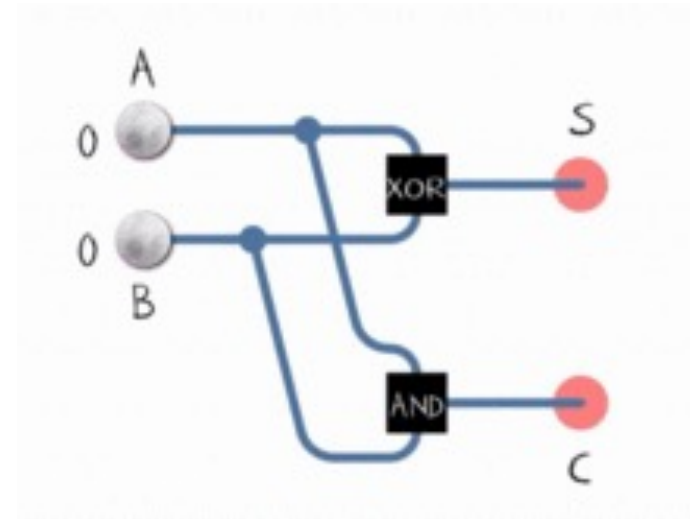
$$0 + 0 = 00$$

$$1 + 0 = 01$$

$$0 + 1 = 01$$

$$1 + 1 = 10$$

Input		Output	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

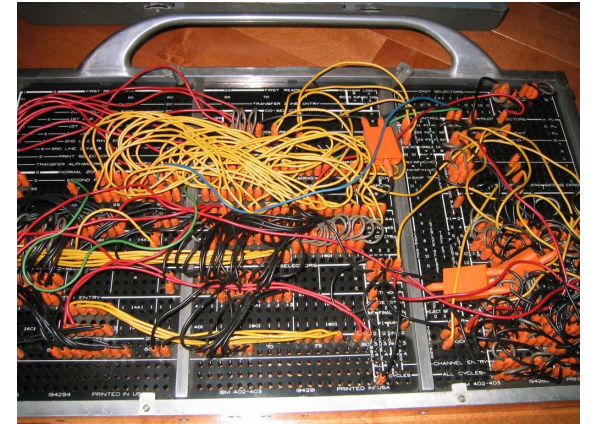


- Implementing more complex arithmetic is similar
 - e.g., multiplier ...

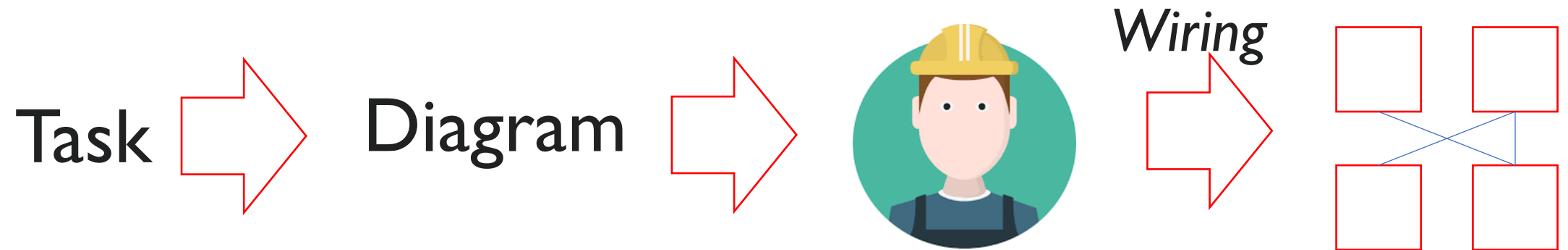
From Arithmetic Circuits to Computer

Computers in 1940's

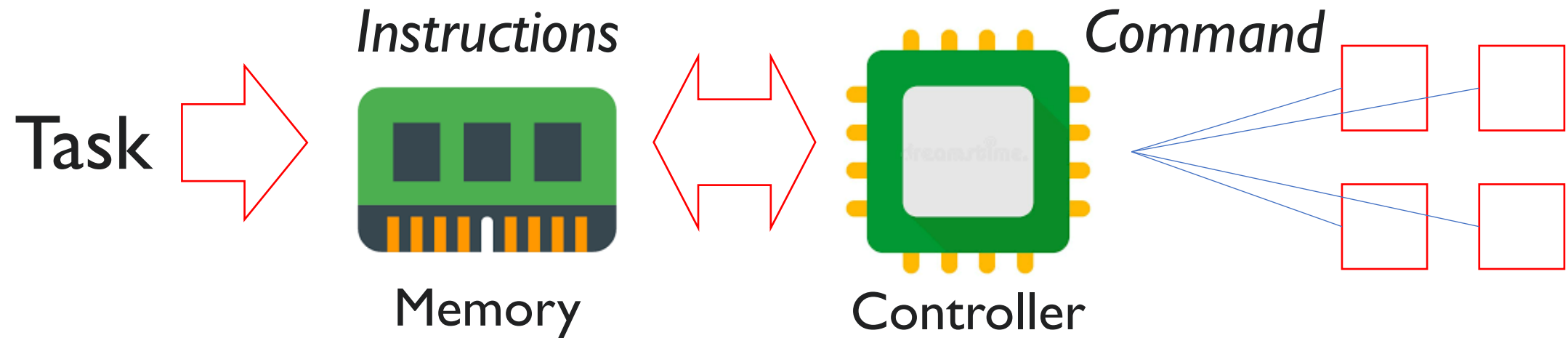
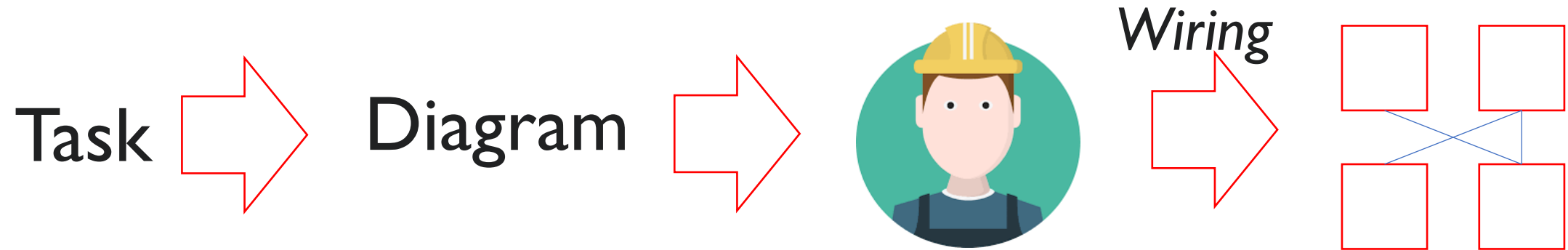
- A large collection of arithmetic machines
- A computer program in 1940's is an interconnection of arithmetic machines, which require significant **wiring**



Stored Program Computer

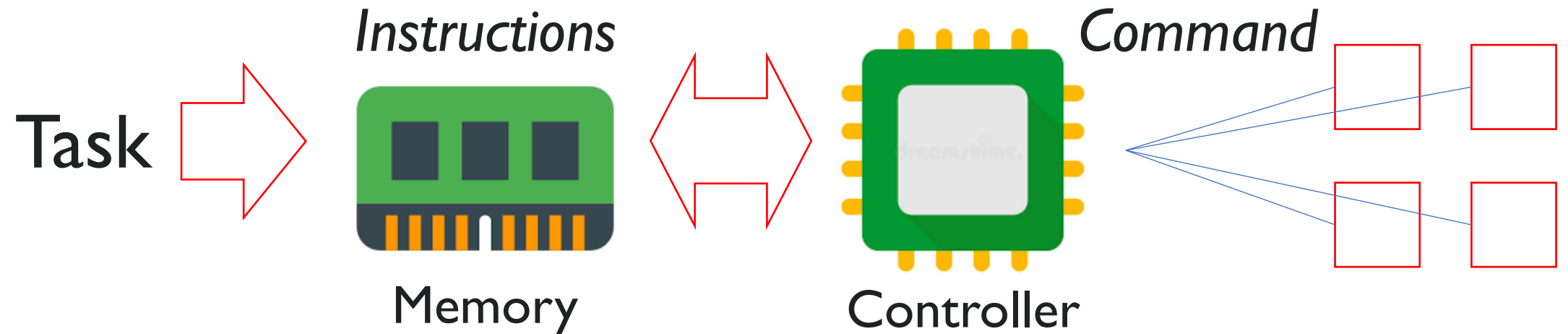


Stored Program Computer



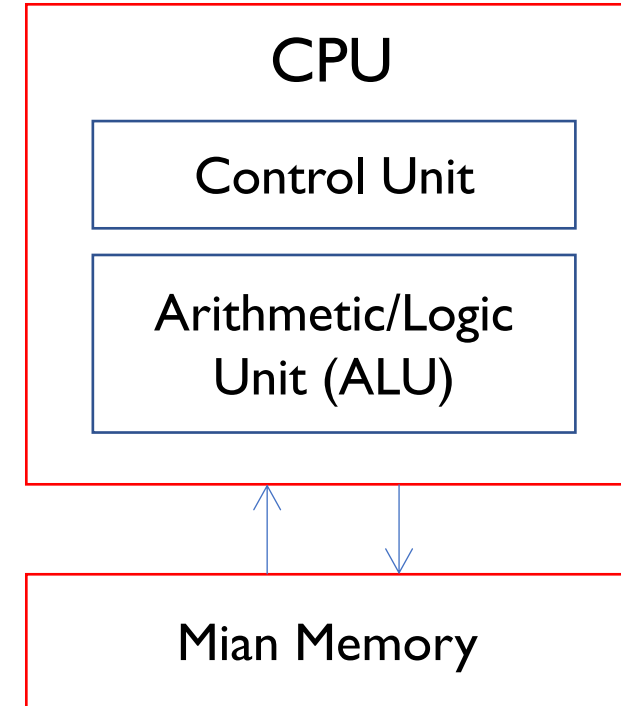
Stored Program Computer

- A.k.a *Von Neumann machine* (proposed in 1945)
- How does it work?



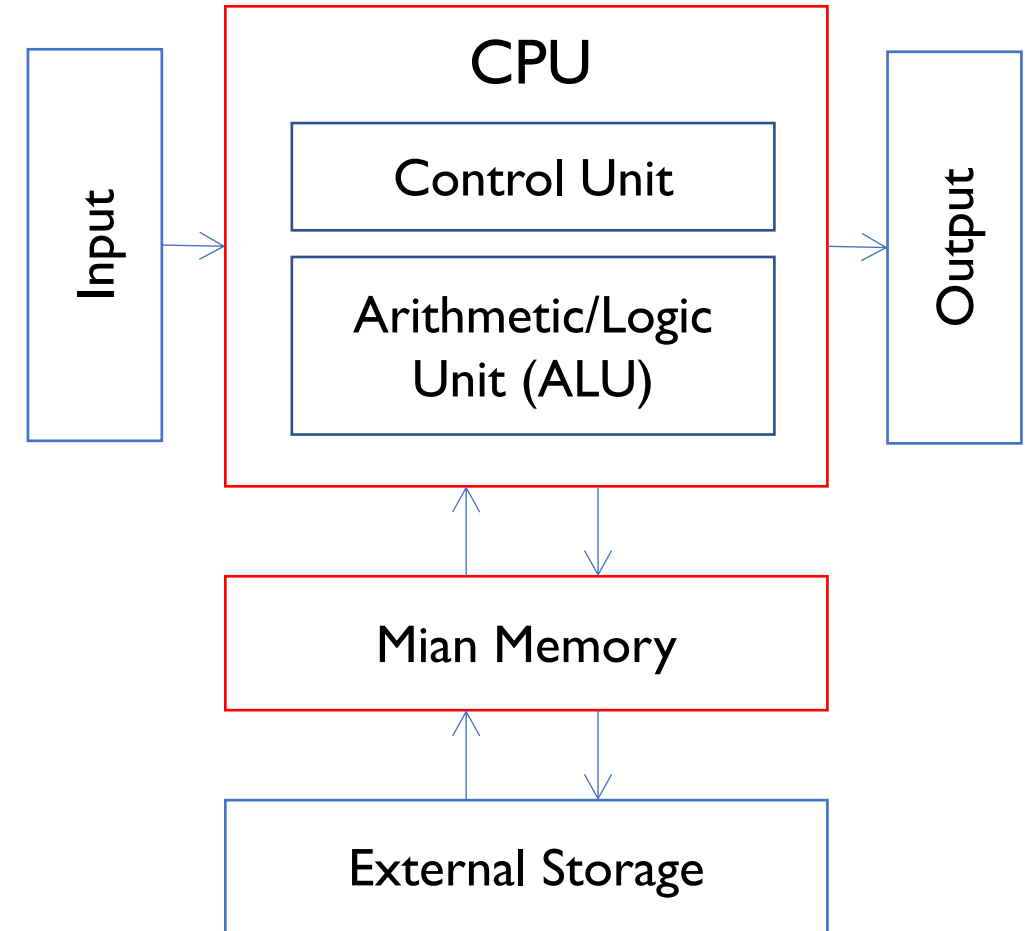
Stored Program Computer

- *Main Memory*: stores both data and program, i.e., a list of instructions
- *CPU (Central Processing Unit)*:
 - *ALU*: performs arithmetic and bitwise operations
 - *Control Unit*: read instructions from memory, direct ALU to execute instructions



Stored Program Computer

- *Main Memory*: stores both data and program, i.e., a list of instructions
- *CPU (Central Processing Unit)*:
 - *ALU*: performs arithmetic and bitwise operations
 - *Control Unit*: read instructions from memory, direct ALU to execute instructions
- *External storage*: (slow) mass storage
- *Input/output*: keyboard, microphone, display, speaker...



Instruction

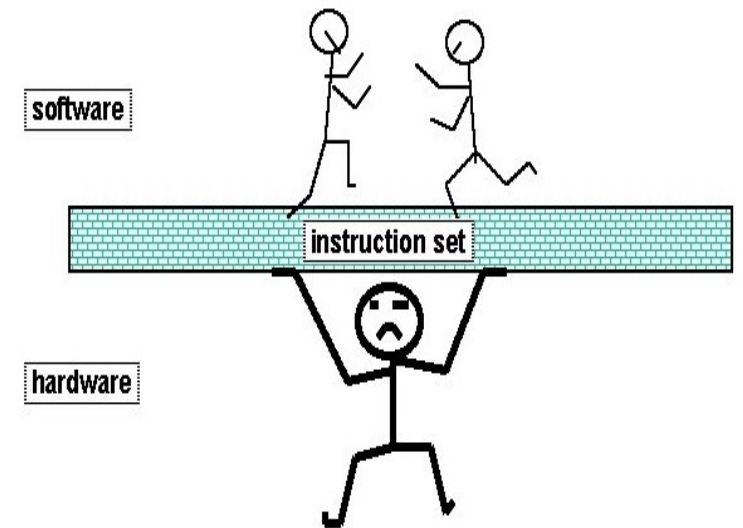
- A sequence of bits that defines a single operation of a CPU
 - E.g., addition, subtraction, read, write
 - Instruction size can be fixed (e.g., 16-bit, 32-bit, 64-bit) or variable

Example: a 32-bit Add Immediate instruction



Instruction Set Architecture (ISA)

- ISA: the specification of instructions supported by a CPU
 - The dictionary of CPU language
 - E.g., x86, RISC ...
- ISA is the interface between hardware and software
 - Different CPUs can run the same computer program as long as they support the same ISA
 - Example: 8086 (1978) and Skylake (2015) both implement x86 ISA, so they can run the same software



Why is Software Programming Important?

- It is almost impossible to run an electronic device without software
 - Printers, MRI machines, disk drives, remote controls, vehicle control system, etc.
- More cost effective to implement functionality in software than hardware
- Software bugs easy to fix
 - Systems increasingly complex, bugs unavoidable
- Allows new features to be added later



Outline for Today

- What's a computer
- Programming languages
- Basics of computer programming

Programming Languages

- Computer program
 - A sequence of instructions for a computer to execute
- Programming language
 - Notation for writing computer programs

Programming Languages



Machine Language

Language directly understood by the computer.

Defined by ISA

x86, RISC ...

PROGRAM I-I The Multiplication Program in Machine Language

1		00000000	00000100	000000000000000000
2	01011110	00001100	11000010	000000000000000010
3		11101111	00010110	000000000000000101
4		11101111	10011110	000000000000001011
5	11111000	10101101	11011111	00000000000010010
6		01100010	11011111	00000000000010101
7	11101111	00000010	11111011	00000000000010111
8	11110100	10101101	11011111	00000000000011110
9	00000011	10100010	11011111	00000000000100001
10	11101111	00000010	11111011	00000000000100100
11	01111110	11110100	10101101	
12	11111000	10101110	11000101	00000000000101011
13	00000110	10100010	11111011	00000000000110001
14	11101111	00000010	11111011	00000000000110100
15		01010000	11010100	00000000000111011
16			00000100	00000000000111101

Cons?

Pros?

Programming Languages



Machine Language

Language directly understood by the computer

Defined by ISA

x86, RISC ...

Symbolic Language

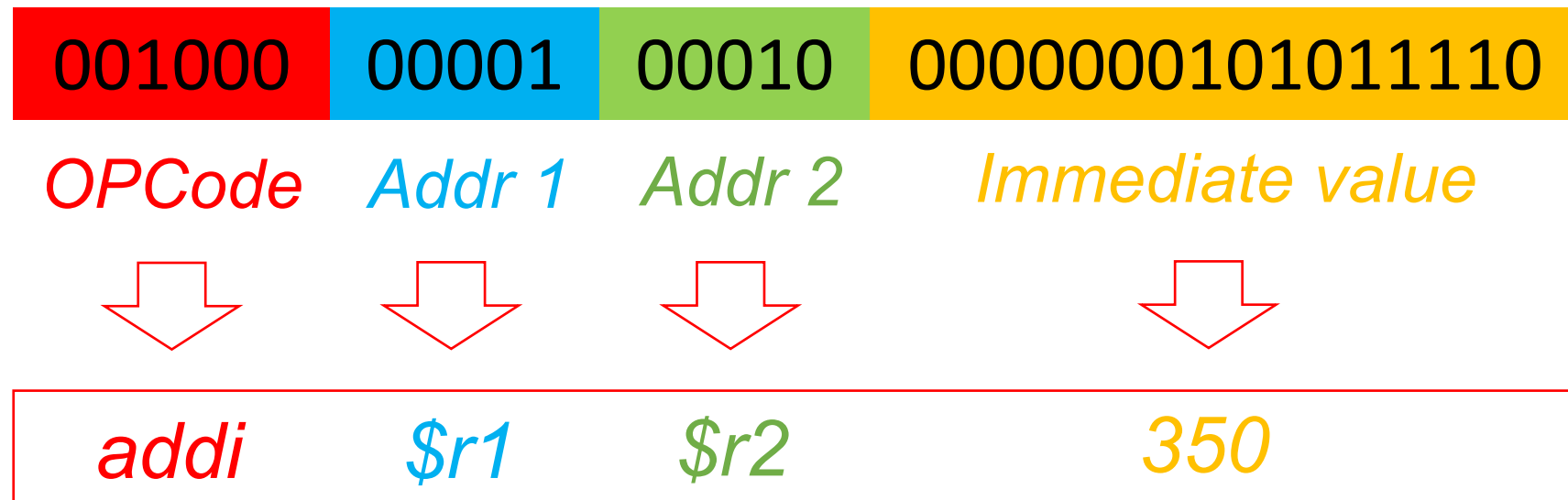
English-like *abbreviations* representing elementary computer operations

Assembly language

Programming Languages

- *Symbolic language* uses symbols to represent the various machine language instructions

Example: a 32-bit Add Immediate instruction



PROGRAM I-2 The Multiplication Program in Symbolic Language

```
1      entry   main, ^m<r2>
2      subl2   #12, sp
3      jsb     C$MAIN_ARGS
4      movab   $CHAR_STRING_CON
5
6      pushal   -8(fp)
7      pushal   (r2)
8      calls   #2, SCANF
9      pushal   -12(fp)
10     pushal   3(r2)
11     calls   #2, SCANF
12     mull3    -8(fp), -12(fp), -
13     pusha    6(r2)
14     calls   #2, PRINTF
15     clrl    r0
16     ret
```

Programming Languages



Machine Language

Language directly understood by the computer

Defined by ISA

x86, RISC ...

Symbolic Language

English-like *abbreviations* representing elementary computer operations

Assembly language

High-level Language

Close to human language.

Example: $a = a + b$

[add values of a and b , and store the result in a , replacing the previous value]

C/C++, Java, Python

PROGRAM I-3 The Multiplication Program in C

```
1  /* This program reads two integers from the keyboard
2     and prints their product.
3     Written by:
4     Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Definitions
11     int number1;
12     int number2;
13     int result;
14
15 // Statements
16     scanf ("%d", &number1);
17     scanf ("%d", &number2);
18     result = number1 * number2;
19     printf ("%d", result);
20     return 0;
21 } // main
```

Compiler

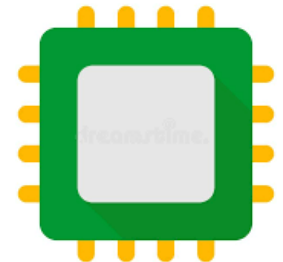
- Computers only understand machine language (binary code)
- Human write programs using high-level programming language



```
1  /* This program reads two integers from the keyboard
2     and prints their product.
3     Written by:
4     Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Definitions
11     int number1;
12     int number2;
13     int result;
14
15 // Statements
16     scanf ("%d", &number1);
17     scanf ("%d", &number2);
18     result = number1 * number2;
19     printf ("%d", result);
20     return 0;
21 } // main
```



```
1      00000000 00000100 0000000000000000
2 01011110 00001100 11000010 0000000000000010
3      11101111 00010110 00000000000000101
4      11101111 10011110 00000000000001011
5 11111000 10101101 11011111 0000000000010010
6      01100010 11011111 00000000000010101
7 11101111 00000010 11111011 0000000000010111
8 11110100 10101101 11011111 0000000000011110
9 00000011 10100010 11011111 0000000000100001
10 11101111 00000010 11111011 0000000000100100
11 01111110 11110100 10101101
12 11111000 10101110 11000101 0000000000101011
13 00000110 10100010 11111011 0000000000110001
14 11101111 00000010 11111011 0000000000110100
15      01010000 11010100 0000000000111011
16      00000100 0000000000111101
```



Compiler

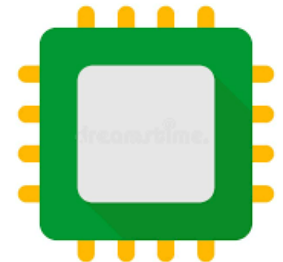
- Computers only understand machine language (binary code)
- Human write programs using high-level programming language
- Need a *compiler* to translate programs written in high-level programming language to binary code



```
1  /* This program reads two integers from the keyboard
2     and prints their product.
3     Written by:
4     Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10     // Local Definitions
11     int number1;
12     int number2;
13     int result;
14
15     // Statements
16     scanf ("%d", &number1);
17     scanf ("%d", &number2);
18     result = number1 * number2;
19     printf ("%d", result);
20     return 0;
21 } // main
```

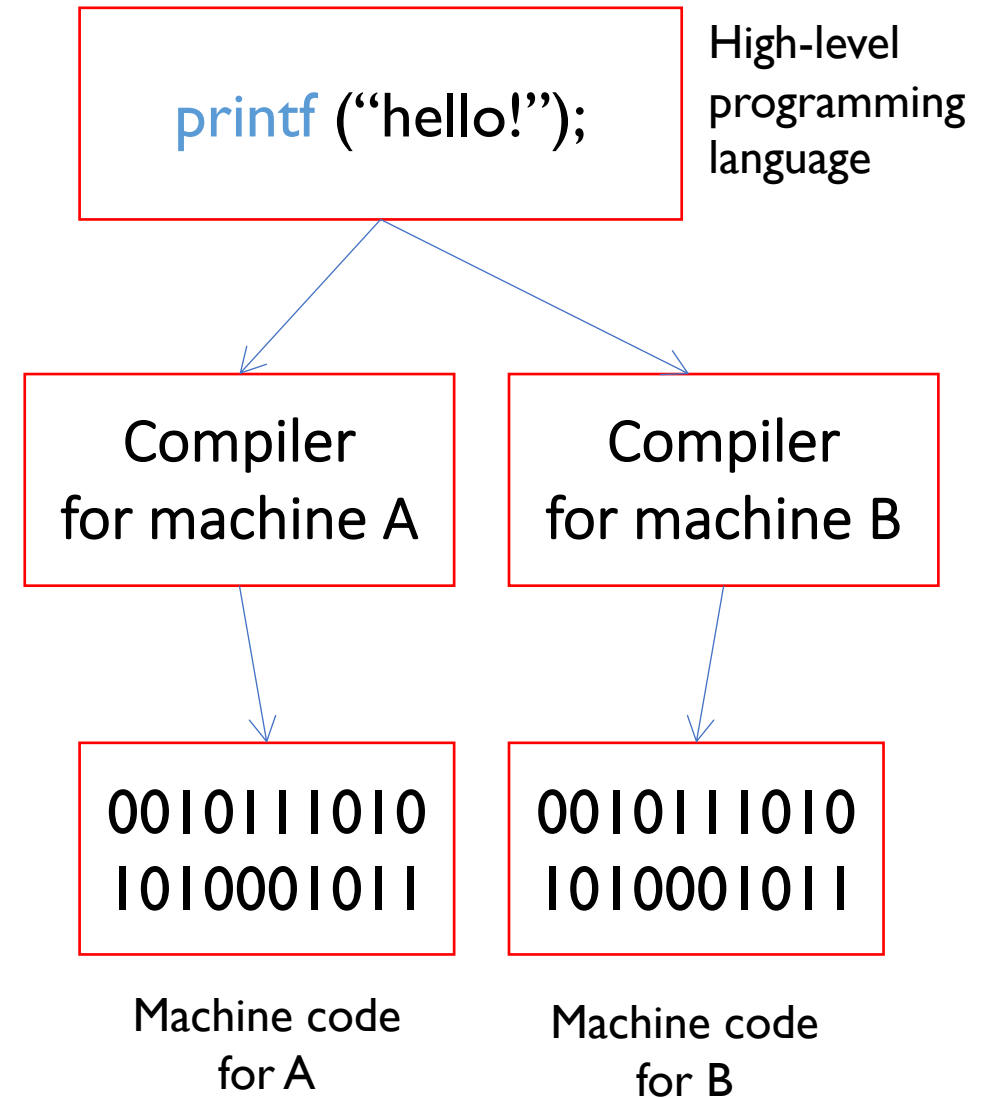


```
1      00000000 00000100 0000000000000000
2 01011110 00001100 11000010 0000000000000010
3      11101111 00010110 00000000000000101
4      11101111 10011110 00000000000001011
5 11111000 10101101 11011111 0000000000010010
6      01100010 11011111 00000000000010101
7 11101111 00000010 11111011 0000000000010111
8 11110100 10101101 11011111 0000000000011110
9 00000011 10100010 11011111 0000000000100001
10 11101111 00000010 11111011 0000000000100100
11 01111110 11110100 10101101
12 11111000 10101110 11000101 0000000000101011
13 00000110 10100010 11111011 0000000000110001
14 11101111 00000010 11111011 0000000000110100
15      01010000 11010100 0000000000111011
16      00000100 0000000000111101
```



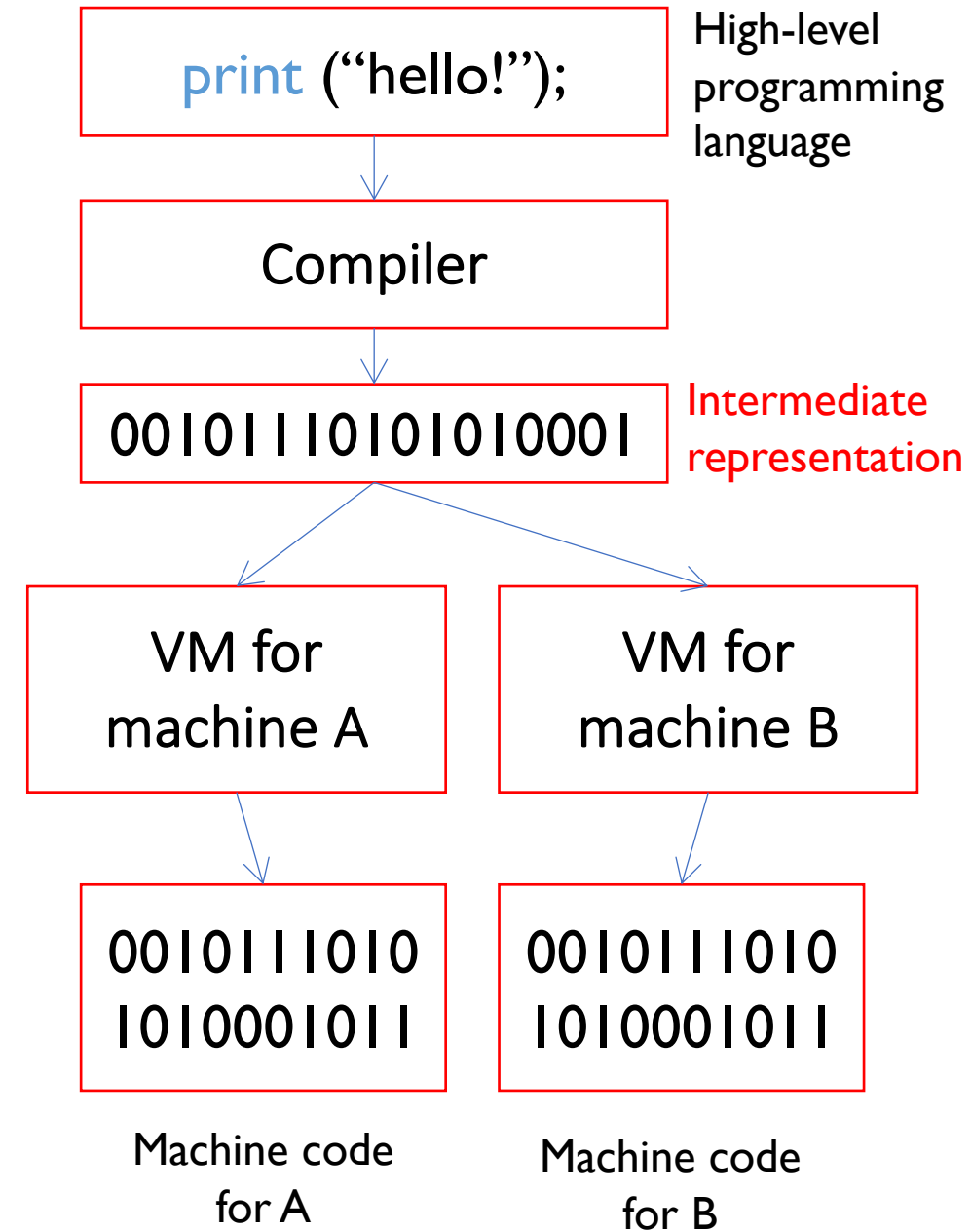
Compiled Languages

- Programs are compiled to binary machine code (for a specific hardware), which is then *directly executed* by the hardware
- E.g., C/C++, Rust, Pascal ...
- *Cons*: need to *re-compile* a program before you execute it on a different hardware
- *Pros*: fast



Interpreted Languages

- Programs are first compiled to an *intermediate representation (IR)*, which is then converted by a *Virtual Machine (VM)* to machine code and executed on the hardware
- E.g., Java, Python...
- *Pros*: better portability
- *Cons*: slow



Different Programming Languages

ActionScript Ada **ASP.NET** Assembler Basic
c **C++** **C#** Cobol Cobra CODE ColdFusion
Delphi Eiffel Fortran FoxPro GPSS **HTML** J#
J++ **Java** JavaScript **JSP** LISP Logo LUA
MEL Modula-2 Miranda Objective-C **Perl**
PHP Prolog **Python** **SQL** Visual Basic
Visual Basic.NET VBA Visual-FoxPro

Syntax

- Programming languages differ in *syntax* and *libraries*

C/C++

```
if (a < b) {  
    printf("a is smaller than b\n");  
} else {  
    printf("b is not greater than a\n");  
}
```

Python

```
if a < b:  
    print("a is smaller than b")  
else:  
    print("b is not greater than a")
```

Syntax

- Programming languages differ in *syntax* and *libraries*
- Syntax is **well-defined, NO** exceptions
 - `if (...) {...} else {...}`
 - `for (;;;) {...}`
- Basic components
 - Variable/structure/function declaration
 - Variable/structure/function access
 - Conditional statement
 - Iteration statement



Libraries

- Programming languages differ in *syntax* and *libraries*
- Most programming languages have an associated core library
- Libraries typically include definitions for
 - Commonly used algorithms (e.g., sorting)
 - Data structures (e.g., lists, trees, and hash tables)
 - Commonly used constants and functions (e.g., math)
 - Input/output

```
if (a < b) {  
    printf ("a is smaller than b\n");  
} else {  
    printf ("b is not greater than a\n");  
}  
printf("%f\n", cos(M_PI));
```

For CS2311: C/C++

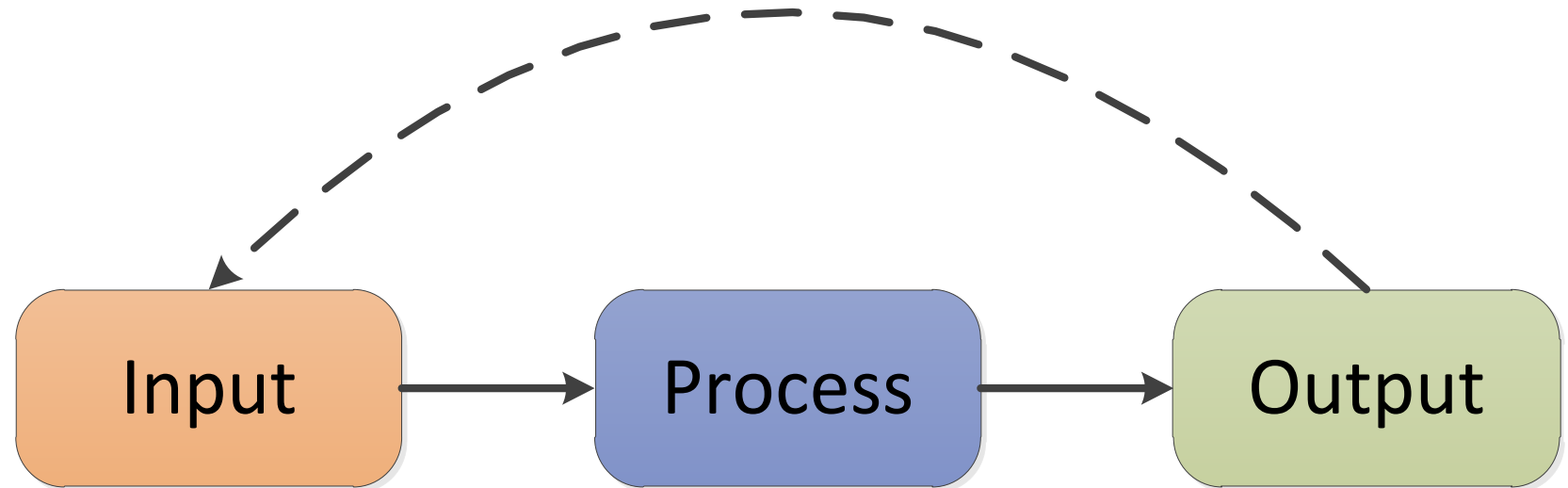
- Created in 1970s by Dennis Ritchie in Bell Labs
- Found lasting use in
 - Supercomputers, microcontrollers, embedded systems
 - Operating systems, device drivers, network stacks
- Many programming languages are based on/influenced by C/C++
 - Go, Java, JavaScript ...
 - Easy to move from C++ to other languages
 - But often not in the other direction
- *Cons*: requires explicit low-level manipulation
- *Pros*: very efficient

Outline for Today

- What's a computer
- Programming languages
- Basics of computer programming

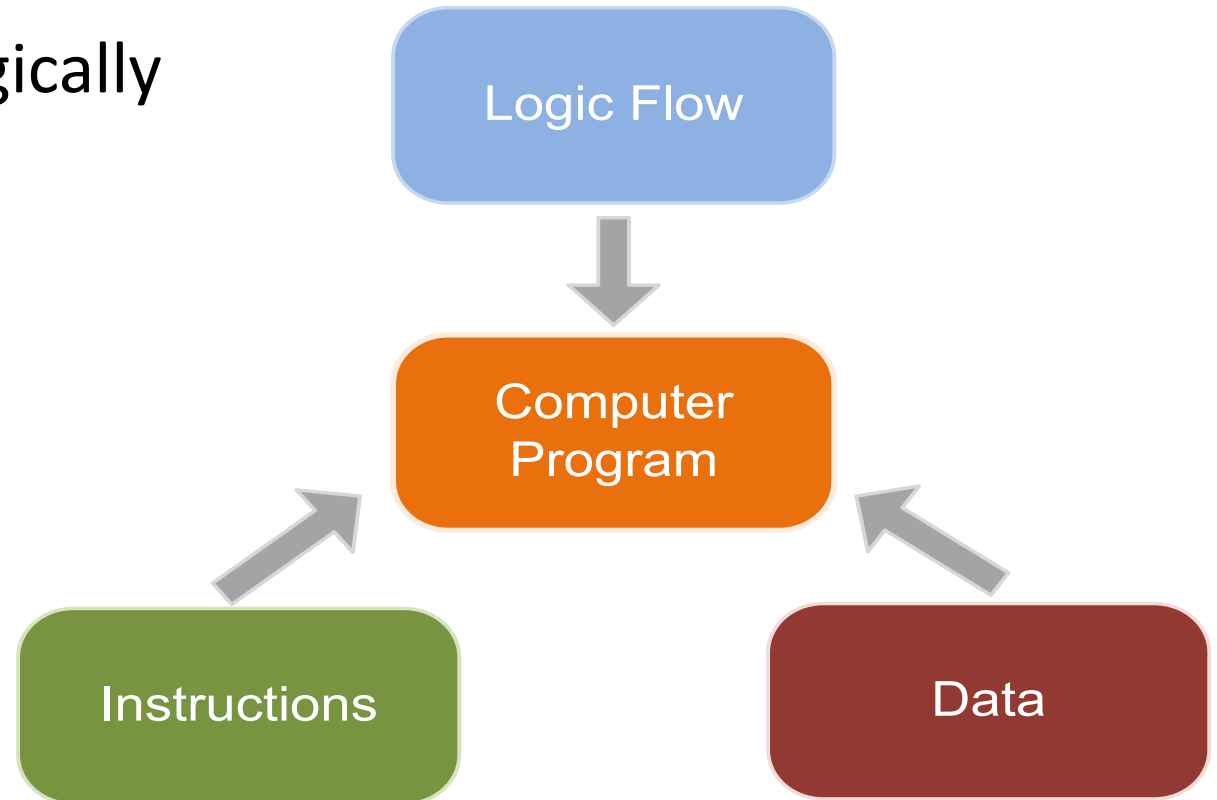
Computer Program (External View)

- Basic elements of a computer program
 - Input
 - Process
 - Output



Computer Program (Internal View)

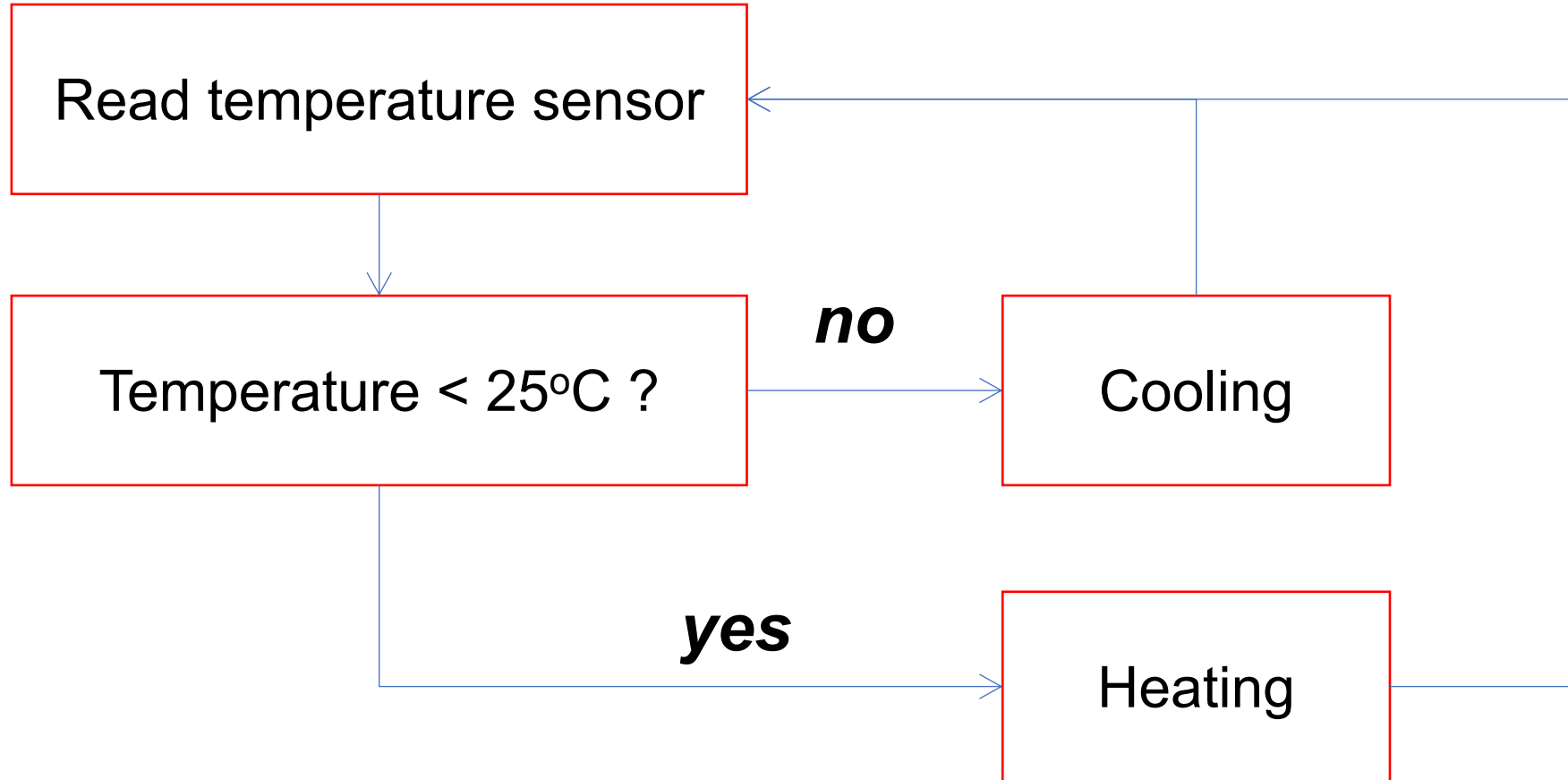
- A list of instructions ordered logically
- Usually involve data access



Logic Flow

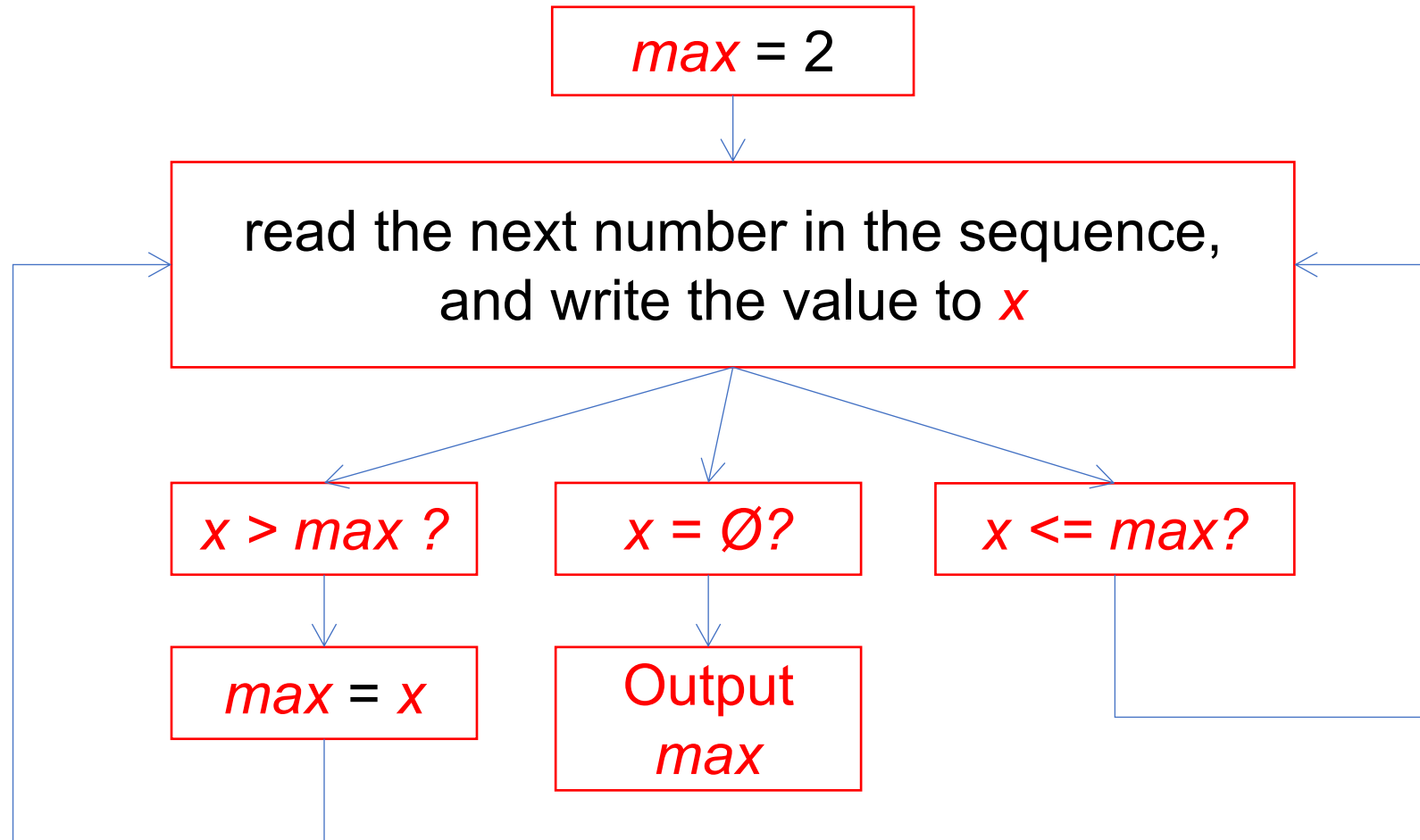
- The logic of problem solving
 - You may implement the same logic flow using different languages
- E.g. Calculate BMI (Body Mass Index)
 1. Read weight from keyboard
 2. Read height from keyboard
 3. $\text{Weight} \times \text{weight} / \text{height}$
 4. Write BMI to screen

Logic Flow



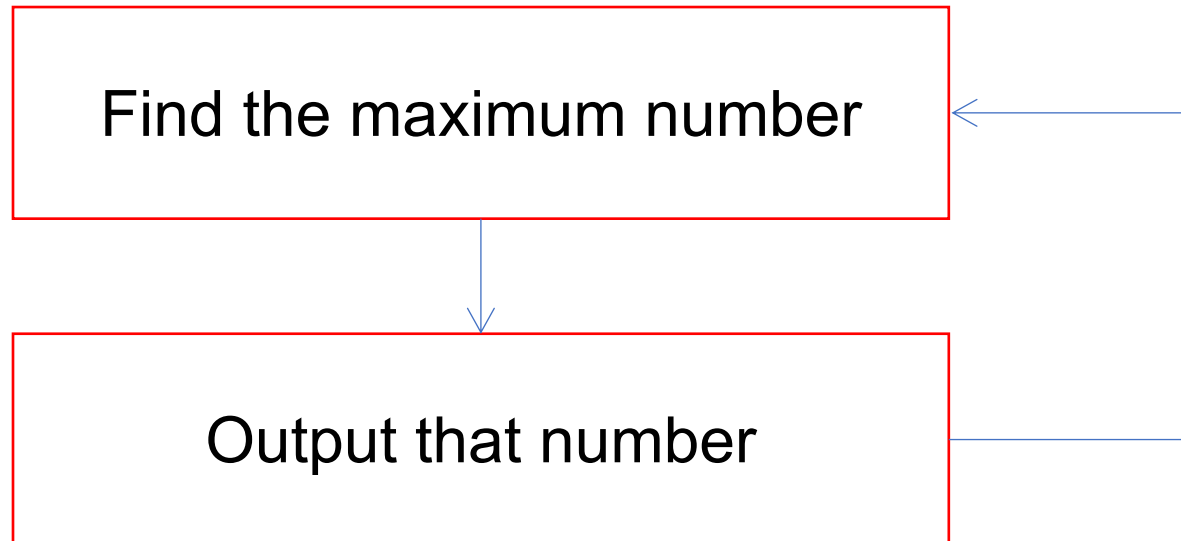
Logic Flow

- Find the maximum number in 2, 1, 3, \emptyset



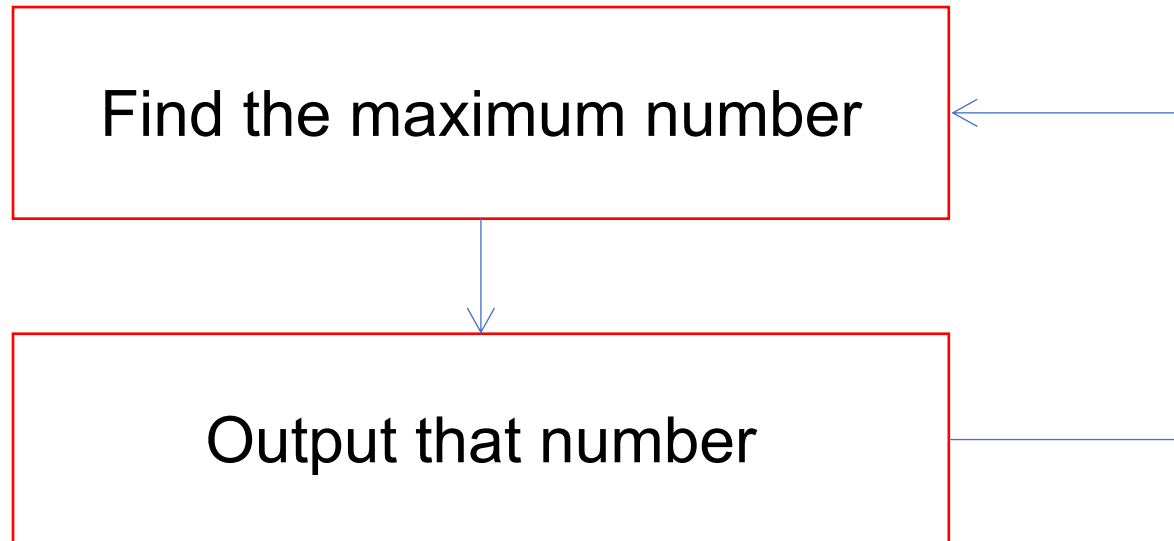
Logic Flow

- Sort 8, 5, 2, 9, 4 in descending order
8, 5, 2, 9, 4



Logic Flow

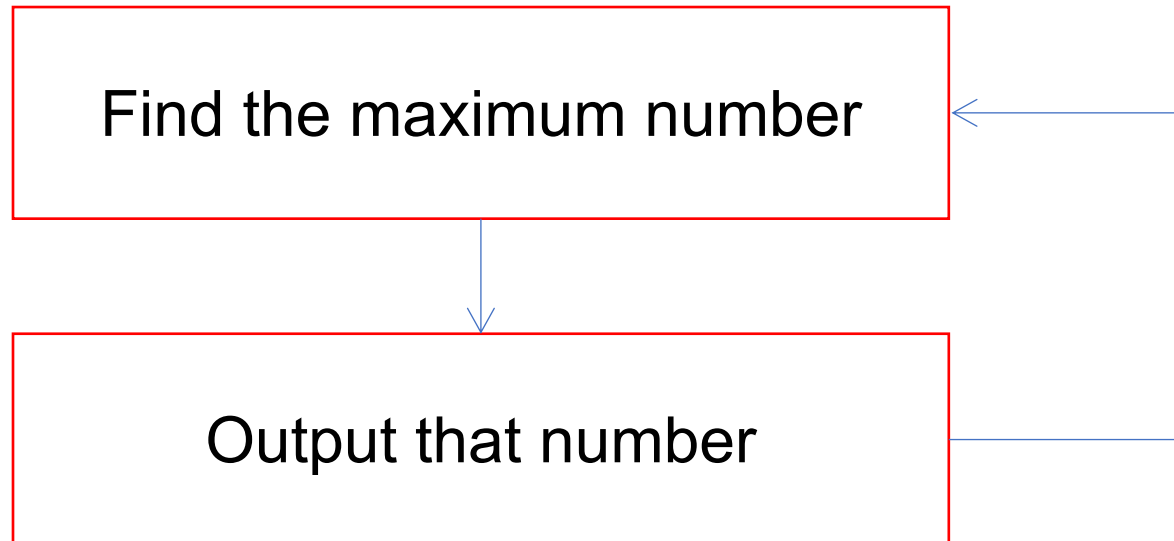
- Sort 8, 5, 2, 9, 4 in descending order
8, 5, 2, 9, 4



9,

Logic Flow

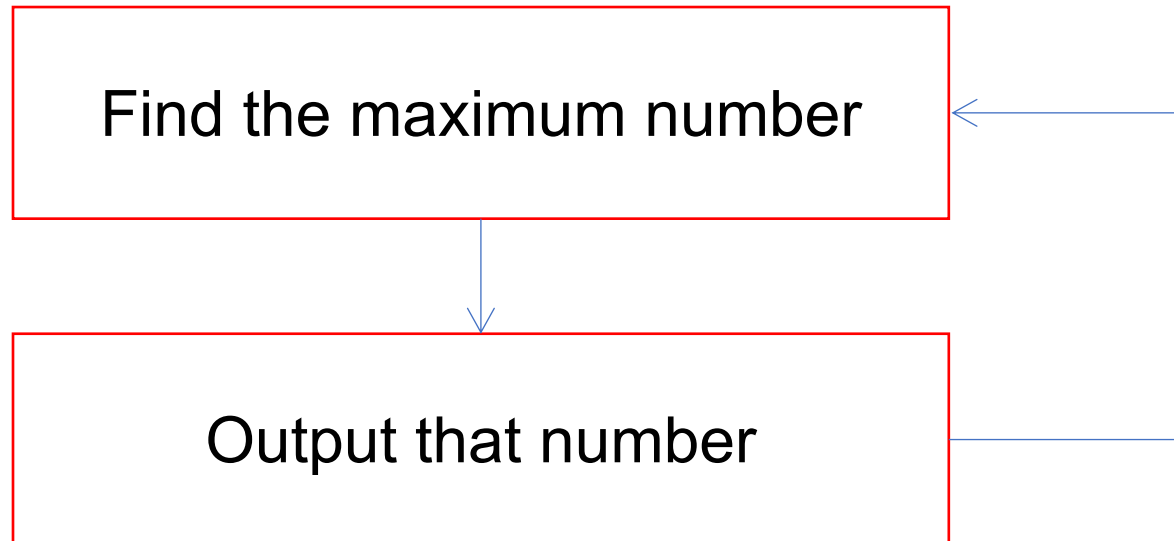
- Sort 8, 5, 2, 9, 4 in descending order
8, 5, 2, 4



9, 8,

Logic Flow

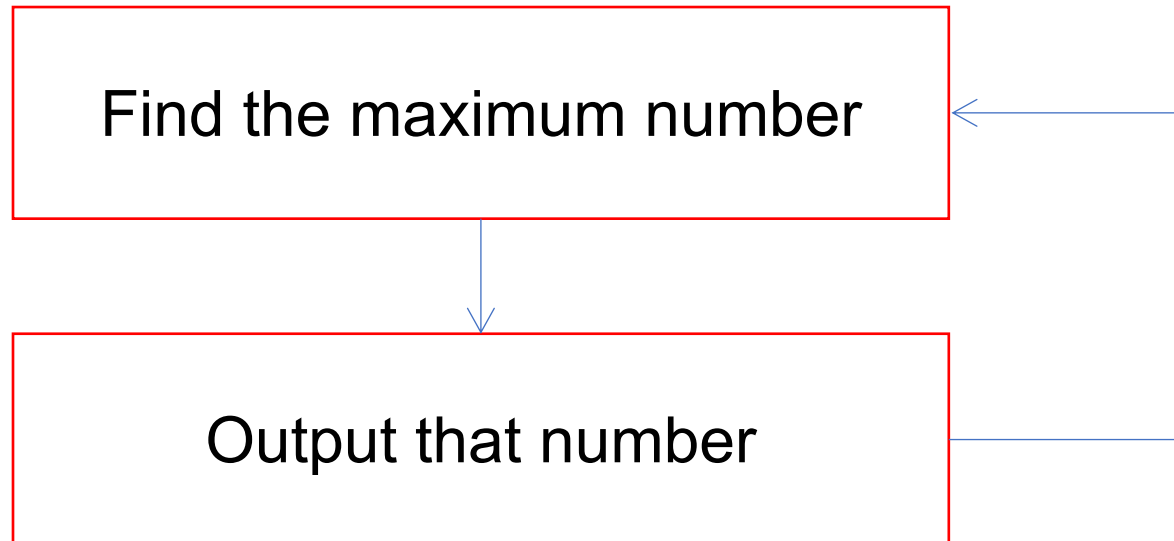
- Sort 8, 5, 2, 9, 4 in descending order
~~5~~, 2, 4



9, 8, 5,

Logic Flow

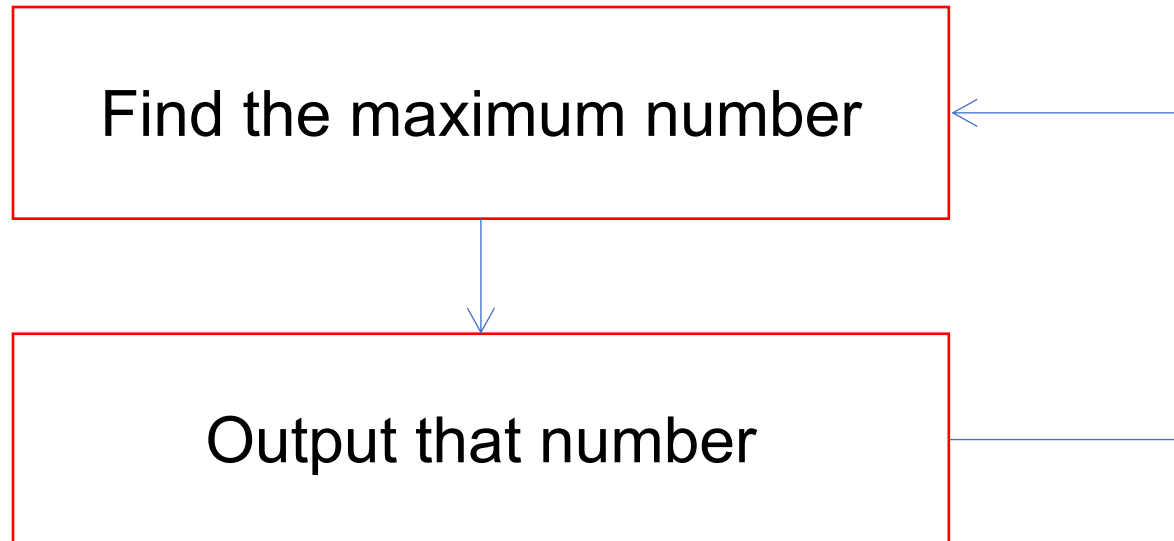
- Sort 8, 5, 2, 9, 4 in descending order
2, 4



9, 8, 5, 4,

Logic Flow

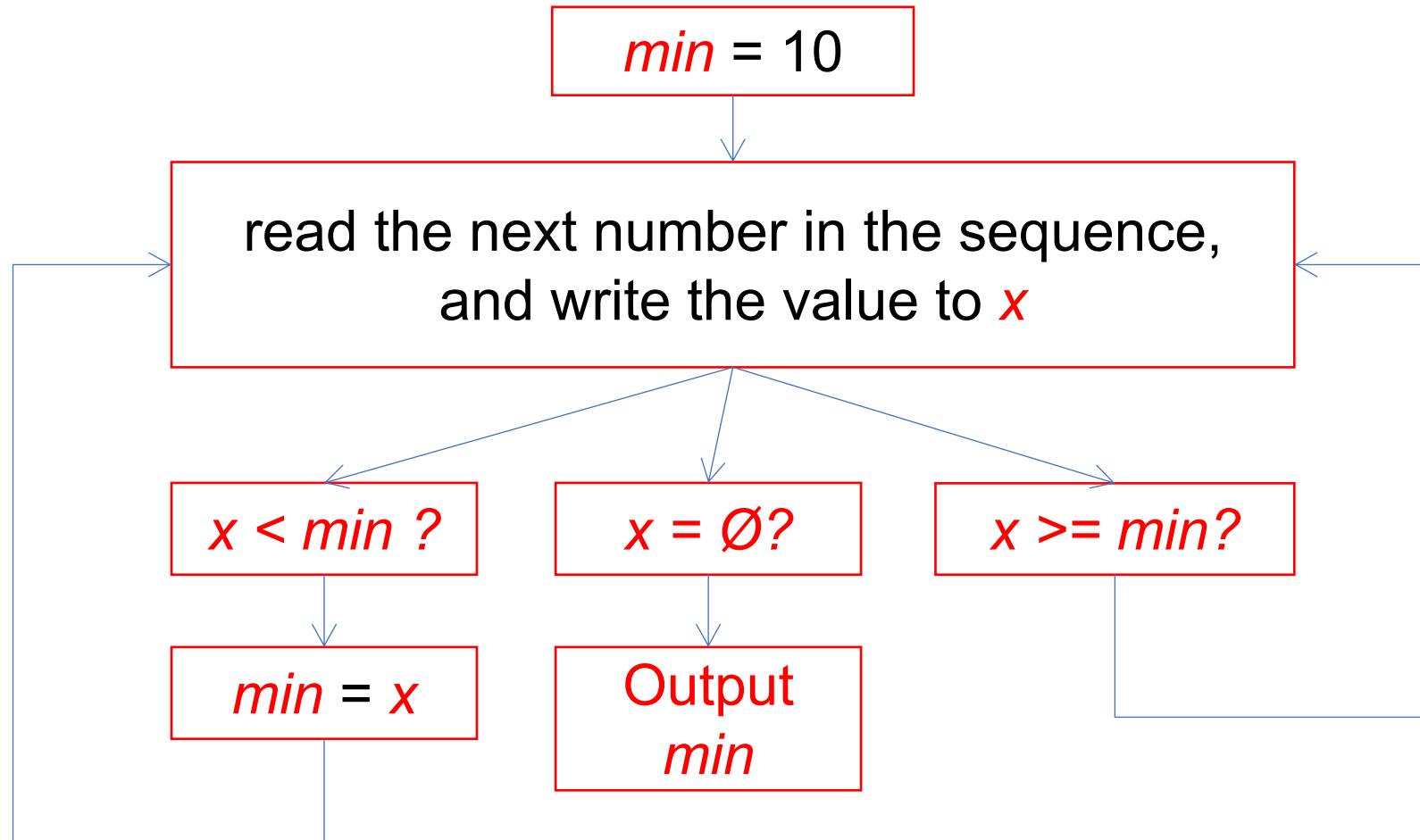
- Sort 8, 5, 2, 9, 4 in descending order
2,



9, 8, 5, 4, 2

Logic Flow - Exercise

- Show the logic flow of finding the minimum number in **10, 7, 6, 1, \emptyset**



Developing a C++ Program

1. Coding

- Write your *source code* into a file
 - e.g., “*hello.cpp*”
- You can use different *editors* or *IDEs*

Editors and IDEs

- Editor: simply where you write your codes
 - Vim, Emacs, nano, vi, or even notepad ...
- IDE: integrated development environment
 - Integrate compiler, build tools, and debuggers
 - With syntax highlighting
 - Popular IDEs
 - **Visual Studio**, Visual Studio Code, JetBrains Clion, Apple Xcode ...



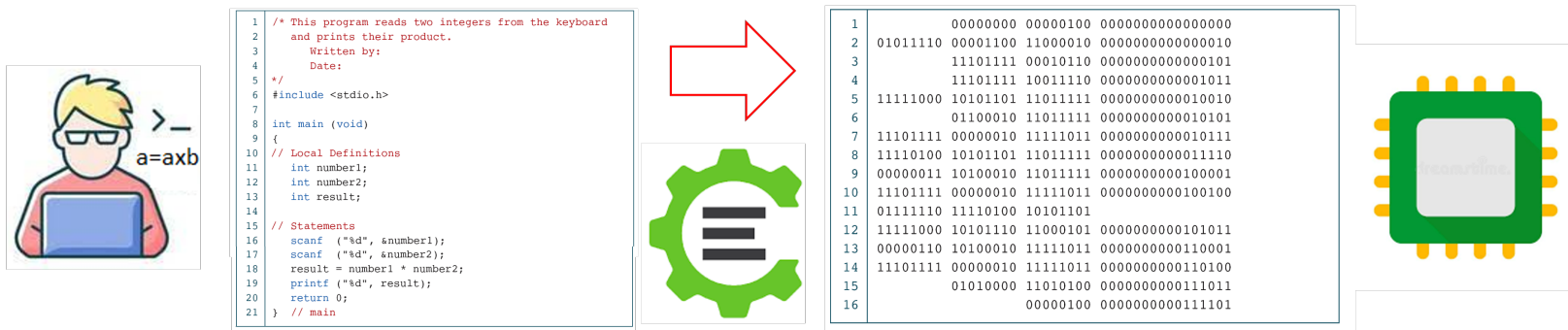
Developing a C++ Program

1. Coding

- Write your *source code* into a file
 - e.g., “*hello.cpp*”
- You can use different *editors* or *IDEs*

2. Compile your source code

- Check grammatical rules (syntax)
- Source code is converted to *object code* in machine language
 - e.g., “*hello.obj*”



C++ Compilers

- **MSVC (Microsoft Visual C++)**
 - Microsoft's compiler for their custom implementation of the C++ standard, known as Visual C++
- **GCC/g++**
 - Mainly targets Unix-like platforms
 - Windows support is provided through the Cygwin or MinGW runtime libraries
- **Clang**
 - Strict adherence to C++ standards
 - Minimal modification to source code's structure during compilation
 - GCC-compatible or MSVC-compatible through compiler drivers

Developing a C++ Program

1. Coding

- Write your *source code* into a file
 - e.g., “*hello.cpp*”
- You can use different *editors* or *IDEs*

2. Compile your source code

- Check grammatical rules (syntax)
- Source code is converted to *object code* in machine language
 - e.g., “*hello.obj*”

3. Link

- Combines objects and libraries to create a *binary executable*
 - e.g., “*hello.exe*”

An Example

```
/* The traditional first program in honor of Dennis Ritchie who
   invented C at Bell Labs in 1972 */
#include <iostream>
using namespace std;
void main()
{
    cout << "Hello world!" << endl;
}
```

Function

- A sequence of instructions grouped together (contained within braces { and }), which implement a specific task

Syntax:

```
ReturnType FunctionName (input parameters)
{
    instructions within function body
}
```

```
/* The traditional first program in
   honor of Dennis Ritchie who
   invented C at Bell Labs in 1972 */
```

```
#include <iostream>
using namespace std;
```

```
void main()
{
    cout << "Hello world!" << endl;
}
```

Example: void main()

- **main** is the name of the function
 - main is a special function
 - The starting point of a C++ program. Every C++ program must have a main
 - Note: C/C++ is case sensitive
 - E.g., Main() or MAIN() is incorrect
- **void** means there is *NO return value*

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */
```

```
#include <iostream>  
using namespace std;
```

```
void main()  
{  
    cout << "Hello world!" << endl;  
}
```

Statement

- A syntactic unit that expresses some action to be carried out
- Ended with a semicolon “;”

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */  
  
#include <iostream>  
using namespace std;  
  
void main()  
{  
    cout << "Hello world!" << endl;  
}
```

Example: cout

- An output function provided by **iostream** library
- **cout**: “Console OUTput” allows our program to output values to the standard output stream (the screen)
- **<<**: output operator, which output values to an output device
- The right-hand side of the **<<** (i.e., **Hello world!** between a pair of double quotes) is the string to output
- **endl**: end of the line. advance the cursor on the screen to the beginning of the next line

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */  
  
#include <iostream>  
using namespace std;  
  
void main()  
{  
    cout << "Hello world!" << endl;  
}
```

#include

- Include the libraries you want to use

Syntax: **#include** <library name>

- For example
 - **iostream** has implemented commonly used functions for input/output, including `cout`
 - **cmath** is another commonly used library that has math functions like `sin()`, `cos()`, `log()` ...

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */
```

```
#include <iostream>
```

```
using namespace std;
```

```
void main()  
{  
    cout << "Hello world!" << endl;  
}
```


using namespace

- **namespace**: a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it

- Declare namespace to avoid writing the full name

Syntax: **using namespace** xxx

- For example
 - **Standard (std)** namespace is used such that we can write **cout** instead of **std::cout**

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */
```

```
#include <iostream>
```

```
using namespace std;
```

```
void main()
```

```
{  
    cout << "Hello world!" << endl;  
}
```

Comments

- Enclosed by “/*” and “*/”
- Or begin with “//”
 - Single line comments
- Comments improves readability of source code
- Will *NOT* be compiled into machine code

```
/* The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */
```

```
#include <iostream>  
using namespace std;  
  
void main()  
{  
    cout << "Hello world!" << endl;  
}
```

Syntax errors

```
/*The traditional first program in  
honor of Dennis Ritchie who  
invented C at Bell Labs in 1972 */
```

```
#include <iostream>
```

```
using namespace std;
```

```
void main()
```

```
{
```

```
cout < Hello world! < endl
```

```
}
```

The texts to output should be placed in a pair of double quotes "texts".

< is not an operator of cout. We need to use <<

We need ; at the end of each statement

Another Example

Any errors?

Guess the functionality?

```
#include <iostream>
#include <cmath>
using namespace std;

double cosofsum(double x, double y) {
    double sum = x+y;
    // cos() is a cosine function from cmath
    double result = cos(sum);
    return result;
}

int main() {
    double a = 0.2;
    // M_PI is defined in cmath, ~= 3.14
    cout << cosofsum(a, M_PI) << endl;
    return 0;
}
```

Outline for Today

- What's a computer
 - Numbering system
 - Logic gates and circuits
 - Stored Program Computer and ISA
- Programming languages
 - Machine/ Symbolic/High-level Language
 - Compiler
- Basics of computer programming
 - Logic Flow
 - Developing: Coding / Compilation/ Linking
 - C++ Syntax: Function/Statement/...