Procedural Programming

Theory Week 1

Maastricht University | Department of Advanced Computing Sciences

Learning objectives

- You know the four components of the Von Neumann Architecture
- You understand the execution cycle on the Von Neumann Architecture
- You understand the binary representation
- You know the basics of assembler
- You know the difference between assembly and high-level programming languages
- You know the difference between compilation vs interpreted code

Let's take it from the start

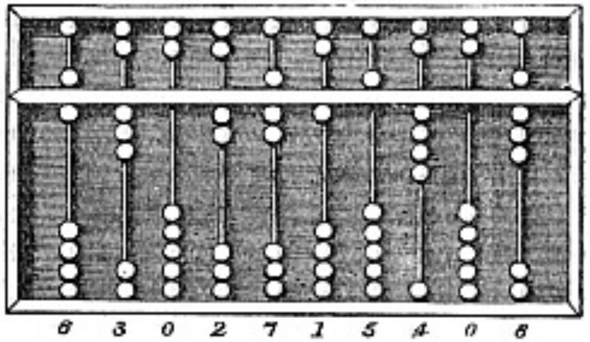
- What is a computer?

- "A device that computes...
  especially a programmable electronic machine that performs high-speed mathematical or logical operations or that assembles, stores, correlates, or otherwise processes information"
  - *From American Heritage® Dictionary of the English Language, 4th Edition*

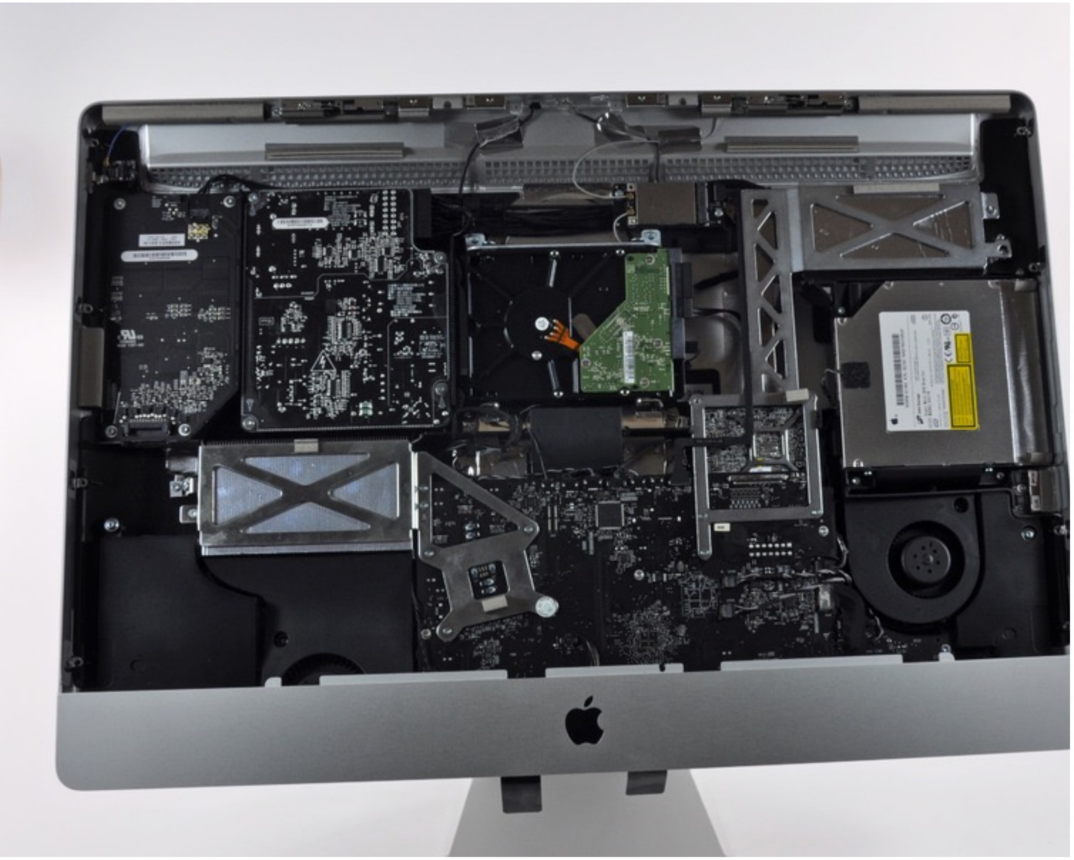
The first computers

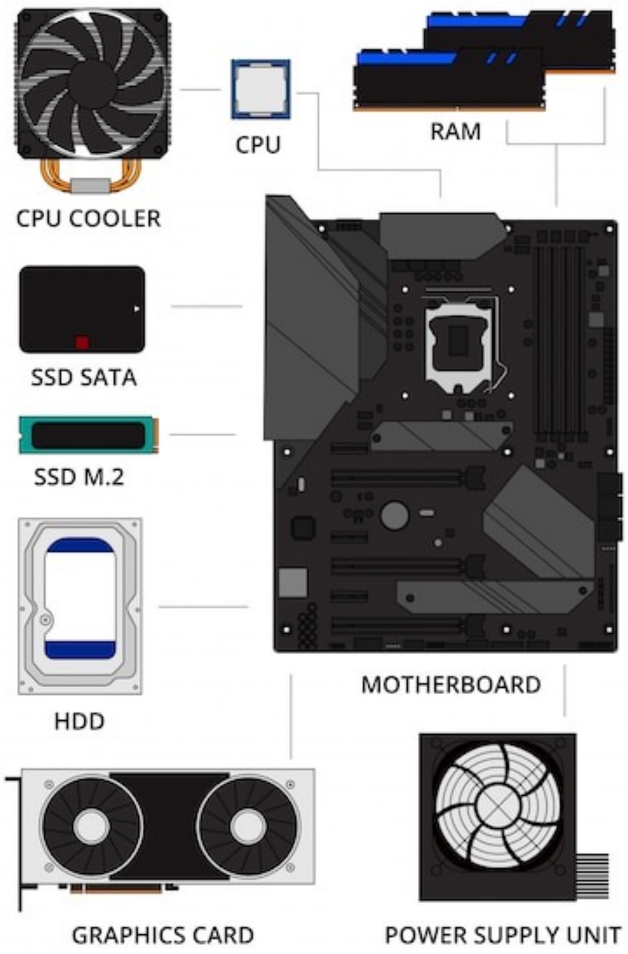
- Scales – computed relative weight of two items
  - Computed if the first item’s weight was less than, equal to, or greater than the second item’s weight

- Abacus – performed mathematical computations
  - Primarily thought of as Chinese, but also Japanese, Mayan, Russian, and Roman versions
  - Can do square roots and cube roots

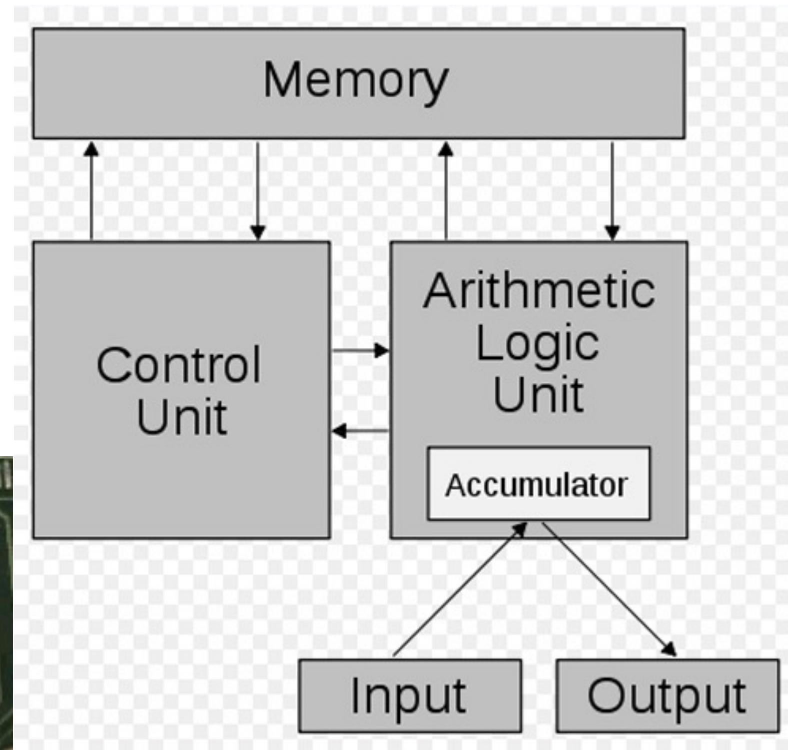


Basic computer architecture





Von Neumann Architecture



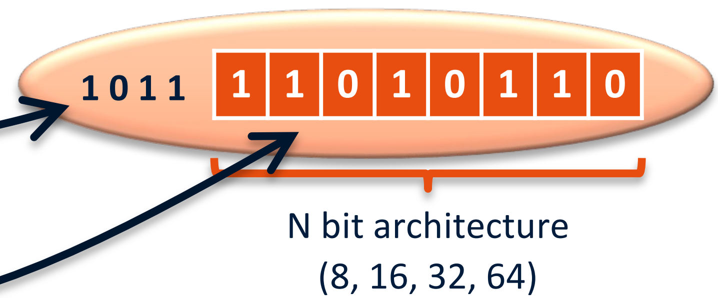
Von Neumann Computer

- 4 components
  1. Control Unit
  2. Arithmetic/Logic Unit
  3. Memory
  4. Input/Output
- **Stored program concept**
- **Sequential execution of instructions**

Memory

Functional unit of a computer that stores and can retrieve instructions and data

- Consists of circuits that represent cells capable of storing N bits
- Each cell has
  - an address
  - content



Internal memory

- RAM (random access memory)
  - reading and writing data and instructions
  - volatile
- ROM (read only memory)

Each cell has unique address

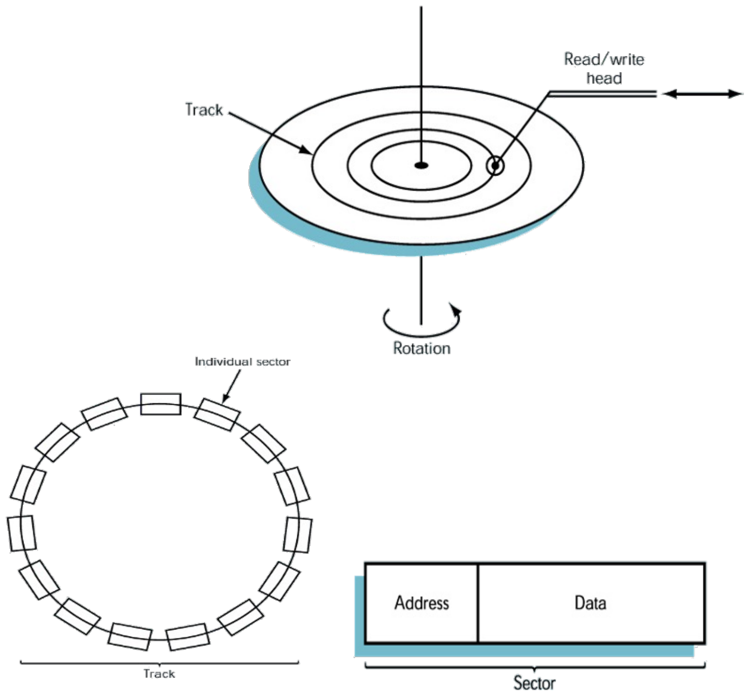
1111	1	1	0	1	0	1	1	0
1110	1	0	0	1	0	1	1	1
1101	0	1	0	1	1	1	0	0
1100	0	1	0	0	0	1	1	1
…	…							
0001	1	1	0	1	0	1	1	0
0000	0	0	0	1	1	0	0	0

Fast! 10-15 nsec (1 nano second = 10^{-9} sec)

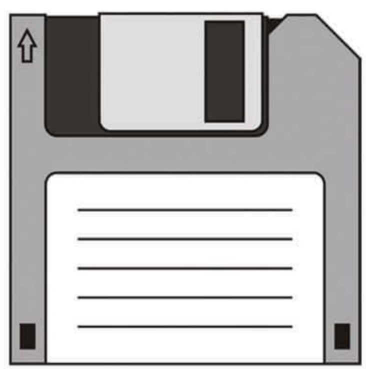
External Memory

Mass storage devices

- (Direct) access
  - USB flash drives
  - solid state drives
  - hard disks
  - optical disks
- Sequential access
  - tape drives



What's a floppy?



(Wikipedia)

Their size (3.5-Inch) was 1.44MB!

Binary Representations

- What?
  - everything inside a computer
    numbers, text, programs, pictures, music, video, …
- Why?
  - information is stored using voltage levels
  - using decimals requires 10 reliable distinct levels
  - much cheaper to only use 2 levels (but more components)

Bits, Bytes,

1 byte	8 bits
1 kilobyte (KB)	2^{10} = 1024 bytes
1 megabyte (MB)	2^{20} = 1048760 bytes
1 gigabyte (GB)	2^{30} = 1073741824 bytes
1 terabyte (TB)	2^{40} = 1099511627776 bytes
1 petabyte (PB)	2^{50} bytes
1 exabyte (EB)	2^{60} bytes
1 zetabyte (ZB)	2^{70} bytes
1 yottabyte (YB)	2^{80} bytes

Size of the entire WWW is 1 yottabyte (2^{80} bytes)

It would take approx. 11 trillion years to download a yottabyte file from the internet using a high-power broadband connection

Decimals

Base 10

(based on our number of fingers?)

- Decimal digits: 0,1,2,3,4,5,6,7,8,9

- Positional system – position represents power

Example:

$3845_{10} = 3 \text{x} 10^{3} + 8 \text{x} 10^{2} + 4 \text{x} 10^{1} + 5 \text{x} 10^{0}$

Binary

Base 2

- Binary digits (bits): 0,1
- Similar positional system – different base

Example

$$1101_2 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 13_{10}$$

Negative numbers?

Leftmost bit of a number represents the sign

0 for +	0	1	0	1	0	1	1	0	$= +86_{10}$
1 for -	1	1	0	1	0	1	1	0	$= -86_{10}$
	0	0	0	0	0	0	0	0	
	1	0	0	0	0	0	0	0	

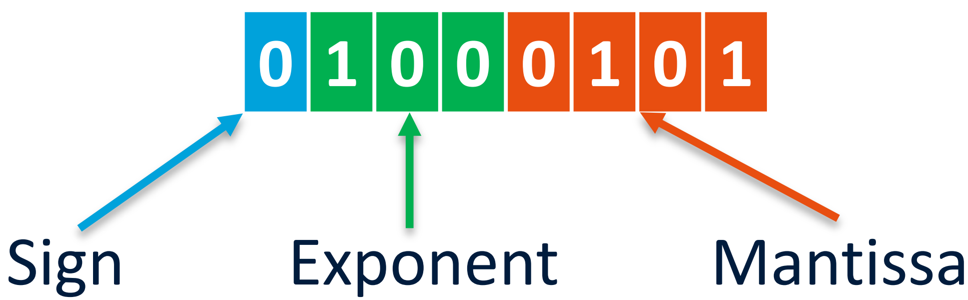
Arithmetic overflow

0	1	1	1	1	1	1	1	+	0	0	0	0	0	0	0	1	=	?

Trouble arises when the result requires more than the number of available bits…

Floating point numbers

- But, how can we represent the decimal numbers?



$$value = 1.0101_2 * 2^1$$

$$= \left(1 + \frac{1}{4} + \frac{1}{16}\right) * 2^1$$

$$= \left(1 + \frac{5}{16}\right) * 2^1$$

$$= \left(1 + \frac{5}{16}\right) * 2^1$$

$$= 21 * 2^{-3}$$

$$\approx 2.625$$

https://www.geeksforgeeks.org/introduction-of-floating-point-representation/

ASCII & UNICODE

- Also text uses a binary representation
- Each character is translated into a bitstring
  - ASCII uses 8 bits
  - UNICODE uses 16 bits

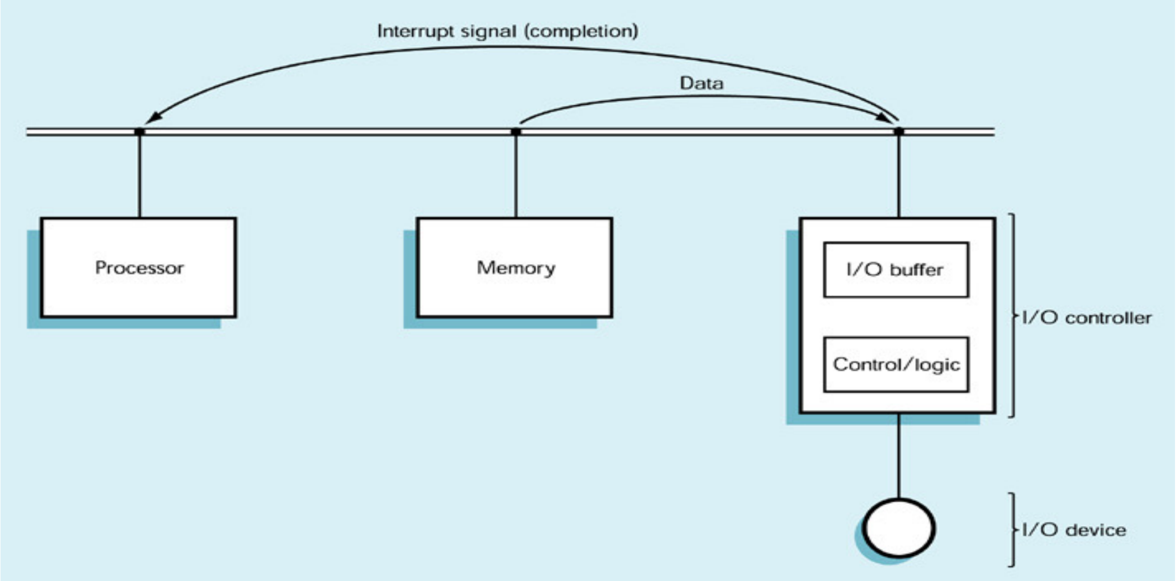
ASCII examples: A = 01000001 a = 01100001
@ = 01000000 ÿ = 11111111

01010111 01100101 01101100 01100011
01101111 01101101 01100101 00100001

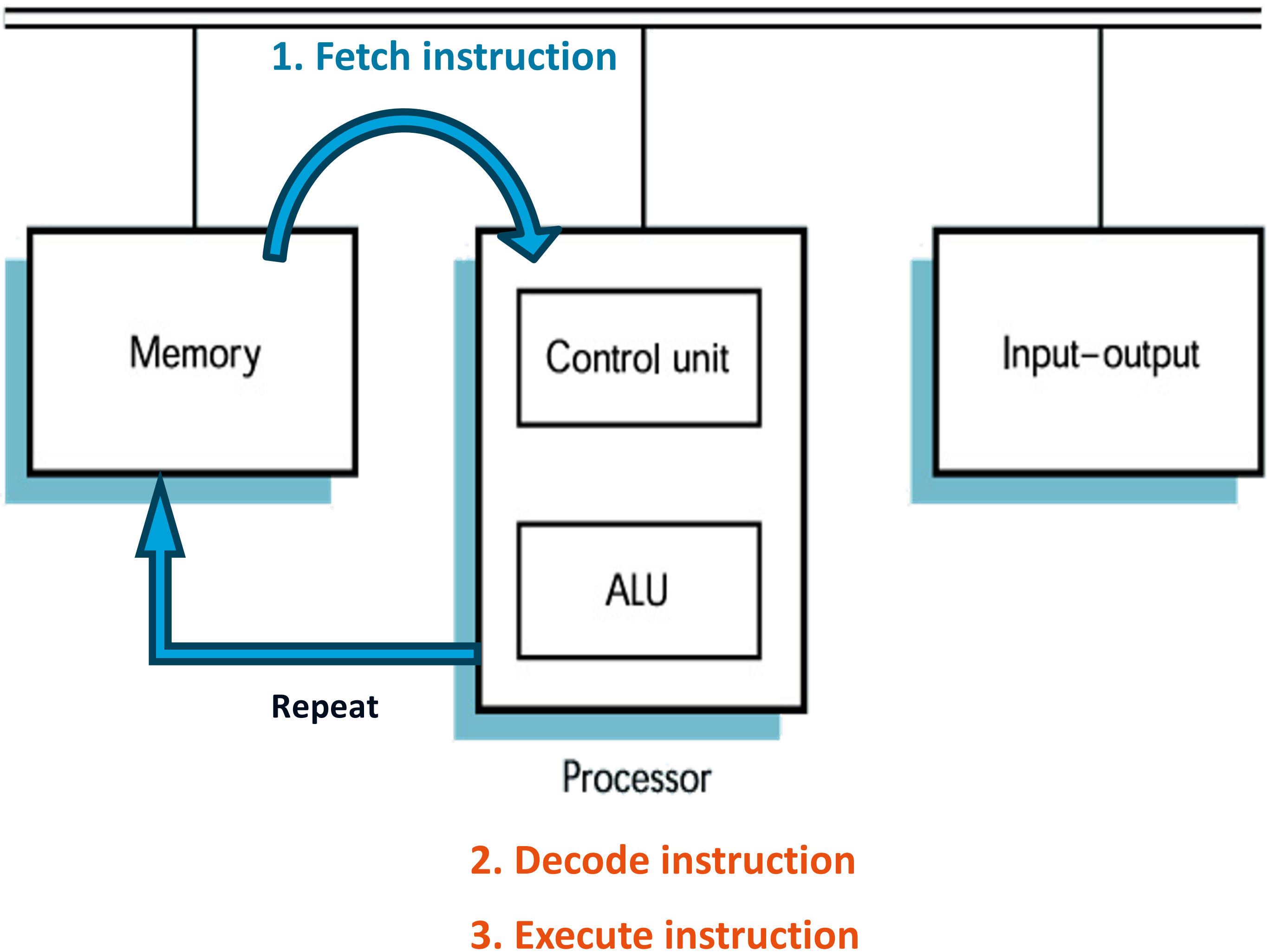
https://www.ascii-code.com/

Input/Output controllers

Treated as part of memory space
- need to compensate for speed differences

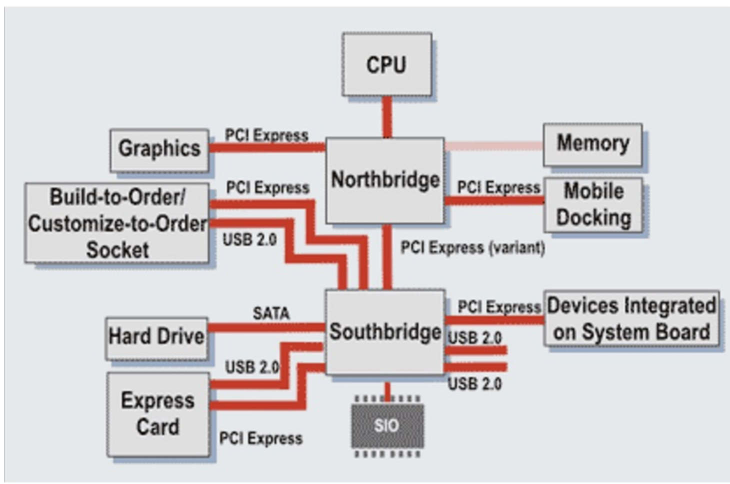


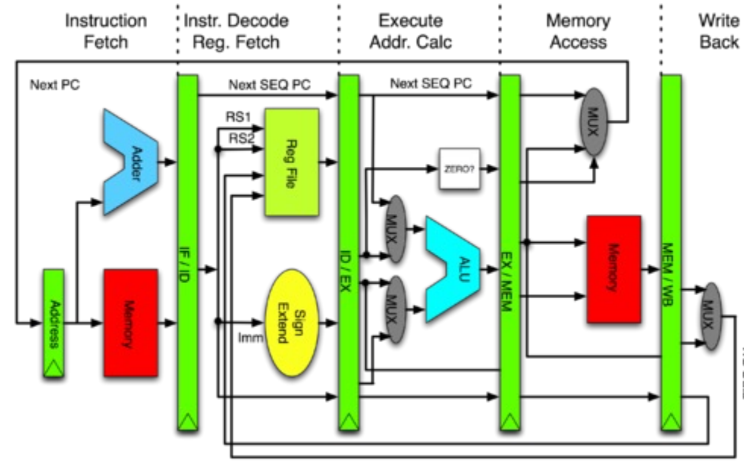
Execution cycle



Modern Computer Architecture

... is a bit more complex ...



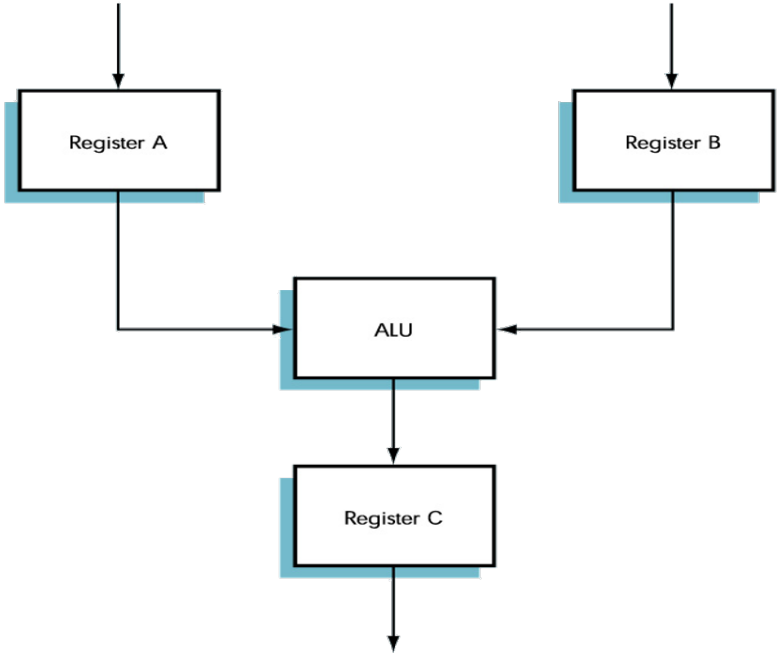


... but the principles remain the same

Arithmetic/Logic Unit (ALU)

Performs primitive arithmetic and logic operations

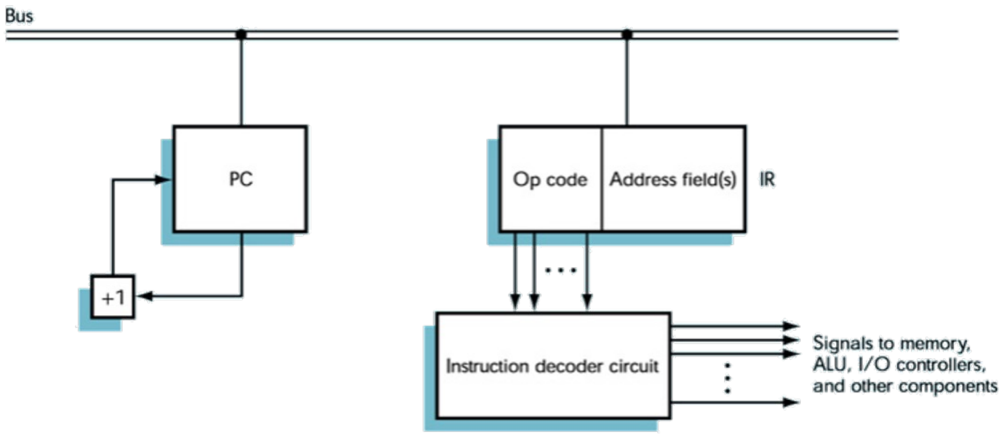
- Registers
- ALU circuit



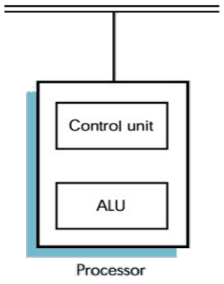
Control Unit (CU)

Implements the sequential execution of instructions

- program counter
- fetches instructions
- decodes instructions
- ensures execution



Processor = ALU + CU



Machine Language

Language/representation used for instructions inside the computer

Instructions or opcodes for:

- input and output
- moving data between RAM and registers
- arithmetic and logic operations
- comparisons and conditional outcomes

Instruction Set Examples

Binary opcode	Instruction
0000	LOAD X
0001	STORE X
0010	CLEAR X
0011	ADD X
0100	INCREMENT X
0101	SUBTRACT X
0110	DECREMENT X
0111	COMPARE X
1000	JUMP X
1001	JUMPGT X
1010	JUMPEQ X
1011	JUMPLT X
1100	JUMPNEQ X
1101	IN X
1110	OUT X
1111	HALT

Assembler

A *human readable* representation of machine language

Can differ for each specific processor

<u>Instruction</u>

```
LOAD X
STORE X
CLEAR X
ADD X
INCREMENT X
SUBTRACT X
DECREMENT X
COMPARE X
JUMP X
JUMPGT X
JUMPEQ X
JUMPLT X
JUMPNEQ X
IN X
OUT X
HALT
```

Assembler programming

Programming in assembler is possible …

but not recommended if sanity is something you enjoy or value

- microscopic view of tasks
- manual management of data movement
- machine-specific

- only used for ultra-high-performance requirements of small subroutines

High-level programming languages

Each language statement can correspond to **many** machine instructions

- (more) macroscopic view of a task
- (more) portable between machines
- closer to natural language descriptions

Assembly Language

```
load x
add y
store z
halt
```

→

High-Level Programming Language

```
z = x + y
```

https://survey.stackoverflow.co/2022/#technology-most-popular-technologies

Source Code

- Computer files containing high-level programming language statements
- Can be compiled and executed (or possibly interpreted directly)

```
private void advance() {
    boolean[][] newgrid = new boolean[grid.length][grid[0].length];

    for (int i=0; i<grid.length; i++)
        for (int j=0; j<grid[0].length; j++)
            newgrid[i][j] = false;

    for (int i=0; i<grid.length; i++)
        for (int j=0; j<grid[0].length; j++)
            if ((grid[i][j]) && (nbrOfNeighbors(i,j) < 2))
                newgrid[i][j] = false;
            else if ((grid[i][j]) && (2 <= nbrOfNeighbors(i,j)) && (nbrOfNeighbors(i,j) <= 3))
                newgrid[i][j] = true;
            else if ((grid[i][j]) && (3 < nbrOfNeighbors(i,j)))
                newgrid[i][j] = false;
            else if ((!grid[i][j]) && (nbrOfNeighbors(i,j) == 3))
                newgrid[i][j] = true;

    grid = newgrid;
}

private int nbrOfNeighbors(int x, int y) {
    int result = 0;
    if ((0 <= x-1) && (0 <= y-1) && (grid[x-1][y-1])) result++;
    if ((0 <= x-1) && (grid[x-1][y])) result++;
    if ((0 <= x-1) && (y+1 < grid[0].length) && (grid[x-1][y+1])) result++;
    if ((0 <= y-1) && (grid[x][y-1])) result++;
    if ((y+1 < grid[0].length) && (grid[x][y+1])) result++;
    if ((x+1 < grid.length) && (0 <= y-1) && (grid[x+1][y-1])) result++;
    if ((x+1 < grid.length) && (grid[x+1][y])) result++;
    if ((x+1 < grid.length) && (y+1 < grid[0].length) && (grid[x+1][y+1])) result++;
    return result;
}
```

Editors

Source code files are text files

- contain only ascii/unicode characters
- very different from e.g. a Word file

Programming editors

- edit text files
- supply syntax highlighting
- more

```
for (int i = 0; i < array.length; i++) {
    for (int j = 0; j < array[i].length; j++) {
        if (array[i][j]>=0){
            positive[i][j] = true;

        else {
            positive[i][j] = false;
        }
    }
}
```

Source code to running program

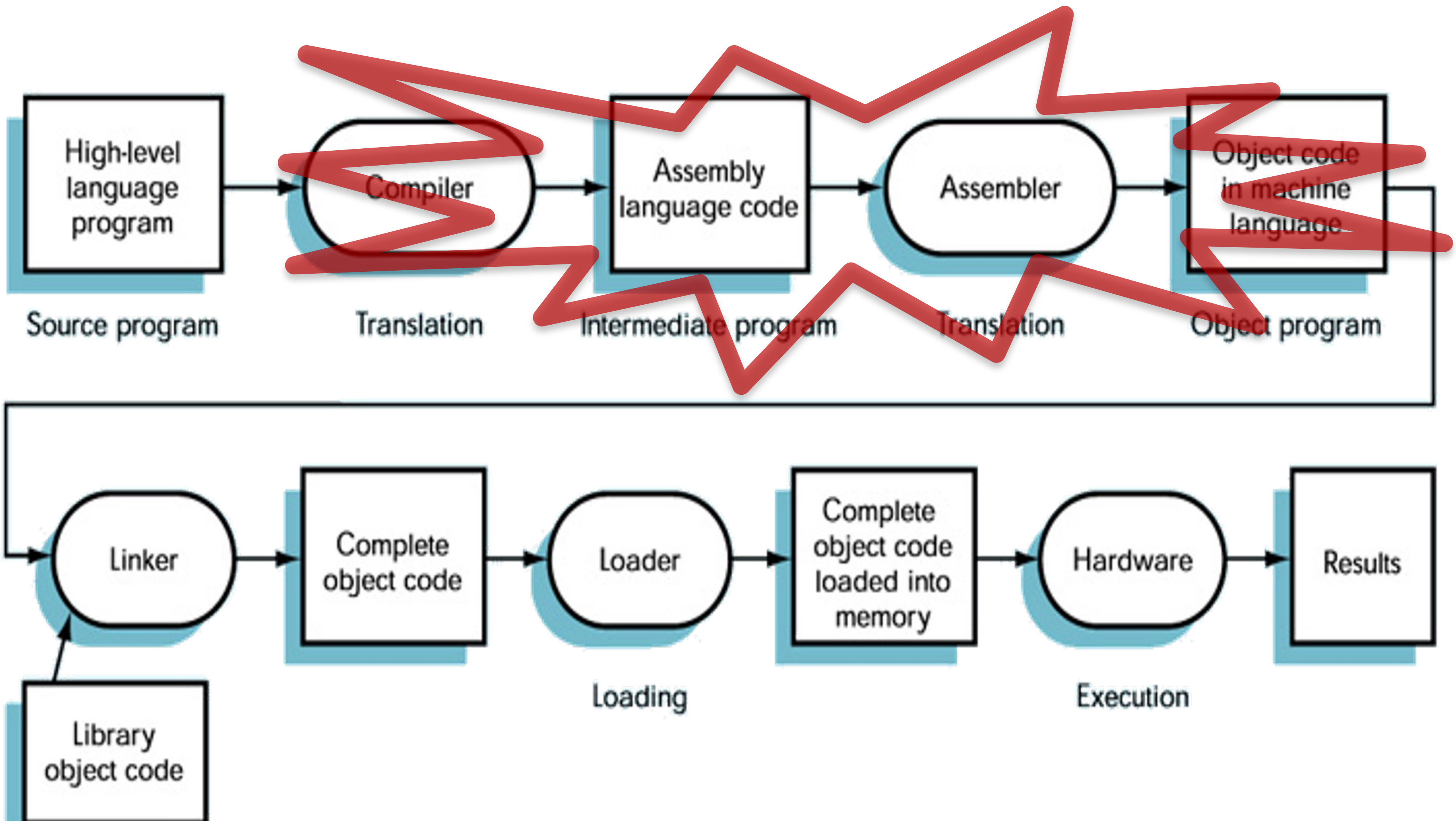
Program compilation

- the process of converting a high-level language program into machine language; done at once before the program runs by a program called “compiler”

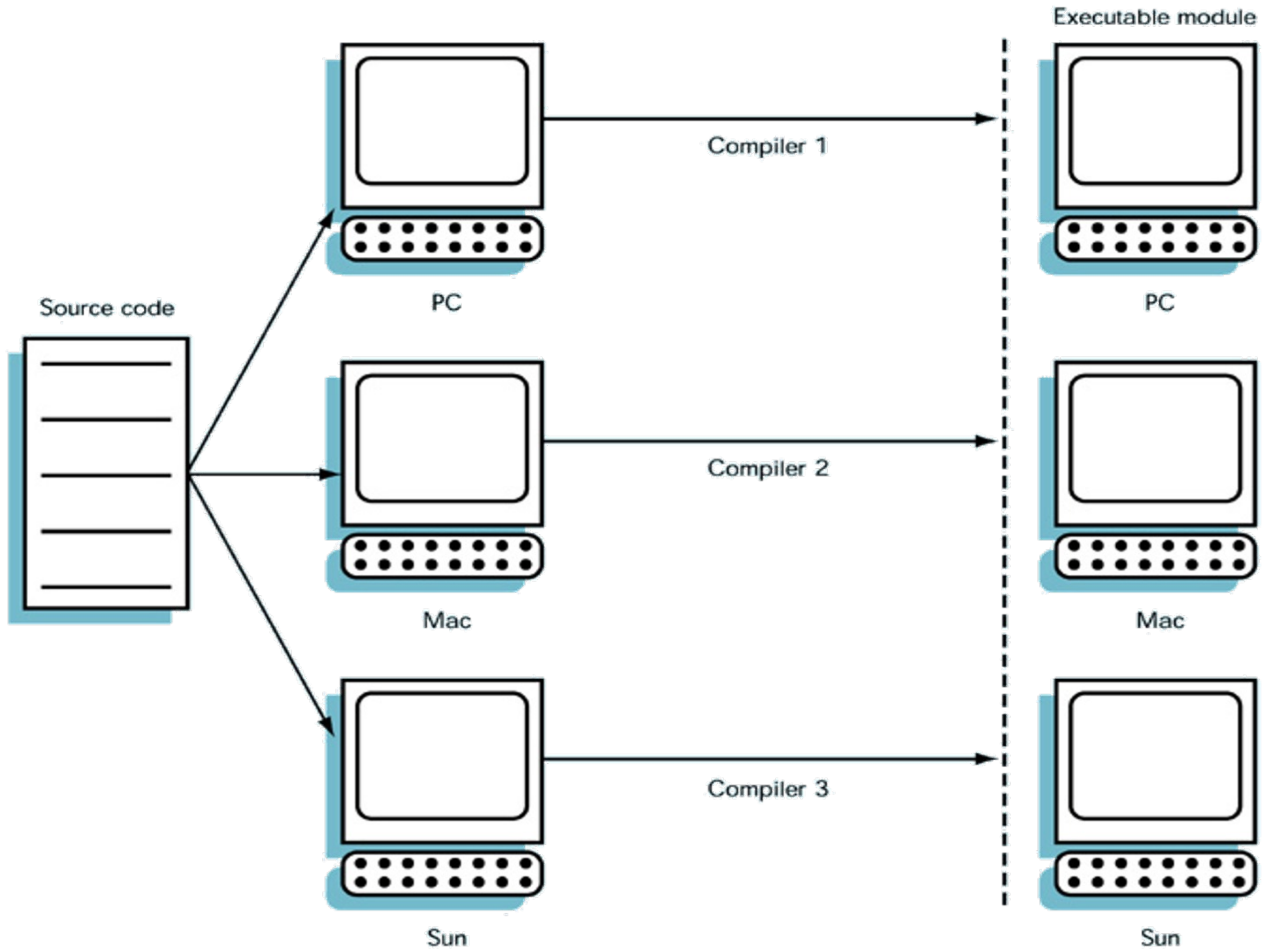
Program interpretation

- the process of executing a program by another program called “interpreter”; converts our code line-by-line into machine code during program run

The compilation & running process



Compilation for different computers



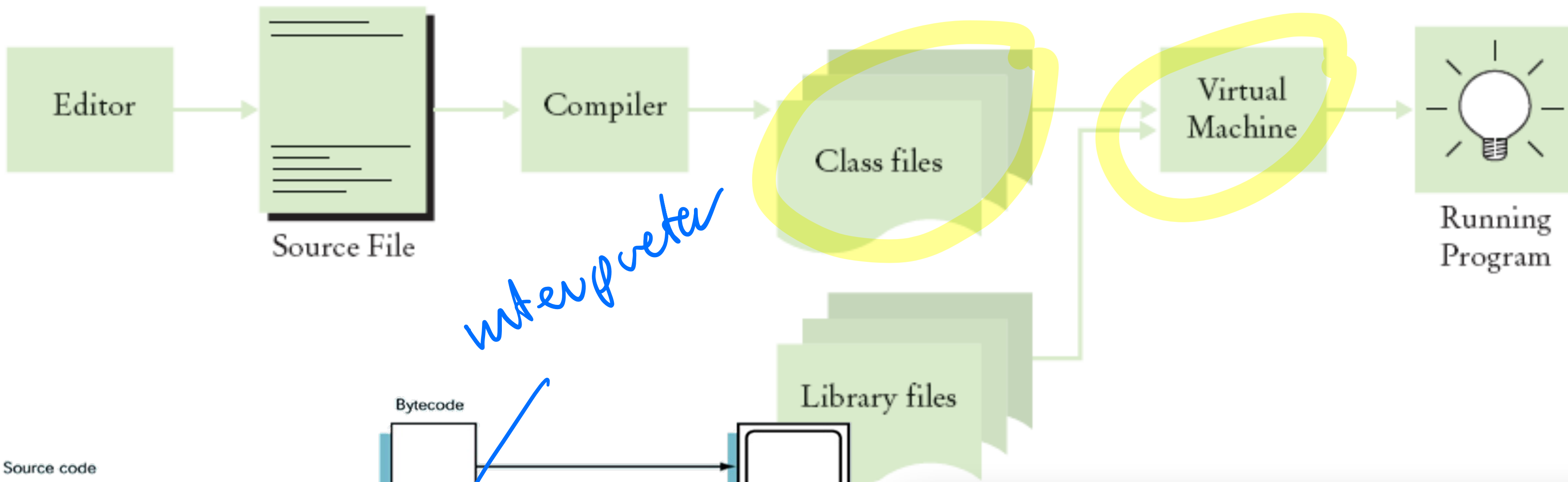
Java

is (according to Sun Microsystems) a simple, object-oriented, distributed, interpreted, secure, architecture-neutral, portable, multithreaded, general-purpose language developed by Sun Microsystems in 1995

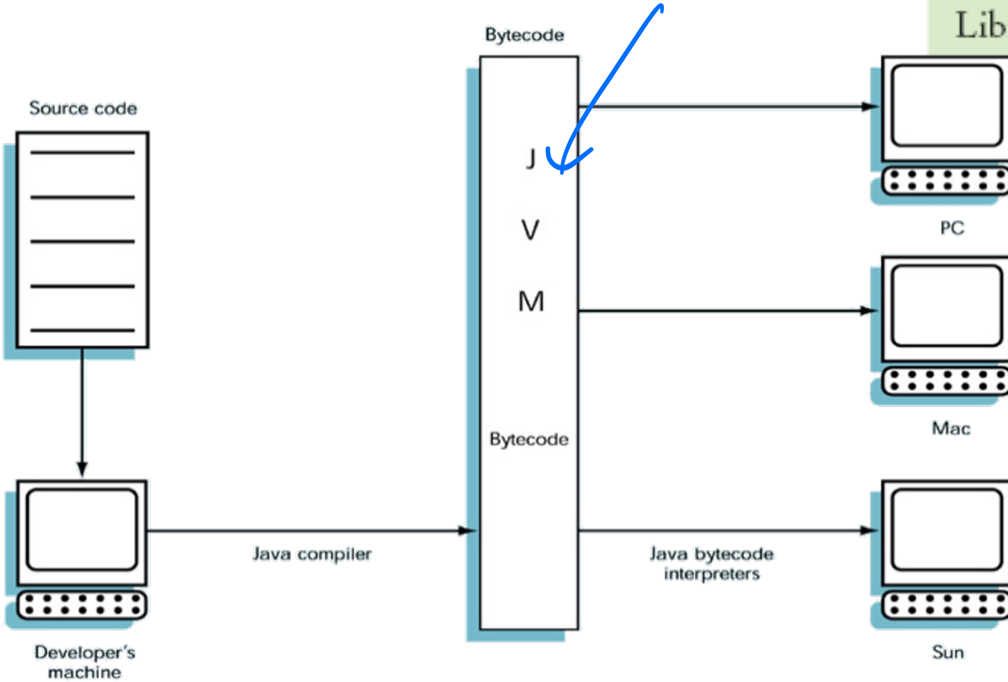
- ✓ well known and popular
- ✓ widely used (also for teaching)
- ✓ rich library
- ✓ designed for the internet

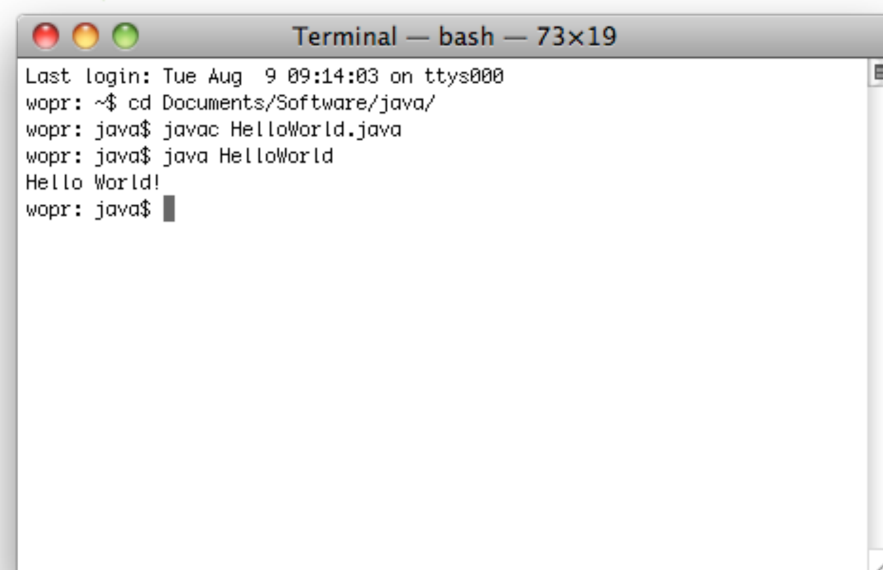
not designed for teaching

Compilation & running process for Java



interpreter





Summary

- Computer architecture
- Binary representations and machine language
- High level programming languages
- Compilation, interpretation and execution

<u>Book:</u> Chapter 1

<u>Quiz:</u> 1a

<u>Assignment 0:</u> Hello Visual Studio Code!

Learning objectives

- You know the four components of the Von Neumann Architecture
- You understand the execution cycle on the Von Neumann Architecture
- You understand the binary representation
- You know the basics of assembler
- You know the difference between assembly and high-level programming languages
- You know the difference between compilation vs interpreted code

Next up

- Watch the videos about compilation and prepare for variables and methods
- Live coding lecture tomorrow
- The first tutorial is on Friday. Prepare it in advance!
  - Get Visual Studio Code installed and running
  - Read Game Lab 1 Student Handbook