CMSC131
Lecture Set 0:
Course Introduction

Topics in this set:

1. Course information
2. Computer terminology basics
3. Tools needed for this course

Basic Info

- **Name**: "Object-Oriented Programming I"
- **Class meetings**: Lab and Lecture
- **Instructor**: Tom Reinhardt
- 4 **TAs**: Ahmed Elgohary, Ujjwal Goel, Ugur Koc, Xuetong Sun
- **Office Hours**
  - Will be Posted
  - All in AVW building:
    - 1112 (TA's), 3239 (Tom Reinhardt)

What Is This Course?

- A *fast-paced* introduction to techniques for writing computer programs!
  - Skill Development in Programming
  - Conceptual Understanding of Programming
  - Beginning of “computer science”
- Intensive, but assumes you are starting at level 0.
- Keys to success
  - Attend all classes and lab sections
  - Start assignments early – and continue until you truly understand
  - Get help early if you are having trouble – instructor & TAs
  - Study every day
    - it doesn’t work to cram for these exams
    - ask questions as soon as you realize you are confused
  - *Check announcements every day*

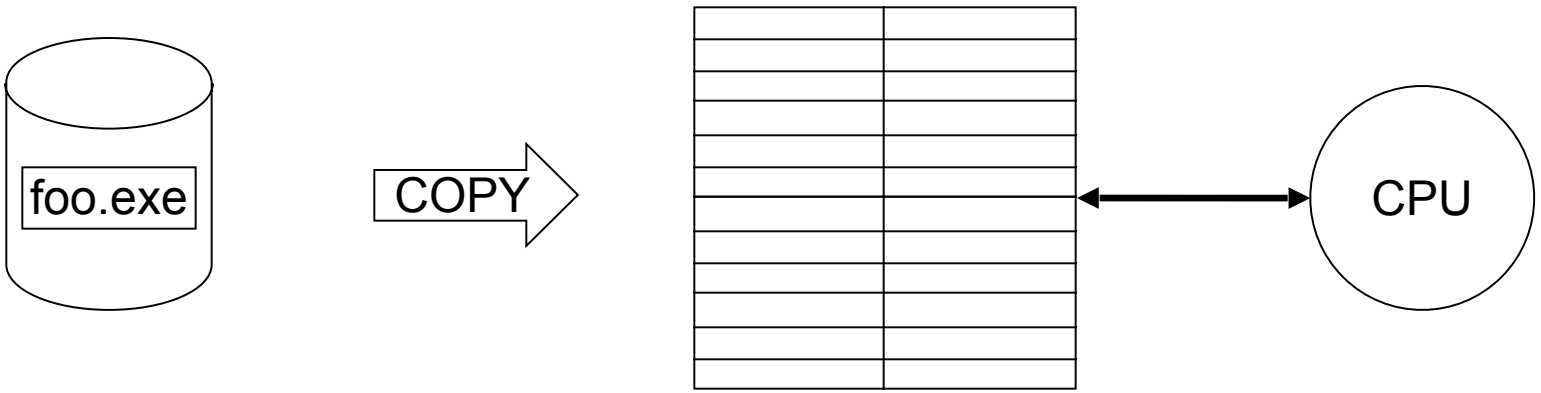
Course Software

- Eclipse
  - An IDE (integrated development environment)
  - We will use it for writing Java™ programs
  - Access to Eclipse (it's free!)
    - You can install it on your own machine: http://www.cs.umd.edu/eclipse
    - Also accessible in some labs around campus
- CVS (Concurrent Versions System)
  - A version-management system
  - You will use it for submitting your projects
- Both of these – Demonstrations on Wednesday

Computer Organization

- Hardware:
  - physical parts of computer
  - examples
    - Monitor, mouse, keyboard
    - Chips, boards
    - Cables, cards
    - etc.
- Software:
  - non-physical (“logical”) parts of computer
  - Programs = instructions for computer to perform

How Programs Are Executed



Program “foo” initially stored in secondary storage

Program copied into main memory

CPU executes program instruction-by-instruction

Hardware Overview

- **CPU** = central processing unit
  - Executes the "instructions“ in programs
- **Main memory** = random-access memory = “RAM”
  - Stores data that CPU accesses, including instructions
  - FAST, but smaller and temporary; wiped out when computer is shut off!
- **Secondary memory**: Hard disks, CDs, DVDs, flash memory, etc.
  - Stores data that can be loaded into main memory
  - SLOWER, but larger and permanent

- **I/O devices**
  - How you communicate with your machine
  - Keyboard, monitor, mouse, speakers, etc.
- **Networking equipment**
  - How others communicate with your machine
  - Networking “cards”, cables, etc.

Main Memory

- Computer data consists of off and on pieces (often written as 0's and 1's)
- bit: A single cell in main memory that can hold either a 0 or 1
- *byte*: A sequence of 8 bits
- word: Unit of memory (size varies by computer - often a sequence of 4 bytes)
- Main memory: table of bytes indexed by "addresses"

Address	Byte value
1	1 0 0 1 1 1 0 1
2	0 0 0 1 1 0 0 1
3	1 1 1 1 1 1 0 1
4	1 1 0 0 0 1 0 0

How Many Different Values can be stored in a…

- Bit?
  2
- Two bits?
  4 = 2 x 2
- Byte?
  256 = 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 28
- Word?
  4,294,967,296 = 232

Other Standard Terminology

Prefixes for bit and byte multiples

Decimal			Binary			
Value	SI		Value	IEC		JEDEC
1000	k	kilo-	1024	Ki	kibi-	K kilo-
1000^2	M	mega-	1024^2	Mi	mebi-	M mega-
1000^3	G	giga-	1024^3	Gi	gibi-	G giga-
1000^4	T	tera-	1024^4	Ti	tebi-	
1000^5	P	peta-	1024^5	Pi	pebi-	
1000^6	E	exa-	1024^6	Ei	exbi-	
1000^7	Z	zetta-	1024^7	Zi	zebi-	
1000^8	Y	yotta-	1024^8	Yi	yobi-	

One kilobyte is approximates one kibibyte which is approx 1000 bytes (actual 1024 bytes).
210 = 1024
220 = 10242
230 = 10243
230=1,073,741,824

How Are Characters, Etc., Represented?

- Via *encoding schemes*
- Example: ASCII
  - American Standard Code for Information Interchange
  - Early standard for encoding a single character in a bytes
  - In ASCII:
    - ‘A’ 01000001, ‘B’ 01000010, ‘C’ 01000011, …
    - ‘a’ 01100001, ‘b’ 01100010, ‘c’ 01100011,…
    - ’1’ 00110001, ‘2’ 00110010, ‘3’ 00110011, …
    - ‘,’ 00101100
    - etc.

Other Character Encodings

- International support?
  - Unicode
- Most common variation: UTF-8
  - Backwards compatible with ASCII

Unicode	Byte1	Byte2	Byte3	Byte4	Example
U+0000–U+007F (0 to 127)	0xxxxxx x				'$' U+0024 → 00100100 → 0x24
U+0080–U+07FF (128 to 2,047)	110yyyx x	10xxxxx x			'¢' U+00A2 → 11000010,10100010 → 0xC2,0xA2
U+0800–U+FFFF (2,048 to 65,535)	1110yyy y	10yyyyx x	10xxxxx x		'€' U+20AC → 11100010,10000010,10101100 → 0xE2,0x82,0xAC
U+10000–U +10FFFF (65,536 to 1,114,111)	11110zz z	10zzyyy y	10yyyyx x	10xxxxx x	'𤭢' U+024B62 → 11110000,10100100,10101101,10100010 → 0xF0,0xA4,0xAD,0xA2

Software Overview

- **Operating system**: manages computer's resources; typically runs as soon as computer is turned on.
  Typical responsibilities:
  - *Process management*
    - Determines when, how programs will run on CPU time
  - *Memory management*
    - Controls access to main memory
  - *I/O, window system, network control*
    - Performs low-level drawing, communication operations
  - *Security*
    - Manages user IDs, passwords, file protections, etc.

- **Applications**: programs users interact directly with; usually are explicitly run.
  Examples:
  - Word processors
  - Games
  - Spreadsheets
  - Music software,
  - Etc

Programming Languages

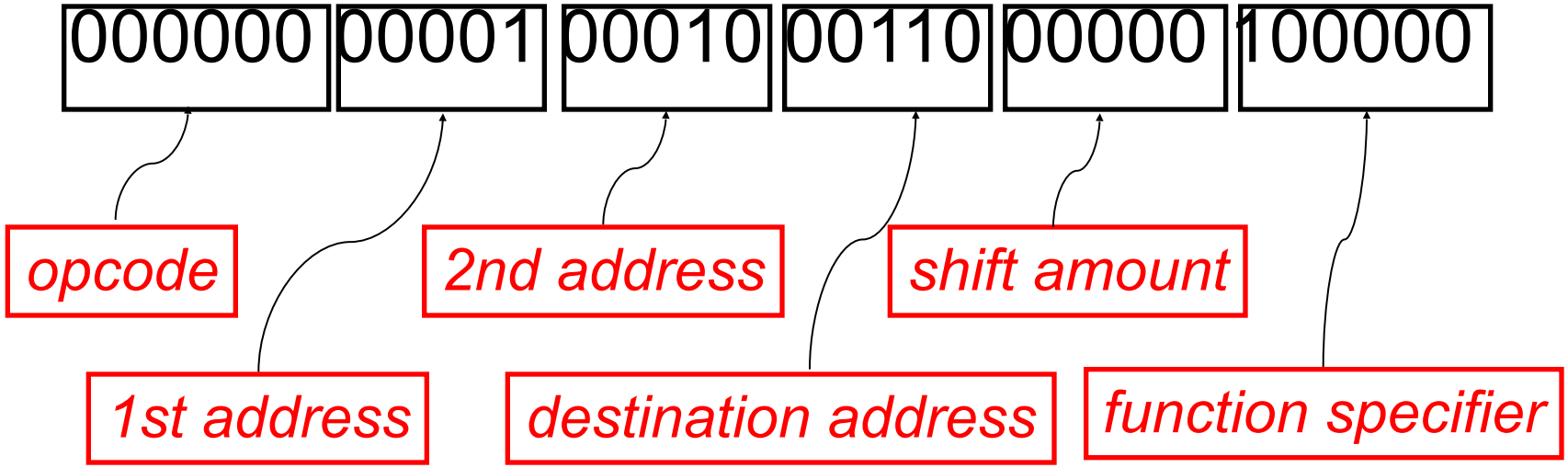
- Used to write programs that run on computers
- Generations of programming languages
  - 1st (1GL): machine code
  - 2nd (2GL): assembly code
  - 3rd (3GL): procedural languages
  - 4th (4GL): application-specific languages
  - 5th (5GL): constraint languages

1st Generation: Machine Code

- Recall: computer data is 0's and 1's.
- In machine code, so are programs!
  - Program: sequence of instructions
  - Machine code: instructions consist of 0's and 1's
- Next slide: example machine code instruction from MIPS (= "Microprocessor without interlocked pipeline stages") architecture
  - Popular in mid-, late 90s
  - Instructions are 4 bytes long

Example MIPS Instruction

- "Add data in addresses 1, 2, store result in address 6":
  00000000001000100011000000100000
- broken into parts:



Programming in 1GLs

Courtesy of Microsoft Encarta Encyclopedia Online. Copyright (c) Microsoft Encarta Online

2nd Generation: Assembly

- Problem with 1GLs: Who can remember those opcodes, addresses, etc. as 0's, 1's?
- Solution (1950s): *assembly language*
  - *mnemonics* = descriptive character strings for opcodes
  - Let programmers give descriptive names to addresses
- MIPS example revisited:

```
add $1, $2, $6
```

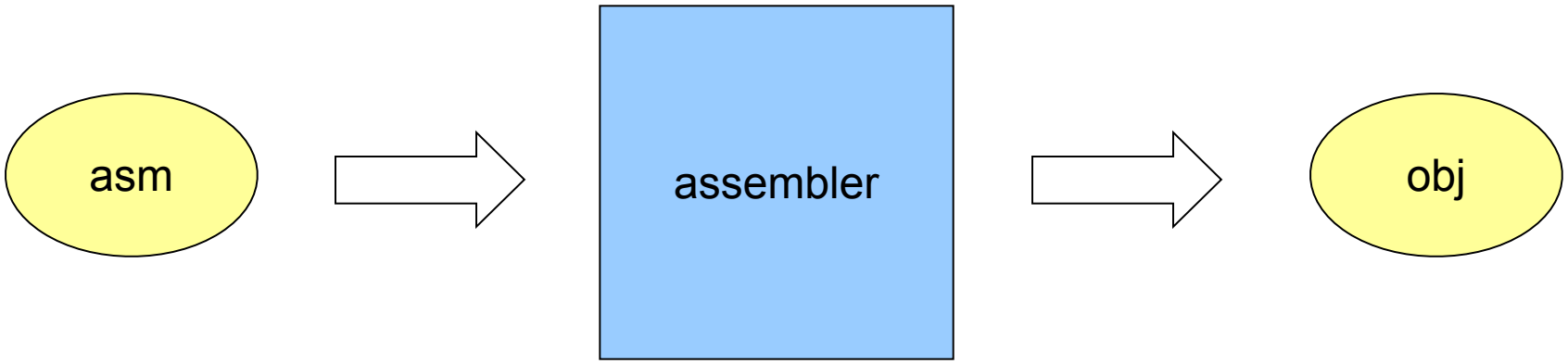
instead of

00000000001000100011000000100000

for "add contents of addresses 1, 2, store result in 6"

Assemblers

- Computers still only work on machine code (1GL)
- Assembly language is not machine code
- *Assemblers* are programs that convert assembly language to machine code (= “object code”)



3rd Generation: Procedural Languages

- Problems with 2GLs
  - *Platform dependency*
    - Different kinds (*architectures*) of computers use different instruction formats
    - E.g. x86, Pentium, 68K, MIPS, SPARC, etc.
    - 1GL / 2GL programs written for one kind of machine will not work on another
  - *Low level*: programs difficult to understand

- Solution (1960s -- now): *procedural languages*
  - Higher-level, "universal" constructs
  - Examples: Cobol, Fortran, Algol, Pascal, C, C++, Java, C#

Compilers

- Computers can only execute machine code
- *Compilers* are programs for translating 3GL programs ("source code") into assembler / machine code