CMSC 131

Object-Oriented Programming I

Computer Organization

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Computer Organization

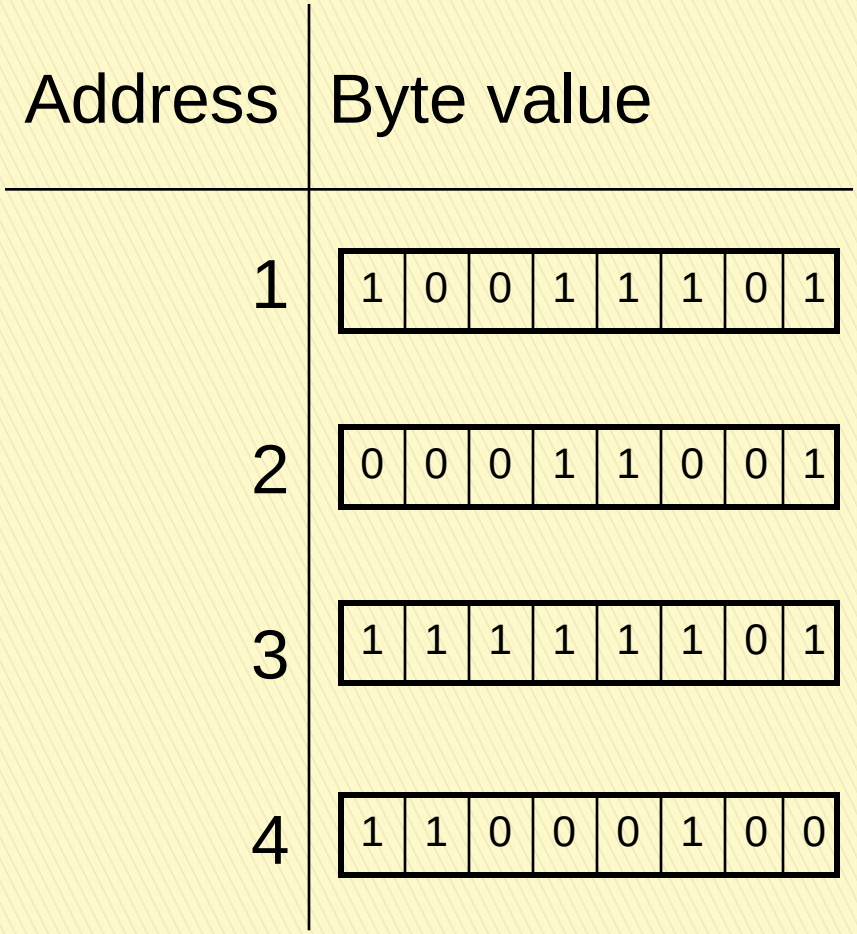
- Hardware: physical parts of computer
  - Monitor, mouse, keyboard
  - Chips, boards
  - Cables, cards
  - etc.
- Software: non-physical (“logical”) parts of computer
  - Programs = instructions for computer to perform

Hardware Overview

- **CPU** → central processing unit "brain"
  - Executes the "instructions" in programs
- **Main memory** → random-access memory = "RAM"
  - Stores data that CPU accesses, including instructions
  - FAST, but 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 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 (often a sequence of 4 bytes)
- Main memory: table of bytes indexed by "addresses"



How Many Different Values 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* = 2^8
- Word?
  4,294,967,296 = 2^{32}
- In general k bits can represent 2^k *values*

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 approximatly one kibibyte which is approximately 1000 bytes.

$2^{10} = 1024$

$2^{20} = 1024^2$

$2^{30} = 1024^3$

How Are Characters, Etc., Represented?

Via *encoding schemes*

Example: ASCII (American Standard Code for Information Interchange)

- Standard for encoding character values as bytes
- In ASCII:
  - ‘A’ 01000001
  - ‘a’ 01100001
  - ‘,’ 00101100
  - etc.

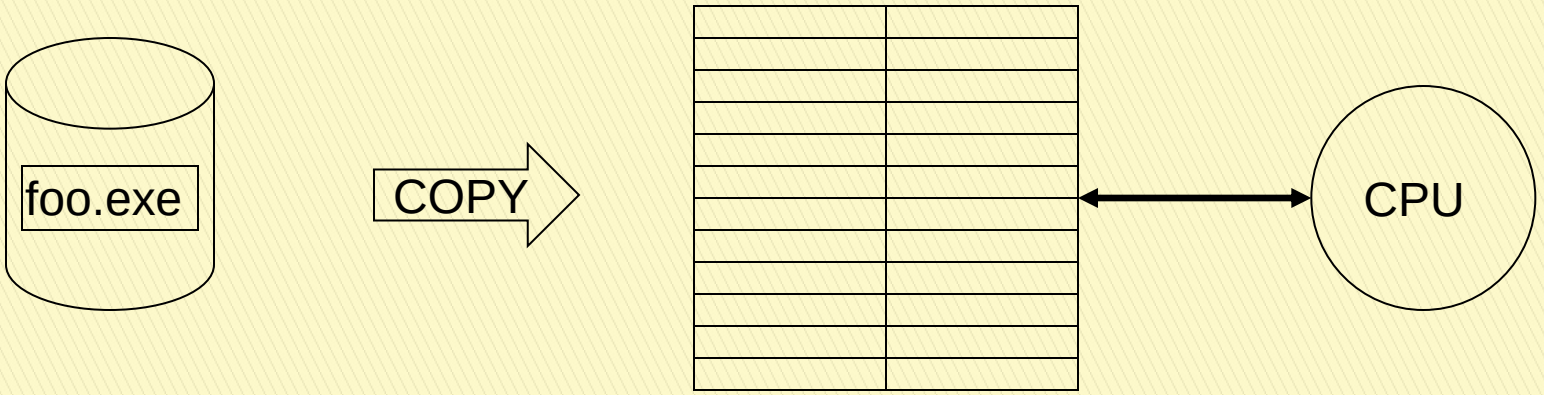
Other Character Encodings

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

Software Overview

- **Two levels →** Operating System and Application
- **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*
  - *I/O, window system*
  - *Network control*
  - *Security*
- **Applications →** programs users interact directly with; usually are explicitly run. Examples:
  - *Word processors*
  - *Games*
  - *Spreadsheets*
  - *Music software*
  - *Java Programs*
  - *Etc*

How Programs Are Executed



Program “foo” initially stored in secondary storage

Program copied into main memory

CPU executes program instruction-by-instruction

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

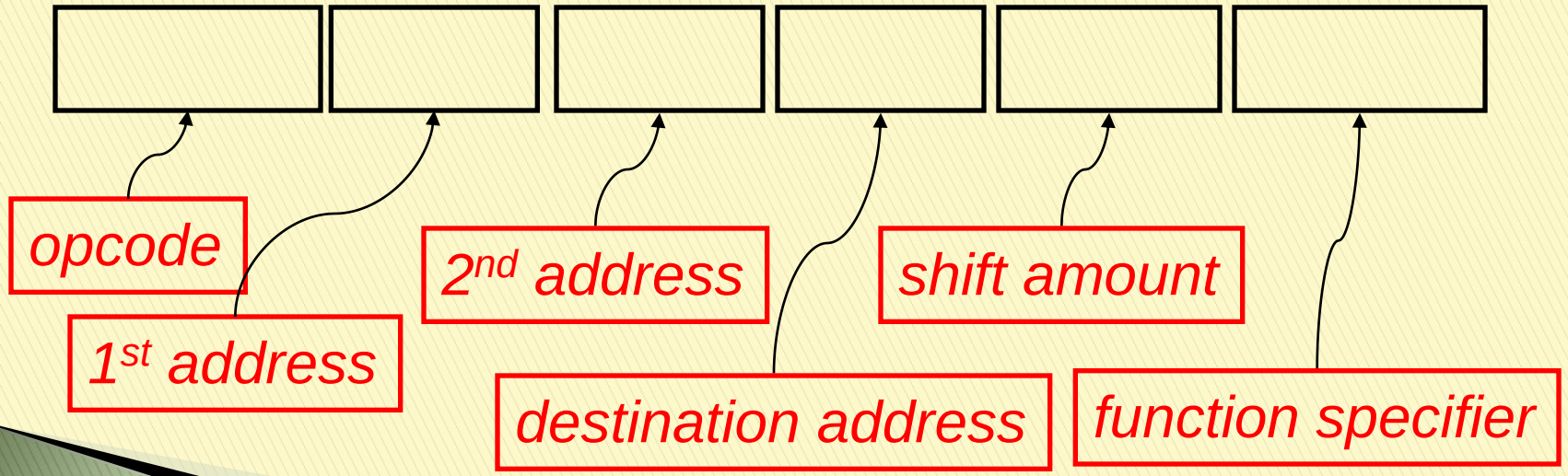
1st Generation: Machine Code

- 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
- ???

  000000 00001 00010 00110 00000 100000



2nd Generation: Assembly

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

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

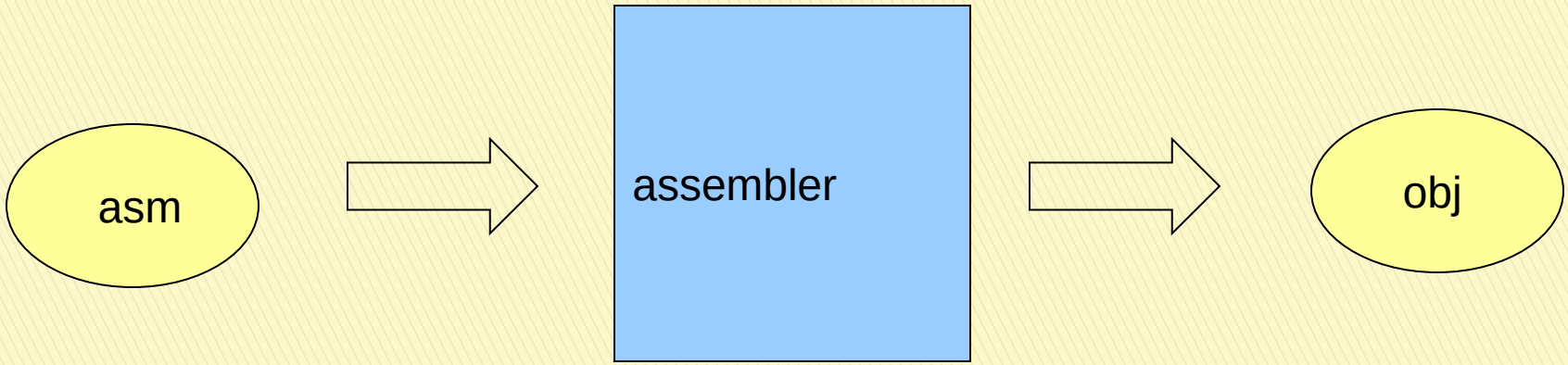
instead of

```
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

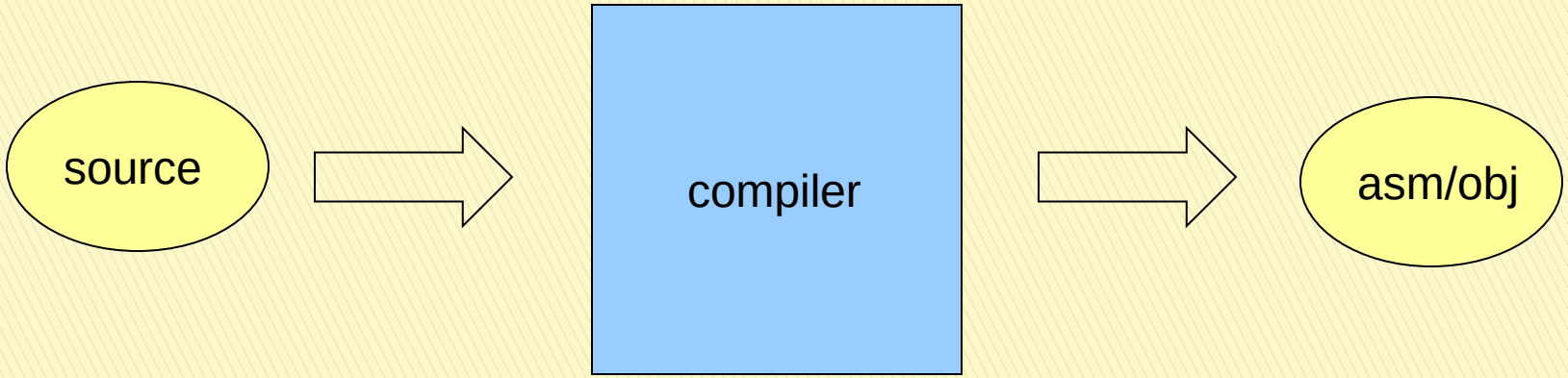
      E.g. x86, Pentium, 68K, MIPS, SPARC, etc.
    - 1GL / 2GL programs written for one kind of machine will not work on another
  - *Low level* → programs difficult to understand
- Solution (60s → now): *procedural languages*

3rd Generation: Procedural Languages

- Procedural languages
- Higher-level, “universal” constructs
- Examples:
  - 1950's → early 60s : Fortran, Cobol
  - 1958 → Lisp, invented for AI, still used!
  - late 60's → Algol, first language that "looks" modern
  - 70's → Pascal (like Algol, but simpler), used for teaching
  - 80's → C became popular (although it was "invented" in 70's)
    C++ "C with classes , object oriented”
  - 90's → Java, Fully object oriented
  - 00's → C# (Microsoft's answer to java)
- List of computer programming languages
  - http://en.wikipedia.org/wiki/List_of_programming_languages

Compilers

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



Interpreters

- Another way to execute 3GL programs
  - Interpreters take source code as input
  - Interpreters execute source directly
  - Much slower than compiled programs
- *Debuggers* are based on interpreters
  - Debuggers support step-by-step execution of source code
  - Internal behavior of program can be closely inspected
- Common interpreter?

Object-Oriented Terminology

- Procedural-oriented languages
  - Programming centers around "actions" (verbs)
- Object Oriented Languages
  - Centered on objects (nouns)
- Object
  - Principal entities that are manipulated by the program (nouns)
- Class
  - A "blueprint"/recipe that defines the structure for one or more objects
- Method
  - java term for a "function", a "procedure" or a "subroutine"
  - this is the code that does something (verbs)
- Why we prefer the object-oriented approach?
  - One big reason: recycling
- System Example