Chapter 1: Computer Abstractions and Technology

1.1 – 1.4: Introduction, great ideas, Moore's law, abstraction, computer components, and program execution

ITSC 3181 Introduction to Computer Architecture

https://passlab.github.io/ITSC3181/

Department of Computer Science
Yonghong Yan
yyan7@uncc.edu
https://passlab.github.io/yanyh/

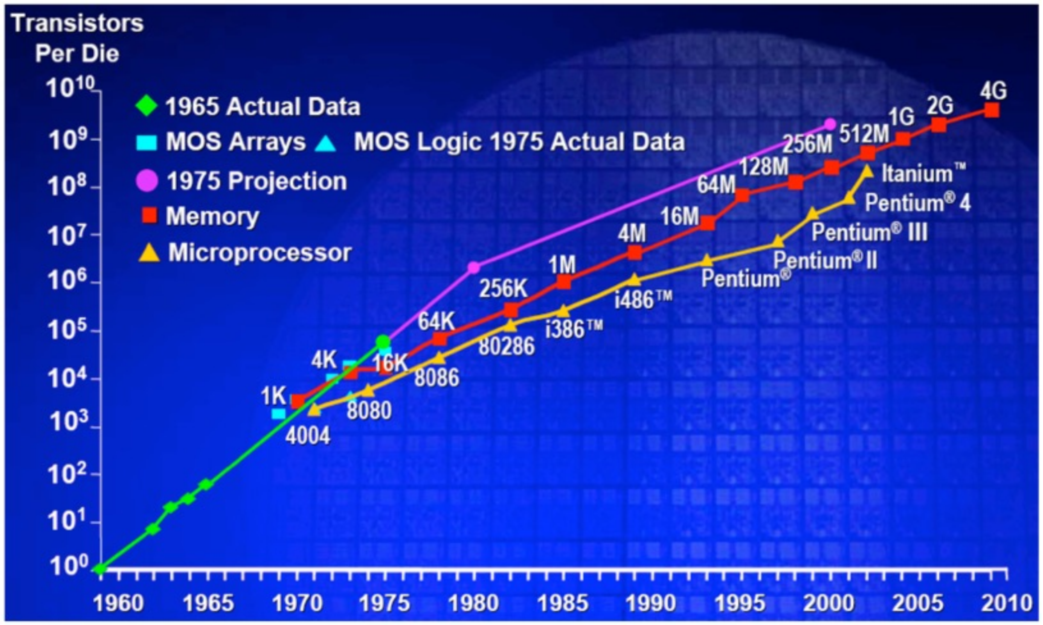
Lectures for Chapter 1 and C Basics
Computer Abstractions and Technology

- ☛ Lecture 01: Chapter 1
  – 1.1 – 1.4: Introduction, great ideas, Moore's law, abstraction, computer components, and program execution

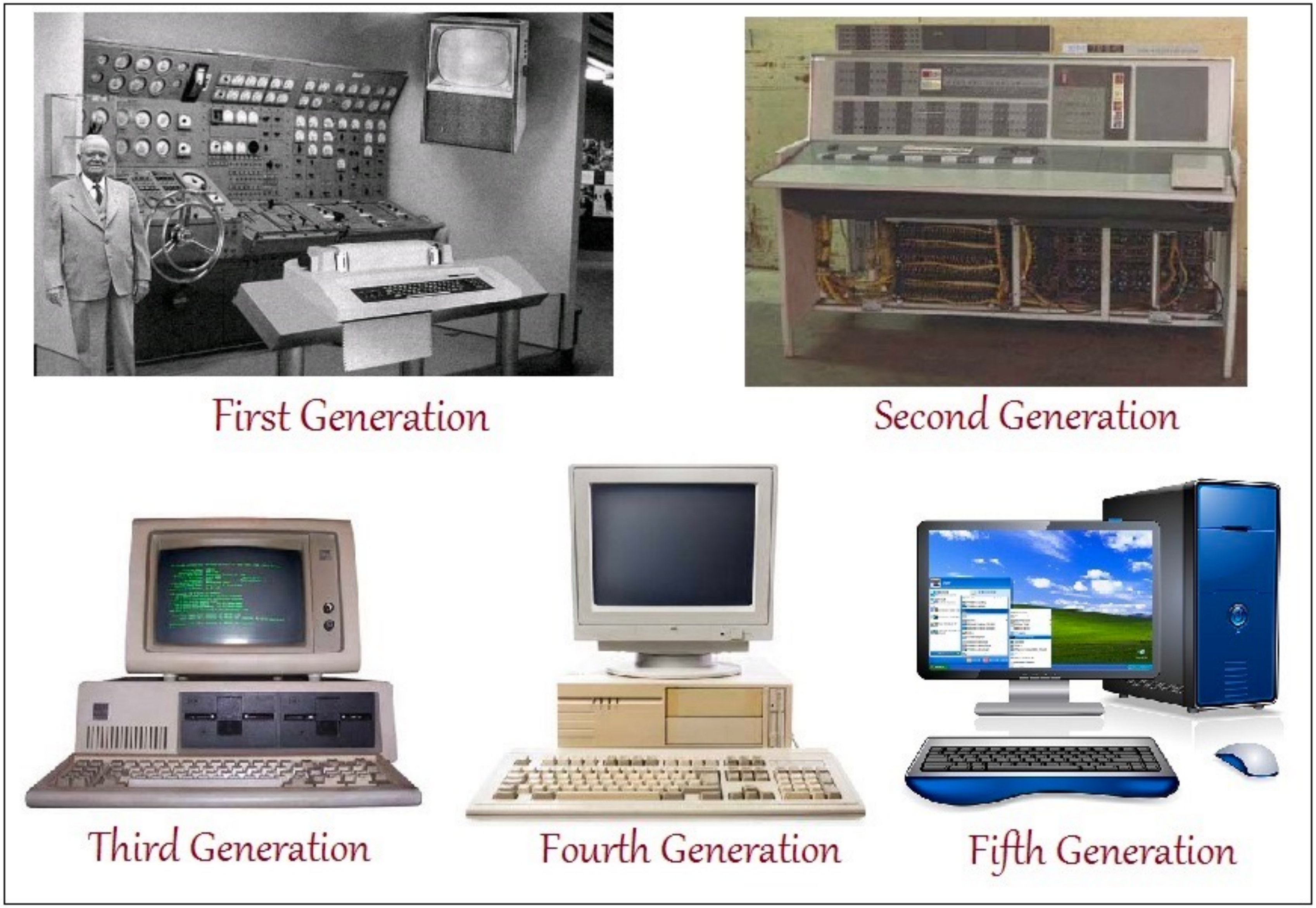
- Lecture 02: Number System, Compilation, Assembly, Linking and Program Execution
- Lecture 03: C Basics; Memory and Binary Systems
- Lecture 04: Chapter 1
  – 1.6 – 1.7: Performance, power and technology trends
- Lecture 05:
  – 1.8 - 1.9: Multiprocessing and benchmarking

The Computer Revolution

- **Progress in computer technology**
  - **Underpinned by Moore's Law**
    - **Every two years, circuit density ~= increasing frequency ~= performance, double**
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive



Generation Of Computers



https://solarrenovate.com/the-evolution-of-computers/

New School Computer

Classes of Computers

- Personal computers (PC) --> computers are PCs today
  - General purpose, variety of software
  - Subject to cost/performance tradeoff

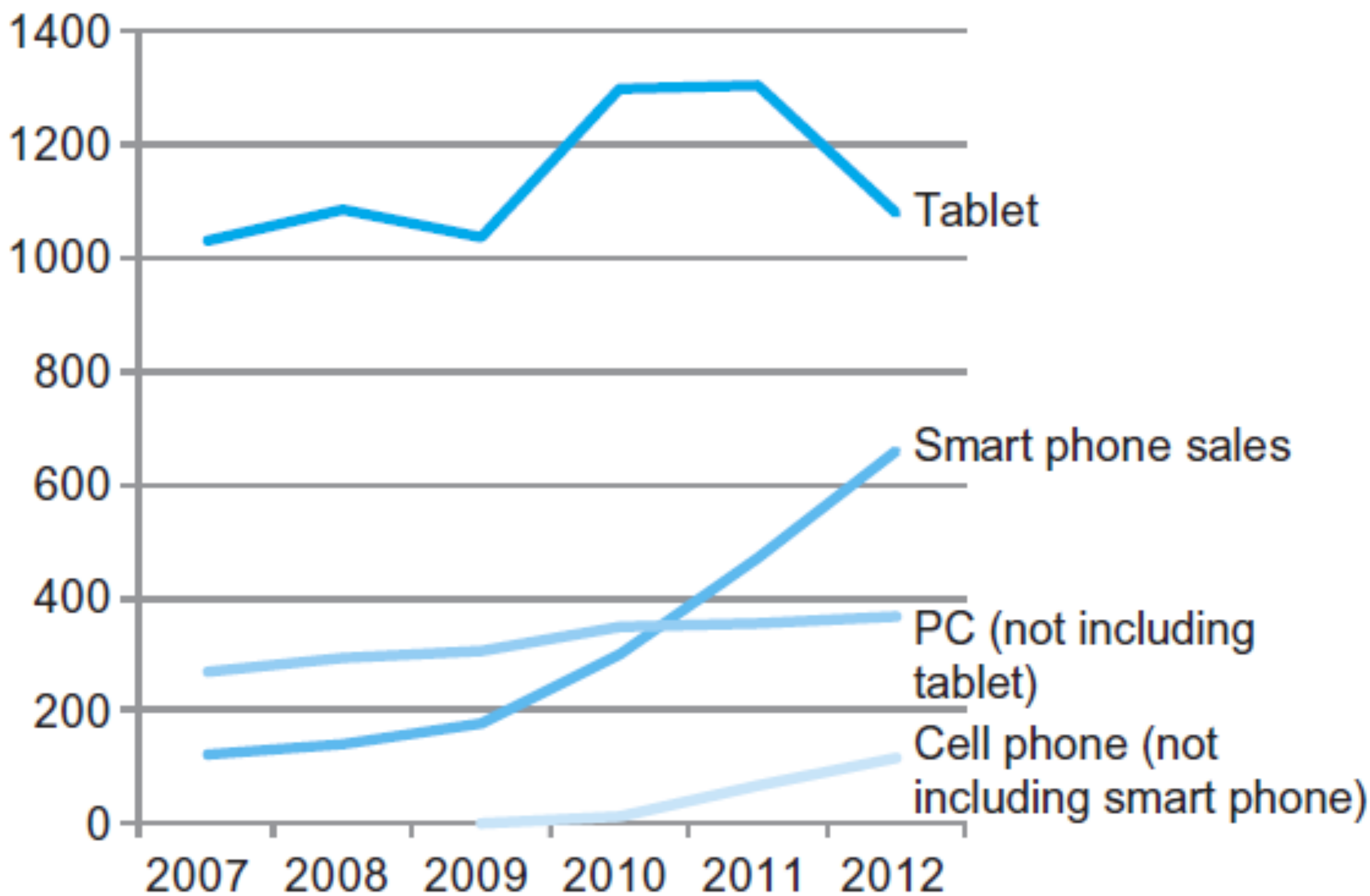
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

Classes of Computers

- Supercomputers
  - High-end scientific and engineering calculations, e.g. for forecasting weather and hurricane
  - Highest capability but represent a small fraction of the overall computer market

- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The PostPC Era



The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses

- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

What You Will Learn

- How programs are translated into the machine language → **Usability**
  - And how the hardware executes them
- The hardware/software interface
- What determines program **performance**
  - And how it can be improved
- How hardware designers improve **performance**

All those that make you more than a programmer, and much more.

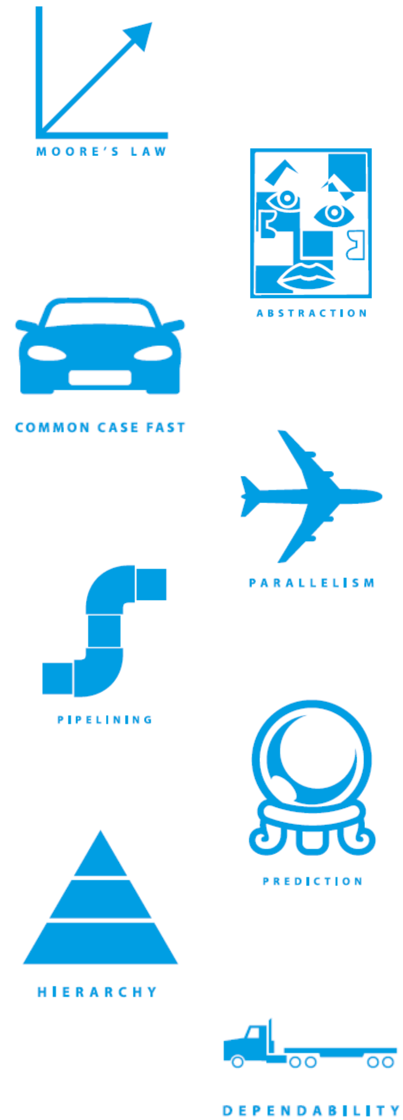
Understanding Performance

- **Performance:**
  - **Hardware performance, peak or theoretical performance, e.g. frequency**
  - **Application performance, user experience, how long to get a computation done**
- **Performance** is like nutrition of food: what is in the raw food is (much) less than what you would digest in your body
  - The process of transformation
  - Application performance you see is less than the hardware/vendo

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Eight Great Ideas

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



Great Idea: "Moore's Law"

Gordon Moore, Founder of Intel

- 1965: since the integrated circuit was invented, the number of transistors/inch^2 in these circuits roughly doubled every year
- **From 1975: Circuit complexity doubles every two years**
  - **→ In a room, number of persons double every two years**
  - **How: shrink the person by half every two years (who can?)**
- **Increasing circuit density ~= increasing frequency ~= increasing performance**

- Transparent to users
- An easy job of getting better performance: buying faster processors (higher frequency)

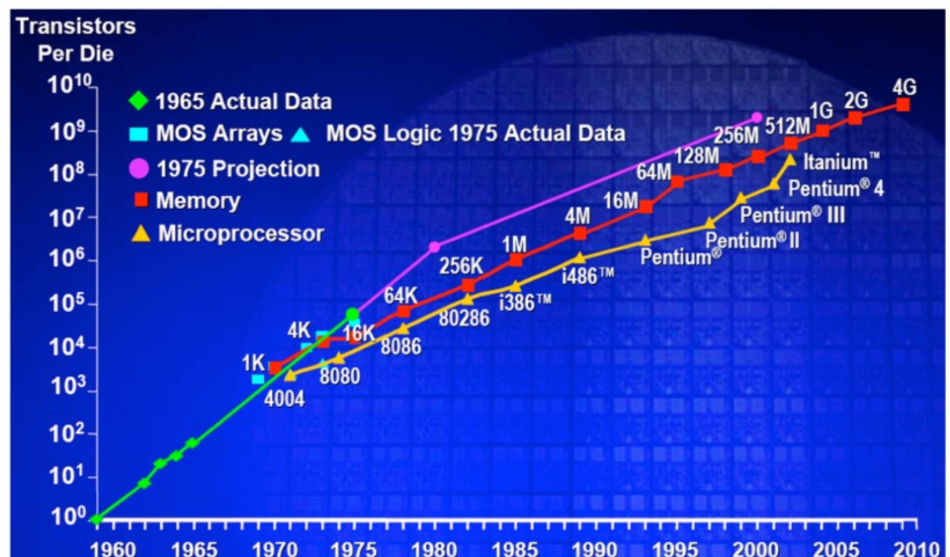


Image credit: Intel

Moore's Law in Reality and Test

Year	Transistors/chip	Transistor tech (size)	CPU Speed (frequency)
1998			
2000	500 M	200 nm	2 GHz
2002			
2004			

M1 Chip

- Made using TSMC's 5nm process (N5)
- 16 billion transistors
- 4 high-performance "Firestorm" cores
- 4 energy-efficient "Icestorm" cores
- 3.2GHz CPU clock speed
- CPU cores first seen in the iPhone 12 lineup's A14 Bionic chip
- 8-core GPU
- Support for 8GB or 16GB unified

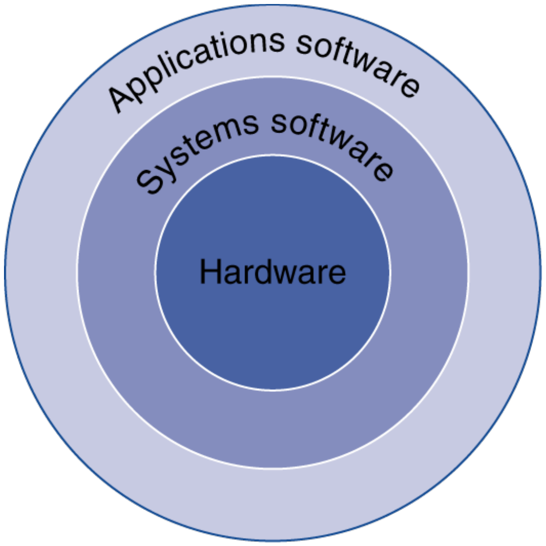
M2 Chip

- Made with TSMC's enhanced 5nm process (N5P)
- 20 billion transistors
- 4 high-performance "Avalanche" cores
- 4 energy-efficient "Blizzard" cores
- 3.49GHz CPU clock speed
- CPU cores first seen in the iPhone 13 lineup's A15 Bionic chip
- 10-core GPU

https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures
https://www.macrumors.com/guide/m1-vs-m2-chip/

Below Your Program

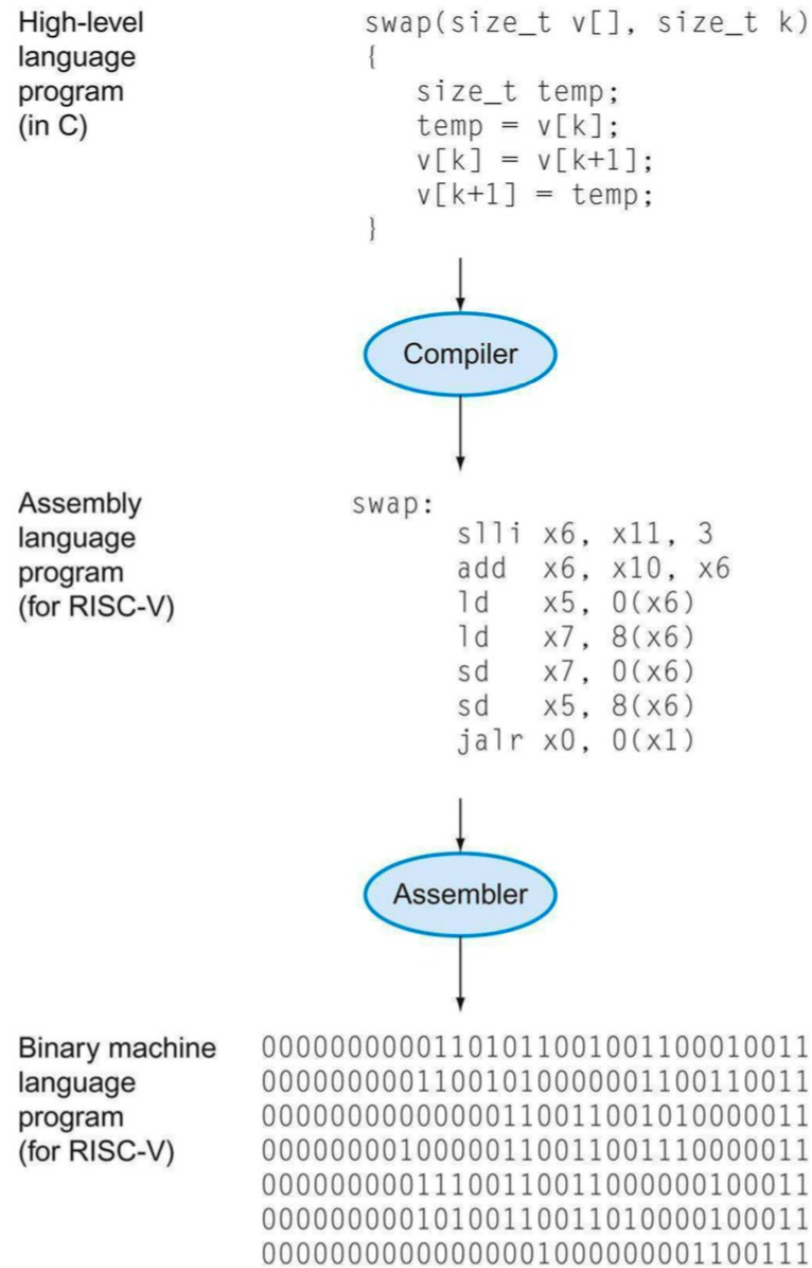
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers



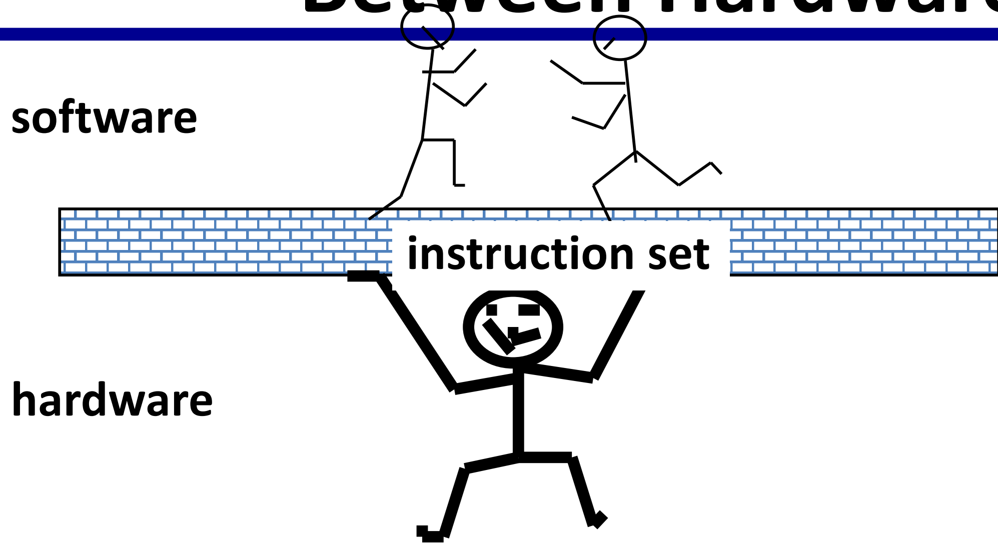
Levels of Program Code

Another Great Idea: Abstraction

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for **productivity and portability**
- **Assembly language**
  - **Textual representation of instructions**
  - **Interface between HW and SW**
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

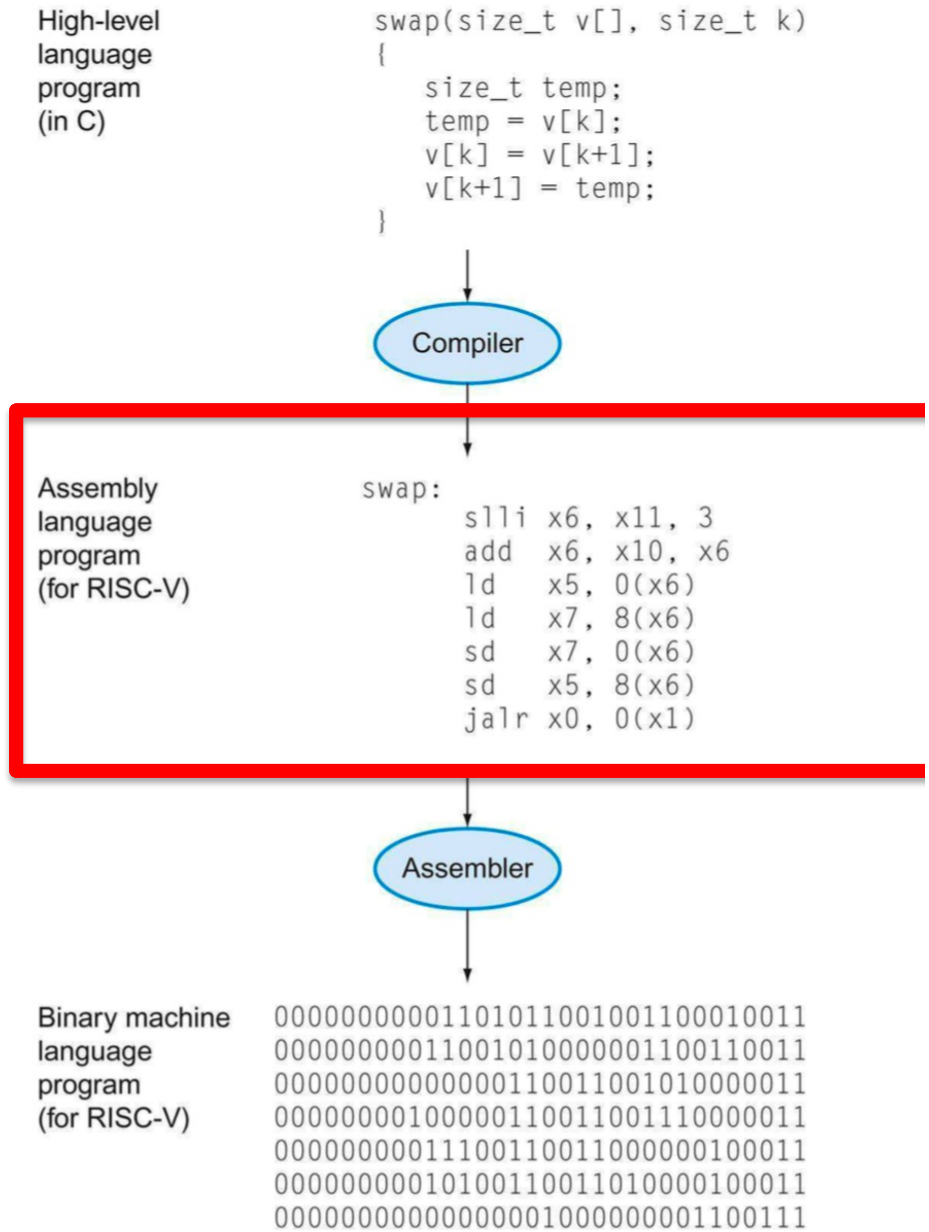


Instruction Set Architecture: The Interface Between Hardware and Software



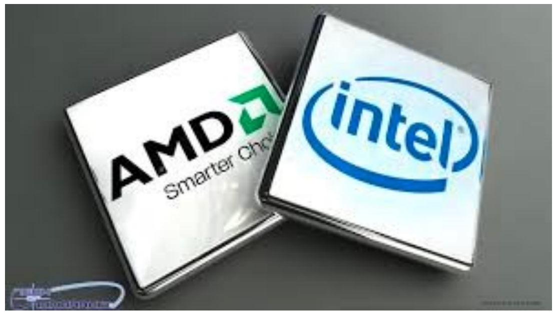
- The words of a computer language are called *instructions*, and its vocabulary/dictionary is called an *instruction set*
  - lowest software interface, assembly level, to the users or to the compiler writer

Instruction Set Architecture – A type of computers



Major Types of ISA (Computers)

- X86: Intel and AMD, Desktop, laptop, server market



- ARM: embedded, smart pad, phone, etc, now moving to laptop/server

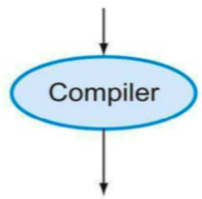
- Power (mainly IBM) and SPARC (mainly Oracle and Fujitsu): server market
- **RISC-V: fastest growing one, embedded so far**
  - **This class uses**

RISC-V®

Levels of Program Code to Multiple Target Architectures

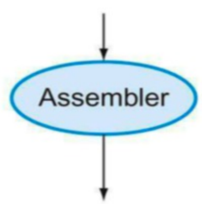
High-level language program (in C)

```
swap(size_t v[], size_t k)
{
    size_t temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



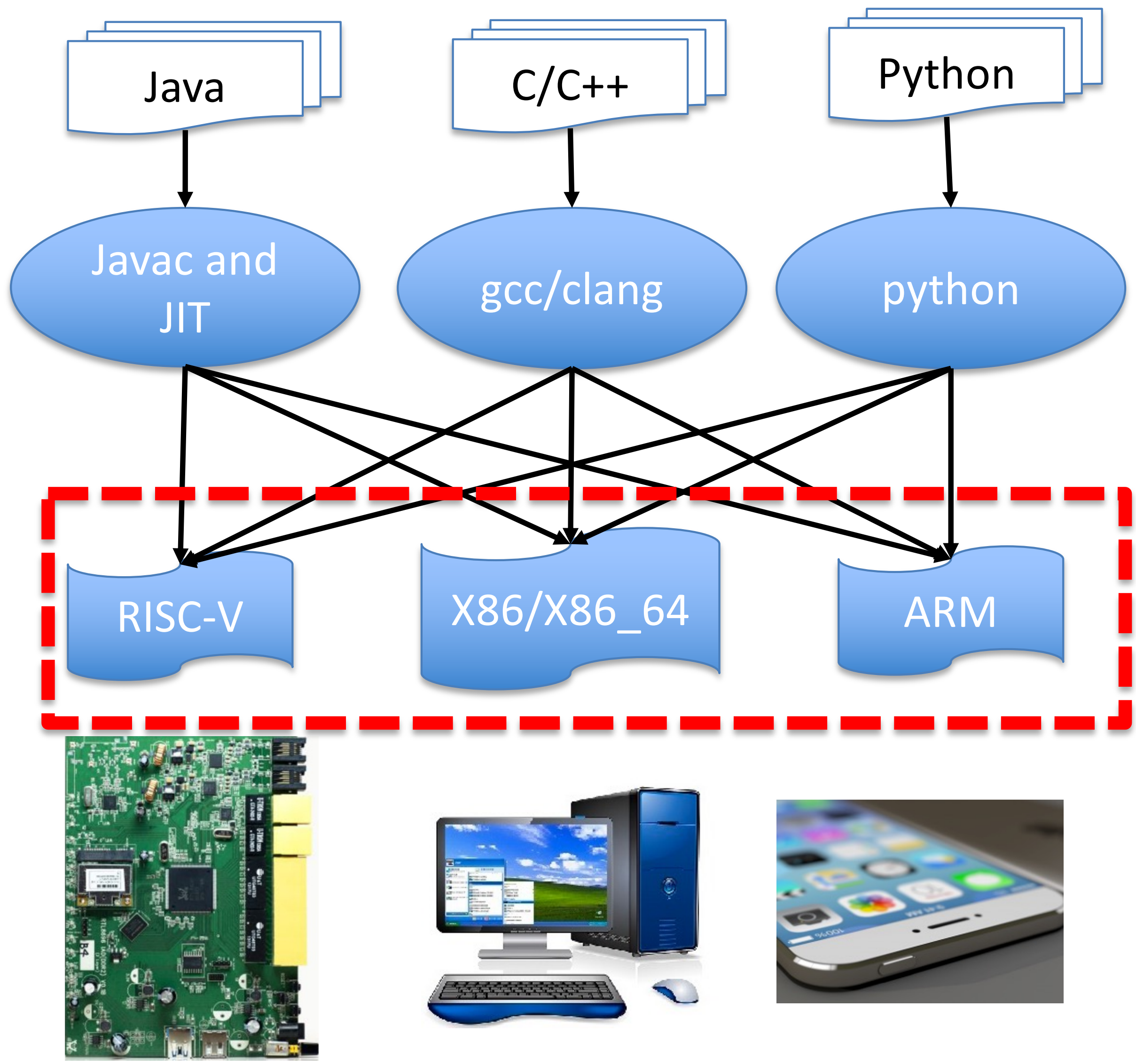
Assembly language program (for RISC-V)

```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      ld   x5, 0(x6)
      ld   x7, 8(x6)
      sd   x7, 0(x6)
      sd   x5, 8(x6)
      jalr x0, 0(x1)
```



Binary machine language program (for RISC-V)

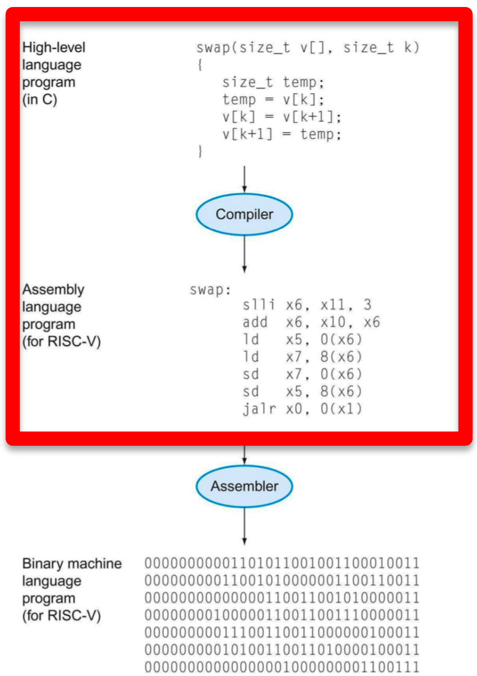
```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```



X86_64 Assembly Example

Using “-S” compiler flag to translate high-level code to assembly instructions

```
yanyh@vm:~$ uname -a
Linux vm 4.4.0-170-generic #199-Ubuntu SMP Thu Nov 14 01:45:04 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
yanyh@vm:~$ gcc -S swap.c
yanyh@vm:~$ cat swap.s
        .file   "swap.c"
        .text
        .globl  swap
        .type   swap, @function
swap:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movl    %esi, -28(%rbp)
        movl    -28(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %eax
        movl    %eax, -4(%rbp)
        movl    -28(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rax, %rdx
        movl    -28(%rbp), %eax
        cltq
        addq    $1, %rax
        leaq    0(,%rax,4), %rcx
        movq    -24(%rbp), %rax
```



- X86_64 is ISA Architecture for most Intel and AMD desktop/server CPUs
- RISC-V is one ISA
- ARM is another ISA
  - Most cellphone/smartphone are ARM CPUs

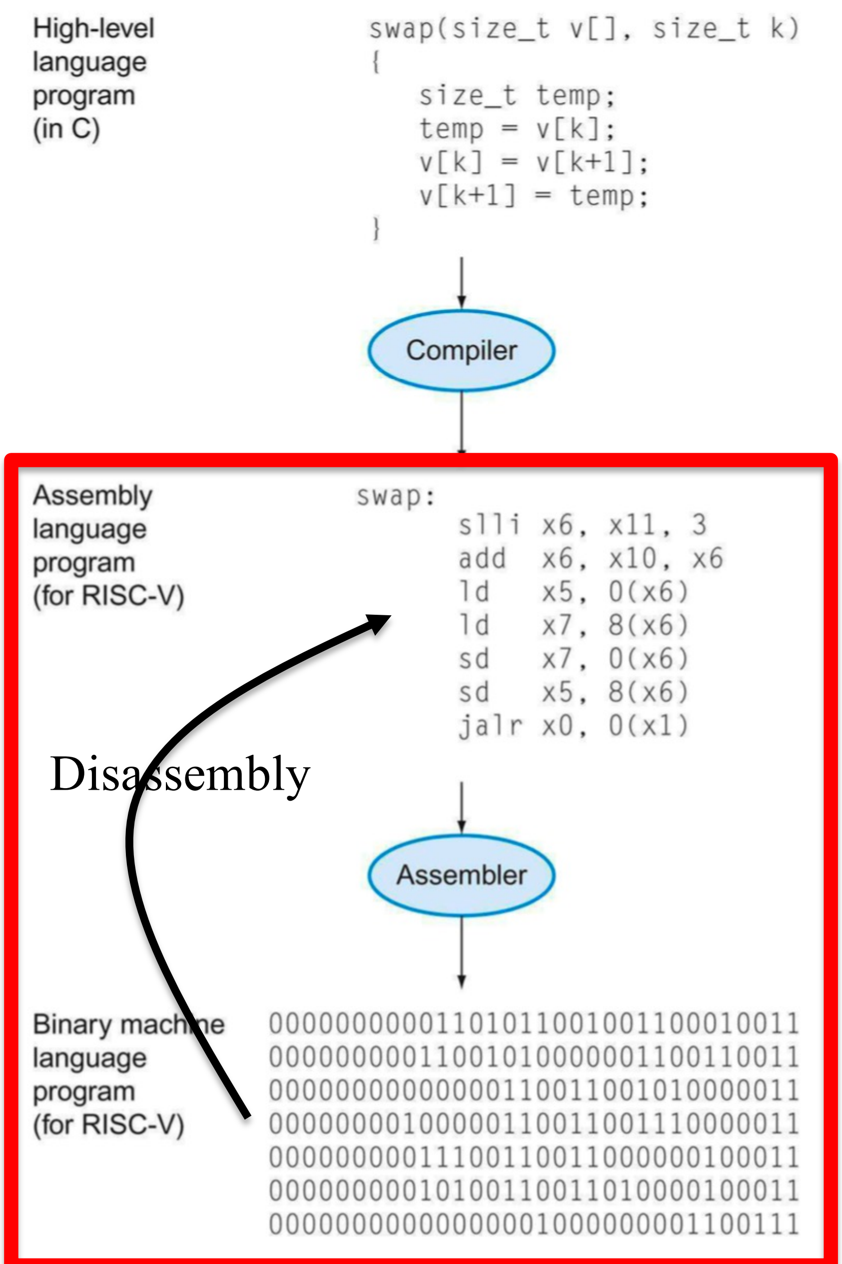
Try the highlighted command for swap.c from the terminal of https://repl.it/languages/c

https://passlab.github.io/ITSC3181/exercises/swap/

X86_64 Assembly Example

https://repl.it/languages/c

Disassembly a machine binary code to assembly instructions using "objdump"




```
yanyh@vm:~$ gcc -c swap.c
yanyh@vm:~$ objdump -D swap.o

swap.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <swap>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 89 7d e8             mov    %rdi,-0x18(%rbp)
   8:   89 75 e4                mov    %esi,-0x1c(%rbp)
   b:   8b 45 e4                mov    -0x1c(%rbp),%eax
   e:   48 98                   cltq
  10:   48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
  17:   00
  18:   48 8b 45 e8             mov    -0x18(%rbp),%rax
  1c:   48 01 d0                add    %rdx,%rax
  1f:   8b 00                   mov    (%rax),%eax
  21:   89 45 fc                mov    %eax,-0x4(%rbp)
  24:   8b 45 e4                mov    -0x1c(%rbp),%eax
  27:   48 98                   cltq
  29:   48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
  30:   00
  31:   48 8b 45 e8             mov    -0x18(%rbp),%rax
  35:   48 01 c2                add    %rax,%rdx
  38:   8b 45 e4                mov    -0x1c(%rbp),%eax
  3b:   48 98                   cltq
  3d:   48 83 c0 01             add    $0x1,%rax
  41:   48 8d 0c 85 00 00 00    lea    0x0(,%rax,4),%rcx
  48:   00
  49:   48 8b 45 e8             mov    -0x18(%rbp),%rax
  4d:   48 01 c8                add    %rcx,%rax
  50:   8b 00                   mov    (%rax),%eax
```

Exercise: Inspect ISA for swap

- Swap example
  - https://passlab.github.io/ITSC3181/exercises/swap/
- Check
  - swap.x86_64.s,
  - swap.x86_64_objdump.txt

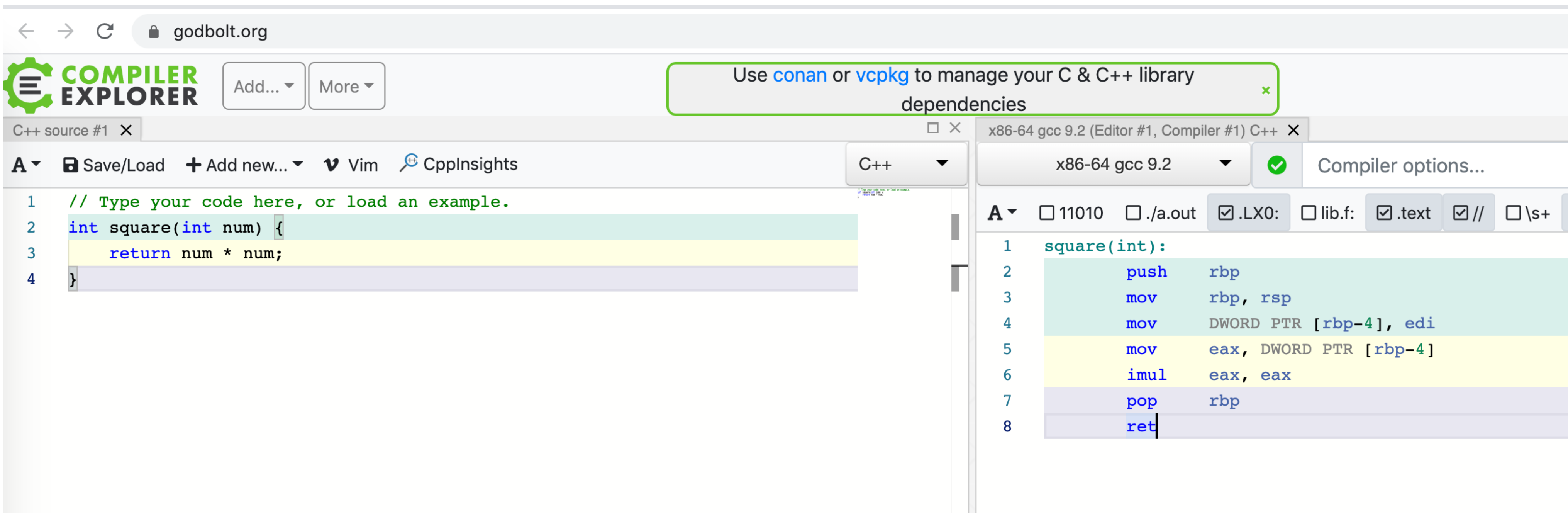
- Generate and execute
  - gcc -s swap.c -o swap.x86_64.s
  - gcc -c swap.c
  - objdum -D swap.o > swap.x86_64_objdump.txt

- For how to compile and run Linux program
  - https://passlab.github.io/ITSC3181/notes/lecture01_LinuxCProgramming.pdf

- Other system commands:
  - cat /proc/cpuinfo to show the CPU and #cores
  - top command to show system usage and memory

Compiler Explorer

- Explore other ISA assembly from Compiler Explorer at https://godbolt.org/
- Work on Lab 01 Tomorrow

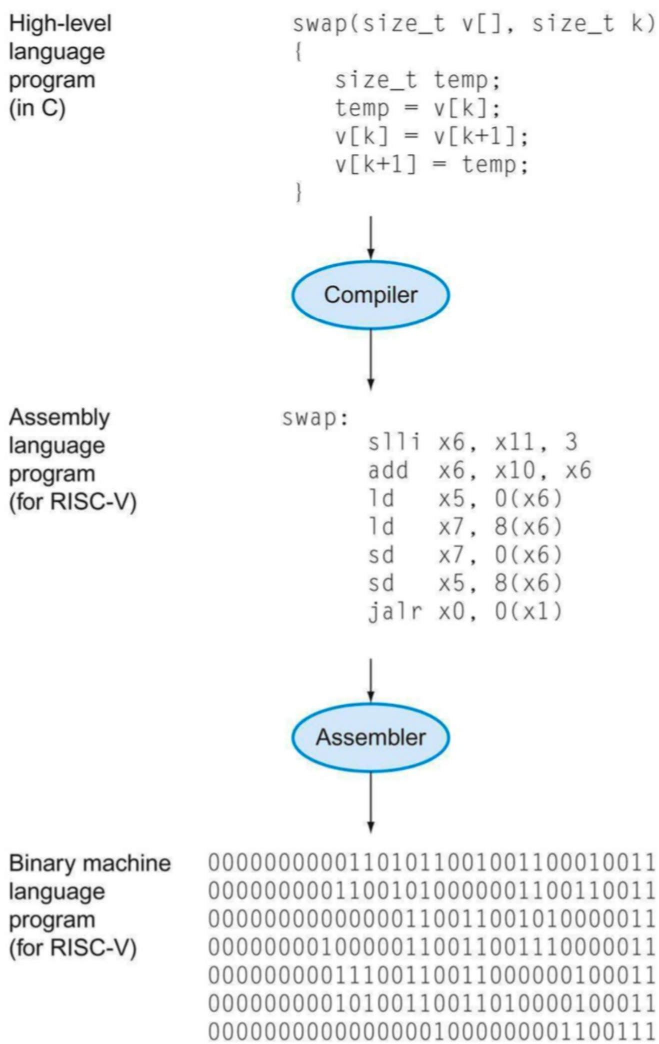


Great Idea: More on Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

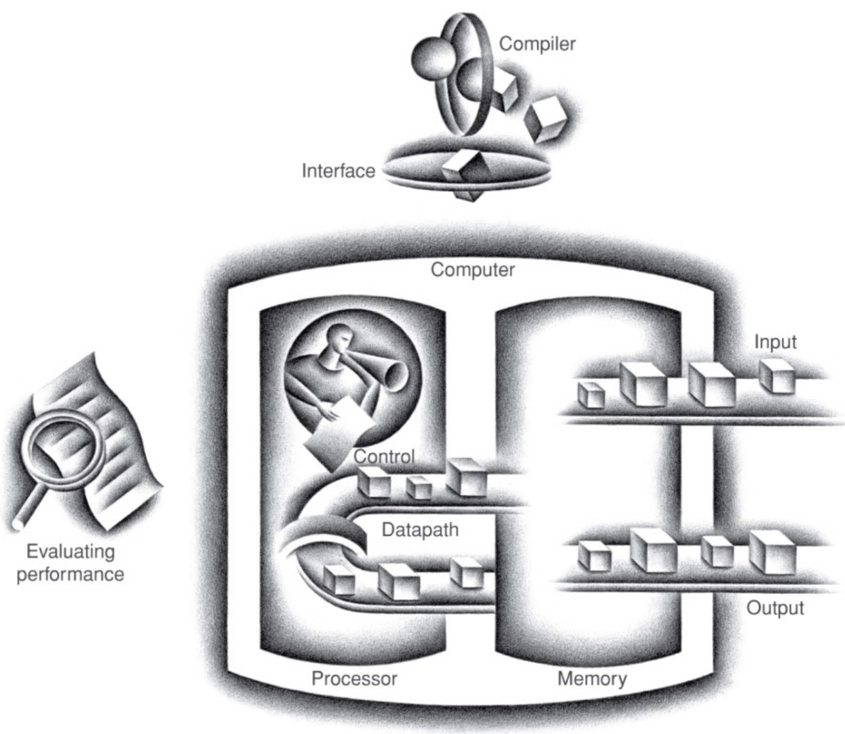
- Another example of abstraction:
  - Java Interface and Class



Components of a Computer

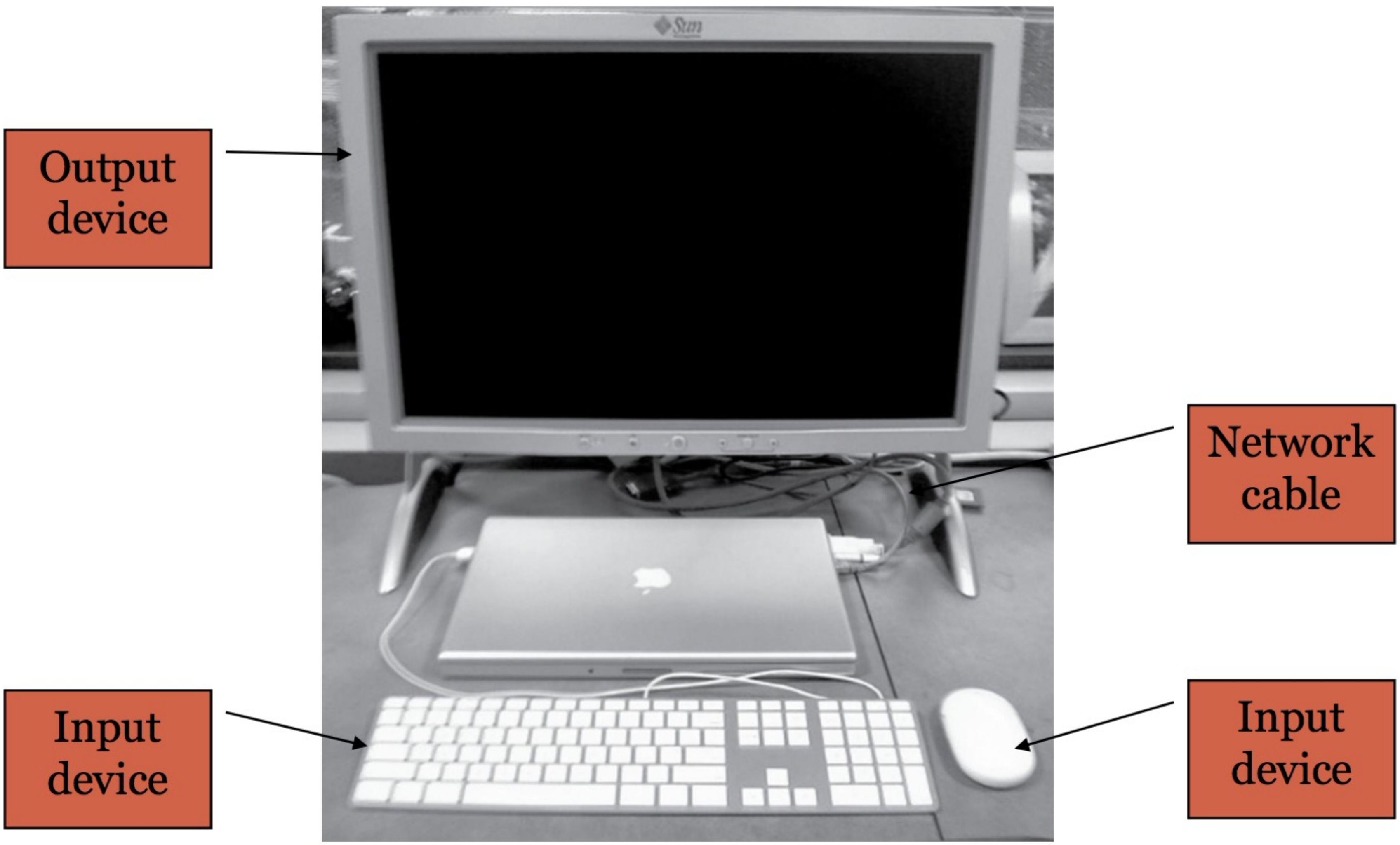
§1.4 Under the Covers

The BIG Picture

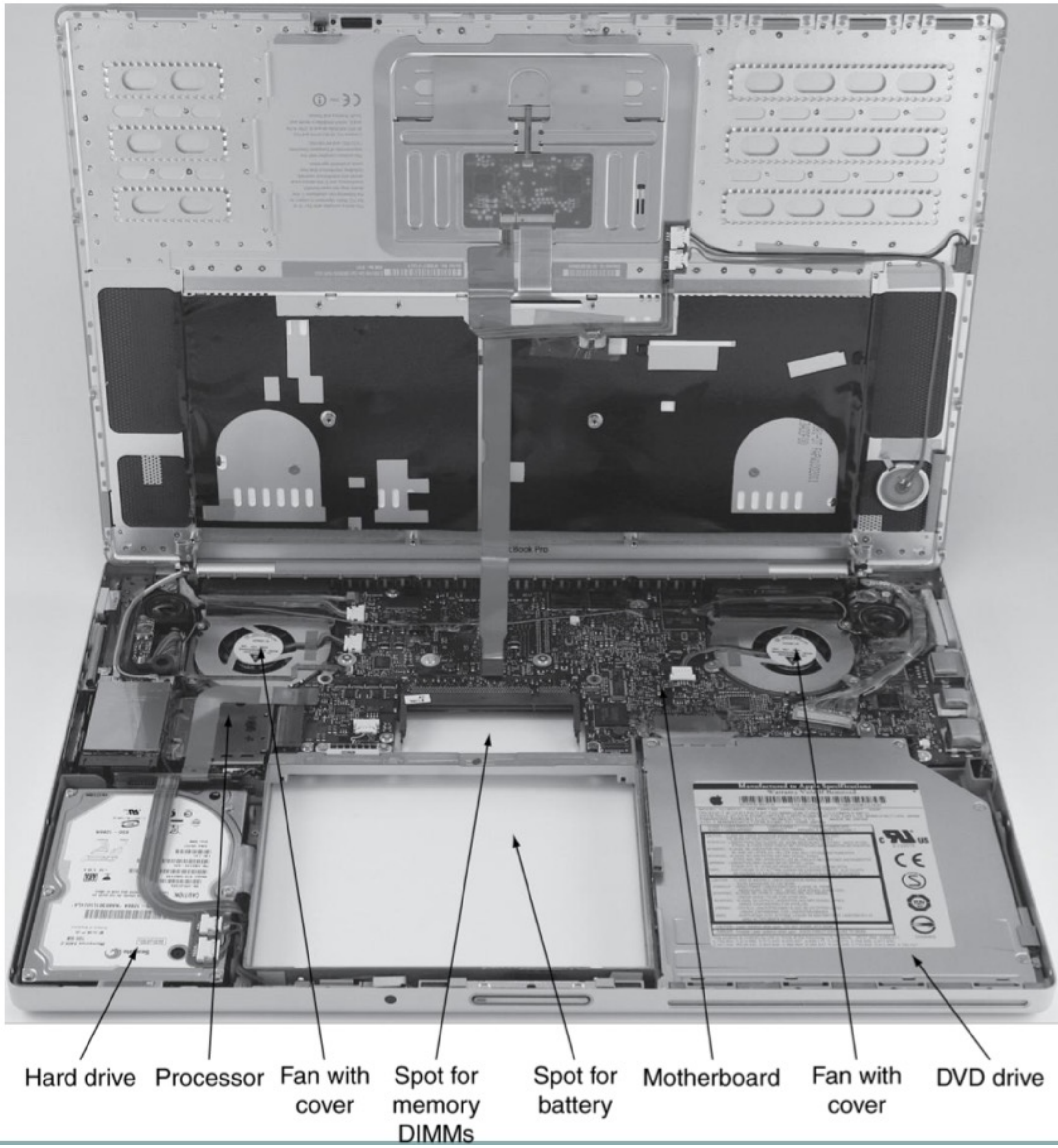


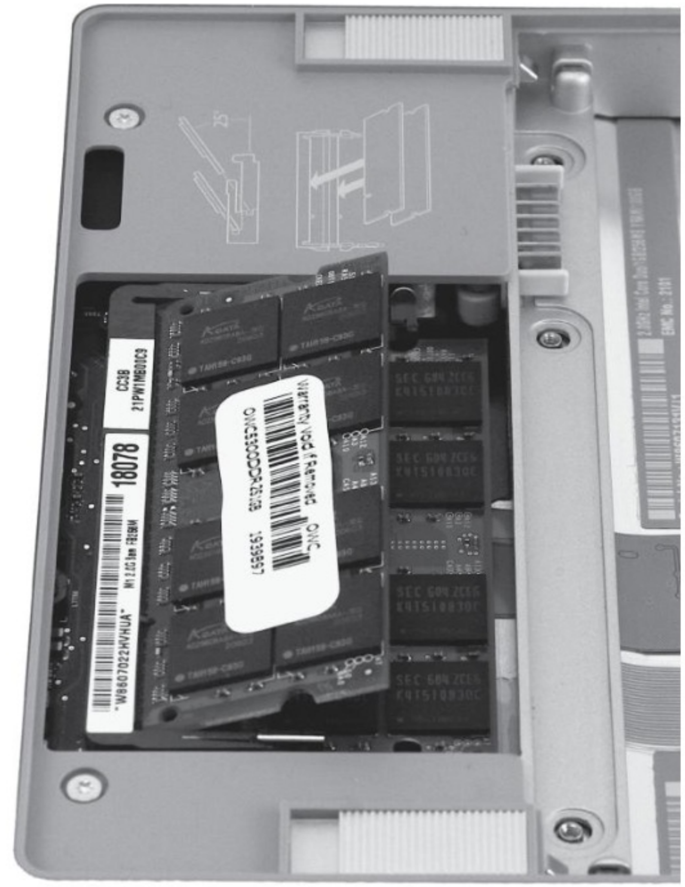
- Same components for ALL kinds of computer
  - Processor (functional unit, control logic and data path)
  - Memory
  - Input/out devices
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Components of a Computer: Input/Output



Open the Box: a Laptop



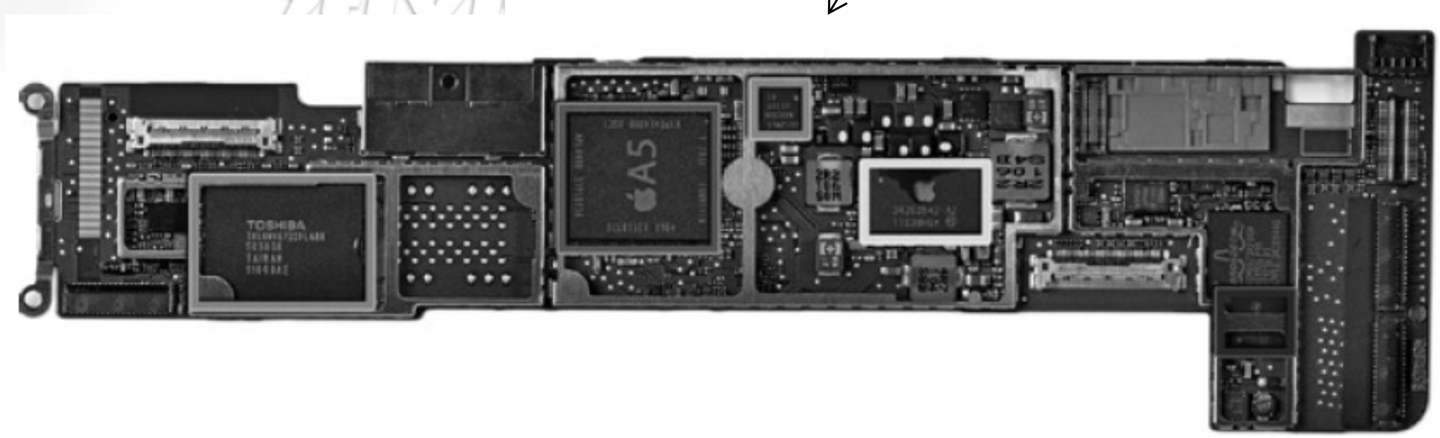


Opening the Box: an IPhone

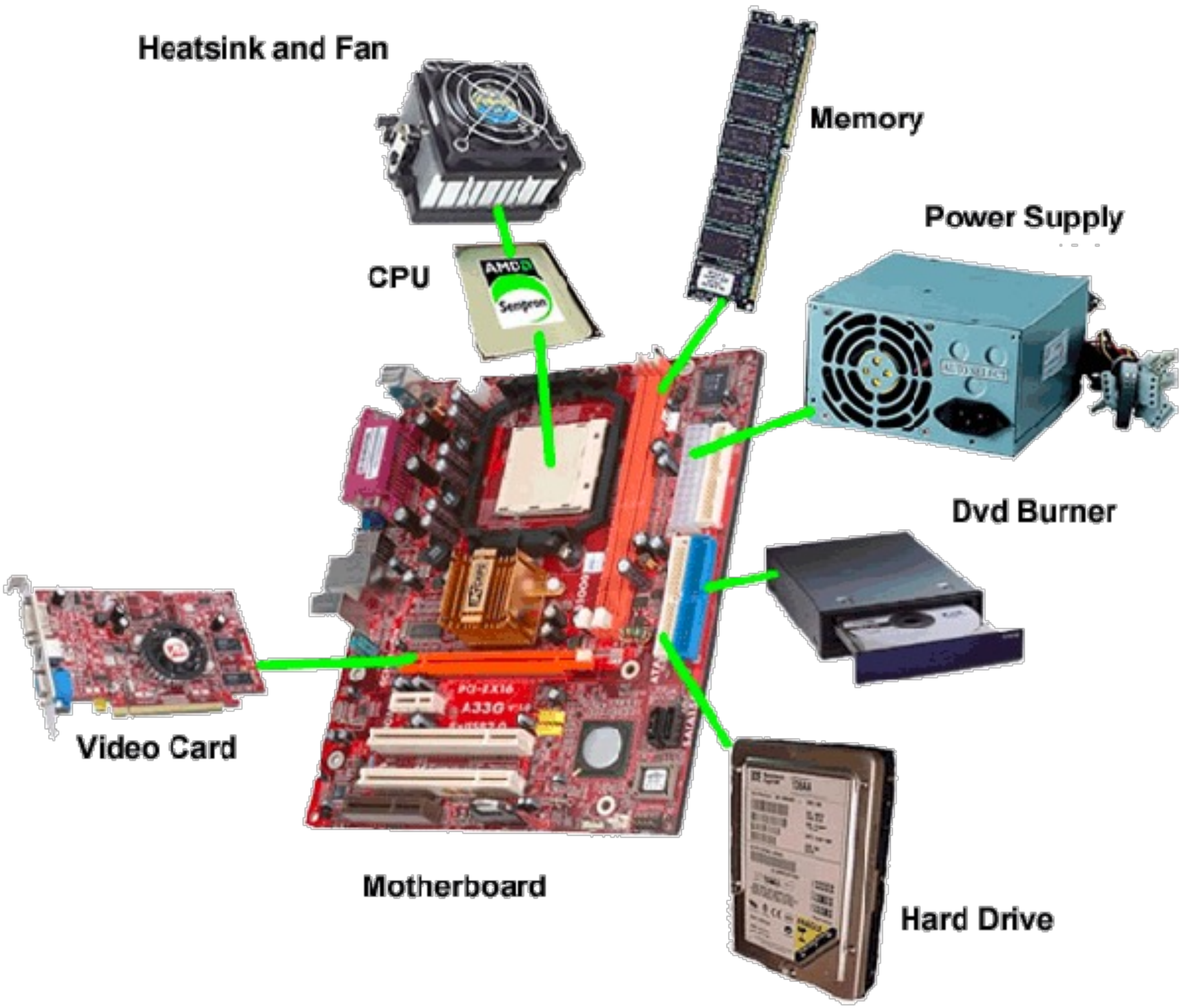
Capacitive multitouch LCD screen

3.8 V, 25 Watt-hour battery

Computer board

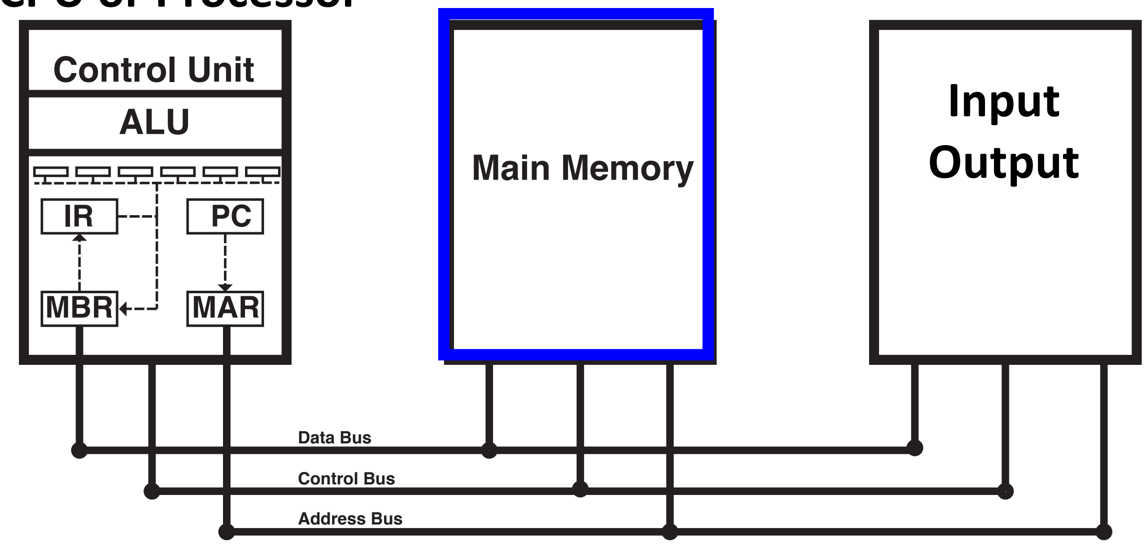


Desktop Computer Components

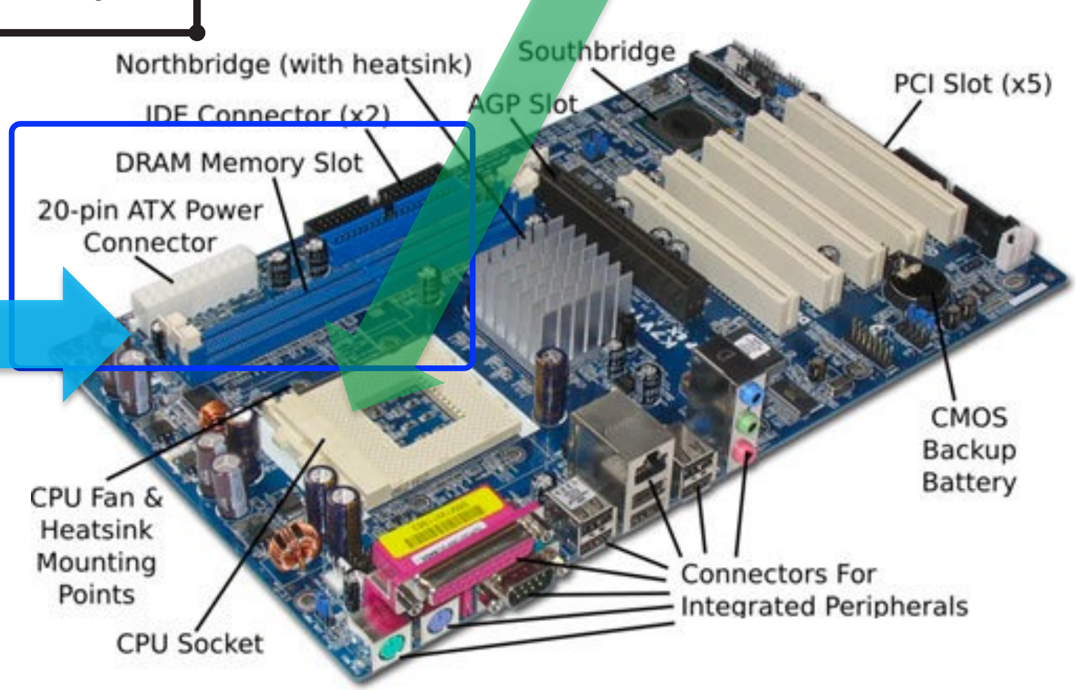


Main Memory (DRAM) of a Computer

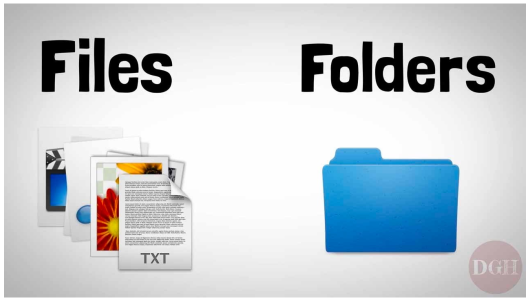
CPU or Processor



CPU is also called a chip.



Everything is Data Stored in Files



- Source code, executable, object are all files
  - Files: Hello.c, sum_full.c, sum
  - Folder: ., .., /home/yanyh, etc
- Compiler, OS kernel, etc are all stored as files
  - gcc, vmlinuz-4.4.0-104-generic
- Information about files/folders and data are also files
  - Metadata

- **Files need to be loaded to memory in order to be processed**
  - **An app or executable is a file (multiple files) that contains the instructions in binary form and other data needed to execute the program.**

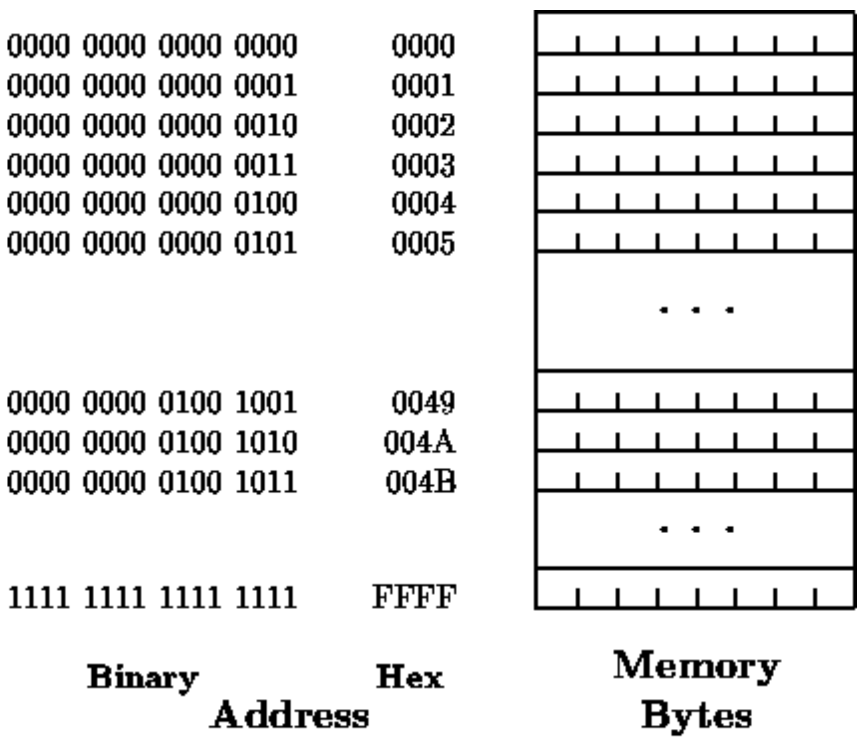
Loading a file for a command to Memory

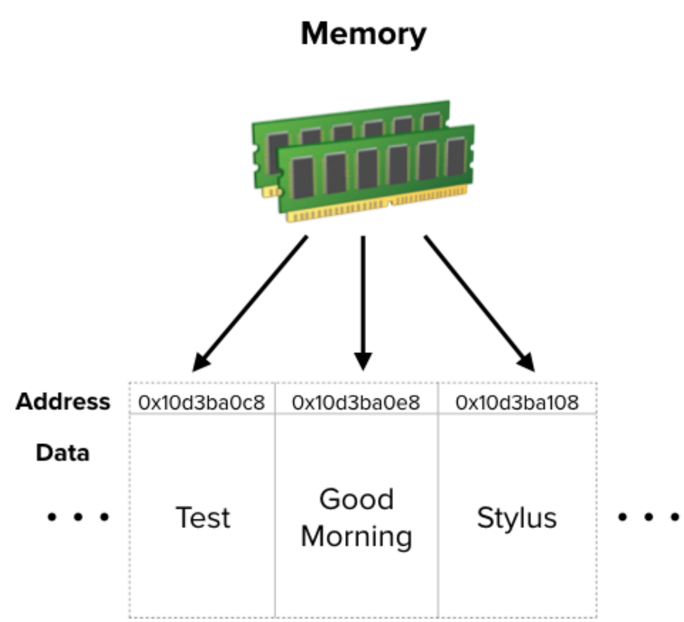
- To load a file from disk into memory

- Loading: To run an app=> load the app executable file to memory and run the instructions of the program
  - `yanyh@vm:~/sum$ ./sum 1000000`
    - ./ is to specify the path of sum file
  - To execute any linux command, e.g. “ls, cd”, etc.
  - Double click an icon to execute app:
- The runtime instance of an executable is called a “**process**”
  - It occupies memory, and uses resources (files, sockets, etc).
  - It executes its threads (machine instructions).
  - See the processes of the system using “ps” command, Windows “task manager”, and Mac OS X “Activity Monitor”

Memory and Address

- Memory are accessed via the address of memory cells that store the value
  - int a = A[i]; //a, A[i] are symbolic representation of memory addresses
    - Read value from a memory location whose address is represented by A[i];
    - Write value to a memory location whose address is represented by a



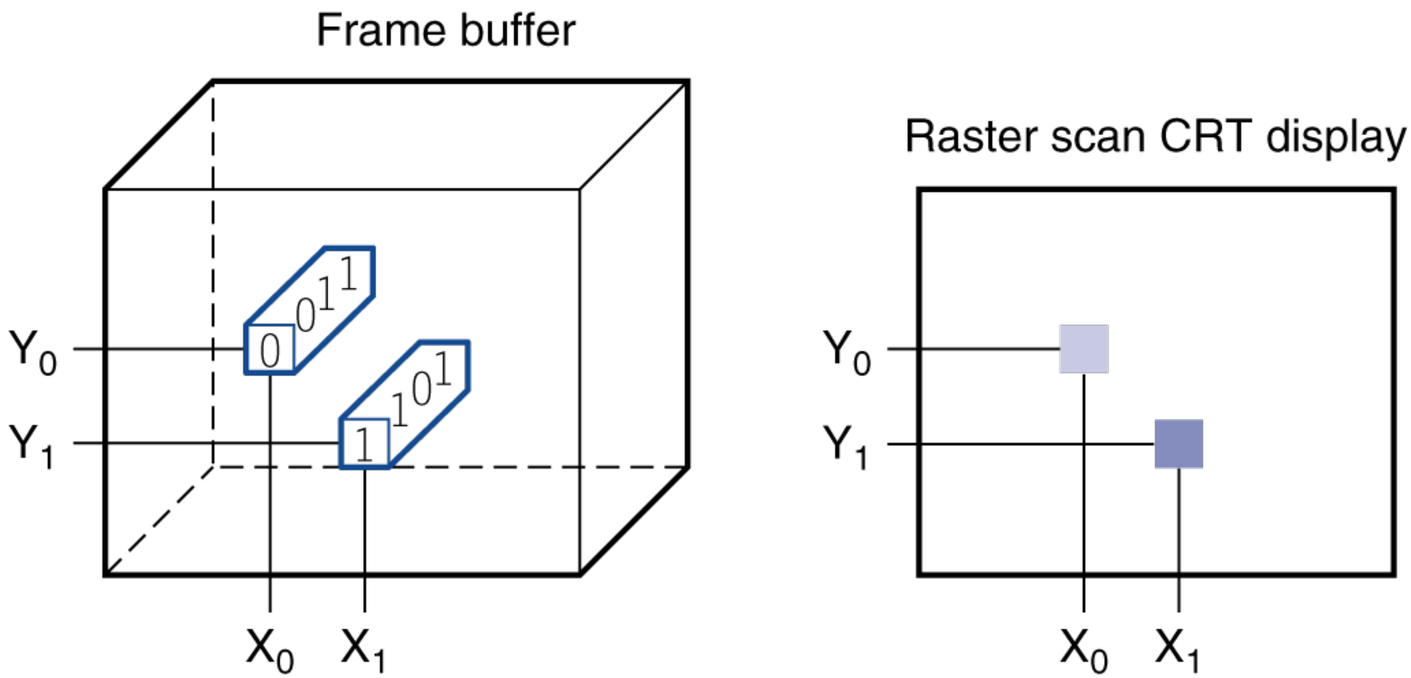


Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

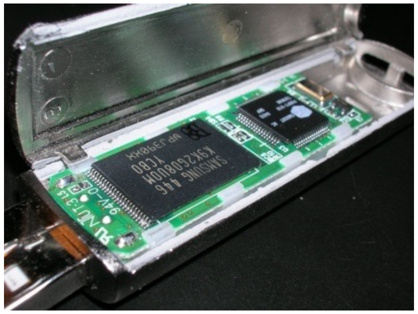
Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



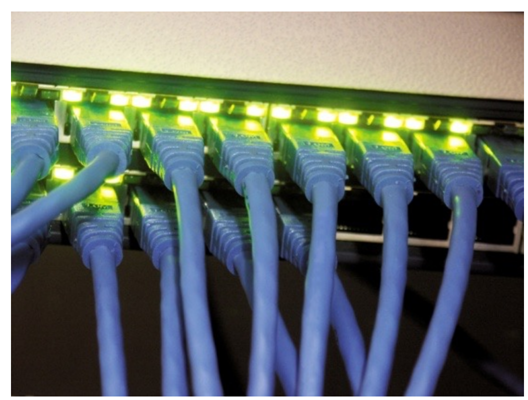
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



End of Lecture 01

Inside the Processor (CPU)

- Functional units: performs computations
- Datapath: wires for moving data
- Control logic: sequences datapath, memory, and operations
- Cache memory
  - Small fast SRAM memory for immediate access to data

Apple A5