

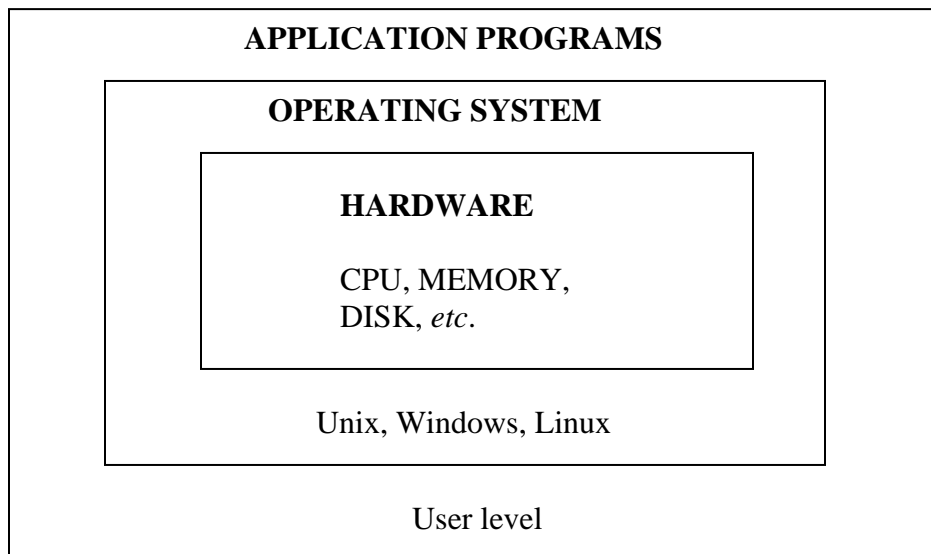
# Lecture 1

## Introduction

Why do we study Computer Organization?

- Most people think of computer as a gadget/machine.
- Inside View vs. Outside View
  - o Outside: Any user, application level, input/output devices
  - o Inside: Processor, memory, buses, storage
  - o Example: Car. Any user can drive it, however it takes a mechanic or knowledgeable person to understand/fix the insides.
- This course is an introduction to the inside of the computer: the components and their functionality.
- Goal: To make you a better computer professional, and not just a programmer.
  - o If you understand the inter-workings of the device you are programming for, you will be a better programmer.

### Simplified View of a Computer's Structure



# Computer Organization

Modern Machines are multi-level.

## -----APPLICATION PROGRAMMER-----

### **Level 6: Application - Problem-oriented language – High Level Language**

(C#, C++, C, Java, Python, LISP, Prolog, etc)

→ Translated by Compiler

### **Level 5: Assembly Language**

Symbolic form for underlying language. Easier to understand.

→ Translated by Assembler (conversion between assembly and ISA)

## -----SYSTEM PROGRAMMER-----

### **Level 4: Operating System machine**

Instructions directly in ISA directly run on hardware, otherwise it is interpreted by OS into instructions for ISA

→ Partial Interpretation (OS)

### **Level 3: Instruction Set Architecture (ISA) – Machine Language**

Machine's Instruction Set (different per machine, instructions that the micro-architecture understands)

→ Interpretation (microprogram) or direct execution

### **Level 2: Micro-architecture Level (Registers, ALU → Data Path)**

Control: older machines microprogram instructions; modern machines partial hardware control.

S/W control: interpreter for the ISA instructions. Fetch, examine, execute.

→ Hardware

### **Level 1: Digital Logic Level (AND, OR's, Registers, Gate Level Logic)**

### **Level 0: Device Level (Transistors)**

## Differences between Levels 0-4 & Levels 5-6:

- **Who works with them: System programmer vs. Application programmer**
- **How the levels are supported: Translation vs. Interpretation**

Translation: program written in language L1. Replace each instruction by equivalent sequence of instructions in language L0. An L0 translation of the program in L1 is thus generated (and then executed).

Interpretation: a program written in L0 takes programs written in L1 as input data. The L0 program then carries the instructions out by examining each L1 instruction in turn and executing the equivalent sequence of L0 instructions directly. There is NO actual generation of a new program in language L0.

- **Nature of Language provided/used.**

Lower levels are numeric languages. Long series of numbers, typically in HEX (hexadecimal – base 16) or in Binary. GREAT for computers, but not for humans! Higher levels: Symbolic form (*add, sub, mux, etc.*)

## **SUMMARY**

- Computers are designed in levels, each built on top of its predecessor.
- Each level is a distinct abstraction, involving different objects & operations.
- Each level allows suppression of detail and reduced complexity, while making it easier for the programmer to understand (similar to programming abstractions in OOP).

**READ Chapter 1, Sections 1.1-1.2**

**Next Time:** Bits, Bytes, words, and ASCII