**Chapter 1**

* **Programming language, compiler, architecture** – determines number of computer instructions for each source-level statement
* **Processor and memory system** – determines how fast instructions can be executed
* I/O system (hardware & operating system) – determines how fast I/O operations can be executed

**EIGHT GREAT IDEAS**

* **Moore’s Law** – integrated circuit resources double every 18-24 months
* Use abstraction to simplify design – lower-level details are hidden to offer a simpler model at higher levels.
* **Make the common case fast** – focus on making common case fast rather than optimizing rare case (common case is usually simpler than rare case; easier to enhance)
* **Performance via parallelism** – compute operations in parallel
* **Performance via pipelining** – a particular case of parallelism
* **Performance via prediction** – on average, can be faster to guess and start working rather than waiting until certain. Prediction should be relatively accurate and mechanism to recover should not be too expensive
* **Hierarchy of memories** – fastest, smallest, most expensive memory-per-bit at top of hierarchy, slowest, largest, cheapest memory-per-bit at bottom
* **Dependability via redundancy** – include redundant components to take over when failure occurs AND to help detect failures

**Systems software** – software to provide commonly-useful services (operating systems, compilers, loaders, assemblers)

**Operating system** – interface between user and hardware

-Handles basic input/output operations

-allocates storage and memory

-provides protected sharing of computer among multiple applications using it simultaneously

**Compiler** – translates high-level language statements into assembly language statements

**Binary digit (bit)** – one of two numbers in base 2 that are components of information

**Instruction** – command that computer hardware understands and obeys

**Assembler** – program that translates symbolic version of instructions into binary version

**Assembly language** -symbolic representation of machine instructions

**Machine language** – binary representation of machine instructions; binary language that the machine understands

**High-level programming language** – portable language composed of words and algebraic notation that can be translated by a compiler into assembly language and then onwards to machine language

1. High-Level Programming Language
2. Translated to Assembly Language through Compiler
3. Translated to Machine Language through Assembler

**FIVE CLASSIC COMPONENTS OF COMPUTER**

1. Input
2. Output
3. Memory
4. Datapath
5. Control

**Datapath** and **Control** can be combined and referred to as **Processor**

**Integrated Circuit (chip)** – device combining dozens to millions of transistors

**Central Processing Unit (CPU) or Processor** – Active part of computer containing datapath and control that adds numbers, tests numbers, signals I/O devices to activate, etc.

**Datapath** – component of processor that performs arithmetic operations

**Control** – component of processor that commands datapath, memory, and I/O devices according to instructions of program

**Memory** – storage area in which programs are kept when running and that contains data needed by the running programs

**Dynamic Random Access Memory (DRAM)** – Memory built as integrated circuit; provides random access to any location

**Cache Memory** – small, fast memory acting as buffer for larger, slower memory

**Static Random Access Memory (SRAM)** – also built as integrated circuit, but faster and less dense than DRAM

**Instruction Set Architecture (Architecture)** – abstract interface between hardware and lowest-level software encompassing all information necessary to write a machine language program that runs correctly, including instructions, registers, memory access, I/O, etc.

**Application Binary Interface (ABI)** – User portion of instruction set + operating system interfaces used by application programmers; defines standard for binary portability across computers

**Implementation** – Hardware that obeys architecture abstraction

**Main Memory** – volatile

**Secondary Memory** – non-volatile

**1.6 – PERFORMANCE**

**Response Time (Execution Time) –** total time required for the computer to complete a task

**Throughput (Bandwidth)** – # of tasks completed per unit of time

**FORMULAS FOR DEFINING PERFORMANCE**

**performance(x) = 1/execution time(x)**

**COMPARING PERFORMANCE OF X AND Y**

**If performance(x) > performance(y)**

Then,

**1/execution time(x) > 1/execution time(y)**

Then,

**execution time(y) > execution time(x)**

Therefore,

**performance(x)/performance(y) = execution time(y) / execution time(x) = n**

**MEASURING PERFORMANCE**

**CPU Execution Time (CPU Time)** – Actual time computer spends computing for specific task

**Clock Cycle** **or Clock Period** – time for one CLOCK PERIOD, usually of processor clock, running at constant rate

**Clock cycles per instruction (CPI**) – average number of clock cycles per instruction

**Instruction count** – number of instructions executed by a program

**CPU PERFORMANCE AND ITS FACTORS**

**Clock cycle** = 1 / clock rate

**Clock rate** = 1 / clock cycle time

**CPU execution time** = # of CPU clock cycles \* clock cycle time

**OR**

**CPU execution time** =# of CPU clock cycles / clock rate

**CPU clock cycles** = instructions \* CPI

**CPU time** = instruction count \* CPI \* clock cycle time OR (instruction count \* CPI) / clock rate