Chapter 1

Computer Abstractions and Technology

What is a Computer ?

- … ???

What is a Computer ?

Can you solve

$$7x^3 + 5x^2 + 3x + 1 = 0$$

???

What is a Computer ?

Can you solve

$$7x^3 + 5x^2 + 3x + 1 = 0$$

???

If you do not know how to solve a problem, you cannot tell the computer how to do it !!

What is a Computer ?

- A tool invented to help us do calculations
- Computer is capable of basic logic and arithmetic,

  … and nothing else !!!
- Computers are programmed according to algorithms that solve problems …

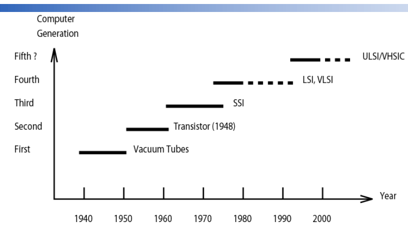
What is a Computer ?

- Computers are conceived/designed with an application domain in mind.
- “General Purpose Computers” have wide range of applications.
- The quest for speed leads to special designs: e.g. DSP, VLIW, SuperScalar.

A Fact of Life

- The faster the computer, the more special purpose it becomes …
- RISC is the right compromise for general purpose computers

A Little History



First (1938-1953): Electronic Numerical Integrator & Computer (ENIAC)
Electronic Discrete Variable Automatic Computer (EDVAC) ← 1st Stored Program

Assembly Language, single user, fixed point arithmetic, CPU assisted I/O

Second (1955-1964): TRADIC (Bell Labs), Stretch (IBM 7030, inst. lookahead & error correction), IBM 7090, CDC 1604, Univac LARC.

HLL; FORTRAN (1956), Cobol (1959), Algol (1960), Compilers, Libraries, Batch, Monitor, Floating Point, I/O Processors, Multiplexed Memory Access

Third (1965-1974): IBM 360/370, CDC 6600, TI ASC, PDP-8,
ILLIAC 4 (1968) ← (8x8 Mesh Connected Parallel Computer)

Microprogramming, Cache, Multiprogramming, Times-shared OS, Intelligent Compilers, Virtual Memory

Fourth (1975-1990): VAX 9000, Cray X-MP, IBM 3090, BBN TC2000
MPP (Goodyear/NASA)

HLL for Scalar & Vector Data, Vectorizing Compilers, Languages and Environments for Parallel Processing

Adapted from K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., Reading, 1993.

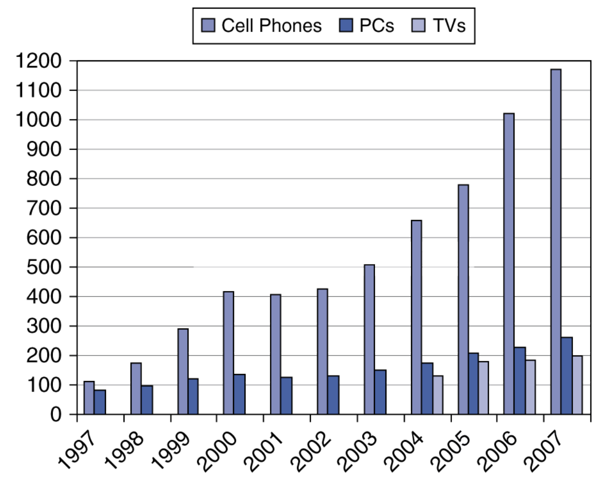
Technology Drives Design

- Computers designed using the latest technological advances.
- Designs always tend to provide best balance of technologies for different parts of the computer (CPU, Memory, I/O)

Classes of Computers

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

The Processor Market



The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Modern Computer Revolution

- Just like John Von Neumann, David Patterson and John Hennessy revolutionized the computer when they introduced

RISC

Modern Computer Revolution

- Computer Architecture is an art …
  - No one design method leading to best computer…
- RISC is a design method/philosophy … derived from observation (experimental computer science): *computers execute 20% of their instructions 80% of the time.*

Modern Computer Revolution

§ 1.1 Introduction

- In this course, we shall design a processor based on the RISC design method.
- Processor is MIPS, an actual commercially available processor …
    - Focus is on design and not on programming. Assumption: you know assembly language programming …

Brief Outline of course

- Define Performance to have a cost function to evaluate designs
- Define the general Instruction Set Architecture, for performance
- Study number crunching and define the arithmetic of the processor
- Implement the processor:
    - Single Cycle
    - Multi-Cycle
    - Pipelining

  The Memory Hierarchy: Technology rules !!

  Advanced topics (time permitting)

What You Will Learn

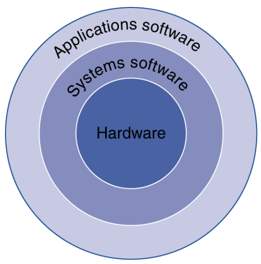
- Programs are translated into the machine language (ISA) – [Compilers course]
  - How does the hardware execute them ?
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance: CPU design – Memory hierarchy design
- Balancing all the components of a system: CPU-Memory-I/O
- What is parallel processing

Understanding Performance

- 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

Below Your Program

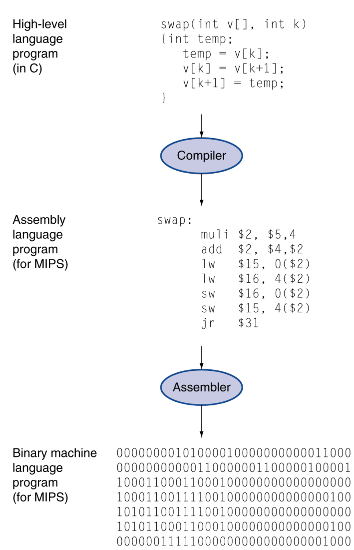
§ 1.2 Below Your Program



- 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

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



Abstractions

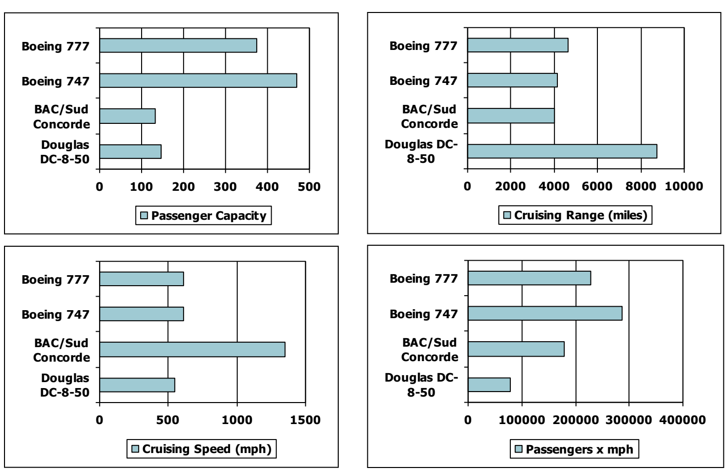
The BIG Picture

- 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

Defining Performance

§1.4 Performance

- Which airplane has the best performance?



Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

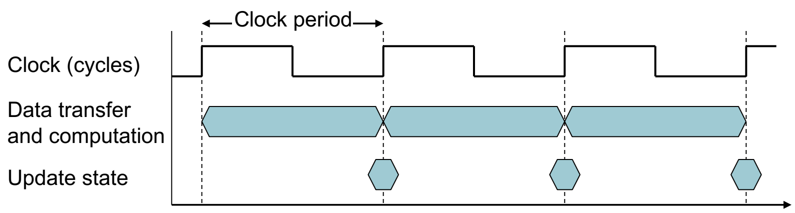
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$
$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$
$$= 10s \times 2GHz = 20 \times 10^9$$
$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n}\left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}}\right)$$

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2 × 1 + 1 × 2 + 2 × 3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4 × 1 + 1 × 2 + 1 × 3
    = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c