CS1101: Lecture 10

Contemporary Multilevel Machines (Part 2)

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie

Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101

Lecture Outline

- A Six-Level Computer (continued)
- Computer Architecture
- Multilevel Machines: Hardware
- Multilevel Machines: Software
- The Hardware/Software Boundary
- Finite Precision Numbers
- **Reading**: Tanenbaum, Chapter 1 & Appendix A.

A Six-Level Computer

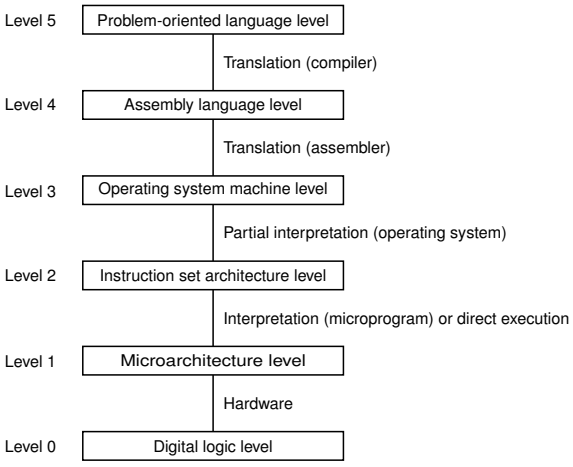

Figure 1.2 A six-level computer

Computer Architecture

- Thus, computers are designed as a series of levels, each one built on its predecessors;
- Each level represents a distinct abstraction, with different operations and objects present;
- This view allows us to handle the complexity of computer design;
- The set of data types, operations and features of each level is called its **architecture**;
- Implementation aspects, such as what kind of chip technology is used to implement the memory, are not part of the architecture;
- The study of how to design those parts of a computer system that are visible to the programmers are called **computer architecture**;

Multilevel Machines: Hardware

- Programs written in a computer's true machine language (level 1) can be directly executed by the computer's electronic circuits (level 0), without any intervening interpreters or translators.
- These electronic circuits, along with the memory and input/output devices, form the computer's hardware.
- Hardware consists of tangible objects:
  - integrated circuits
  - printed circuit boards
  - cables
  - power supplies
  - memories
  - printers
- Hardware is not abstract ideas, algorithms, or instructions.

Multilevel Machines: Software

- Software consists of **algorithms** (detailed instructions telling how to do something) and their computer representations-namely, **programs**
- Programs can be stored on hard disk, floppy disk, CD-ROM, or other media but the essence of software is the set of instructions that makes up the programs, not the physical media on which they are recorded.
- In the very first computers, the boundary between hardware and software was crystal clear.
- Over time, however, it has blurred considerably, primarily due to the addition, removal, and merging of levels as computers have evolved.
- *Hardware and software are logically equivalent*

The Hardware/Software Boundary

- Any operation performed by software can also be built directly into the hardware;
- "*Hardware is just petrified software*"
- Also, any instruction executed by the hardware can also be simulated in software;
- The decision to put certain functions in hardware and others in software is based on such factors as:
  - cost
  - speed
  - reliability and
  - frequency of expected changes
- There are few hard and fast rules to the effect that X must go into the hardware and Y must be programmed explicitly. These decisions change with trends in technology and computer usage.

Finite-Precision Numbers

- On most computers, the amount of memory available for storing a number is fixed at the time that the computer is designed.
- This forces us to deal only with numbers that can be represented in a fixed number of digits. We call such numbers **finite-precision numbers**.
- Consider the set of positive integers representable by three decimal digits, with no decimal point and no sign. This set has exactly 1000 members: 000, 001, 002, 003, ..., 999.
- With this restriction, it is impossible to express certain kinds of numbers, such as:
  - Numbers larger than 999.
  - Negative numbers.
  - Fractions.
  - Irrational numbers.
  - Complex numbers.

Arithmetic using Integers

- One important property of arithmetic on the set of all integers is closure with respect to the operations of addition, subtraction, and multiplication.
- In other words, for every pair of integers i and j, $i + j$, $i - j$ and $i * j$ are also integers.
- The set of integers is not closed with respect to division, because there exist values of i and j for which i/j is not expressible as an integer (e.g., 7/2 and 1/0).
- Finite-precision numbers are not closed with respect to any of these four basic operations, as shown below, using three-digit decimal numbers as an example:
  - 600 + 600 = 1200 (too large)
  - 003 - 005 = -2 (negative)
  - 050 x 050 = 2500 (too large)
  - 007 / 002 = 3.5 (not an integer)

Overflow and Underflow

- The violations can be divided into two mutually exclusive classes;
- Operations whose result is larger than the largest number in the set (overflow error) or smaller than the smallest number in the set (underfiow error), and operations whose result is neither too large nor too small but is simply not a member of the set.
- Of the four violations earlier, the first three are examples of the former, and the fourth is an example of the latter.

Radix Number Systems

- An ordinary decimal number consists of a string of decimal digits and, possibly, a decimal point.
- The choice of 10 as the base for exponentiation, called the **radix**, is made because we are using decimal, or base 10, numbers.
- When dealing with computers, it is frequently convenient to use radices other than 10.
- The most important radices are 2, 8, and 16. The number systems based on these radices are called binary, octal, and hexadecimal, respectively.

The General Form of a Decimal Number

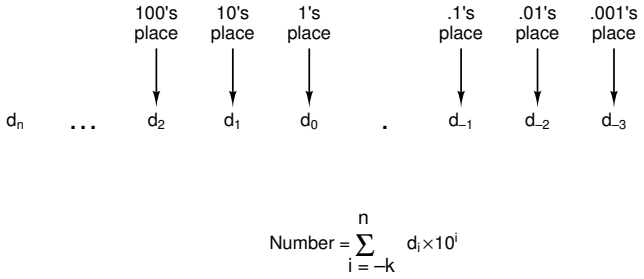

Figure A.1 The general form of a decimal number