Computer Architecture and Operating Systems

First year, Computer Science

Prof. dr. Henri Luchian
Lect. dr. Vlad Rădulescu

- examination
  - two written tests - course
    - one for each half of the semester
  - one practical test - laboratory
    - assembly language
- in order to be accepted to the written tests
  - be present at the laboratory classes
    - any student may miss at most two of them during each half of the semester

Contents: First Half of the Semester

I. Introduction
II. Combinational circuits and Boole functions
III. Sequential circuits and automata
IV. Internal representations
V. Computer architecture and organization

I. Introduction

I.1. Evolution

How Do We Define Computing?

- what operations can be performed?
- evolution
  - abacus: addition
  - toothed wheels (Leibniz, Pascal): addition, multiplication
  - Babbage: external instructions, branch computing
  - von Neumann: memorized program; execution as a sequence of instructions; memory hierarchies
  - parallel computing, quantum computing, etc.

Universal Computing Machines

- a universal computing machine can behave like any particular computing machine
  - so it can solve any problem that a particular computing machine can solve
- example - the computer
  - depending on the program it executes, it solves problems like: matrix computation, graphic design, desktop publishing, etc.

Short History (1)

- positional writing of numbers
  - Indians, Arabs
- Boole algebra
  - George Boole, 1854
- the incompleteness theorem
  - Kurt Gödel, 1935
- the link between Boole algebra and circuits
  - Claude Shannon, 1938

Short History (2)

- The "Neumannian" computer
  - John von Neumann, 1946
- the transistor
  - Shockley, Brittain, Bardeen, 1947
- integrated circuits

I.2. Empirical Laws

Empirical Laws

- in any field of science, the laws depend (one way or another) on experiments or on real world observations
- reproductibility leads to the idea of empirical laws: they are true in most cases, according to observations

Empirical Laws in Computer Science

- the "90:10" law (Donald Knuth)
  - 90% of the execution time of a program is used for 10% of the instructions
- Amdahl's law
  - highest efficiency in improving a system (either concrete or abstract) is achieved when the most intensively used subsystem is optimized
- locality laws - spatial, temporal

Amdahl's Law (1)

- consider a system (hardware or software) and one of its components
- that component works a certain percentage f_a of the system's total work time
- and it is improved, such that if gets to work *a* times faster than before
- how much times faster does the system as a whole become?

Amdahl's Law (2)

$$A(a, f_a) = \frac{1}{(1 - f_a) + \frac{f_a}{a}}$$

- highest increase of the overall speed
  - better improvement of the component (*a*)
  - improving the components with the highest weight in the system's work time (f_a)
    - i.e., the most intensely used ones