Introduction to Computer Organization and Architecture

Topic 1. Introduction

Han, Nguyen Dinh (han.nguyendinh@hust.edu.vn)

Faculty of Mathematics and Informatics
Hanoi University of Science and Technology

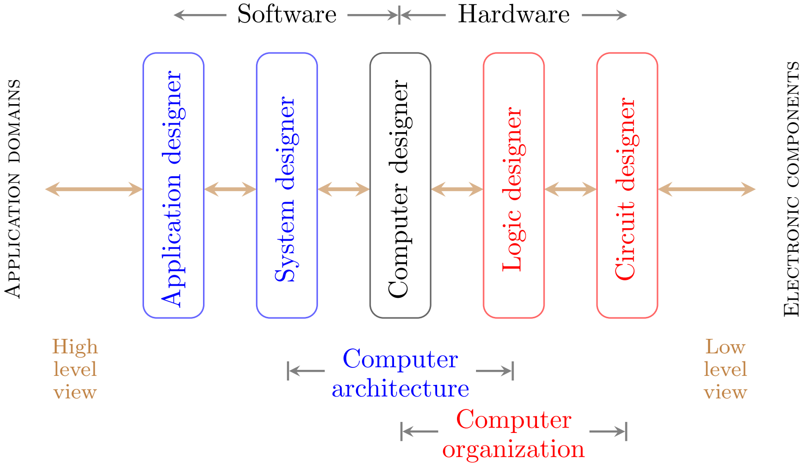
Outline

1. Computer abstraction
2. Computer technology
3. Computer program
4. Hardware and software
5. Computer performance

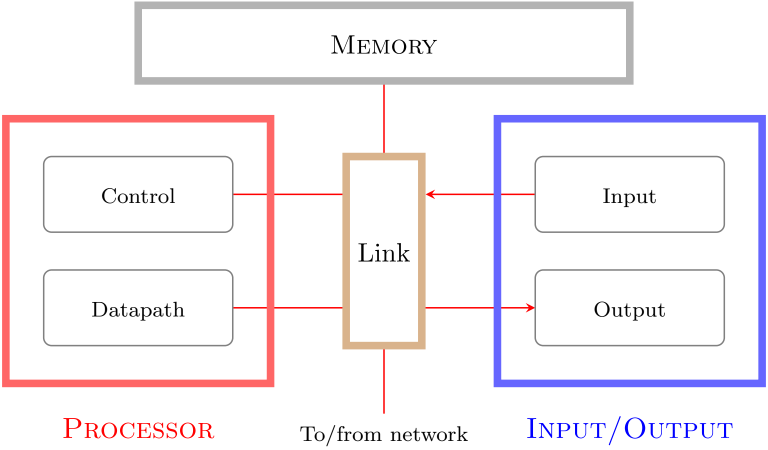
$\boxed{1}$

Computer abstraction

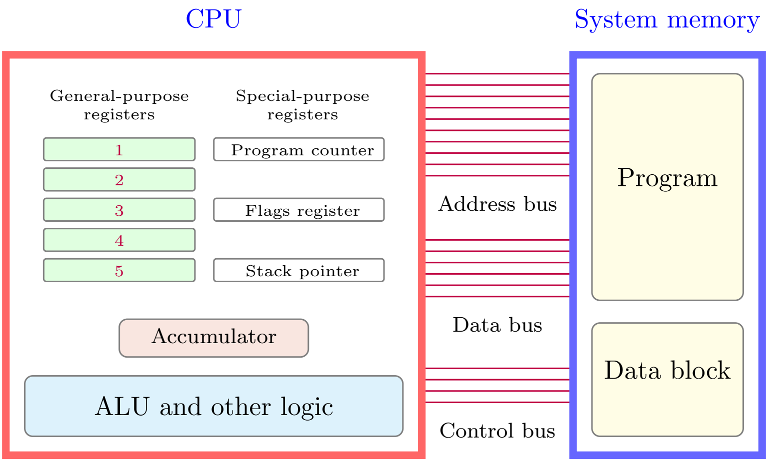
Views in computer system engineering



The (three to six) main units of a digital computer



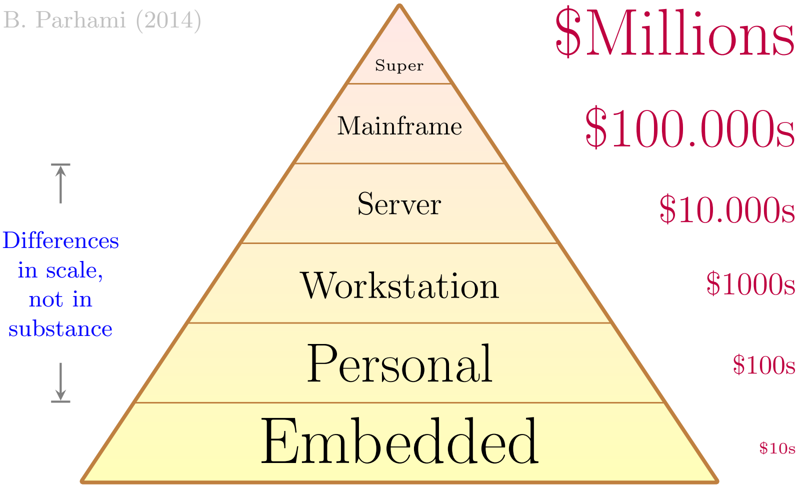
A simplified modern computer



$\boxed{2}$

Computer technology

Classifying computers by computational power and price

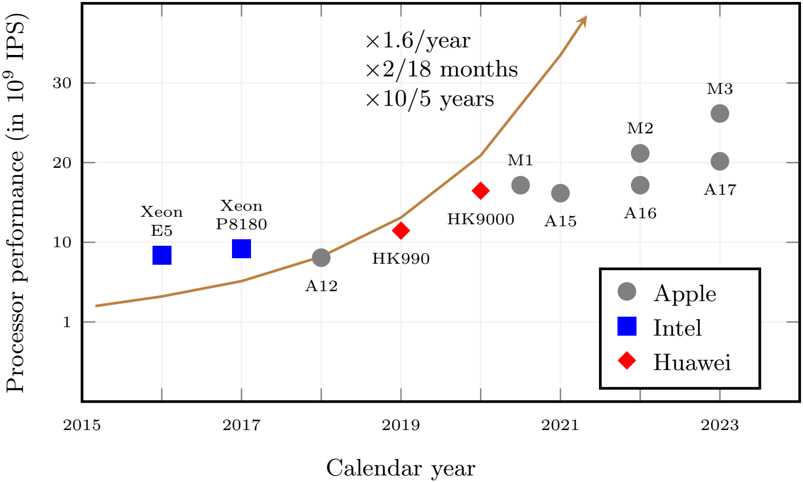


The five generations of digital computers

Generation (begun)	Processor technology	Memory innovations	I/O devices introduced	Dominant look&fell
1 (1950s)	Vacuum tube	Magnetic drum	Paper tape magnetic tape	Hall-size cabinet
2 (1960s)	Transistor	Magnetic core	Drum, printer, text terminal	Room-size mainframe
3 (1970s)	SSI/MSI	RAM/ROM chip	Disk/keyboard, video monitor	Desk-size mini
4 (1980s)	LSI/VLSI	SRAM/ DRAM	Network, CD, mouse, sound	Desktop/ laptop micro
5 (1990s)	ULSI/GSI/ WSI, SoC	SDRAM, flash	Sensor/actuator, point/click	Invisible, embedded

Source: B. Parhami (2014)

Trends in processor performance (Moore's Law)



$\boxed{3}$

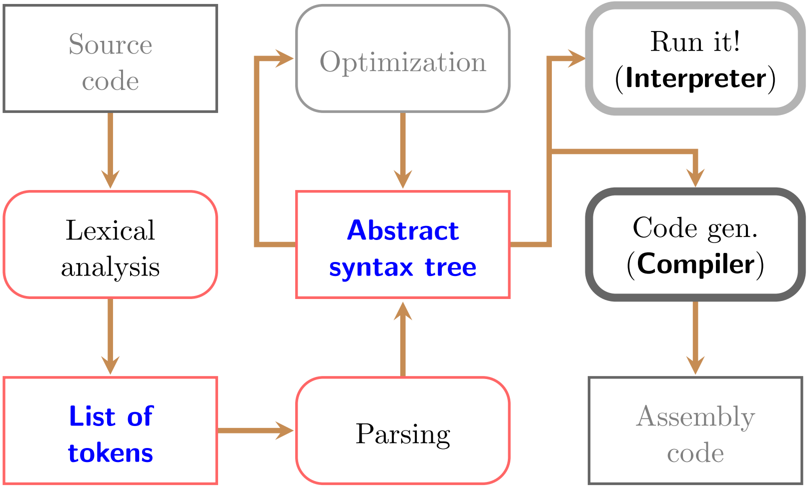
Computer program

We recall that a language is a set of sentences, which in turn are sequences of *terminal symbols*. Every language has syntax and semantics

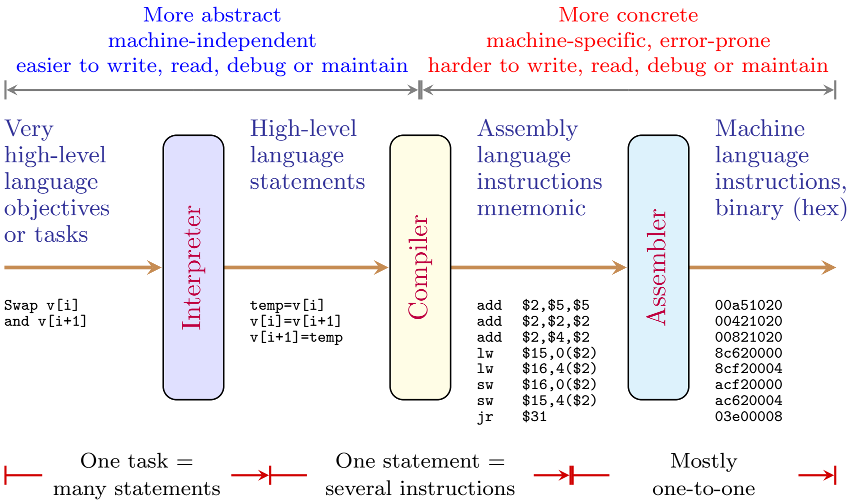
For a programming language, its syntax or *structure* influences how programs are written (i.e. how expressions, commands, declarations, etc., are put together to *form* programs), and its semantics is concerned with the *meaning* of programs

However, before a program can be run, it first must be translated into a form in which it can be executed by a computer. The system for processing programs is called *a language processor*. Examples of language processors are compilers, interpreters, and auxiliary tools like syntax-directed editors

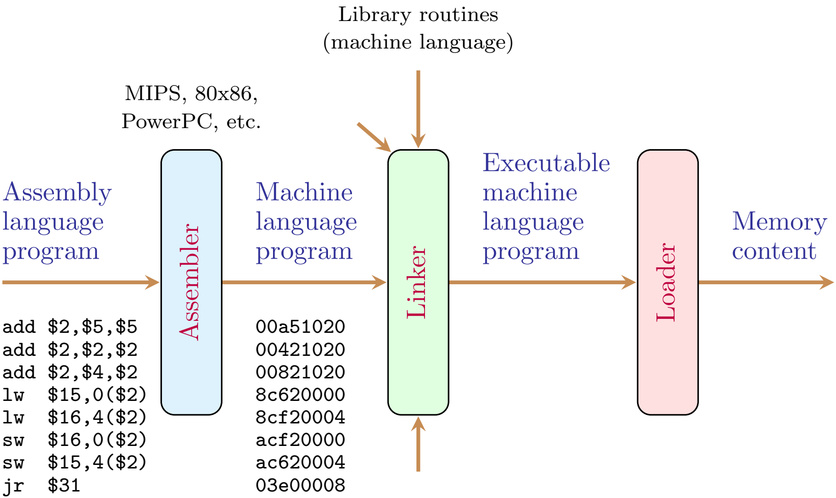
Interpreter and compiler structure



Models and abstractions in programming



Assembly and machine languages



Computer program

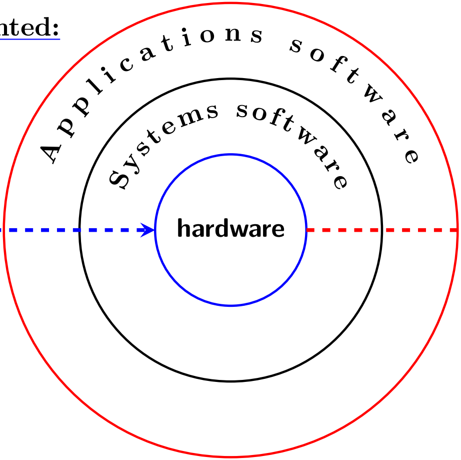
Can anyone learn (if necessary) and then teach us how to write and run some simple programs in VHDL, Verilog or System Verilog?

$\boxed{4}$

Hardware and software

Machine oriented:

generic,
simple,
focus on data

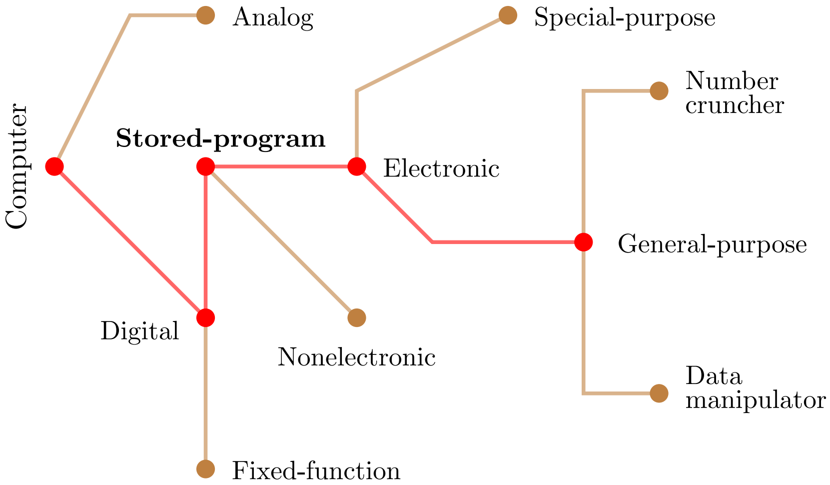


Man oriented:

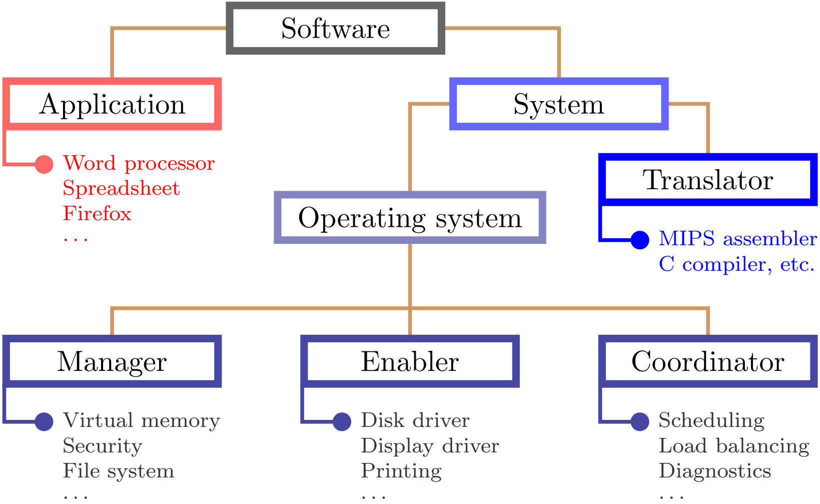
specific,
complex,
focus on users

Source: D.A. Patterson, et al. (2014)

The world of computer hardware



Categorization of software



$\boxed{5}$

Computer performance

Concepts of performance and speedup

	Instruction count	CPI	Clock
Program	●		
Compiler	●	●	
Instruction Set Architecture	●	●	●
Microarchitecture		●	●
Physical Design			●

Concepts of performance and speedup

We recall that the performance of a computer X will be related to its CPU execution time (or simply CPU time):

$$\text{Performance}_X = \frac{1}{\text{CPU execution time}}$$

and the basic performance equation can be written in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time as follows

$$\begin{aligned} \text{CPU time} &= \text{Instructions} \times \text{Average CPI} \times (\text{Secs per cycle}) \\ &= \text{Instructions} \times \text{Average CPI}/(\text{Clock rate}) \end{aligned}$$

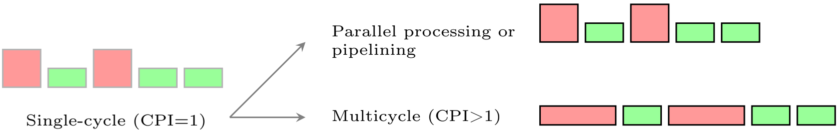
Strategies for speeding up instruction execution

$$\textbf{Performance} = \textbf{Clock rate}/(\textbf{Instructions} \times \textbf{CPI})$$

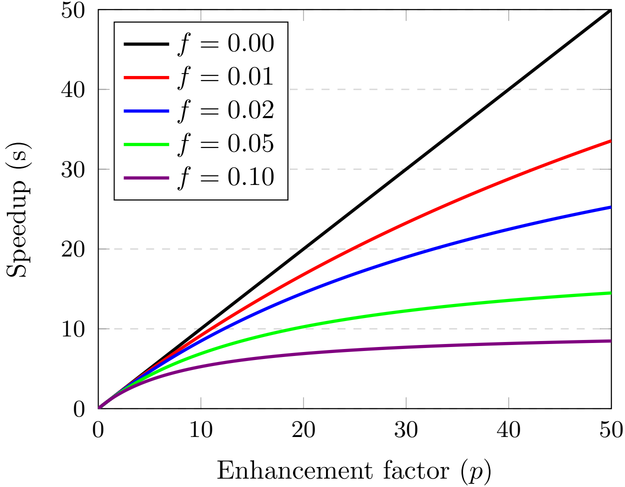
Design memory & I/O structures to support ultra high-speed CPUs

Define a simple enough instruction set to require a small number of cycles and allow high clock rate

Design hardware for CPI = 1; seek improvements with CPI > 1



Performance enhancement (Amdahl's Law)



f: Fraction unaffected

p: Speedup of the rest

s: Overall speedup

$$s = \frac{1}{f + \frac{(1-f)}{p}}$$

$$s \leq \min(p, 1/f)$$

THANK YOU VERY MUCH FOR YOUR ATTENTION!