603AA - Principles of Programming Languages [PLP-2015]

Andrea Corradini

Department of Computer Science, Pisa

Academic Year 2015/16

Admins

- **http://www.di.unipi.it/~andrea/Didattica/PLP-15/**
- 9 CFU/ECTS
- Students enrolled till AY 2013/14 have to integrate the course with a 3 CFU activity
  - To be agreed upon with me
- Office Hours: (was *Monday, 15:30-17:30)*

Evaluation

- 2 midterms
  - November 2-6, 2015
  - December 16-18, 2015
- Written proof
- Oral examination

- Pre-evaluation:
  - Starter kit test: Thu October 1 at 17:00

Course Topics and Goals

- The course presents principles and techniques for the implementation and usage of programming languages.
- First part:
  - formal definition of the syntax of programming languages
  - main phases of a compiler with emphasis on the lexical, syntactical and semantical analysis phases of the front-end.
- Second part:
  - main topics of the structure of programming languages from the viewpoint of the runtime support of its abstract machine and of the expressiveness of the supported linguistic constructs
  - focus on constructs of imperative, functional, object-oriented, and scripting languages

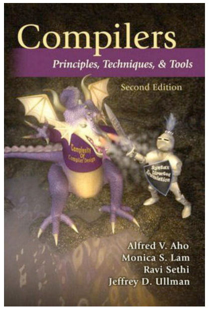
Textbooks

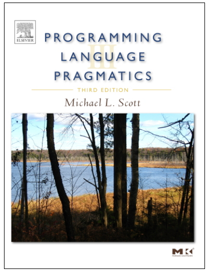
- **[Scott] Programming Language Pragmatics**
by Michael L. Scott, 3rd edition

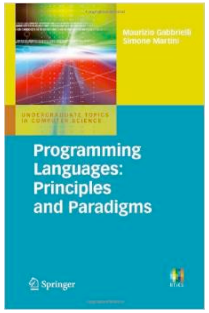
- **[ALSU] Compilers: Principles, Techniques, and Tools**
by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, 2nd edition

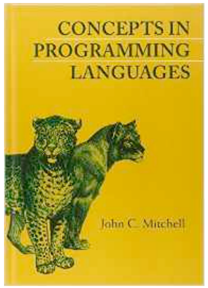
- **[GM] Programming Languages: Principles and Paradigms**
by Maurizio Gabbrielli and Simone Martini

- **[Mitchell] Concepts in Programming Languages**
by John C. Mitchell






Maurizio Gabbrielli
Simone Martini
UNDERGRADUATE TOPICS IN COMPUTER SCIENCE
Programming Languages: Principles and Paradigms
Springer



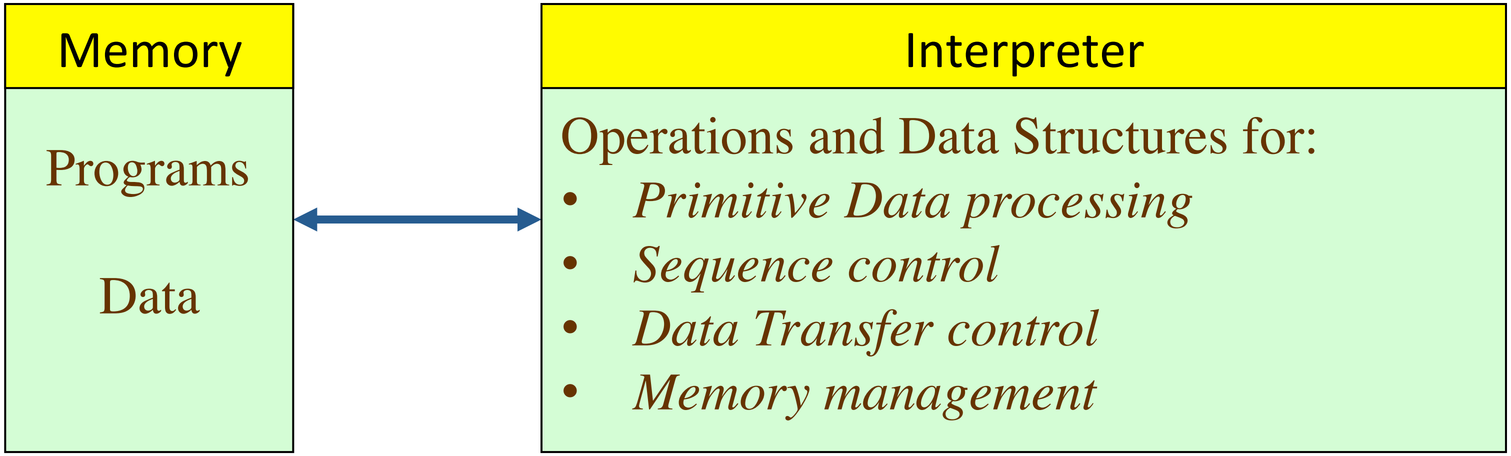
Credits

- Slides freely taken and elaborated from a number of sources:
  - – Marco Bellia (DIP)
  - – Gianluigi Ferrari (DIP)
  - – Robert A. van Engelen (Florida State University)
  - – Gholamreza Ghassem-Sani (Sharif University of Technology)

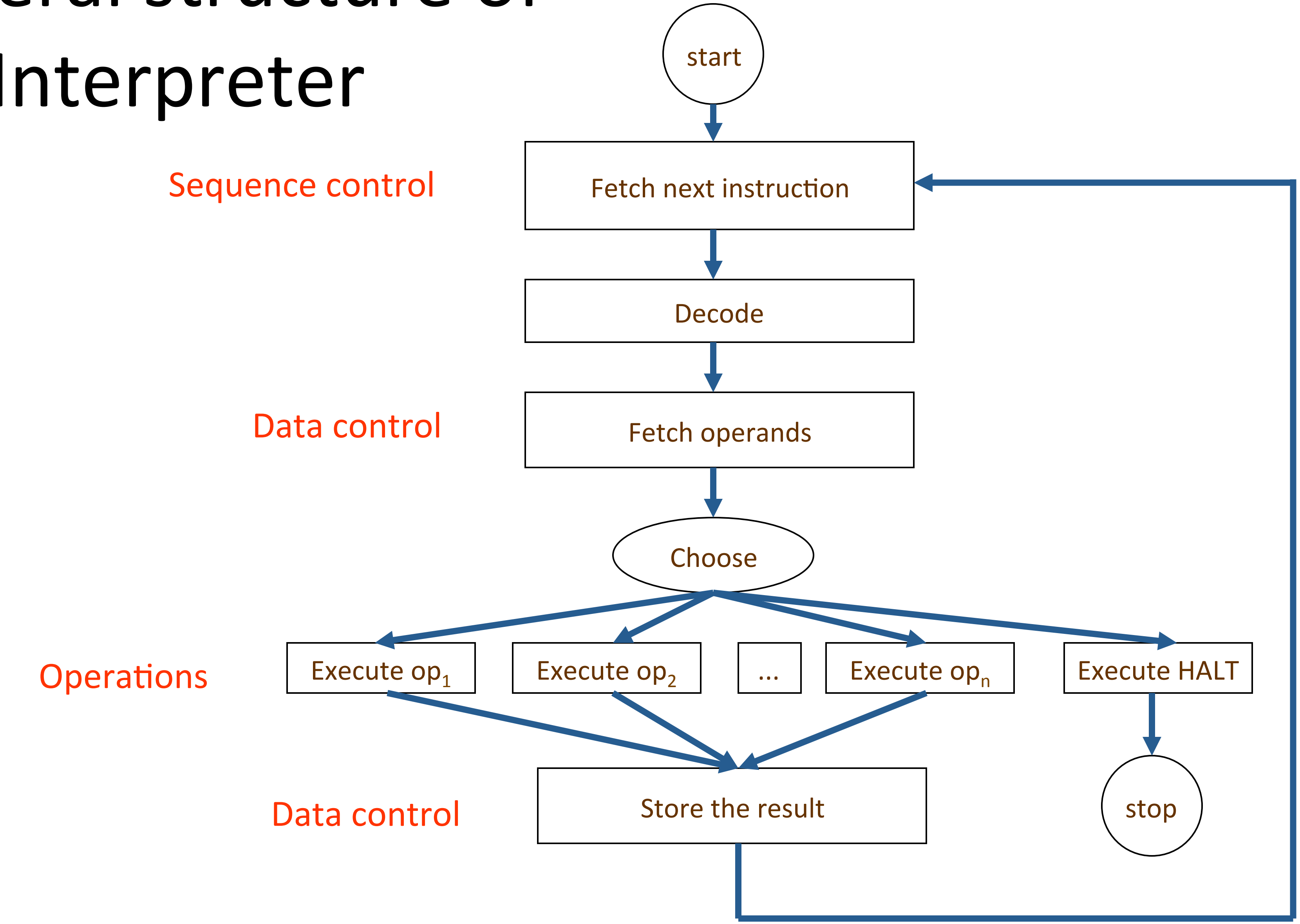
Abstract Machines

Abstract Machine for a Language **L**

- Given a programming language **L**, an **Abstract Machine M_L for L** is *a collection of data structures and algorithms which can perform the storage and execution of programs written in **L***
- An abstraction of the concept of hardware machine
- Structure of an abstract machine:



General structure of the Interpreter



start

Sequence control

Fetch next instruction

Decode

Data control

Fetch operands

Choose

Operations

Execute op$_1$

Execute op$_2$

...

Execute op$_n$

Execute HALT

Data control

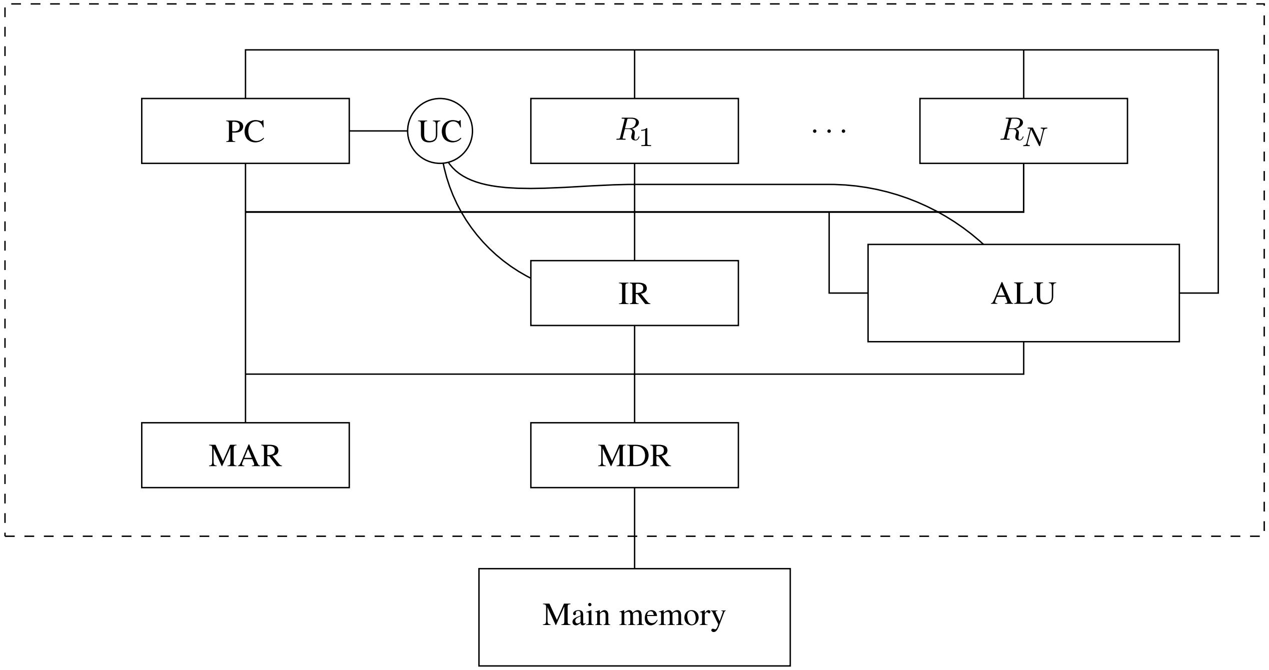
Store the result

stop

The Machine Language of an AM

- Given and Abstract machine **M**, the machine language **L_M** of **M**
  - includes all programs which can be executed by the interpreter of M
- Programs are particular data on which the interpreter can act
- The components of **M** correspond to components of **L_M**, eg:
  - Primitive data types
  - Control structures
  - Parameter passing and value return
  - Memory management
- Every Abstract Machine has a unique Machine Language
- A programming language can have several Abstact Machines

An example: the Hardware Machine



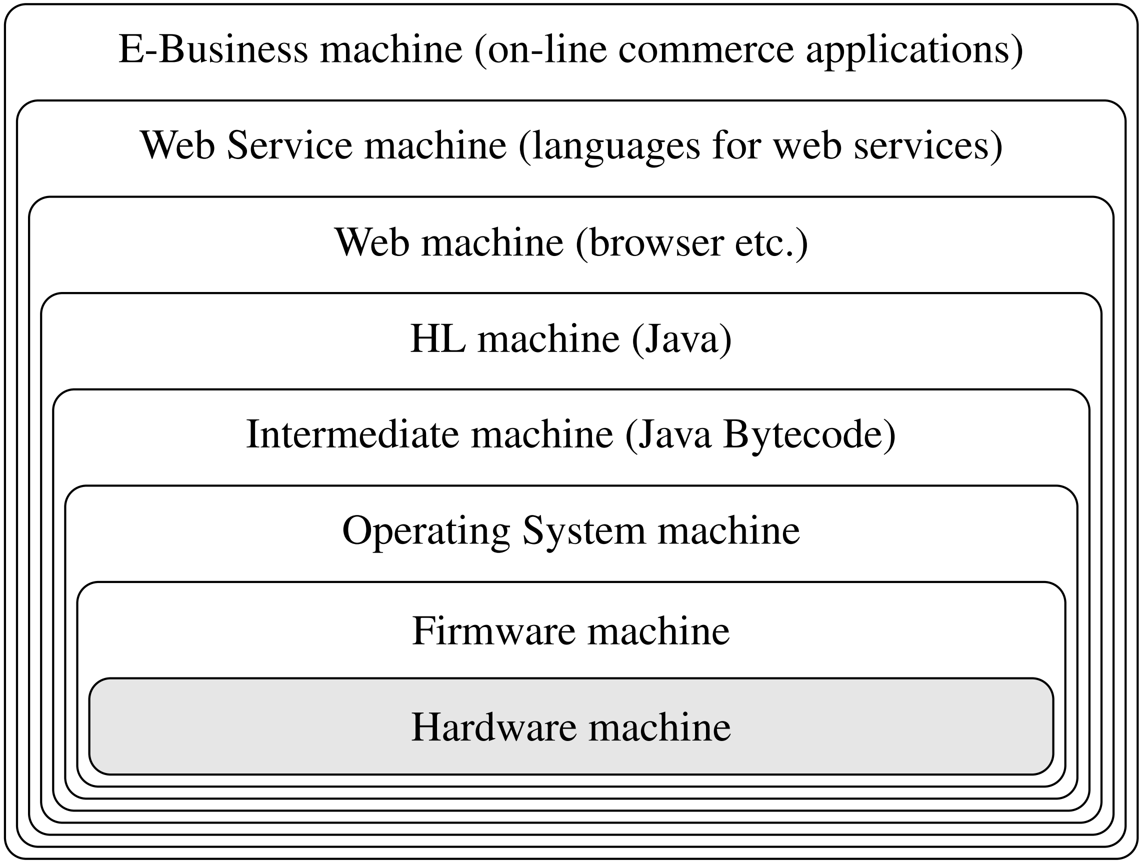
- The language?
- The memory?
- The interpreter?
- Operations and Data Structures for:
  - Primitive Data processing?
  - Sequence control?
  - Data Transfer control?
  - Memory management?

Implementing an Abstract Machine

- Each abstract machine can be implemented in **hardware** or in **firmware**, but if it is high-level this is not convenient in general
- An abstract machine **M** can be implemented over a **host machine $\mathbf{M_O}$**, which we assume is already implemented
- The components of **M** are realized using data structures and algorithms implemented in the machine language of $\mathbf{M_O}$
- Two main cases:
  - The interpreter of **M** coincides with the interpreter of $\mathbf{M_O}$
    - **M** is an **extension** of $\mathbf{M_O}$
    - other components of the machines can differ
  - The interpreter of **M** is different from the interpreter of $\mathbf{M_O}$
    - **M** is **interpreted** over $\mathbf{M_O}$
    - other components of the machines may coincide

Hierarchies of Abstract Machines

- Implementation of an AM with another can be iterated, leading to a hierarchy (onion skin model)
- Example:



E-Business machine (on-line commerce applications)

Web Service machine (languages for web services)

Web machine (browser etc.)

HL machine (Java)

Intermediate machine (Java Bytecode)

Operating System machine

Firmware machine

Hardware machine

Implementing a Programming Language

- **L** high level programming language
- **M_L** abstract machine for **L**
- **M_O** host machine
- **Pure Interpretation**
  - **M_L** is interpreted over **M_O**
  - Not very efficient, mainly because of the interpreter (fetch-decode phases)
- **Pure Compilation**
  - Programs written in **L** are translated into equivalent programs written in **L_O**, the machine language of **M_O**
  - The translated programs can be executed directly on **M_O**
    - **M_L** is not realized at all
  - Execution more efficient, but the produced code is larger
- Two limit cases that almost never exist in reality