

Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

[http://www.cs.uiuc.edu/class
/fa06/cs421/](http://www.cs.uiuc.edu/class/fa06/cs421/)

Based in part on slides by Mattox Beckman, as updated
by Vikram Adve and Gul Agha

Contact Information - Elsa L Gunter

- Office: 2112 SC
- Office hours:
 - Thursdays 9:00am – 10:15am
 - Also by appointment
- Email: egunter@cs.uiuc.edu

Contact Information - TAs

Teaching Assistants Office: 0207 SC

- T. Baris Aktemur
 - **Email:** aktemur@uiuc.edu
 - **Hours:** Mon 9:00am – 10:00am & Thurs 1:00pm – 2:00pm
- Chris Osborn
 - **Email:** cosborn3@uiuc.edu
 - **Hours:** Tues 9:00am – 10:00am & Wed 12:30pm – 1:30pm
- Andrei Popescu
 - **Email:** popescu2@cs.uiuc.edu
 - **Hours:** Tues 4:00pm – 5:00pm & Fri 4:00pm – 5:00pm

Some Course References

No required textbook.

- Compilers: Principles, Techniques, and Tools, (*also known as "The Dragon Book"*); by Aho, Sethi, and Ullman. Published by Addison-Wesley. ISBN: 0-201-10088-6.
- Advanced Programming Language Design, by Raphael A. Finkel. Addison Wesley Publishing Company, 1996.
- Programming Language Pragmatics, by Michael L. Scott. Morgan Kaufman Publishers, 2000.
- Concepts, Techniques, and Models of Computer Programming by Peter Van Roy and Seif Haridi, MIT Press, 2004.
- Essentials of Programming Languages (2nd Edition) by Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes, MIT Press 2001.

Course Grading

- Homework 35%
 - About 10 MPs (in Ocaml) and 4 written assignments
 - MPs submitted by handin
- Midterm 25%
 - In class - **Oct 10**
- **DO NOT MISS EXAM DATE!**
- Final 40% - **Dec 16**
- Percentages are approximate
 - Exams may weigh more if homework is much better

Course Homework

- You may discuss homeworks and their solutions with others
- You may not leave the discussion with a written solution
- You must write your own solution
- You may not look at another written solution when you are writing your own
 - You may look at examples from class and other similar examples

Course Website

- Main page - summary of news items
- Policy - rules governing course
- Lectures - syllabus and slides
- MPs - information about homework
- Exams
- Unit Projects - for 4 credit students
- Resources - tools and helpful info
- FAQ

Personal History

- First began programming more than 35 years ago
- First languages: basic, DG Nova assembler
- Since have programmed in at least 10 different languages
 - Not including AWK, sed, shell scripts, latex, HTML, etc

Personal History - Moral

One language may not last you all day, let alone your whole programming life

Programming Language Goals

- Original Model:
 - Computers expensive, people cheap; hand code to keep computer busy
- Today:
 - People expensive, computers cheap; write programs efficiently and correctly

Programming Language Goals

- *Mythical Man-Month* Author Fred Brookes
 - “The most important two tools for system programming ... are (1) high-level programming languages and (2) interactive languages”

Languages as Abstractions

- Abstraction from the Machine
- Abstraction from the Operational Model
- Abstraction of Errors
- Abstraction of Data
- Abstraction of Components
- Abstraction for Reuse

Why Study Programming Languages?

Helps you to:

- understand efficiency costs of given constructs
- reduce bugs by understanding semantics of constructs
- think about programming in new ways
- choose best language for task
- design better program interfaces (and languages)
- learn new languages

Study of Programming Languages

- Design and Organization
 - Syntax: How a program is written
 - Semantics: What a program means
 - Implementation: How a program runs
- Major Language Features
 - Imperative / Applicative / Rule-based
 - Sequential / Concurrent

Features of a Good Language

- Simplicity – few clear constructs, each with unique meaning
- Orthogonality - every combination of features is meaningful, with meaning given by each feature
- Flexible control constructs

Features of a Good Language

- Rich data structures – allows programmer to naturally model problem
- Clear syntax design – constructs should suggest functionality
- Support for abstraction - program data reflects problem being solved; allows programmers to safely work locally

Features of a Good Language

- Expressiveness – concise programs
- Good programming environment
- Architecture independence and portability

Features of a Good Language

- Readability
 - Simplicity
 - Orthogonality
 - Flexible control constructs
 - Rich data structures
 - Clear syntax design

Features of a Good Language

- Writability
 - Simplicity
 - Orthogonality
 - Support for abstraction
 - Expressivity
 - Programming environment
 - Portability

Features of a Good Language

- Usually readability and writability call for the same language characteristics
- Sometimes they conflict:
 - Comments: Nested comments (e.g. `/* ... /* ... */ ... */`) enhance writability, but decrease readability

Features of a Good Language

- Reliability
 - Readability
 - Writability
 - Type Checking
 - Exception Handling
 - Restricted aliasing

Programming Language Implementation

- Develop layers of machines, each more primitive than the previous
- Translate between successive layers
- End at basic layer
- Ultimately hardware machine at bottom

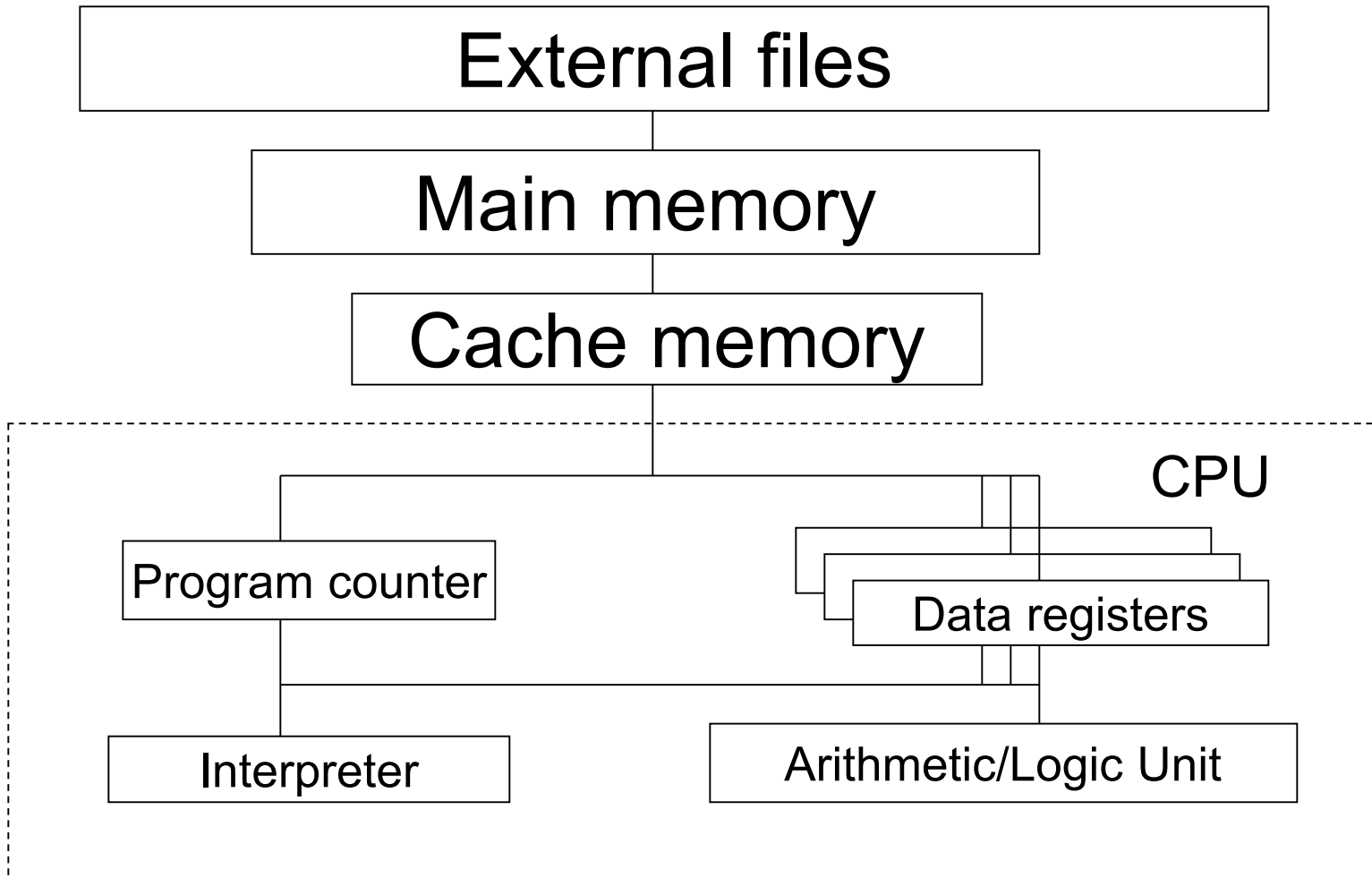
Basic Machine Components

- **Data:** basic data types and elements of those types
- **Primitive operations:** for examining, altering, and combining data
- **Sequence control:** order of execution of primitive operations

Basic Machine Components

- **Data access:** control of supply of data to operations
- **Storage management:** storage and update of program and data
- **External I/O:** access to data and programs from external sources, and output results

Basic Computer Architecture



Interpretation Versus Compilation

- A **compiler** from language L1 to language L2 is a program that takes an L1 program and for each piece of code in L1 generates a piece of code in L2 of same meaning

Interpretation Versus Compilation

- An **interpreter** of L1 in L2 is an L2 program that executes the meaning of a given L1 program
- Compiler would examine the body of a loop once; an interpreter would examine it every time the loop was executed

Program Aspects

- Syntax: what valid programs look like
- Semantics: what valid programs mean; what they should compute
- Compiler must contain both information

Major Phases of a Compiler

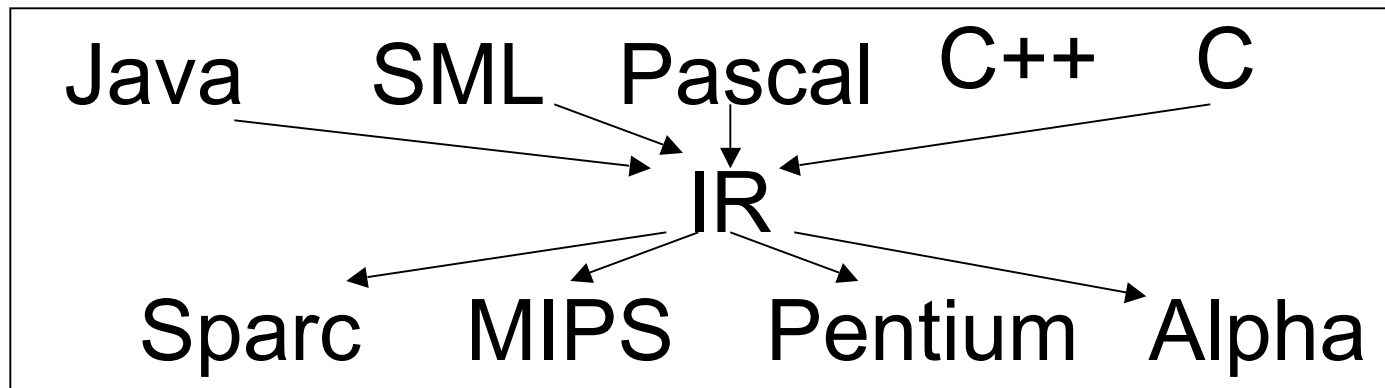
- Lex
 - Break the source into separate tokens
- Parse
 - Analyze phrase structure and apply semantic actions, usually to build an abstract syntax tree

Major Phases of a Compiler

- Semantic analysis
 - Determine what each phrase means, connect variable name to definition (typically with symbol tables), check types

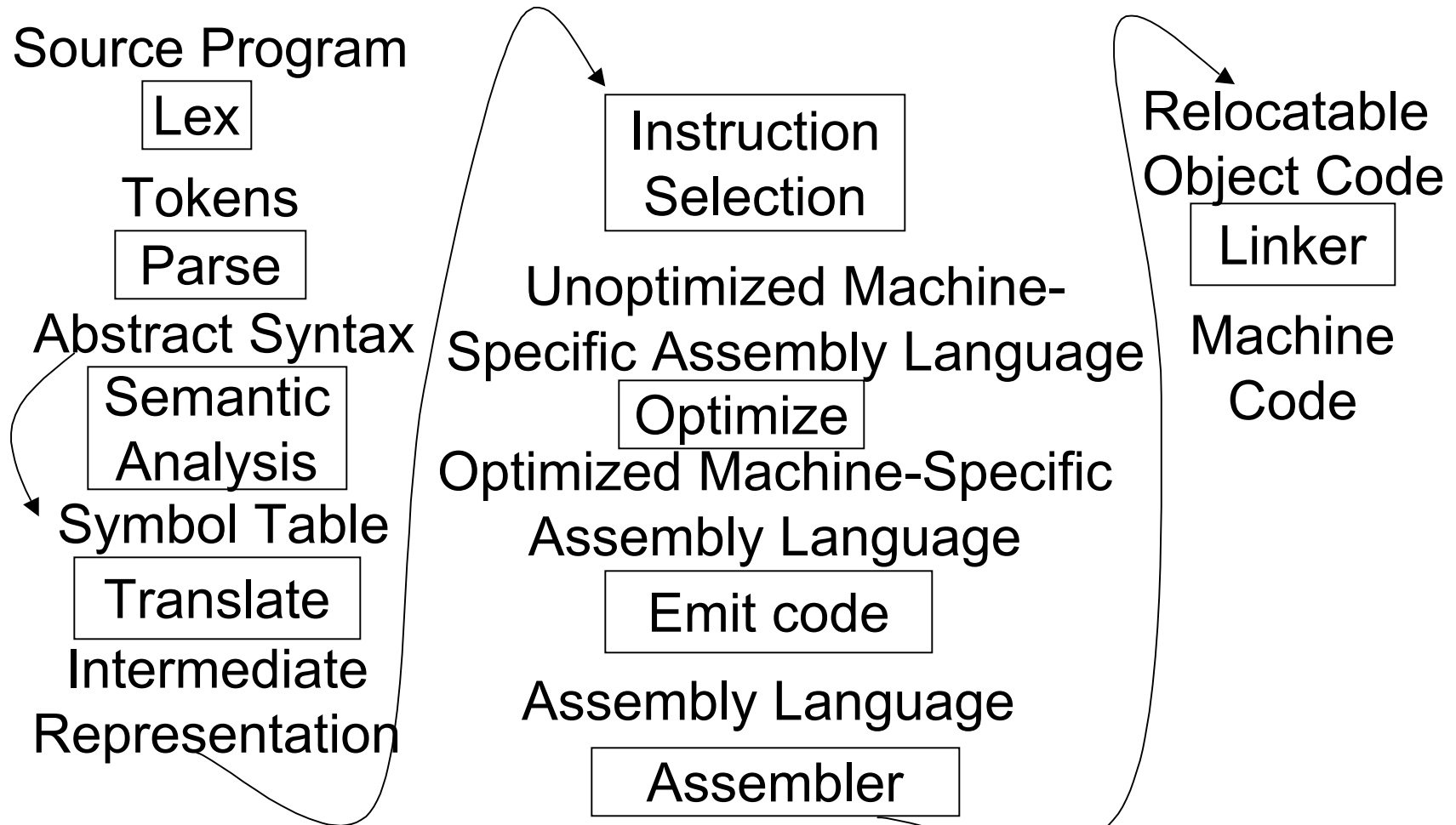
Major Phases of a Compiler

- Translate to intermediate representation



- Instruction selection
- Optimize
- Emit final machine code

Major Phases of a Compiler



Elsa L. Gunter

Modified from "Modern Compiler Implementation in ML", by Andrew Appel

Example of Intermediate Representation

- Program code: $X = Y + Z + W$
 - $\text{tmp} = Y + Z$
 - $X = \text{tmp} + W$
- Simpler language with no compound arithmetic expressions

Example of Optimization

Program code: $X = Y + Z + W$

- | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">• Load reg1 with Y• Load reg2 with Z• Add reg1 and reg2, saving to reg1• Store reg1 to tmp ** | <ul style="list-style-type: none">• Load reg1 with tmp **• Load reg2 with W• Add reg1 and reg2, saving to reg1• Store reg1 to X |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Eliminate two steps marked **