COMP36512 - Compilers
Lecture 1: Introduction

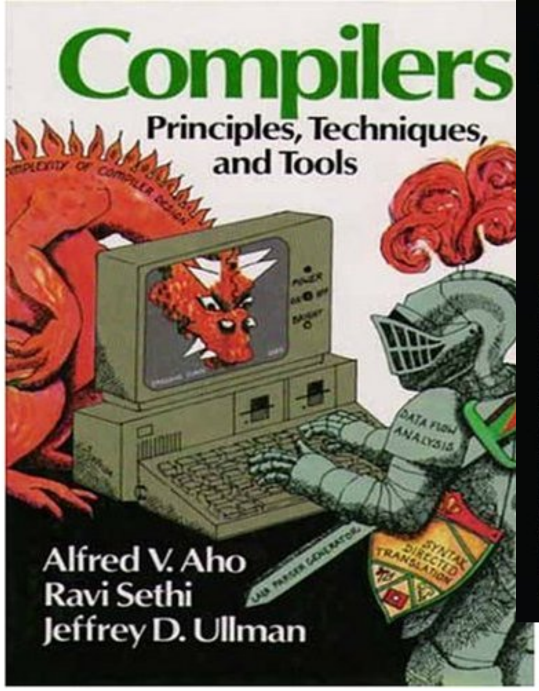
- Module Aims:
  - Any program written in a programming language must be translated before it can be executed. This translation is typically accomplished by a software system called compiler. This module aims to introduce students to the principles and techniques used to perform this translation and the issues that arise in the construction of a compiler.

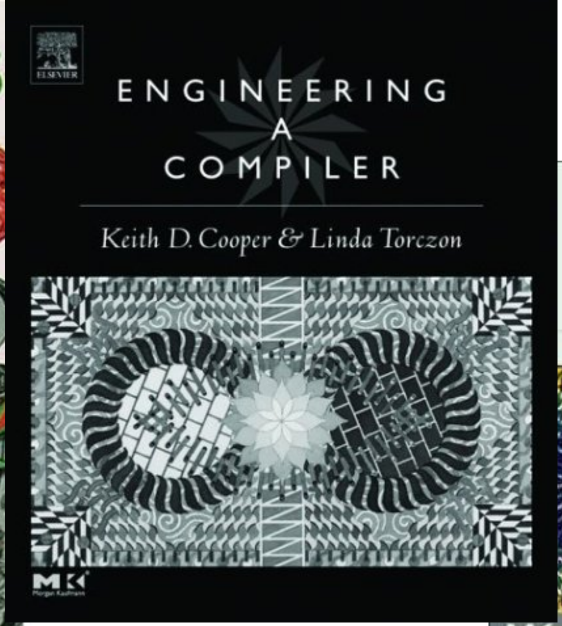
COMP36512 - Compilers (cont.)

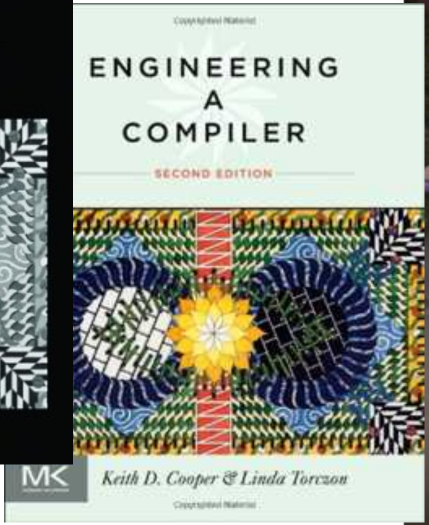
- Learning Outcomes:
  - A student successfully completing this module should be able to:
    - understand the principles governing all phases of the compilation process.
    - understand the role of each of the basic components of a standard compiler.
    - show awareness of the problems of and methods and techniques applied to each phase of the compilation process.
    - apply standard techniques to solve basic problems that arise in compiler construction.
    - understand how the compiler can take advantage of particular processor characteristics to generate good code.

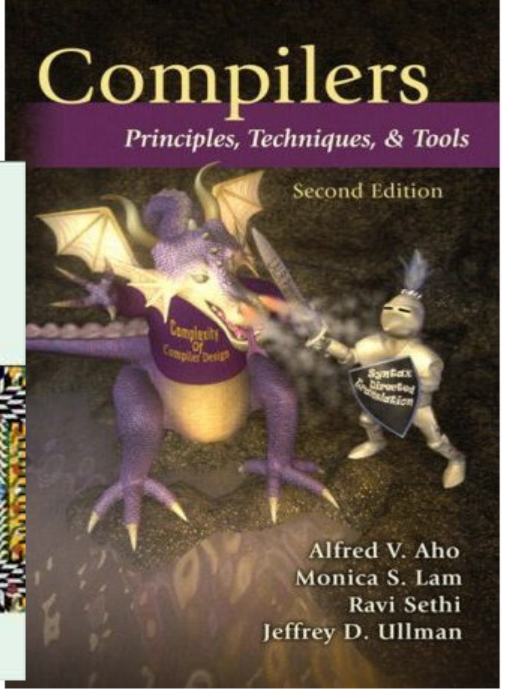
Course Lecturer - and Lectures

- Who am I?
  - Rizos Sakellariou – **`rizos@manchester.ac.uk`**
  - Research: Parallel and Distributed Systems (including scheduling, performance optimisation, etc), and (in the distant past) parallelising compilers
- Lectures: Tuesdays 11:00-13:00 (Kilburn LT 1.1)
  - handouts (and other information) available from: **http://studentnet.cs.manchester.ac.uk/ugt/COMP36512/**
- How to study:
  - Make Your Own Notes, and/or
    - listen, understand, jot down, ask, interrupt, ...
  - Study the book(s), …, …, …
  - Flexible Approach/Engineering: tradeoff/constraints/optimization
- Assessment: 2-hour exam (3 questions [out of 5 – not anymore!])









Course Texts

- **Aho, Lam, Sethi, Ullman. “Compilers: Principles, Techniques and Tools”, 2nd edition**. (Aho2) The **1st edition** (by Aho, Sethi, Ullman – Aho1), the “Dragon Book”, has been a classic for over 20 years.
- **Cooper & Torczon. “Engineering a Compiler”** – an earlier draft has been consulted when preparing this module. The 2nd edition is now available and being assessed (pointers will be provided to the 1st and hopefully to the 2nd edition).
- Other books:
  - Hunter *et al*. “The essence of Compilers” (Prentice-Hall)
  - Grune *et al*. “Modern Compiler Design” (Wiley)

Syllabus

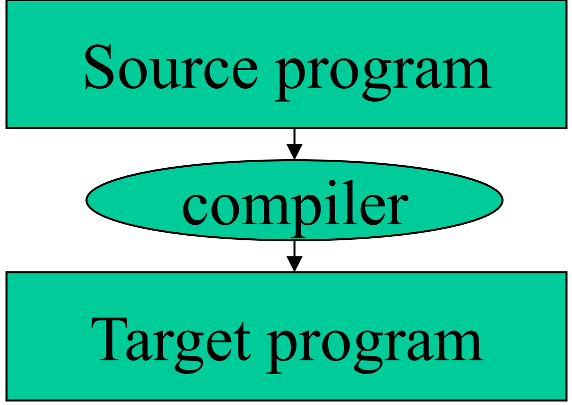
- 2 Introduction
- 3 Lexical Analysis (scanning)
- 1 – Exercises (on your own – optional)
- 3 Syntax Analysis (parsing)
- 1 – Exercises (on your own – optional)
- 1 Semantic Analysis
- 1 Intermediate Representations
- 1 Storage Management
- 1 Exercises (on your own – optional)
- 4 Code Generation
- 1 Code Optimisation
- 1 – Guest Lecture???
- 2 Exam preparation - Conclusion

Definitions

(compile: collect material into a list, volume)

- What is a compiler?
  - A program that accepts as input a program text in a certain language and produces as output a program text in another language, while preserving the meaning of that text (Grune *et al*, 2000).
  - A program that reads a program written in one language (source language) and translates it into an equivalent program in another language (target language) (Aho *et al*)
- What is an interpreter?
  - A program that reads a source program and produces the results of executing this source.
- *We deal with compilers! Many of these issues arise with interpreters!*

Examples



- C is typically compiled
- Lisp is typically interpreted
- Java is compiled to bytecodes, which are then interpreted

Also:

- C++ to Intel Core 2/…/Assembly
- C++ to C
- High Performance Fortran (HPF) to Fortran (parallelising compiler)
- C to C (or any language to itself)

In the general sense:

- What is LaTeX?
- What is ghostview? (PostScript is a language for describing images)

Qualities of a Good Compiler

What qualities would you want in a compiler?

– generates correct code (first and foremost!)
– generates fast code
– conforms to the specifications of the input language
– copes with essentially arbitrary input size, variables, etc.
– compilation time (linearly)proportional to size of source
– good diagnostics
– consistent optimisations
– works well with the debugger

Principles of Compilation

The compiler must:

- *preserve the meaning of the program being compiled.*
- *"improve" the source code in some way.*

Other issues (depending on the setting):

- Speed (of compiled code)
- Space (size of compiled code)
- Feedback (information provided to the user)
- Debugging (transformations obscure the relationship source code vs target)
- Compilation time efficiency (fast or slow compiler?)

Why study Compilation Technology?

- Success stories (one of the earliest branches in CS)
  - Applying theory to practice (scanning, parsing, static analysis)
  - Many practical applications have embedded languages (eg, tags)
- Practical algorithmic & engineering issues:
  - Approximating really hard (and interesting!) problems
  - Emphasis on efficiency and scalability
  - Small issues can be important!
- Ideas from different parts of computer science are involved:
    - AI: Heuristic search techniques; greedy algorithms - Algorithms: graph algorithms - Theory: pattern matching - Also: Systems, Architecture
- Compiler construction can be challenging and fun:
  - new architectures always create new challenges; success requires mastery of complex interactions; results are useful; opportunity to achieve performance.

Uses of Compiler Technology

- Most common use: translate a high-level program to object code
  - Program Translation: binary translation, hardware synthesis, …
- Optimizations for computer architectures:
  - Improve program performance, take into account hardware parallelism, etc…
- Automatic parallelisation or vectorisation
- Performance instrumentation: e.g., -pg option of cc or gcc
- Interpreters: e.g., Python, Ruby, Perl, Matlab, sh, …
- Software productivity tools
  - Debugging aids: e.g, purify
- Security: Java VM uses compiler analysis to prove "safety" of Java code.
- Text formatters, just-in-time compilation for Java, power management, global distributed computing, …

Key: Ability to extract properties of a source program (analysis) and transform it to construct a target program (synthesis)

Summary

- A compiler is a program that converts some input text in a <u>source language</u> to output in a <u>target language</u>.
- Compiler construction poses some of the most challenging problems in computer science.
- Reading:
  - Aho2, 1.1, 1.5; Aho1 1.1; Cooper1 1.1-1.3;
  - Grune 1.1, 1.5

- Next lecture: structure of a typical compiler.