Extra Concepts Introduced

- General Compilation Issues
- Bootstrapping
- Testing and Maintenance
- Current Research Areas in Compilers

Compilation Issues

- Source Language
- Target Language
- Evaluation Criteria

Source Language Issues

- Size of the language
- Stability of the language

Target Language Issues

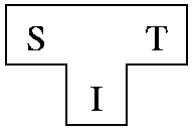
- Maturity of the architecture
  - Architectural problems may exist.
  - May be extensions to the architecture.
- Design of the intermediate language
  - Need to parameterize some decisions.
    - size of data types
    - alignment requirements
    - argument evaluation order

Evaluation Criteria

- Compiler Speed
- Generated Code Quality
  - execution time
  - code size
  - energy usage
- Error Diagnostics
- Portability
  - retargetability
  - rehostability
- Maintainability

Bootstrapping

- Bootstrapping involves using the facilities of the source language to compile the compiler.
- A compiler is characterized by three languages.
  1. S - the source language that it compiles
  2. T - the target language it generates
  3. I - the language in which it is implemented
- Can be represented with a T diagram.

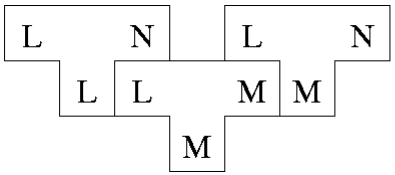


Or with text: S $_{I}$ T

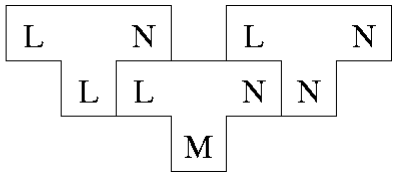
Cross Compiling

- A cross compiler runs on one machine and produces target code for another machine.
- Example: A cross compiler that compiles language L, is written in language L, and produces code for machine N. An existing compiler compiles language L, executes on machine M, and generates code for machine M.

1.

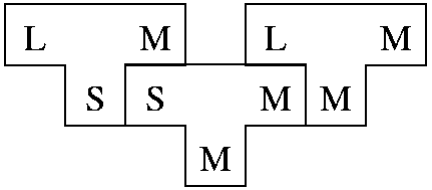


2.



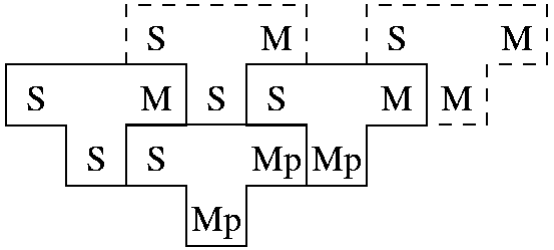
Compiling a New Language

- Can write a simple compiler that consists of a subset S of L that executes on machine M and produces code for machine M. We can then use this subset S to write a compiler that compiles L into M code.



Optimizing a Compiler

- We can use a compiler to compile itself to get a faster compiler. Say we have implemented a compiler that compiles source language S, written in source language S, and produces code for machine M. We also have access to another compiler for S that produces poor code p for machine M (call it Mp) and executes slowly (Mp).



Tools for Compiler Development

- makefiles
- profilers
- source-level debuggers
- cross-reference generators
- static checkers

Testing and Maintenance

- Need a large test suite to attempt to verify the correctness of a compiler.
- Maintenance requires
  - Good programming style
  - Good documentation
  - Configuration management

Current Research Areas in Compilers

- Programming language support
- Computer architecture support
- Static analysis
- Memory management
- Program understanding support
- Tuning applications
- Dynamic compilation

Programming Language Support

- Support for new/popular programming languages
  - Java, C#, PHP, Python, Ruby
- Domain-specific languages and tools

Exploiting Architectural Features

- Instruction level parallelism
  - VLIW, SIMD
- Thread level parallelism
  - multicore, distributed computing, grid computing
- Support for specialized and low power, embedded processors
  - DSPs, GPUs, ASIPs

Static Program Analysis

- Dependence analysis
- Points-to analysis
- Interprocedural analysis

Memory Management Support

- Optimizations to improve memory hierarchy performance on static data.
- Dynamic memory management
  - Explicit storage reclamation
  - Garbage collection

Program Understanding Support

- Source code analysis
- Formal methods
- Verifying security properties
- Performance analysis
- Profiling
- Debugging

Tuning Applications

- Techniques
  - Autotuning
  - Iterative compilation
  - Continuous compilation
- Optimization space exploration
  - Phase and parameter selection
  - Phase order

Dynamic Compilation Systems

- Just-in-time compilation
- Dynamic optimizations