Introduction to Compiler Design

19CSE401 – Compiler Design

Introduction

- The world as we know it depends on programming languages.
- All the software running on the computers is written in some programming language.
- So before a program runs, it first must be translated into a form in which it can be executed by a computer.
- The system software that do this translation is called *Compiler.*

What is a Compiler?

- A system software to convert source language program to target language program.
- Validates input program to the source language specification – produces error messages/warnings.
- Primitive systems did not have Compilers, programs assembly language, hardcoded into machine code
- Compiler design started with FORTAN in 1950s
- Many tools have been developed for compiler design automation.

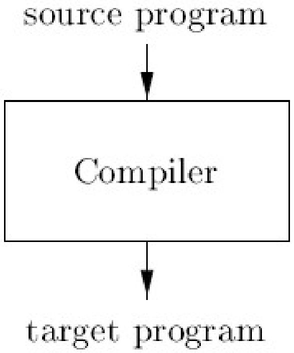
What, When and Why of Compilers

- What:
  - A compiler is a program that can read a program in one language and translates it into an equivalent program in another language.
- When
  - 1952, by Grace Hopper for A-0.
  - 1957, Fortran compiler by John Backus and team.

- Why? Study?
  - A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer.
  - For a computer to execute programs written in these languages, these programs need to be translated to a form in which it can be executed by the computer.

Language Processors

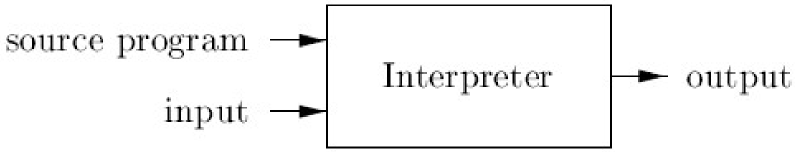
- A ***compiler*** is a program that can read a program in one language *(the source language)* and translate it into an equivalent program in another language *(the target language).*



- *An important role of the compiler is to report any errors in the source program that it detects during the translation process.*

Language Processors

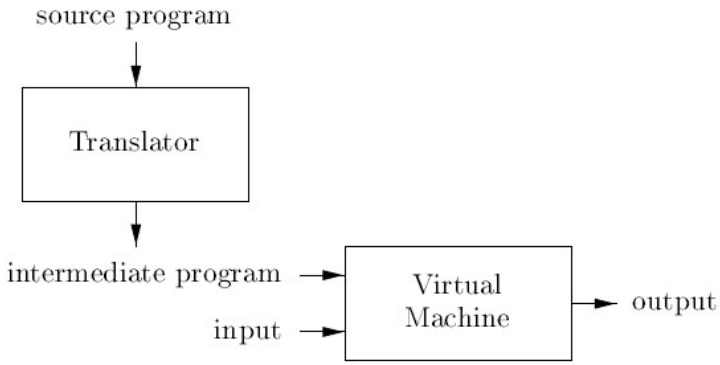
- An interpreter is another common kind of language processor.
- An interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.



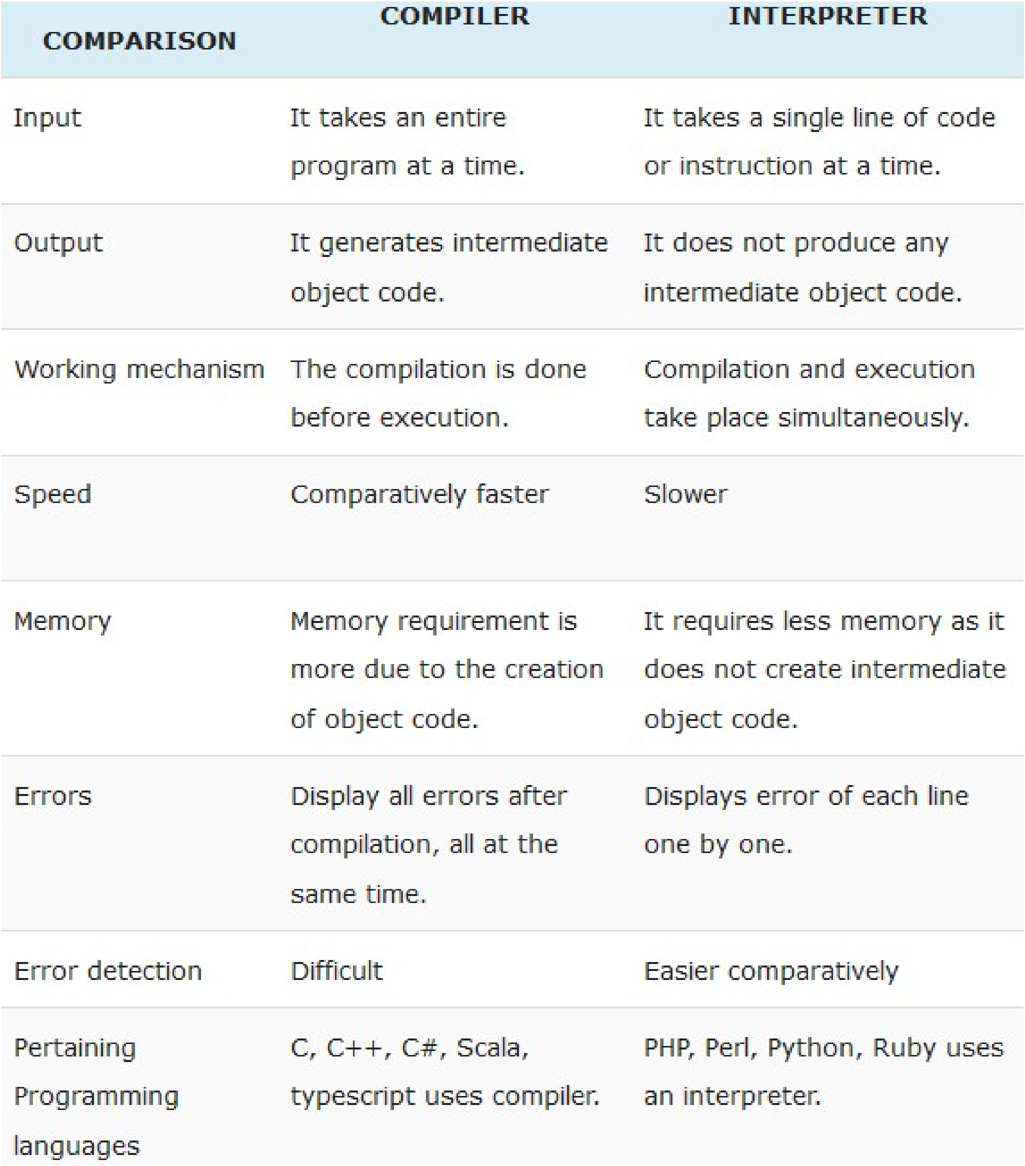
- The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs .
- An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

Java Language Processor

- Java language processors combine compilation and interpretation.

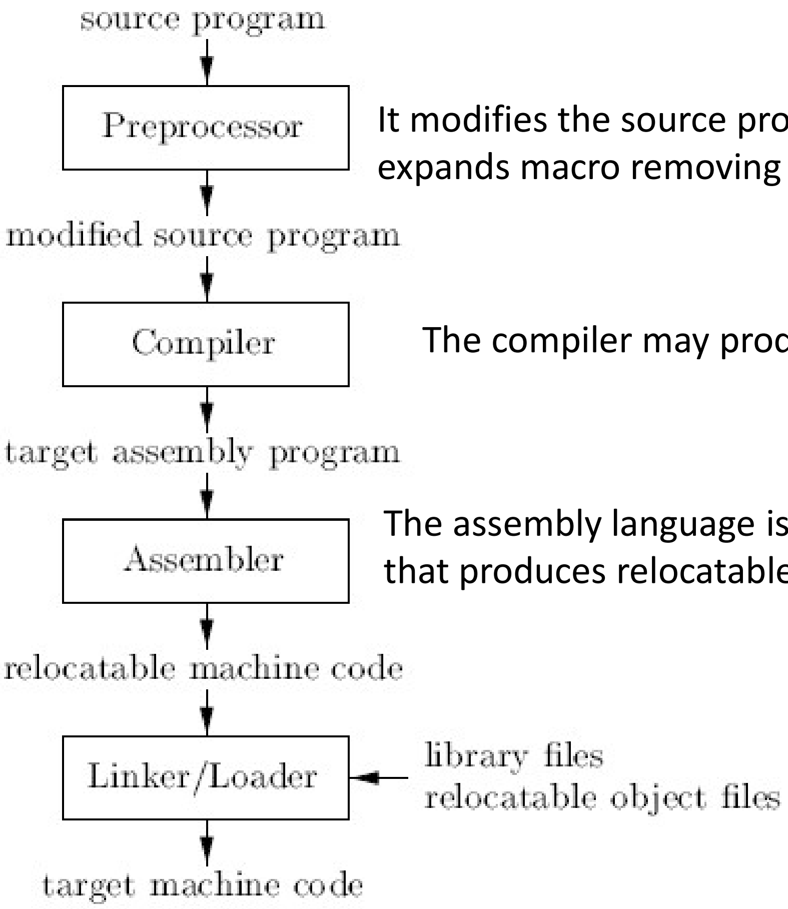


- A Java source program may first be compiled into an intermediate form called bytecodes.
- The bytecodes are then interpreted by a virtual machine.
- Advantage of this is that bytecodes compiled on one machine can be interpreted on another machine, perhaps across a network



COMPARISON	COMPILER	INTERPRETER
Input	It takes an entire program at a time.	It takes a single line of code or instruction at a time.
Output	It generates intermediate object code.	It does not produce any intermediate object code.
Working mechanism	The compilation is done before execution.	Compilation and execution take place simultaneously.
Speed	Comparatively faster	Slower
Memory	Memory requirement is more due to the creation of object code.	It requires less memory as it does not create intermediate object code.
Errors	Display all errors after compilation, all at the same time.	Displays error of each line one by one.
Error detection	Difficult	Easier comparatively
Pertaining Programming languages	C, C++, C#, Scala, typescript uses compiler.	PHP, Perl, Python, Ruby uses an interpreter.

A Language Processing System



It modifies the source program by removing comments,
expands macro removing \n and \t, etc

The compiler may produce an assembly-language program as its output,

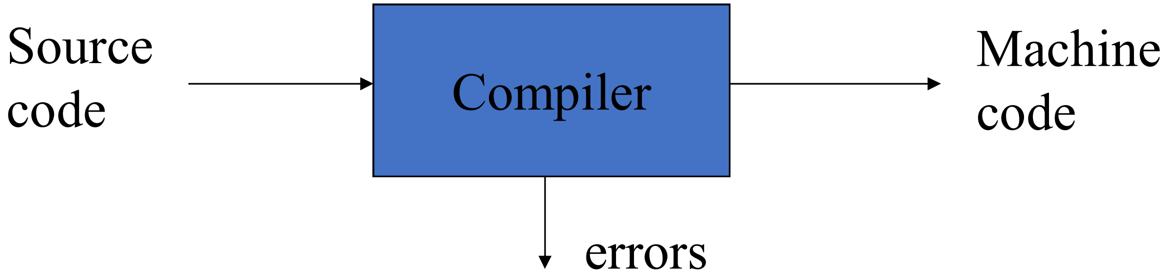
The assembly language is then processed by a program called an ***assembler***
that produces relocatable machine code as its output.

The ***linker*** resolves external memory addresses,
where the code in one file may refer to a location
in another file.
The **loader** then puts together all of the executable object files int

Application of Compiler technology

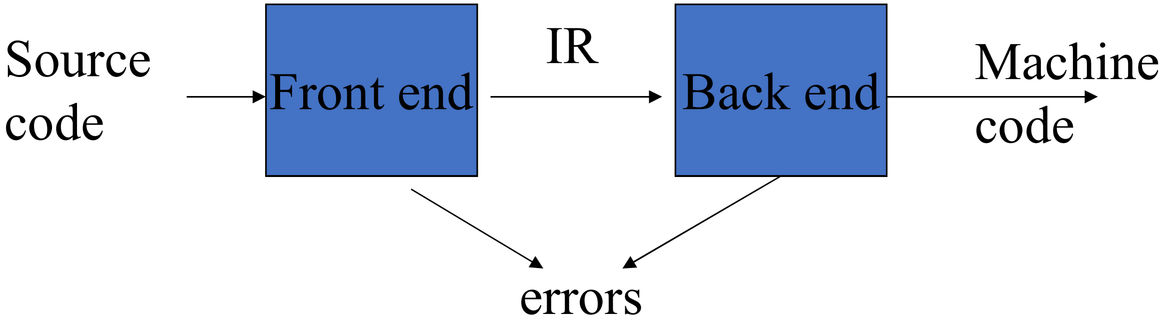
- Parsers for HTML in web browser
- Interpreters for javascript/flash
- Machine code generation for high level languages
- Software testing
- Program optimization
- Malicious code detection
- Design of new computer architectures

Abstract view



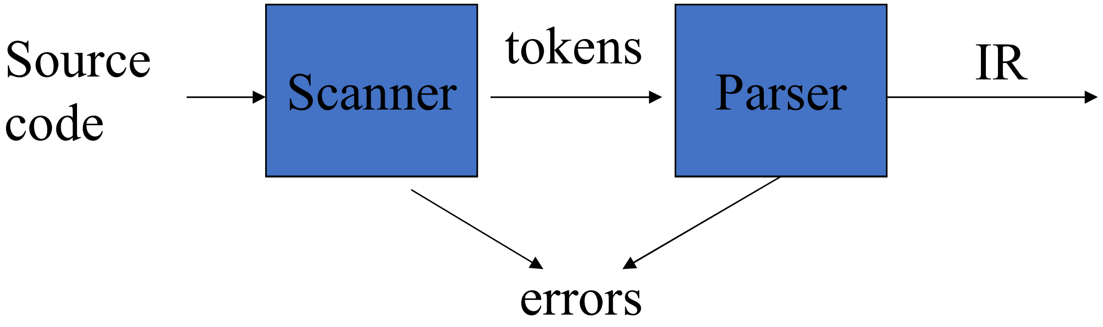
- Recognizes legal (and illegal) programs
- Generate correct code
- Manage storage of all variables and code
- Agreement on format for object (or assembly) code

Front-end, Back-end division



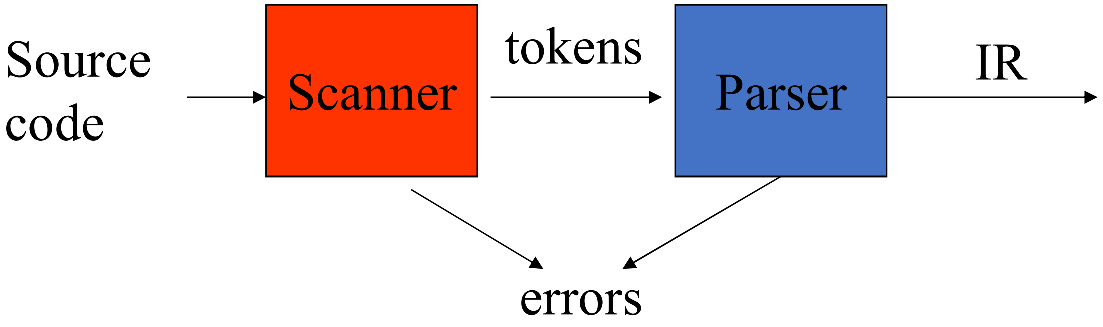
- Front end maps legal code into IR
- Back end maps IR onto target machine
- Simplify retargeting
- Allows multiple front ends
- Multiple passes -> better code

Front end



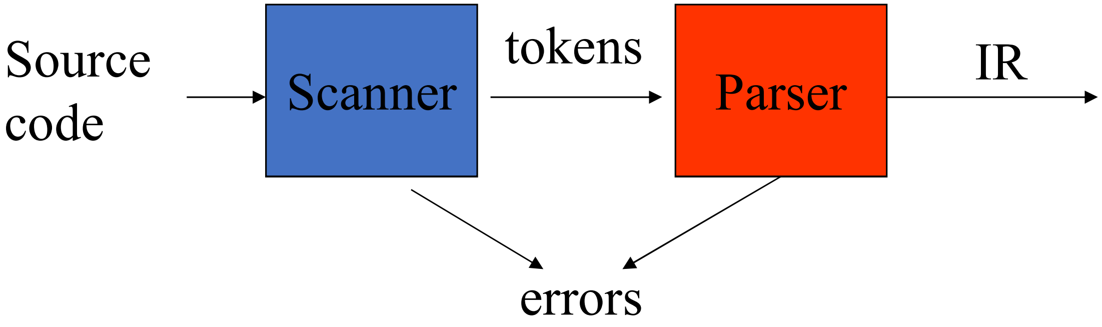
- Recognize legal code
- Report errors
- Produce IR
- Preliminary storage maps

Front end



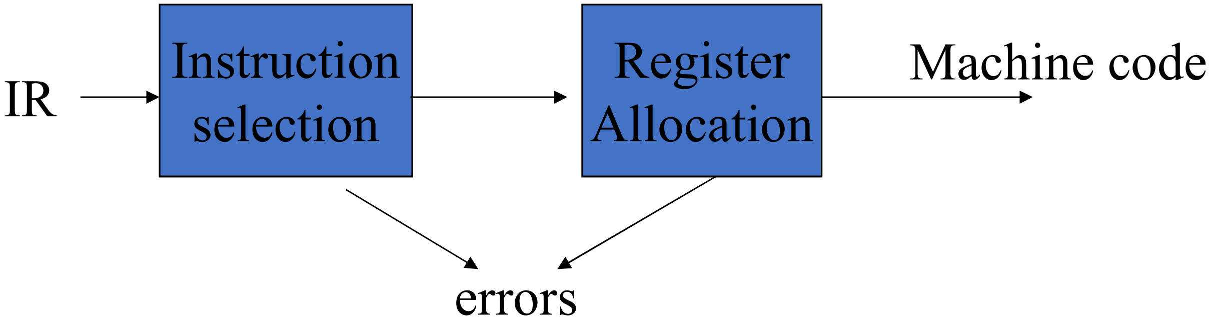
- Scanner:
    - Maps characters into tokens – the basic unit of syntax
        - x = x + y becomes <id, x> = <id, x> + <id, y>
    - Typical tokens: number, id, +, -, *, /, do, end
    - Eliminate white space (tabs, blanks, comments)
- A key issue is speed so instead of using a tool like LEX it sometimes needed to write your own scanner

Front end



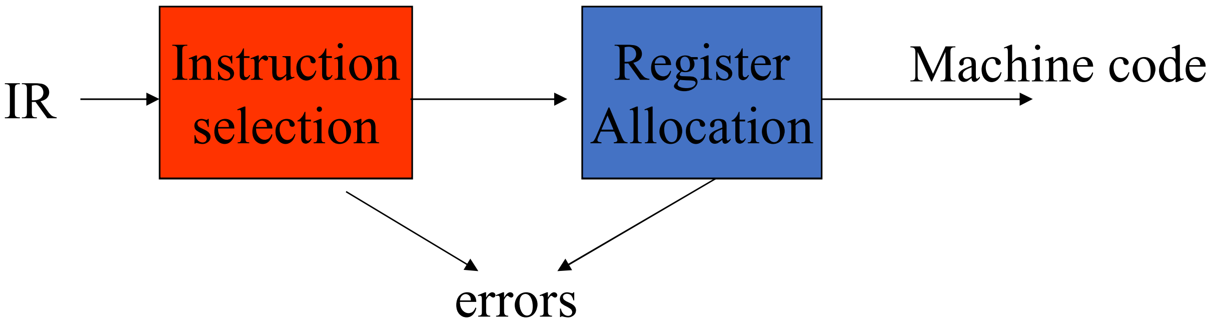
- Parser:
  - Recognize context-free syntax
  - Guide context-sensitive analysis
  - Construct IR
  - Produce meaningful error messages
  - Attempt error correction
- There are parser generators like YACC which automates much of the work

Back end



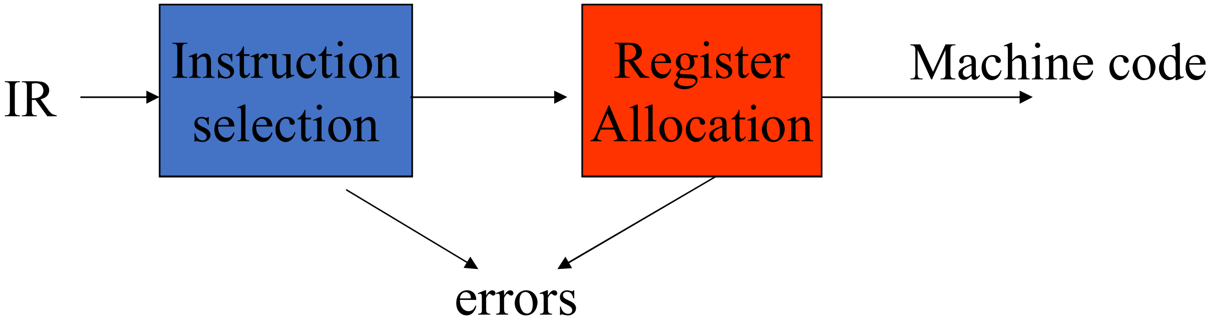
- Translate IR into target machine code
- Choose instructions for each IR operation
- Decide what to keep in registers at each point
- Ensure conformance with system interfaces

Back end



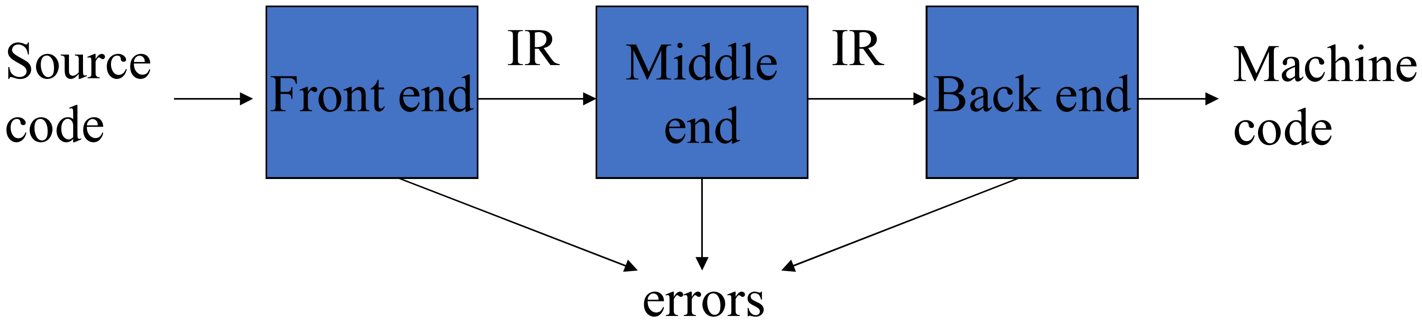
- Produce compact fast code
- Use available addressing modes

Back end



- Have a value in a register when used
- Limited resources
- Optimal allocation is difficult

Traditional three pass compiler



- Code improvement analyzes and change IR
- Goal is to reduce runtime

Middle end (optimizer)

- Modern optimizers are usually built as a set of passes
- Typical passes
  - Constant propagation
  - Common sub-expression elimination
  - Redundant store elimination
  - Dead code elimination