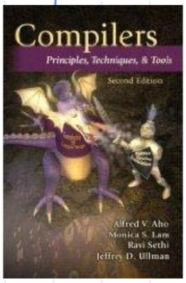
編譯器設計

黃元欣

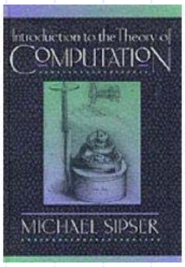
shin@csie.ntust.edu.tw

Syllabus

- Text Books



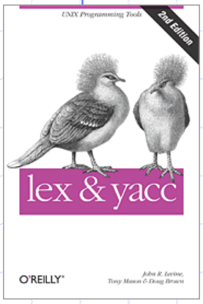
  - *Compilers: Principles, Techniques, and Tools 2e*
    A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman
    Addison-Wesley 2007
    ISBN 0-321-48681-1



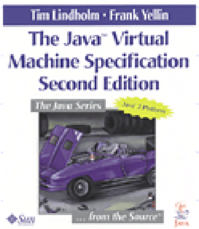
  - *Introduction to the Theory of Computation*
    Michael Sipser
    PWS Publishing 1997
    ISBN 0-534-94728-X

Syllabus

- Reference Books



  - *lex & yacc*
    J.R. Levine, T. Mason, and D. Brown
    O'Reilly 1995
    ISBN 1-56592-000-7



  - *The Java^{TM} Virtual Machine Specification, 2nd Ed.*
    Tim Lindholm and Framk Yellin
    Addison-Wesley 1999
    ISBN 0-201-43294-3
    http://java.sun.com/docs/books/vmspec

Syllabus

- Course Outline
  - Introduction
  - Lexical Analysis (Chap. 3)
  - Syntax Analysis (Chap. 4)
  - Syntax-Directed Translation (Chap. 5)
  - Run-time Organization (Chap.7)
  - Intermediate Code Generation (Chap.6)
  - Code Generation (Chap. 8)

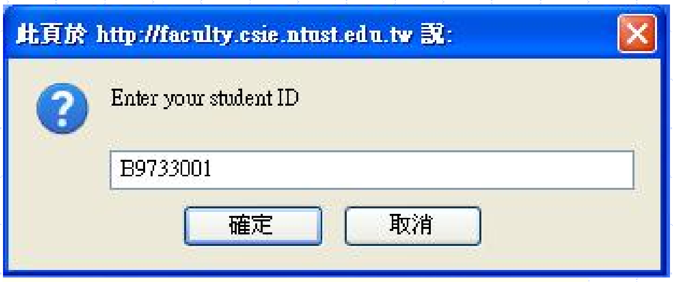
Syllabus

- Grading
  - Programming Assignments 30%
  - Midterm 30%
  - Final 40%
- Office Hours
  - M8, T8 (T4-512, Tel: 6746)

Account Registration

You need to get an account at the class home page in order to submit programming assignments and download lecture slides

http://faculty.csie.ntust.edu.tw/~shin/compilers.html

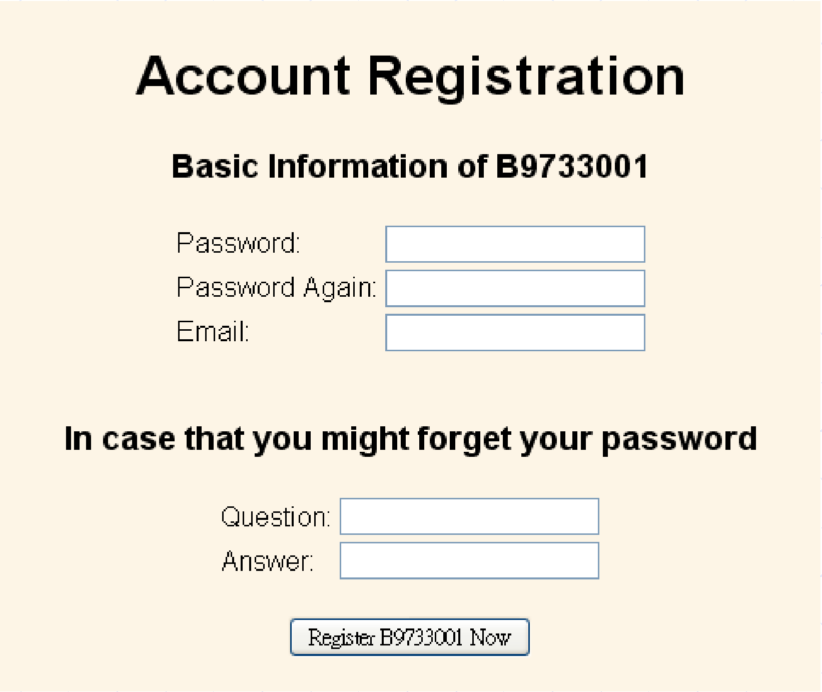


此頁於 http://faculty.csie.ntust.edu.tw 說:

Enter your student ID

B9733001

Account Registration



Account Registration

Basic Information of B9733001

Password:

Password Again:

Email:

In case that you might forget your password

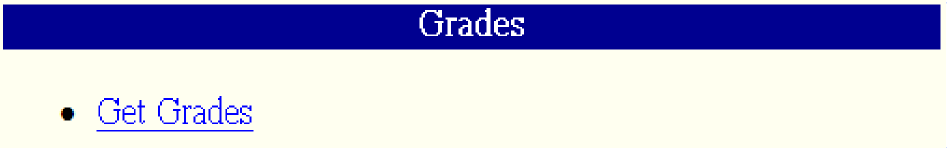
Question:

Answer:

Grades

You can get the scores of all your examinations and programming assignments at the class home page:

http://faculty.csie.ntust.edu.tw/~shin/compilers.html



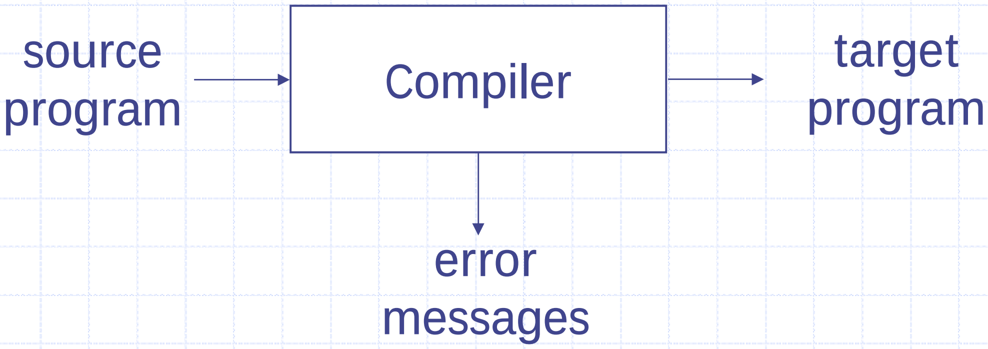
Grades

- Get Grades

編譯器設計

Chapter 1: Introduction

Compilers



source program

Compiler

target program

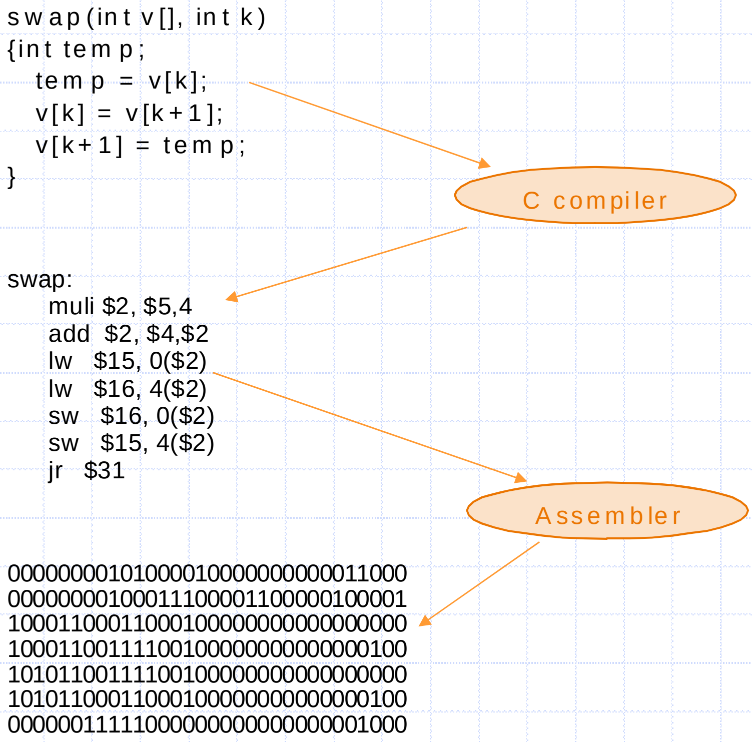
error messages

- A compiler is a program that
  - reads a program written in one language (the *source* language) and
  - translates it into an equivalent program in another language (the *target* language)

Compilers

- Examples
  - Compilers of C/C++, Fortran, Java, etc
  - Text formatters, e.g. TeX, LaTeX
  - Silicon compilers
  - Query interpreters, e.g SQL compilers
  - Preprocessors
  - Assemblers
  - Browsers
  - Parallelizing compilers

Compilers and Assemblers



High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Assembly language program (for MIPS)

```
swap:
     muli $2, $5,4
     add  $2, $4,$2
     lw   $15, 0($2)
     lw   $16, 4($2)
     sw   $16, 0($2)
     sw   $15, 4($2)
     jr   $31
```

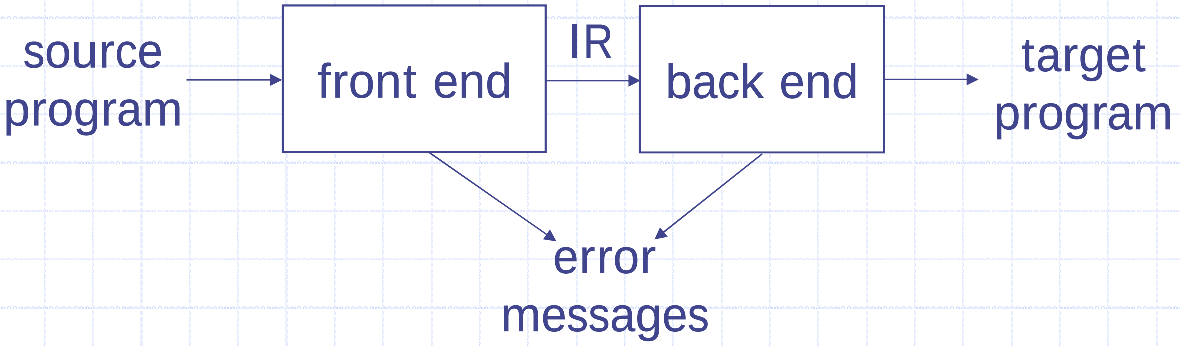
Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Adapted from *Computer Organization & Design* by Patterson & Hennessy. Copyright 1998 Morgan Kaufmann Publishers

Analysis-Synthesis Model

- There are two parts to compilation
  - Analysis (front end)
    - Breaks up the source program into constituent pieces
    - Creates an intermediate representation (IR)
  - Synthesis (back end)
    - Constructs the desired target program from the IR
    - (Optionally) performs optimizations

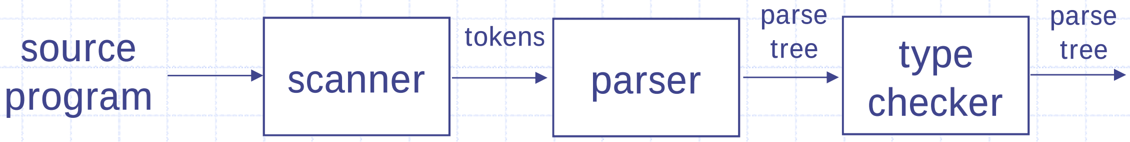


Analysis-Synthesis Model

- Some tools that perform analysis
  - Structure editors
    - Takes a sequence of commands as input to build a source program, e.g. with the user types while, the editor supplies the matching do
  - Pretty printers
    - Analyzes a program and prints it in such a way that the structure of the program becomes clearly visible
  - Static checkers
    - Reads and analyzes a program, and attempts to discover potential bugs without running the program
  - Interpreters
    - Instead of producing a target program as a translation, an interpreter performs the operations implied by the program

Analysis of the Source Program

- Analysis (front end) consists of 3 phases:
  - Linear Analysis (Lexical Analysis)
    - scan characters and group them into tokens
  - Hierarchical Analysis (Syntax Analysis)
    - group tokens into grammatical phrases
  - Semantic Analysis
    - identify semantic errors and gather type information



source program → scanner → tokens → parser → parse tree → type checker → parse tree

Lexical Analysis

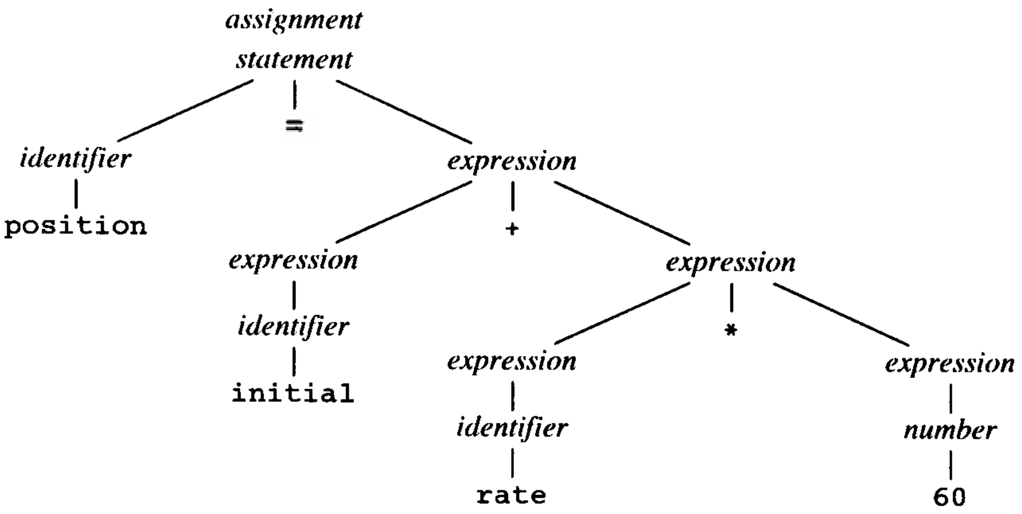
- Mapping characters into tokens
  - Tokens: the basic unit of syntax
  - position = initial + rate * 60

    becomes

    〈id, position〉 = 〈id, initial〉 + 〈id, rate〉 * 60

Syntax Analysis

- Tokens are grouped into grammatical phrases that are used to synthesize output



Syntax Analysis

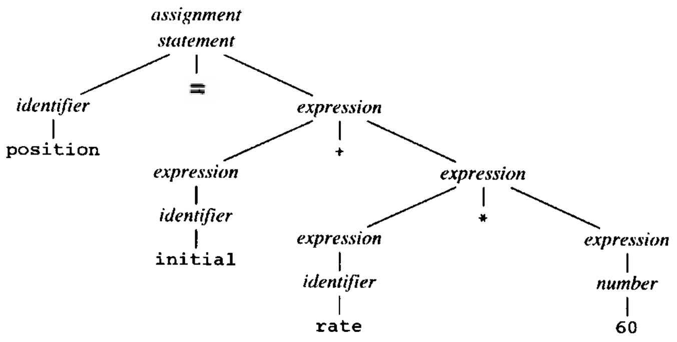
- The hierarchical structure of a program is usually expressed by recursive rules, e.g.
  - Any *identifier* is an expression
  - Any *number* is an expression
  - If *expression*$_1$ and *expression*$_2$ are expressions, so are

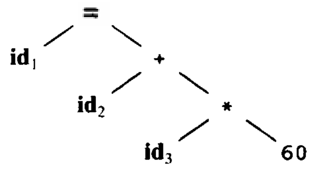
    expression$_1$ op *expression*$_2$

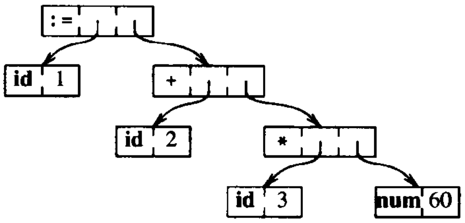
    (*expression*$_1$)

Parse Tree vs. Syntax Tree

- Parse tree
  - describes the syntactic structure of the source program
- Syntax tree
  - A more common internal representation of this syntactic structure
  - A compressed representation of the parse tree

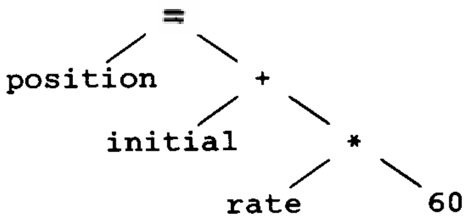


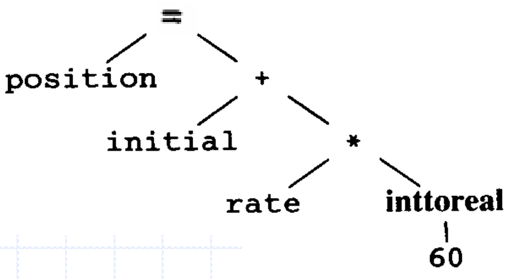




Semantic Analysis

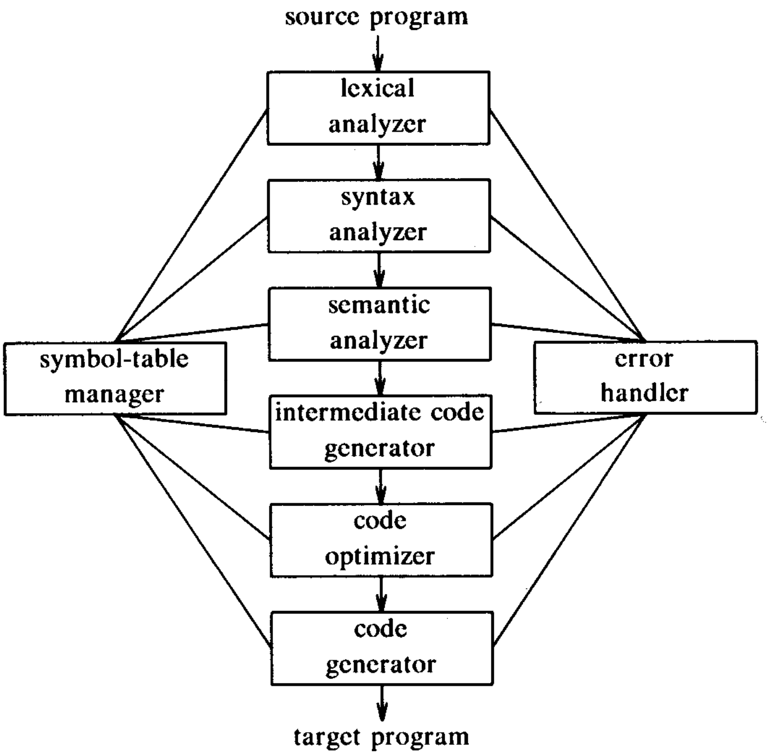
- Checks for semantic errors
- Gathers type information for the subsequent code-generation phase





Phases of a Compiler

- A compiler operates in phases, each of which transforms the source program from one representation to another



Symbol-Table Management

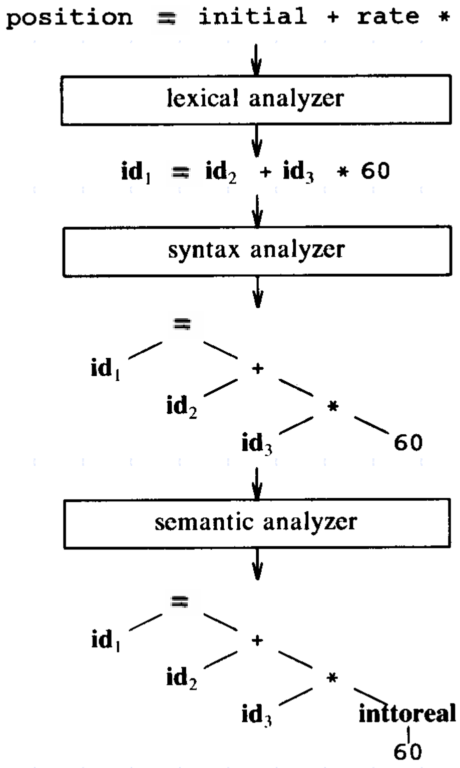
- Essential function of a compiler
  - To record the identifiers used in the source program and collect information about various attributes of each identifier
    - e.g. allocated storage, type, scope, etc.
- Symbol table
  - A data structure containing a record for each identifier, with fields for the attributes
  - When an identifier is detected by the lexical analysis, it is entered into the symbol table
  - The attributes are determined during syntax analysis and semantic analysis
  - e.g. `float position, initial, rate;`

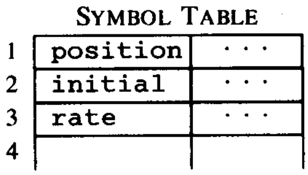
Error Detection and Reporting

- Each phase can encounter errors
  - After detecting an error, a phase must deal with the error, so that compilation can proceed
    - allowing further errors to be detected
  - Lexical phase can detect errors where characters remaining in the input do not form any token
  - Syntax analysis phase detects errors where the token stream violates the syntax of the language
  - Semantic analysis phase tries to detect constructs that have the right syntactic structures but no meaning to the operation involved
    - e.g. `a = b + c;` where b is an array and c an integer

Analysis Phases

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis



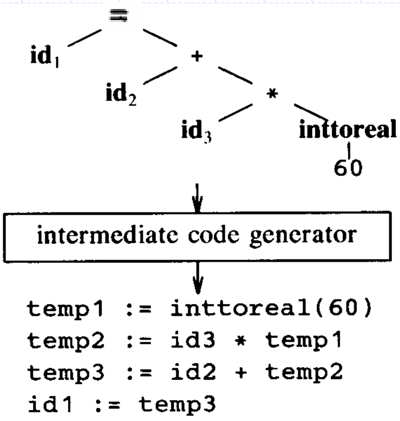


SYMBOL TABLE

1	position	· · ·
2	initial	· · ·
3	rate	· · ·
4		

Intermediate Code Generation

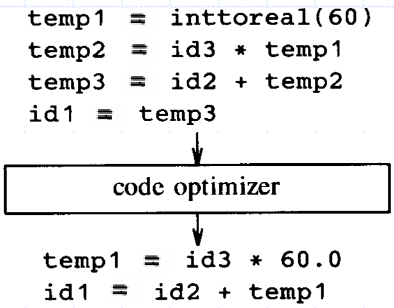
- Two properties
  - Easy to produce
  - Easy to translate into the target program
- Examples
  - Graph representations
  - Postfix notation
  - Three-address code



```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

Code Optimization

- Attempts to improve the intermediate code
  - So that faster-running machine code will result

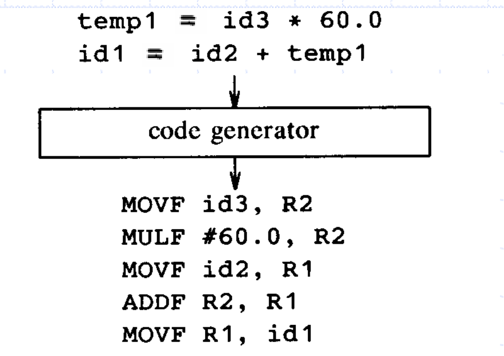


```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

Code Generation

- Generates target code
  - Consisting of relocatable machine code or assembly code



```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

Cousins of the Compiler

- Preprocessors
  - Preprocessors produce input to compilers
  - Macro processing
    - Allows users to define macros
  - File inclusion
    - Includes header files into the program text, e.g. `#include <stdio.h>`
  - “Rational” preprocessors
    - Augment older languages with more modern flow-of-control and data-structuring facilities
  - Language extensions
    - Add capabilities to languages by what amounts to built-in macro
      - e.g. HPF

Cousins of the Compiler

- Assemblers

```
mov a, R1          0001 01 00 00000000
add #2, R1         0011 01 10 00000010
mov R1, b          0010 01 00 00000100
```

- Loaders and Link-Editors

Compiler-Construction Tools

- Some general tools have been created for the automatic design of specific compiler components
  - Parser generators
    - Producing syntax analyzers, normally from input that is based on a context-free grammar
  - Scanner generators
    - Automatically generating lexical analyzers, normally from a specification based on regular expressions
  - Automatic code generators
    - Taking a collection of rules that define the translation of each operation of the intermediate language into the machine language