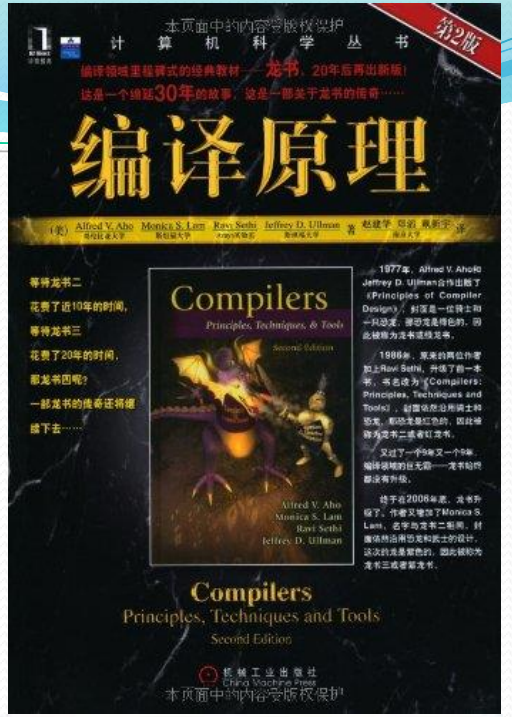
编译原理

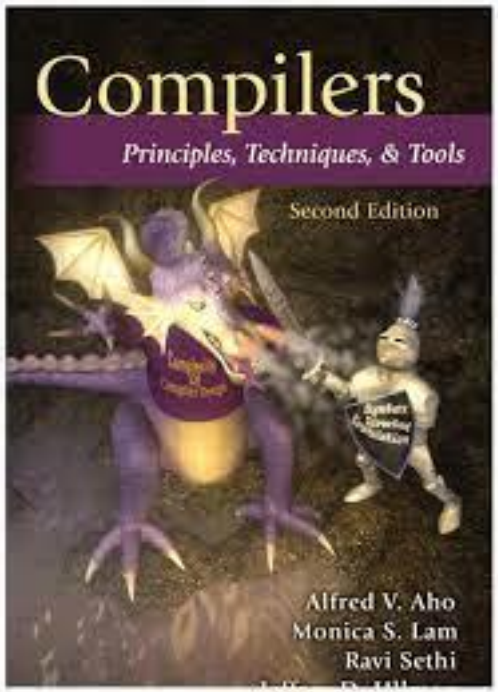
第一章　引　论

南京大学人工智能学院
戴新宇

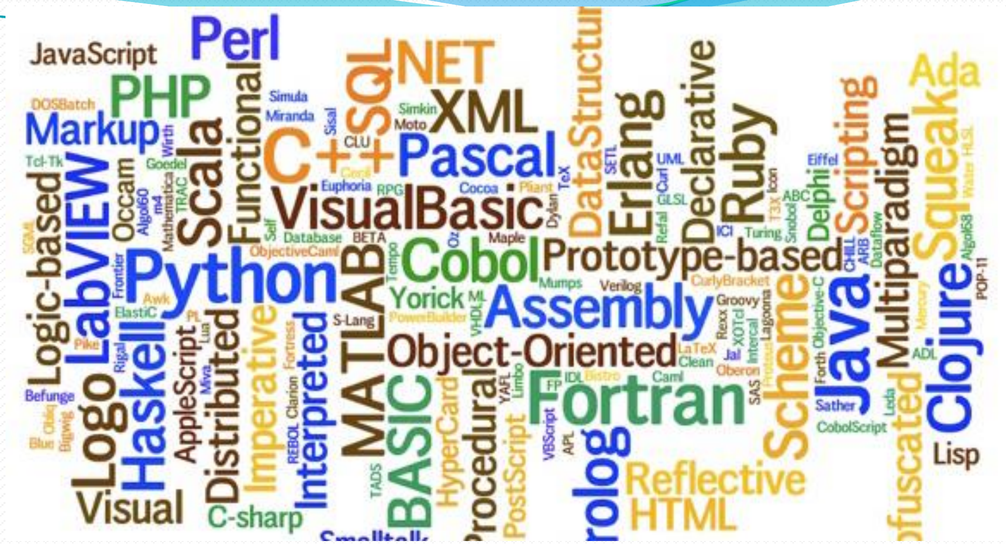
课程概要

- 教材：编译原理（龙书第二版）
- 学时：32学时(课堂)
- 学分：2
- 周二: 10-12am， 仙I-103
- 助教: 徐婷
- 课程考核：
  - 课堂
  - 作业
  - 期末考试

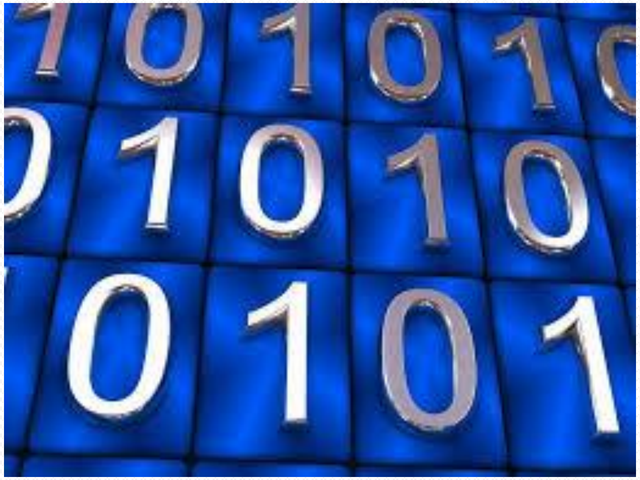




课程第一印象？

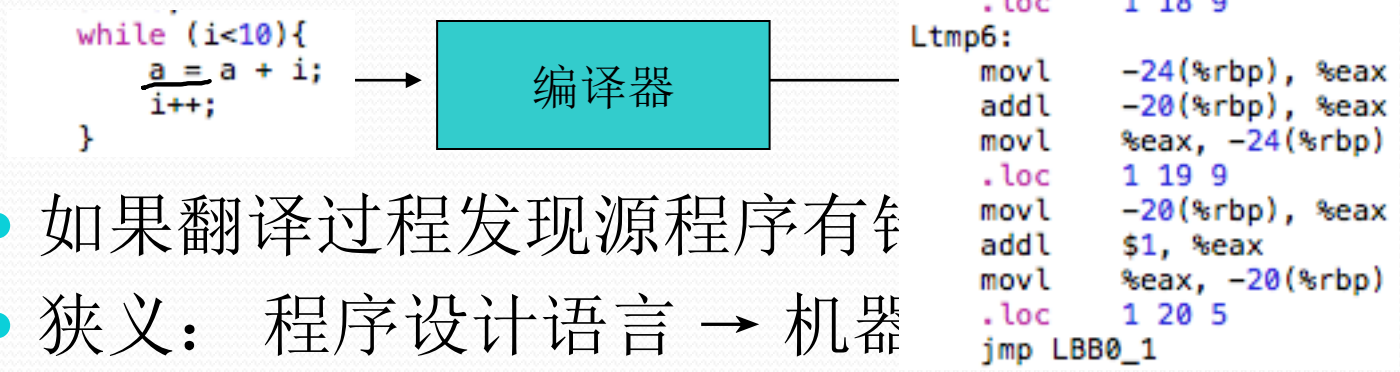

JavaScript
Perl
PHP
.NET
XML
Markup
SQL
C++
Pascal
Scala
Functional
VisualBasic
DataStructure
Erlang
Declarative
Ruby
Ada
Scripting
Squeak
Multiparadigm
Python
Cobol
Prototype-based
Logic-based
LabVIEW
Haskell
MATLAB
Yorick
Assembly
Object-Oriented
Java
Clojure
Logo
Distributed
Imperative
Interpreted
BASIC
Fortran
Scheme
Delphi
Visual
C-sharp
Reflective
HTML
Lisp
Prolog
Procedural
PostScript
HyperCard
AppleScript


CreateRectRgn(80,160,80+RectSizeX
PtInRegion(point))
double xOc,yOc;
for(int i=0;i<NUMCITY;i++)
{
xOc=80+RectSizeX*



什么是编译器

- 一个编译器就是一个程序，读入以某一种语言（源语言）编写的程序，并把该程序[illegible]等价的、用另一种语言（目标语言）[illegible]



- 如果翻译过程发现源程序有[illegible]
- 狭义： 程序设计语言 → 机器[illegible]
- 广义：程序变换 C++ → C →汇编 Python → C

课程目标

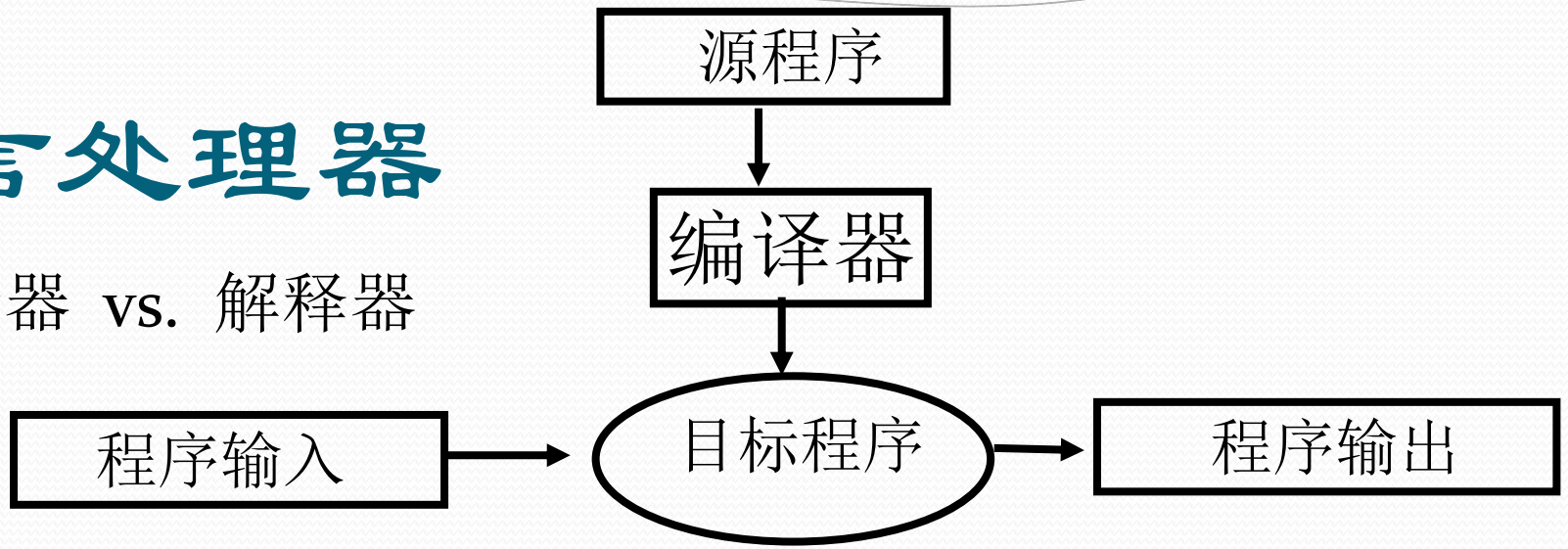
- 目标：掌握编译原理的基础理论和技术
- 此外
  - 帮助我们更好的理解程序设计语言和机器体系结构
  - 帮助我们写出更加高效的程序

下列程序片断哪一个更快?

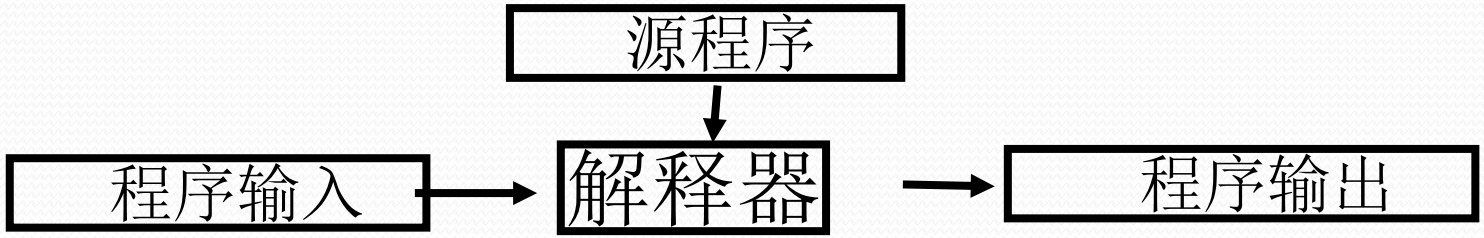
```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
      sum += A[i][j];

 for (j = 0; j < N; j++)
      for (i = 0; i < M; i++)
      sum += A[i][j];
```

语言处理器

- 编译器 vs. 解释器



- 效率高，一次编译，多次运行
- 通常目标程序是可执行的

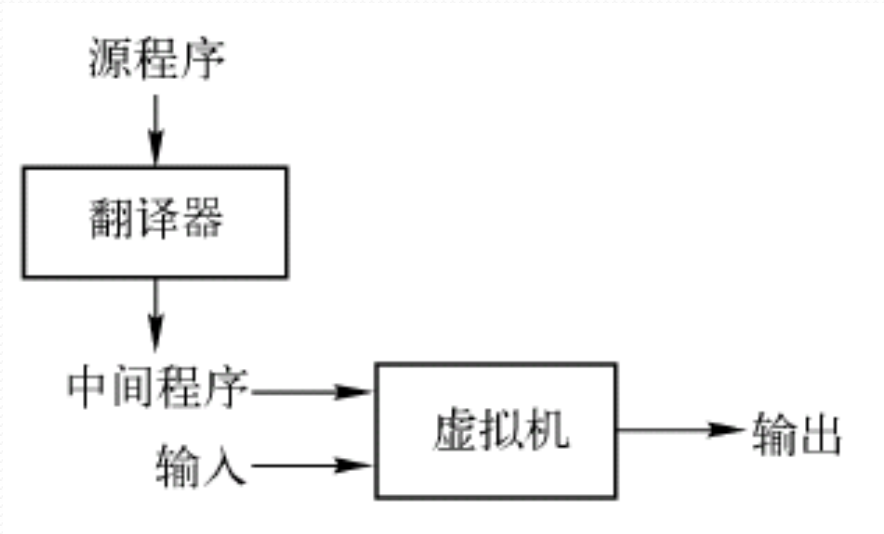


- 直接利用用户提供的输入，执行源程序中指定的操作。
- 不生成目标程序，而是根据源程序的语义直接运行。
- 边解释，边执行，错误诊断效果好。

编译器 VS. 解释器

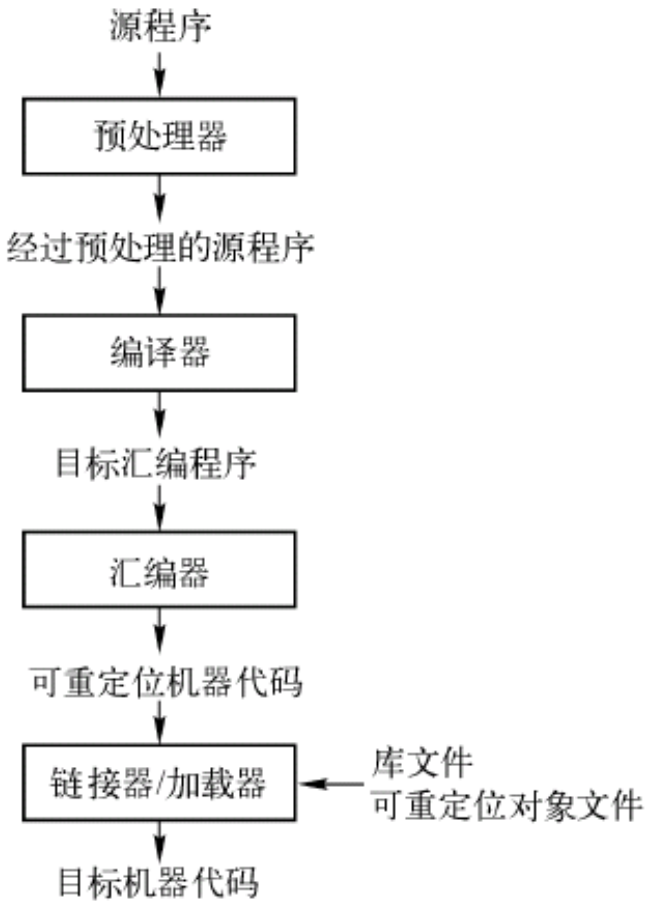
Javac Hello.java
Java Hello

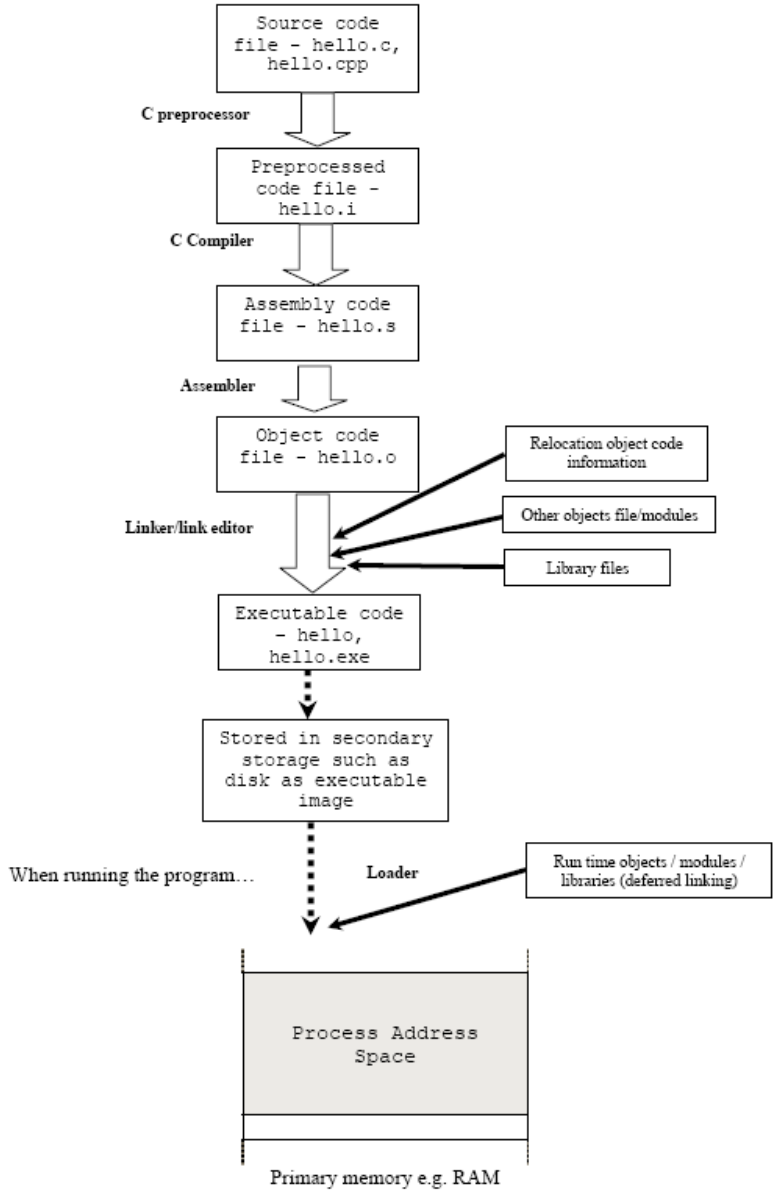
- Java结合了两者：



典型语言（如C）的编译

- 预处理器
- 汇编器
- 链接器
- 加载器



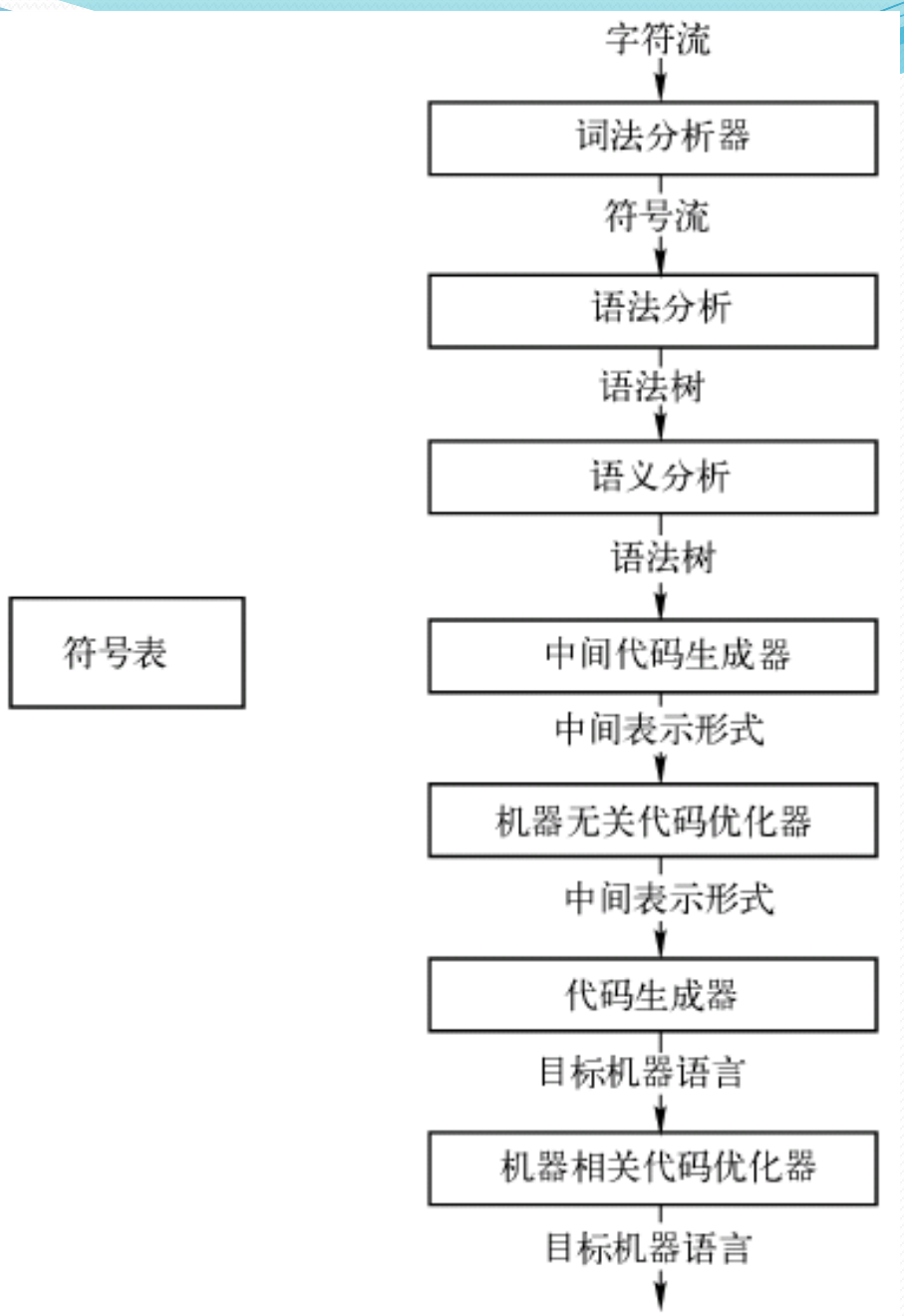


编译器的结构

- 分析部分（Analysis）
  - 源程序 - 语法结构 - 中间表示
  - 搜集源程序中的相关信息，放入符号表
  - 分析、定位程序中可能存在的错误信息（语法、语义错误）
  - 又称编译器的前端（front end），是于机器无关的部分
- 综合部分（Synthesis）
  - 根据符号表和中间表示构造目标程序
  - 又称编译器的后端（back end），是于机器相关的部分

编译器中的若干步骤

- 每个*步骤*把源程序的一种表示方式转换成另一种表示方式。
- 实践中，某些中间表示不需要明确的构造出来。
- 符号表存放源程序的相关信息，可由各个步骤使用



符号表管理

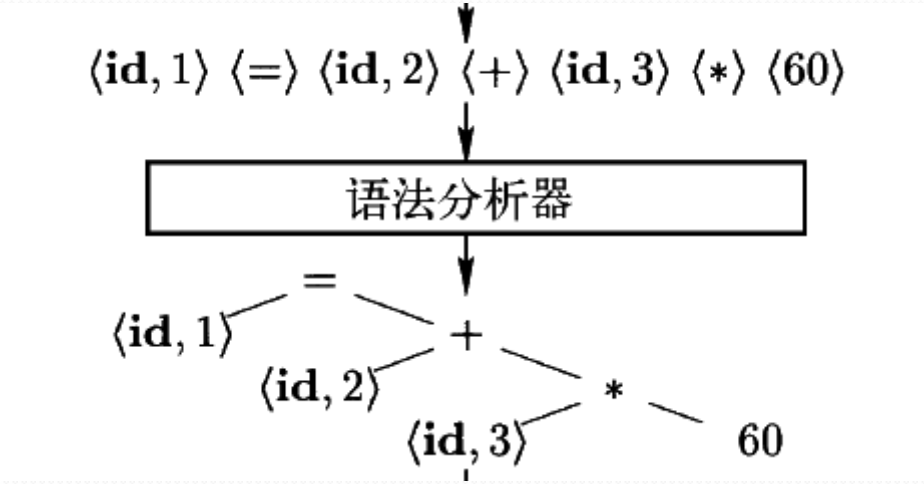
- 记录源程序中使用的变量的名字，收集各种属性
  - 名字的存储分配
  - 类型
  - 作用域
  - 过程名字的参数数量、参数类型等等
- 符号表可由编译器的各个步骤使用

词法分析（lexical analysis, scanning）

- 词法分析/扫描读入源程序的字符流，输出有意义的词素(lexeme)
  - 基于词素，产生词法单元：

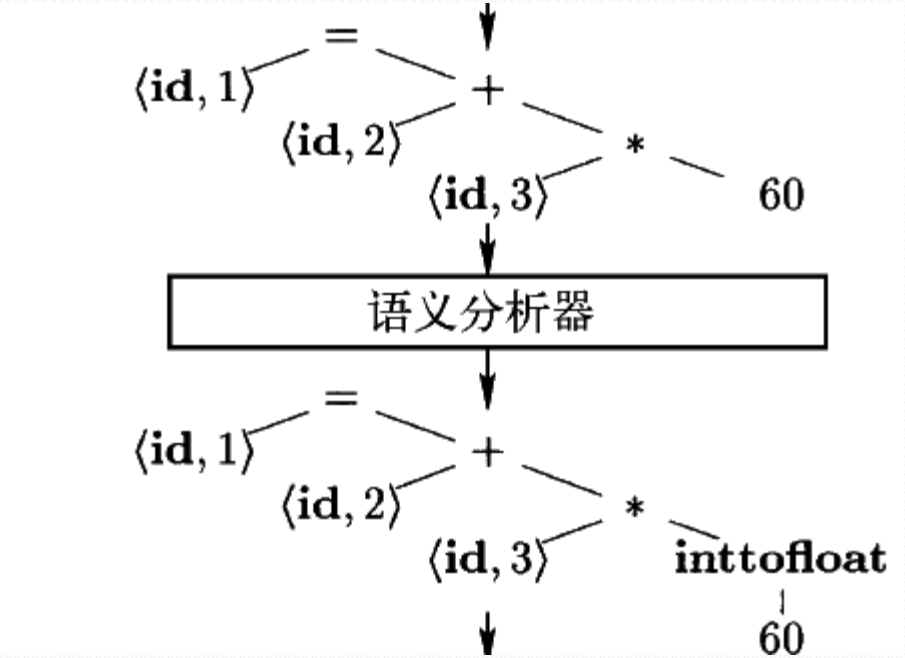
    <token-name, attribute-value>
  - token-name由语法分析步骤使用
  - attribute-value指向相应的符号表条目，由语义分析/代码生成步骤使用
- 例子
  - position = initial + rate * 60
  - <id,1> <=, > <id, 2> <+, > <id,3> <*, > <number, 4>

语法分析（syntax analysis/parsing）



- 词法分析后，需要得到词素序列的语法结构
- 语法分析/解析
  - 根据各个词法单元的第一个分量来创建树形中间表示形式。通常是语法树（syntax tree）
  - 指出了词法单元流的语法结构。

语义分析(Semantic Analysis)



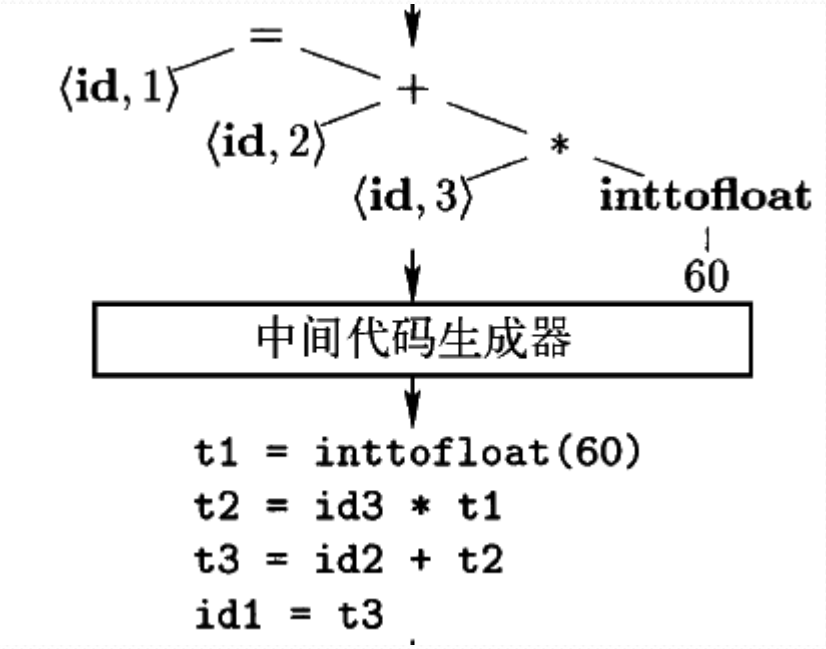
- 得到语义(meaning)，对于编译器来说比较难
- 语义分析
  - 使用语法树和符号表中的信息，检查源程序是否满足语言定义的语义约束。
  - 同时收集类型信息，用于代码生成。
  - 类型检查，类型转换。

语义分析

- Jack said Jerry left his assignment at home.
- Jack said Jack left his assignment at home?

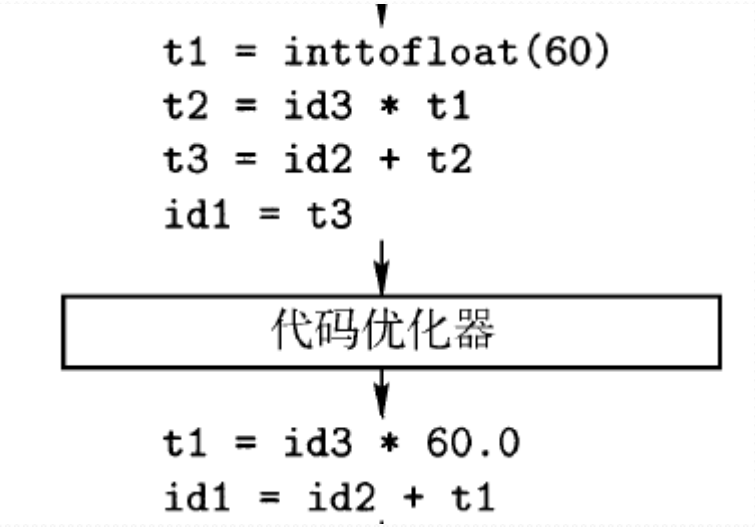
```
{
    int Jack = 3;
    {
        int Jack = 4;
        cout << Jack;
    }
}
```

中间代码生成(Intermediate-Code Generation)



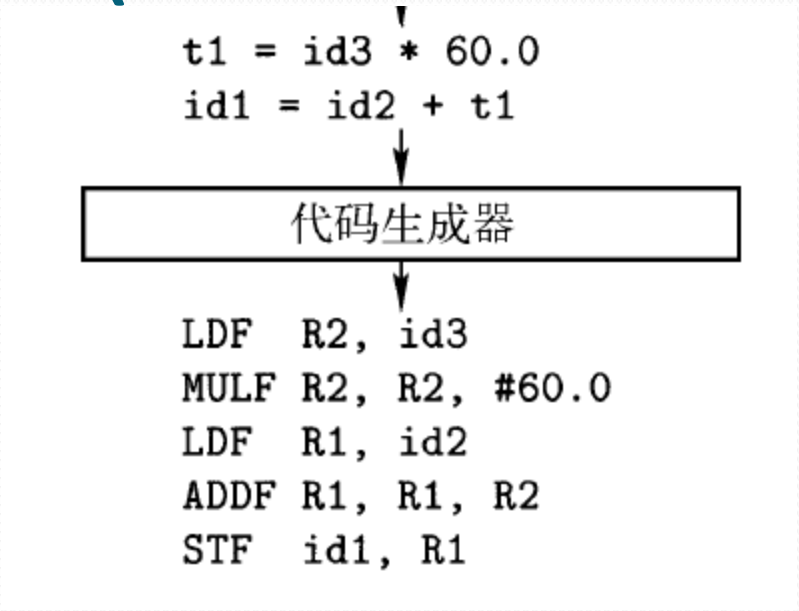
- 根据语义分析的输出，生成类机器语言的中间表示
- 三地址代码：
  - 每个指令最多包含三个运算分量
  - t1 = inttofloat(60)； t2 = id3 * t1; t3 = id2 + t2;

代码优化(Code Optimization)



- 通过对中间代码的分析，改进中间代码，得到更好的目标代码
  - 运行的更快、占用更少的内存：少占资源
- 优化有具体的设计目标

代码生成(Code Generation)



- 把中间表示形式映射到目标语言
  - 寄存器的分配
  - 指令选择
  - 内存分配

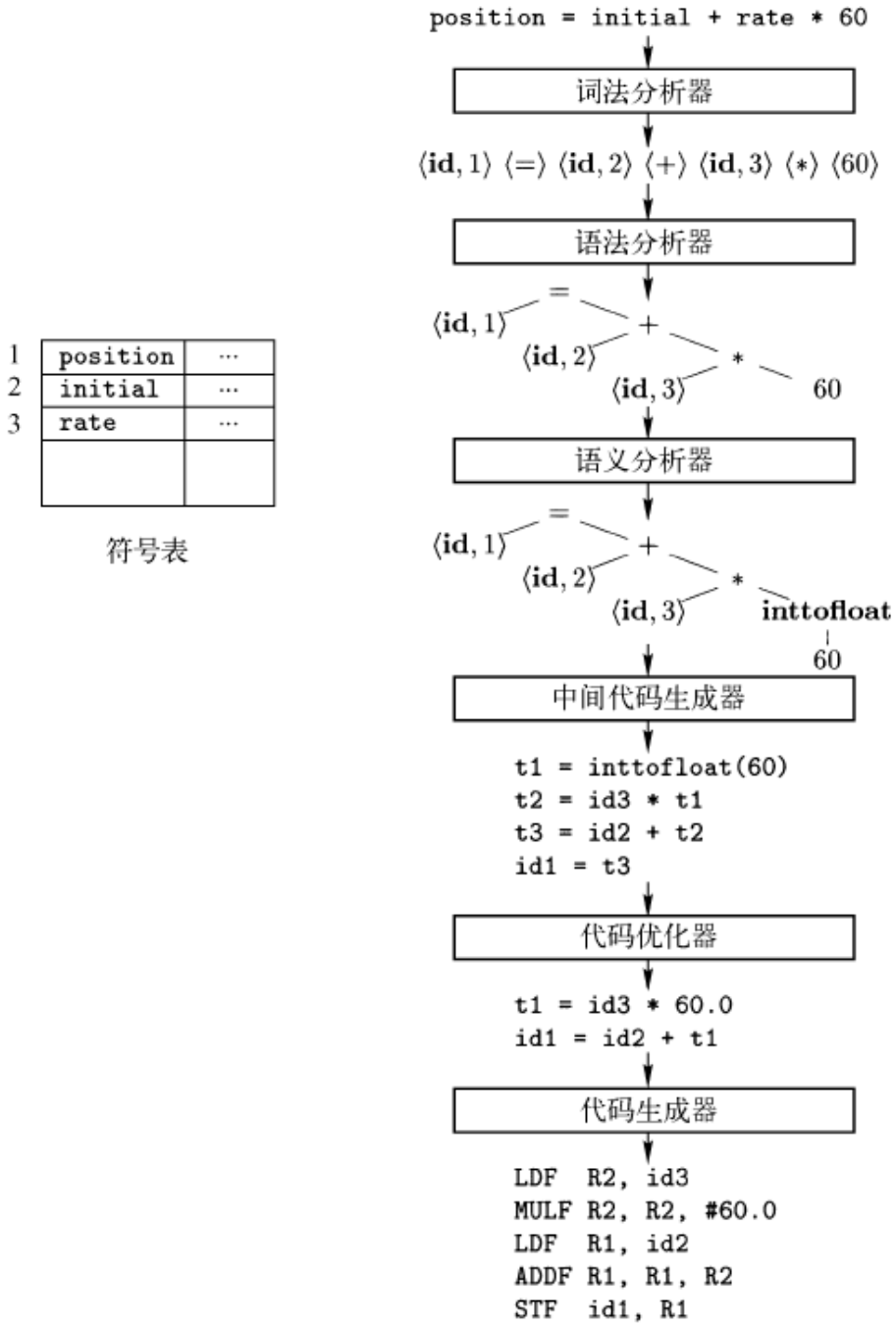


图 1-7　一个赋值语句的翻译

编译器的趟（Pass）

- 趟：以文件为输入输出单位的编译过程的个数，每趟可由一个或若干个步骤构成
- “步骤”是逻辑组织方式
- “趟”和具体的实现相关

编译器简介

- 编译器 vs. 解释器
- 编译器的结构
- 编译的构造工具

编译器的构造工具

- 扫描器的生成器: lex/flex
  - 根据一个语言的词法单元的正则表达式描述生成词法分析器
- 语法分析器的生成器： yacc/bison
  - 根据一个程序设计语言的语法描述自动生成语法分析器
- 语法制导的翻译引擎
  - 生成一组用于遍历分析树并生成中间代码的程序
- 代码生成器的生成器
  - 依据中间语言的每个运算如何翻译成目标机上机器语言的一组规则，生成一个代码生成器
- 数据流分析引擎
  - 收集数据流信息，用于优化
- 编译器构造工具集

编译器的处理对象-程序语言

程序设计语言

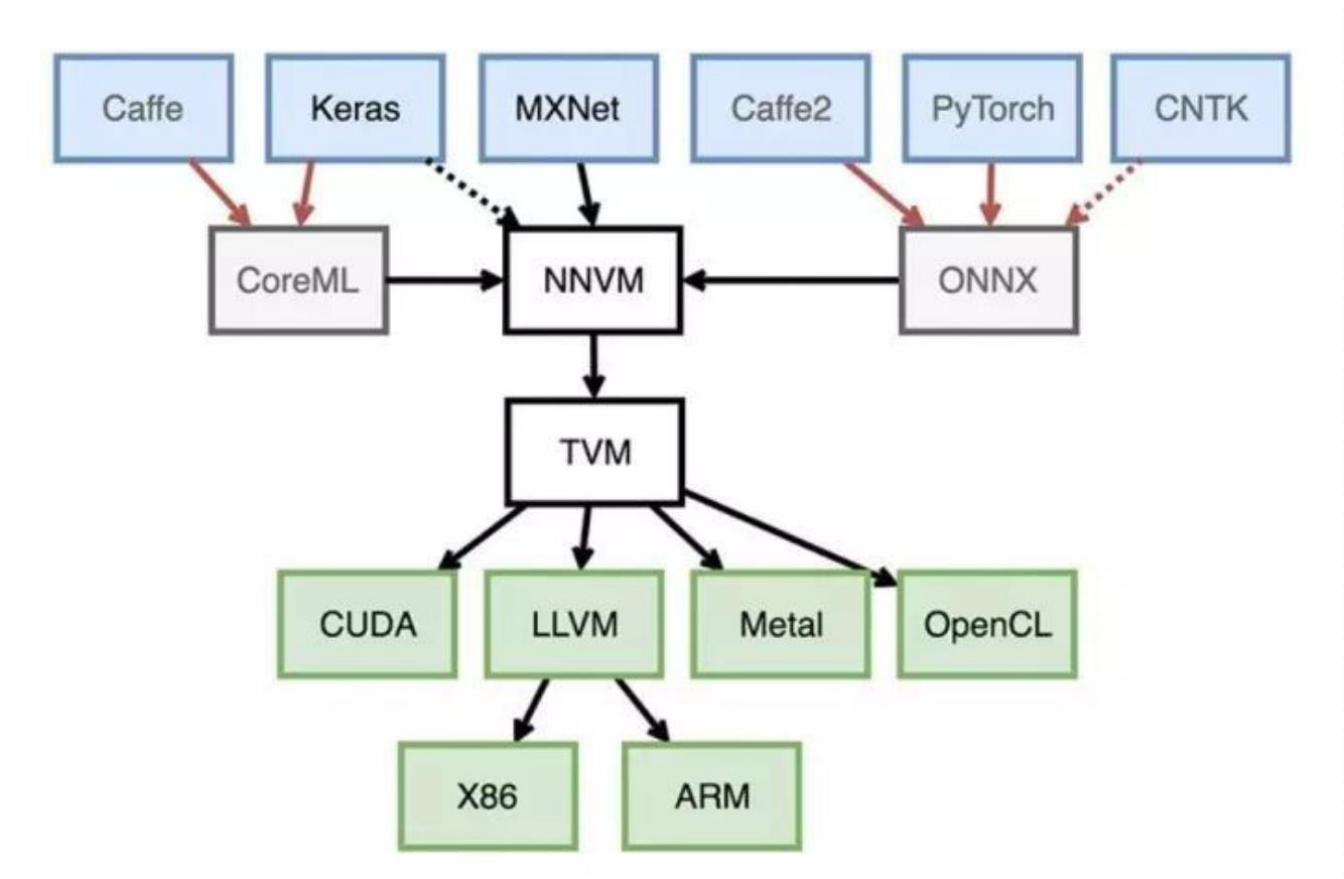
- 语言的代分类
  - 第一代语言：机器语言
  - 第二代语言：汇编语言
  - 第三代语言：高级程序设计语言
    - Fortran, Pascal, Lisp, Modula, C
  - 第四代：特定应用语言：,SQL, Postscript
  - 第五代：基于逻辑和约束的语言，Prolog、OPS5

- 面向对象语言
  - Simula, Smalltalk, Modula3, C++, Object Pascal, Java, C#
  - 数据抽象、继承
- 面向过程语言

程序设计语言和编译器之间的关系

- 程序设计语言的新发展向编译器设计者提出新要求
  - 设计相应的算法和表示方法来翻译和支持新的语言特征
- 通过降低高级语言的执行开销，推动这些高级语言的使用
- 编译器设计者还需要更好地利用新硬件的能力。

人工智能编译器


Caffe
Keras
MXNet
Caffe2
PyTorch
CNTK
CoreML
NNVM
ONNX
TVM
CUDA
LLVM
Metal
OpenCL
X86
ARM

第一课总结


CreateRectRgn(80,160,80+RectSizeX
PtInRegion(point))
double xOc,yOc;
for(int i=0;i<NUMCITY;i++)
xOc=80+RectSizeX*