编译原理

田玲 教授、博导

lingtian@uestc.edu.cn

第五章 编译概述

主要内容

1.编译的一些基本概念

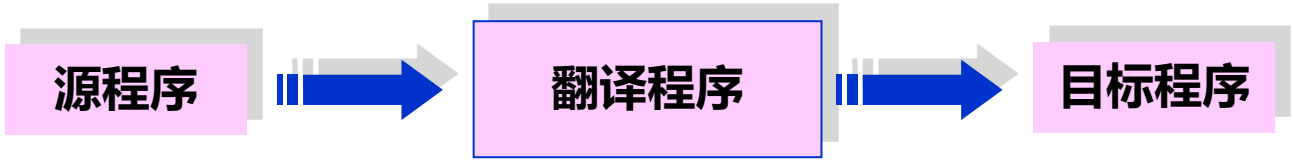
2.编译的五个阶段

翻译，编译，解释等概念

词法分析、语法分析、语义分析、优化、代码生成以及它们的关系

基本概念

- **翻译**:将一种语言编写的程序转换成完全等效的另一种语言编写的程序的过程称为**翻译**（translate）；在计算机中，翻译由一个程序来实现，称为**翻译程序**(translator)。



- **编译**:将高级语言程序翻译成低级语言程序称为**编译**（compile）；实现编译的程序称为**编译程序**(compiler)。
- **汇编程序**：将汇编语言程序翻译成机器语言的程序，称为**汇编程序**。

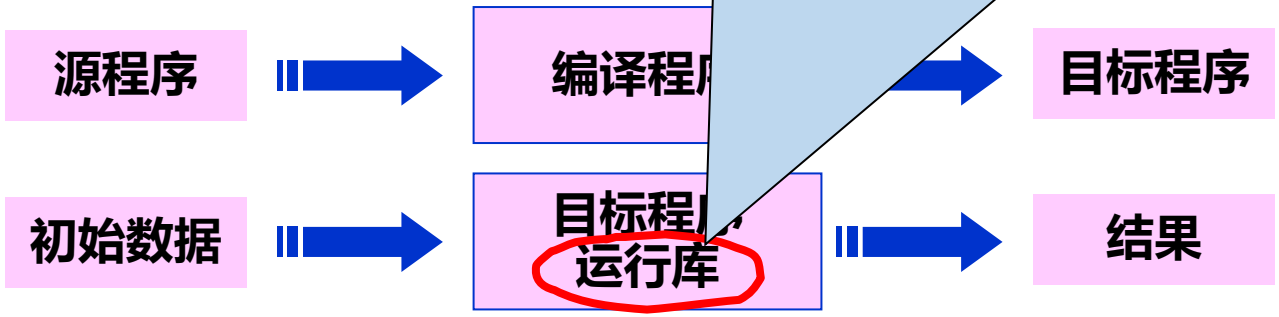
第一节 引言

- **宿主语言**:编写编译程序的语言称为**宿主语言**；源语言、目标语言、宿主语言通常是不同的语言；
- 如果一个编译程序
  则称该编译程序为
- 如果编译程序是用源语言编写的，则称该编译程序是**自编译的**（self-compiling）；
- 如果一个编译程序
  则称为**交叉编译**（

例:在PC上将FORTRAN编写的程序编译成PC机的机器码；而编译器是由C语言编写的；

例:C的编译程序

一系列程序执行时需要的辅助子程序；编译时未将它们插入到目标程序；



第一节 引言

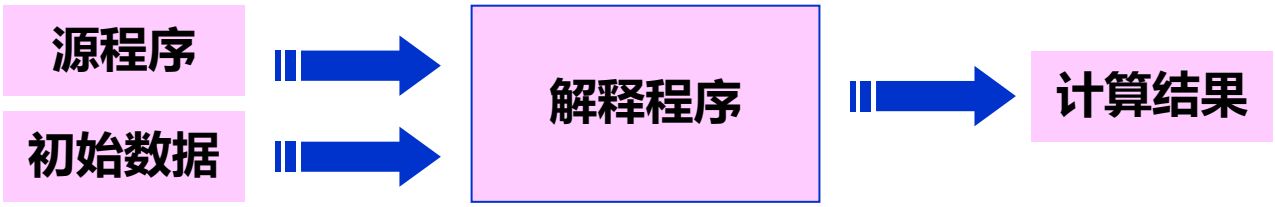
- **解释**:不将源程序翻译成目标程序，而是一边分析，一边执行，这种翻译方法称为**解释**；实现解释的程序，称为**解释程序**(interpreter)

例:BASIC语言解释执行；JAVA语言也是解释执行。

解释的优缺点

1.特别适合于动态语言和交互式环境，便于人机对话

2.解释器边解释边执行，重复执行的程序需要重复翻译，比编译执行花费更多时间，执行效率低；



程序解释执行示意图

第二节 编译步骤

1.**词法分析**：从左向右扫描源程序，并进行分析，识别出符合词法规则的**单词符号**（token）；如果出错，则给出出错信息；

2.**语法分析**：对由词[illegible]语法规则进行分析，识别出**语法单位**，给出[illegible]出出错信息；

如基本字、标识符、常数、运算符和界符等

3.**语义分析**：按照语义[illegible]大多数编译器采用中间语言来描述程序[illegible]

例如表达式、短语、子句、句子和子程序、程序等

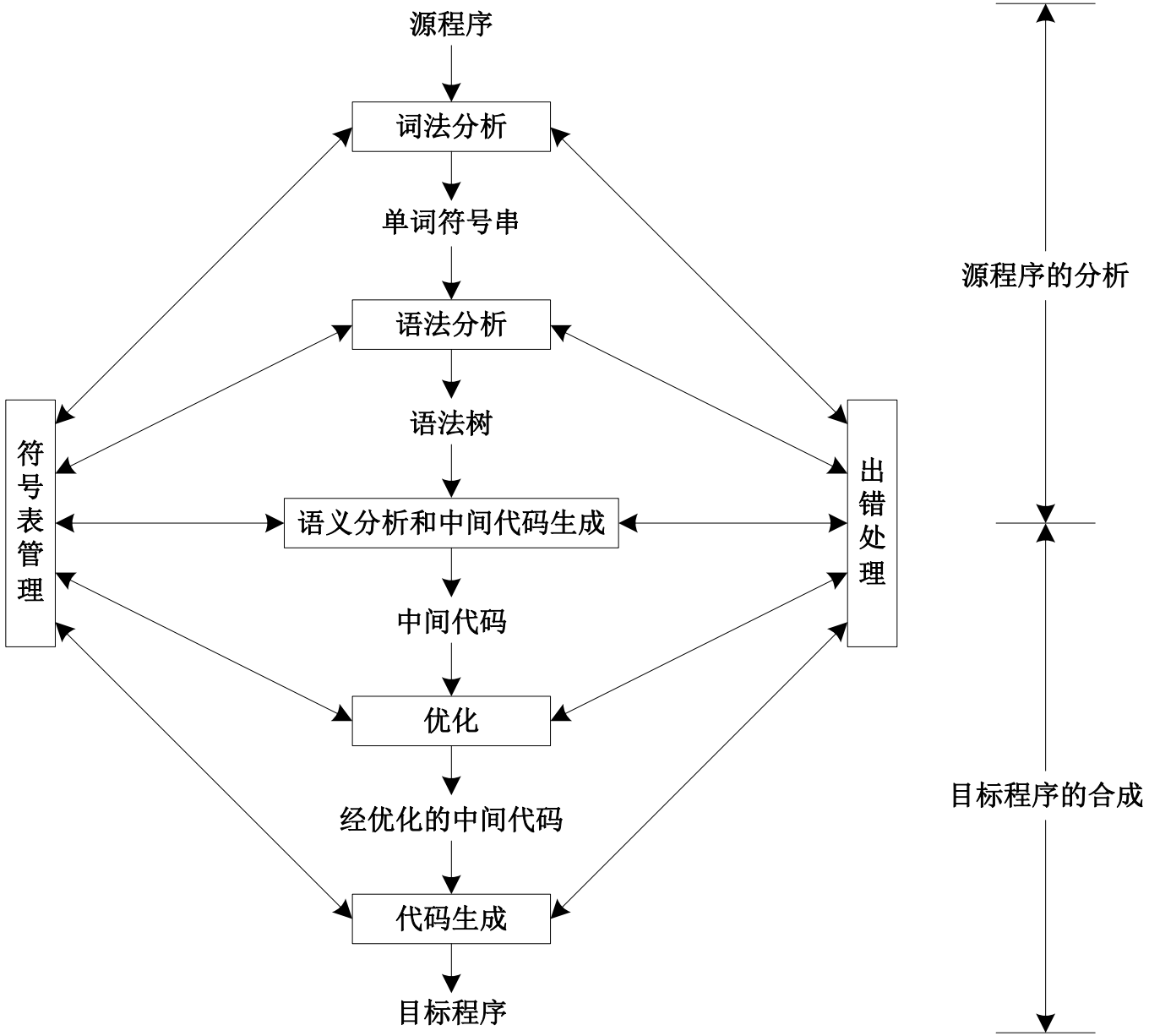
4.优化：对中间代码进行一些等价变换，使生成的目标程序效率更高；

5.目标代码生成：将中间代码转换成目标代码；

6.**符号表管理**：符号表存储程序中各种数据对象和实体的属性，编译程序负责对这些表格进行创立和维护；

7.**出错处理**：编译程序会发现很多的程序错误，尤其在语法分析阶段；出错处理将报告错误的性质、发生错误的地方等；

第二节 编译步骤


源程序
词法分析
单词符号串
语法分析
语法树
语义分析和中间代码生成
中间代码
优化
经优化的中间代码
代码生成
目标程序
符号表管理
出错处理
源程序的分析
目标程序的合成

实例分析

实例：一个C程序（片段）的编译过程。

- …
- int aaa, bbb, ccc;
- …
- aaa = bbb + ccc * 100;
- …

第1步：词法分析

1. 识别出源程序中的单词符号，将aaa、bbb、ccc三个标识符和常数100加入符号表。

符号表

符号编号	符号属性（名称、类型、值、…）
sym1	aaa、标识符、…
sym2	bbb、标识符、…
sym3	ccc、标识符、…
sym4	100、常数、…
…	…

第1步：词法分析

2. 识别赋值语句中的单词符号，生成单词符号串（ 7个单词符号）：

sym1 = sym2 + sym3 * sym4

其中，sym1、sym2、sym3和sym4分别代表aaa、bbb、ccc和100在符号中的编号，通过编号可以查到相应符号的各个属性。

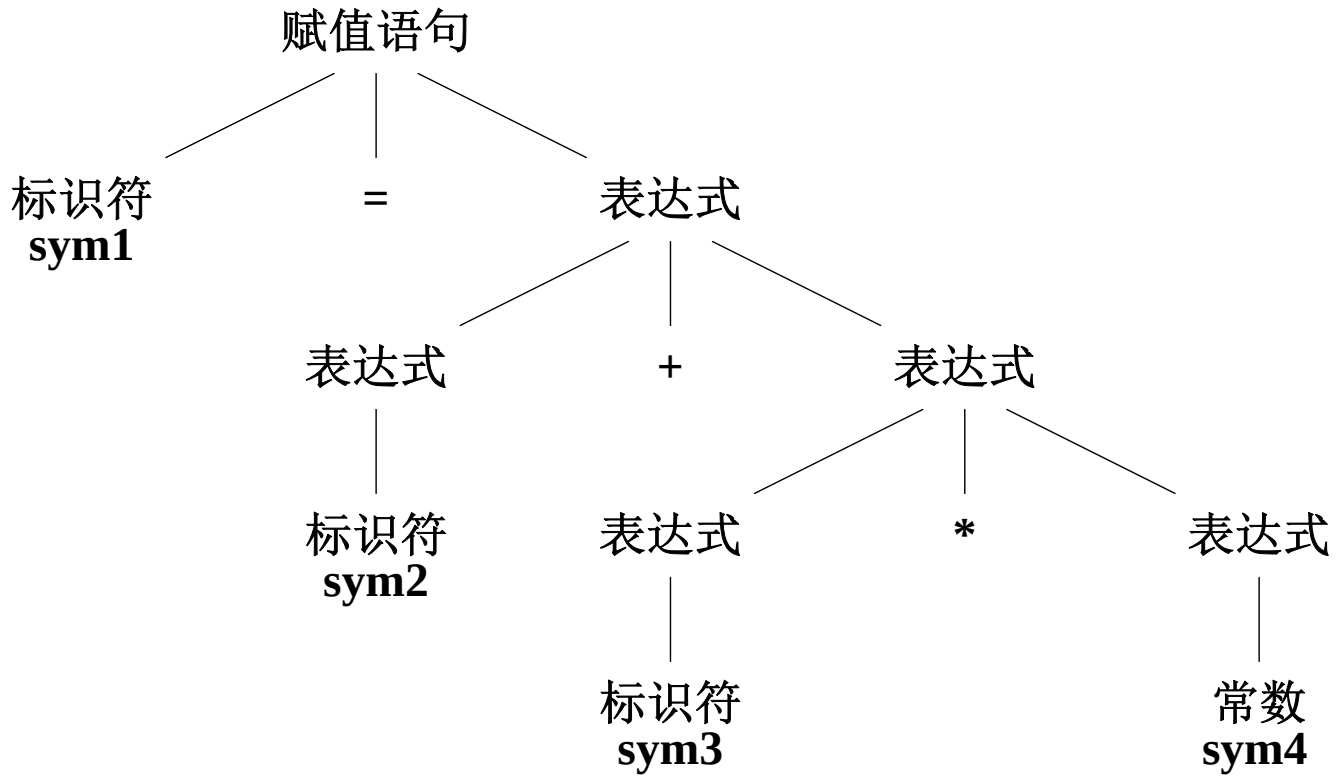
第2步：语法分析

1）进行语法的合法性检查

- “标识符=表达式”是合法的赋值语句
- “标识符”是合法的表达式
- “常数”是合法的表达式
- “表达式+表达式”是合法的表达式
- “表达式*表达式”是合法的表达式

第2步：语法分析

2) 各个语法单位和语法树的形成



第3步：语义分析和中间代码生成

1) 语义分析：语义的合法性检查，例如：类型检查、类型转换。

2) 中间代码生成：生成语义等价，结构更简单的中间代码。

- temp1 = sym4
- temp2 = sym3 * temp1
- temp3 = sym2 + temp2
- sym1 = temp3

第4步：优化

代码的等价变换，取消多余的数据和操作，提高代码的“时空”效率。

- temp1 = sym3 * sym4
- sym1 = sym2 + temp1

第5步：目标代码生成

根据目标语言的规范，生成相应的汇编程序。
关键点：1.变量地址的分配；2.寄存器的分配。

- MOV R2, sym3
- MUL R2, 100
- MOV R1, sym2
- ADD R1, R2
- MOV sym1, R1

符号	地址
sym1	0
sym2	4
sym3	8
…	…

不止是编译

从源程序的编写到最终的执行，除了编译之外，还需要结合以下几个过程，共同完成一个完整的程序处理任务。

- 预处理（Preprocess）
- 汇编（Assemble）
- 连接（Link）
- 装入（Load）
- 运行（Run）

第1步：预处理

在编译之前进行，执行文件包含、宏替换等操作，形成一个源文件。

- ...
- int aaa, bbb, ccc;
- ...
- aaa = bbb + ccc * 100;
- ...

第2步：编译

将源程序翻译成目标程序（汇编程序）。

- MOV R2, sym3
- MUL R2, 100
- MOV R1, sym2
- ADD R1, R2
- MOV sym1, R1

符号	地址
sym1	0
sym2	4
sym3	8
…	…

第3步：汇编

将汇编程序翻译成机器程序。

机器程序由机器指令构成，每条机器指令包括以下4个部分内容：

1) 操作码：读内存0001 、写内存0010 、加运算0011 、乘运算0100

2) 操作数1：寄存器编号

3) 操作数2的寻址模式：立即数寻址00、寄存器寻址01、直接地址寻址10

4) 操作数2：与寻址模式对应的数据

汇编的过程

…

MOV R2, sym3 → 0001 10 10 00001000

MUL R2, 100 → 0100 10 00 01100100

MOV R1, sym2 → 0001 01 10 00000100

ADD R1, R2 → 0011 01 01 00000010

MOV sym1, R1 → 0010 01 10 00000000

…

生成的机器程序

…
0001 10 10 00001000 *
0100 10 00 01100100
0001 01 10 00000100 *
0011 01 01 00000010
0010 01 10 00000000 *
…

其中，“*”为重定位标志。

第4步：连接

将若干个机器程序、库程序合并（连接）起来，解决相互之间的数据共享、交叉引用等问题，形成一个完整的机器程序。

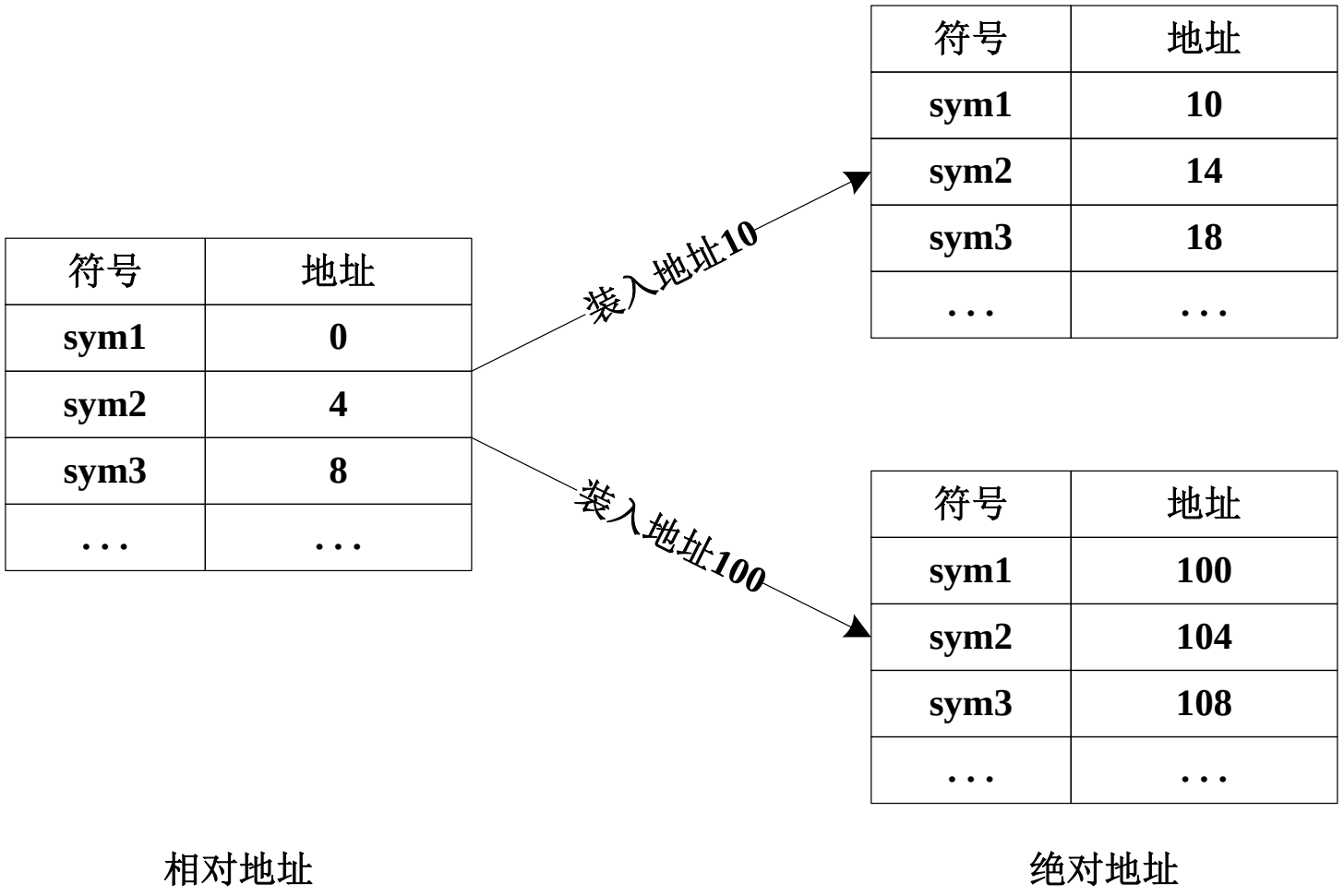
说明：

连接分为静态连接和动态连接；动态连接分为装入时动态链接和运行时动态连接。

第5步：装入

根据当前的内存使用情况确定机器程序即将装入内存的绝对地址，从硬盘（或其它外部存储器）中读入机器程序，更新其中的重定位信息，然后装入指定内存。

重定位实例


符号
地址
sym1
0
sym2
4
sym3
8
...
...
装入地址10
装入地址100
符号
地址
sym1
10
sym2
14
sym3
18
...
...
符号
地址
sym1
100
sym2
104
sym3
108
...
...
相对地址
绝对地址

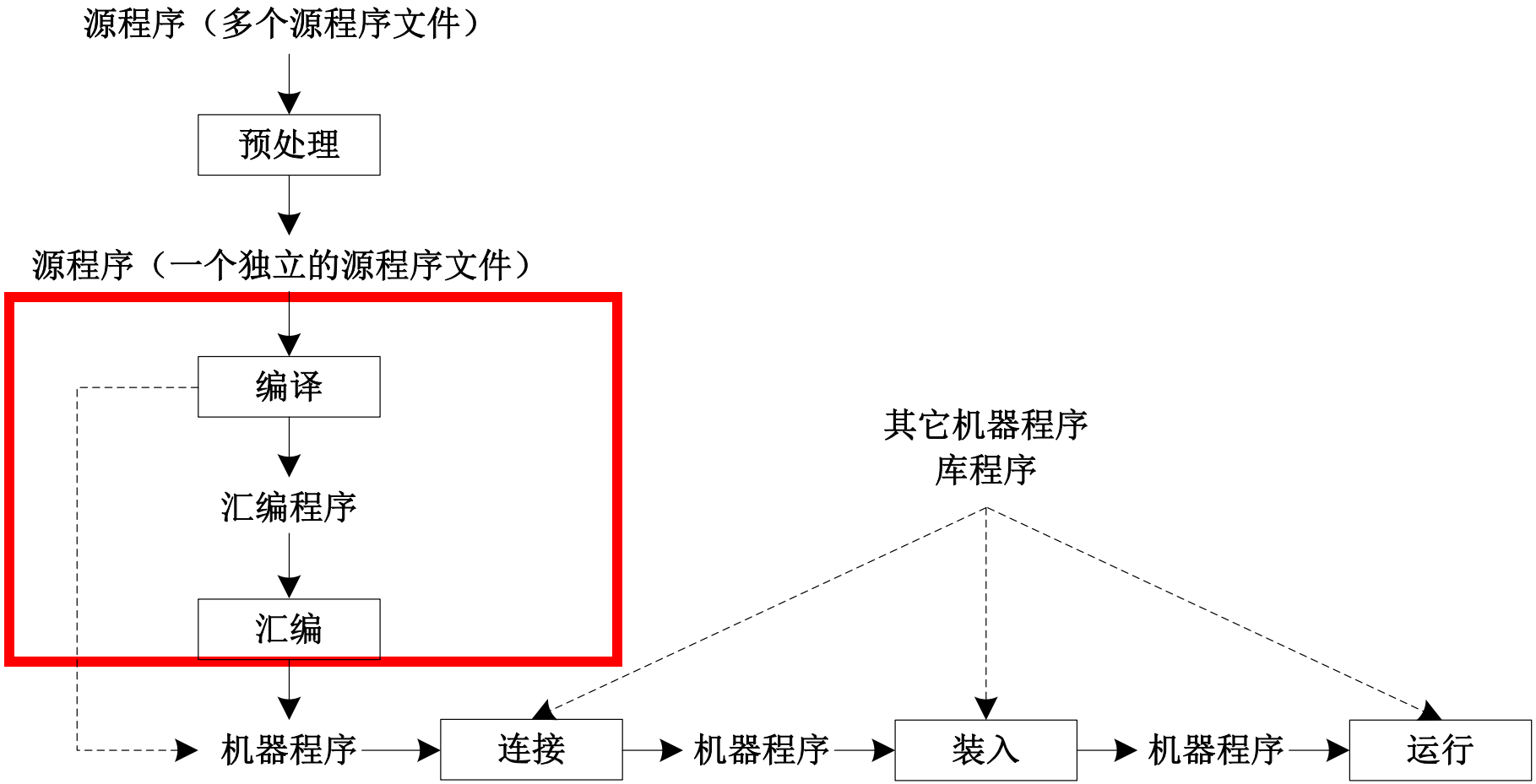
假设装入内存的绝对地址为128，则最终装入内存的机器程序为：

- …
- 0001 10 10 10001000
- 0100 10 00 01100100
- 0001 01 10 10000100
- 0011 01 01 00000010
- 0010 01 10 10000000
- …

第6步：运行

运行时存储空间（代码空间、数据空间）的组织，参数传递，进程切换……

完整的程序处理过程


源程序（多个源程序文件）
预处理
源程序（一个独立的源程序文件）
编译
汇编程序
汇编
其它机器程序
库程序
机器程序
连接
机器程序
装入
机器程序
运行