第1章

引论

Prehistory(40年代) — 机器语言

(Grace Hopper)

(Punch card)

Intel机器码写的阶乘计算程序

```
10111000    00000001 00000000 00000000 00000000
10111010    00000010 00000000 00000000 00000000
00111001    11011010
01111111    00000110
00001111    10101111 11000010
01000010
11101011    11110110
11000011
```

远古(1950) — 汇编语言及汇编器

用文本表示机器语言

机器指令用助记符表示；

内存地址和指令地址用标识符表示；

允许有注释。

汇编器完成汇编语言到机器语言的翻译。

Intel汇编语言的阶乘计算程序

```
;; 输入参数 N 放入寄存器EBX中
;; 计算结果放入寄存器EAX中
Factorial:
    mov eax, 1;; 初始化输出result = 1
    mov edx, 2;; 初始化循环参数index = 2
L1:cmp edx, ebx;; 如果 index <= N ...
    jg L2
            ;; result乘上indeximul eax, edx
    inc edx;; index递增1
    jmp L1;; 转移到循环始点

L2:ret                  ;; 返回
```

高级语言:

命令式语言： Fortran, Algol, Pascal, Basic, C, Modula, Ada,…

面向对象语言： Simula, Smalltalk, C++, Java, C#,…

说明式语言:

函数式语言： Lisp, Scheme, ML, Haskell,…

逻辑程序设计语言： Prolog.

脚本语言： Shell, Awk, Perl, Tcl, Python, Javascript, PHP, …

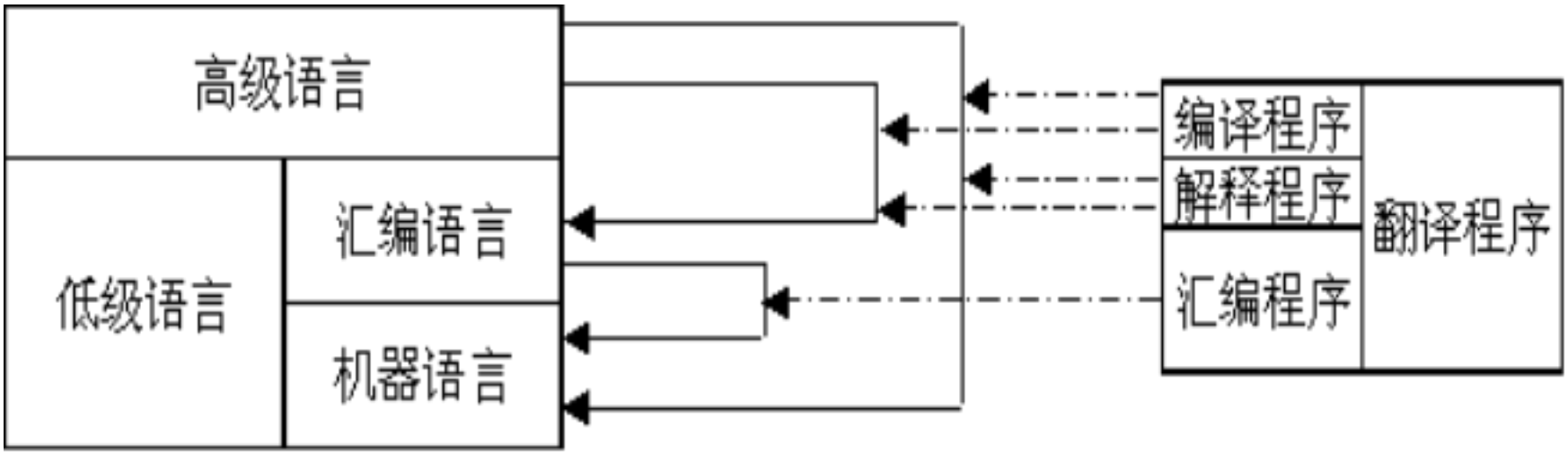
A文字处理语言： TEX, LTEX

超文本语言： HTML, XML;

数据库语言： SQL;

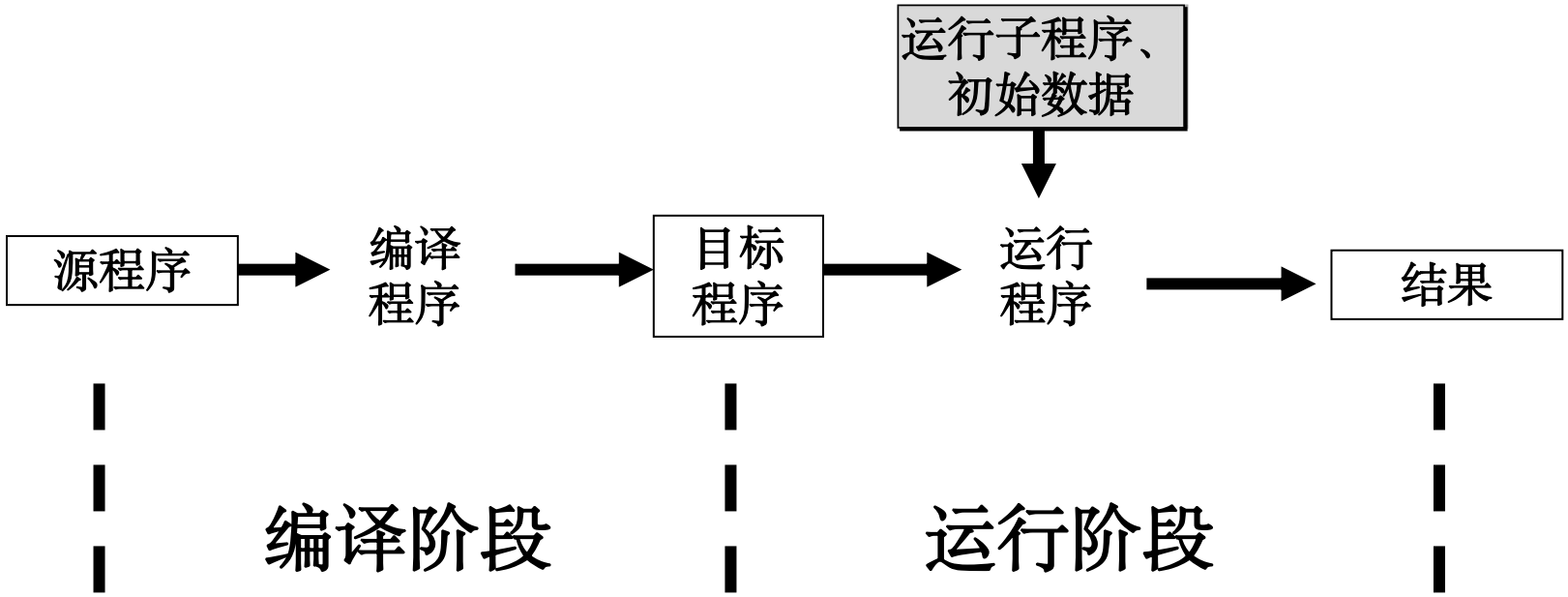
数据表处理语言： VisiCalc, Lotus1-2-3, Excel,…

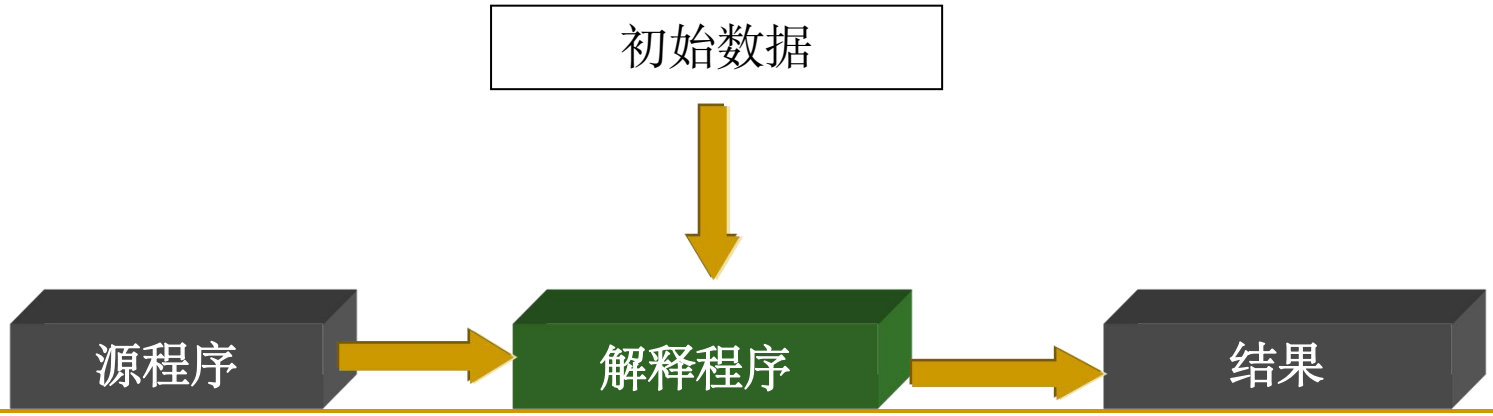
1.1 什么是编译程序?



- 一个编译程序就是一个语言翻译程序。它把一种高级语言（称作源语言）书写的程序翻译成另一种低级语言（称作目标语言）的等价的程序。

高级语言程序的处理方法（编译途径和解释途径）

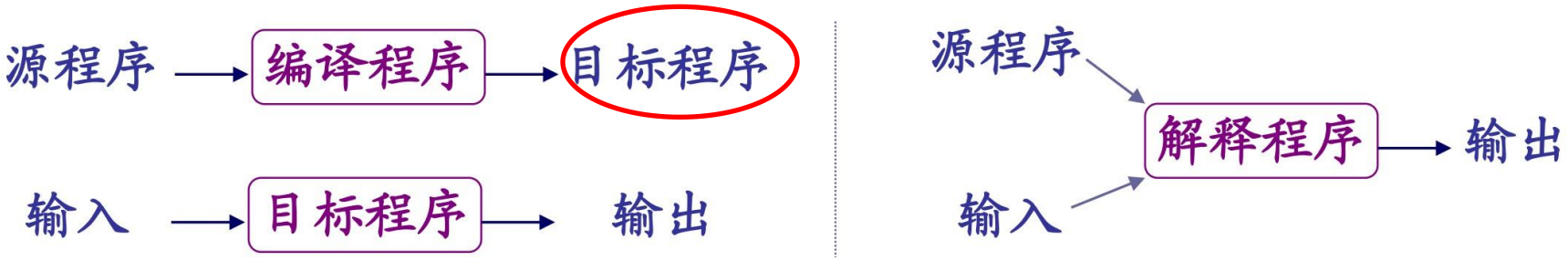


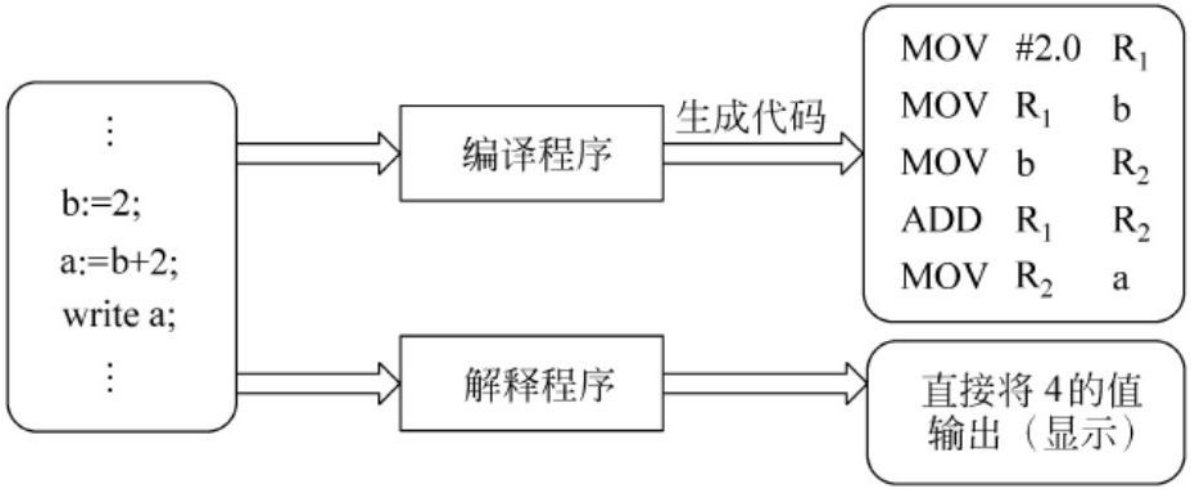


解释程序

- 解释程序不产生目标程序文件
- 解释不区别翻译阶段和执行阶段
- 解释源程序的每条语句后直接执行
- 程序执行期间一直有解释程序守候
- 常用于实常用于实现虚拟机

比较编译程序和解释程序（之一）





比较编译程序和解释程序（之二）

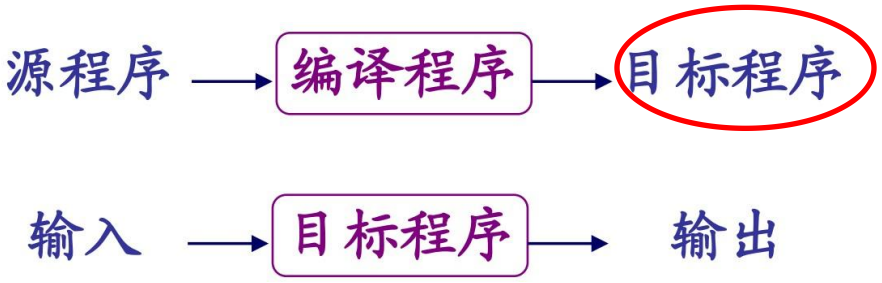
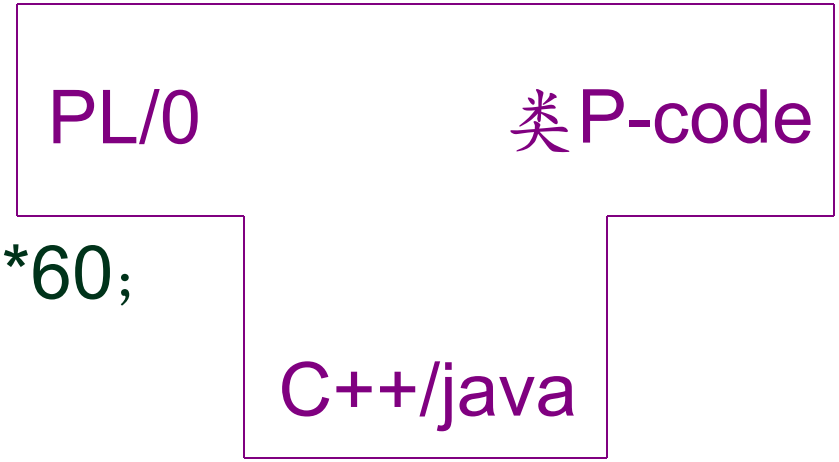


图 1.13　编译程序的编译阶段和运行阶段的存储区内容

图 1.14　解释程序的存储区内容

PL/0编译程序总体结构

T-型图



$id_1=id_2+id_3*60;$

```
mov   id3,R2
mul   #60.0,R2
mov   id2,R1
add   R2,R1
mov   R1,id1
```

PL/0： 编译程序所实现的源语言

类P-code： 目标语言

C++/java： 编译程序的实现语言

PL/0 程序示例

```
CONST A=10;                    /*主程序常量说明部分*/
VAR   B,C;                     /*主程序变量说明部分*/
PROCEDURE  P;                  /*主程序过程说明部分*/
    VAR D;                     /*过程P的局部变量说明部分*/
    PROCEDURE  Q;              /*过程P的局部过程说明部分*/
        VAR X;                 /*过程Q的局部变量说明部分*/
        BEGIN
            READ(X);
            D:=X;
            WHILE X==0
            DO CALL P;
        END;
    BEGIN
        WRITE(D);
        CALL Q;
    END;
BEGIN
    CALL P;
END.
```

Q 的过程体

P 的过程体

主程序的过程体

计算最大公约数

```
var m, n, r, q;
procedure gcd;
    begin
        while r#0  do
            begin
                q := m / n;
                 r := m - q * n;
                m := n;
                n := r;
            end
    end;
begin
        read(m);
        read(n);
        r:=1;
        call gcd;
        write(m);
end.
```

程序设计语言的组成

字符　　Z,A.....

单词　　IF，THEN，ELSE....

语句　　IF A>Z THEN A--

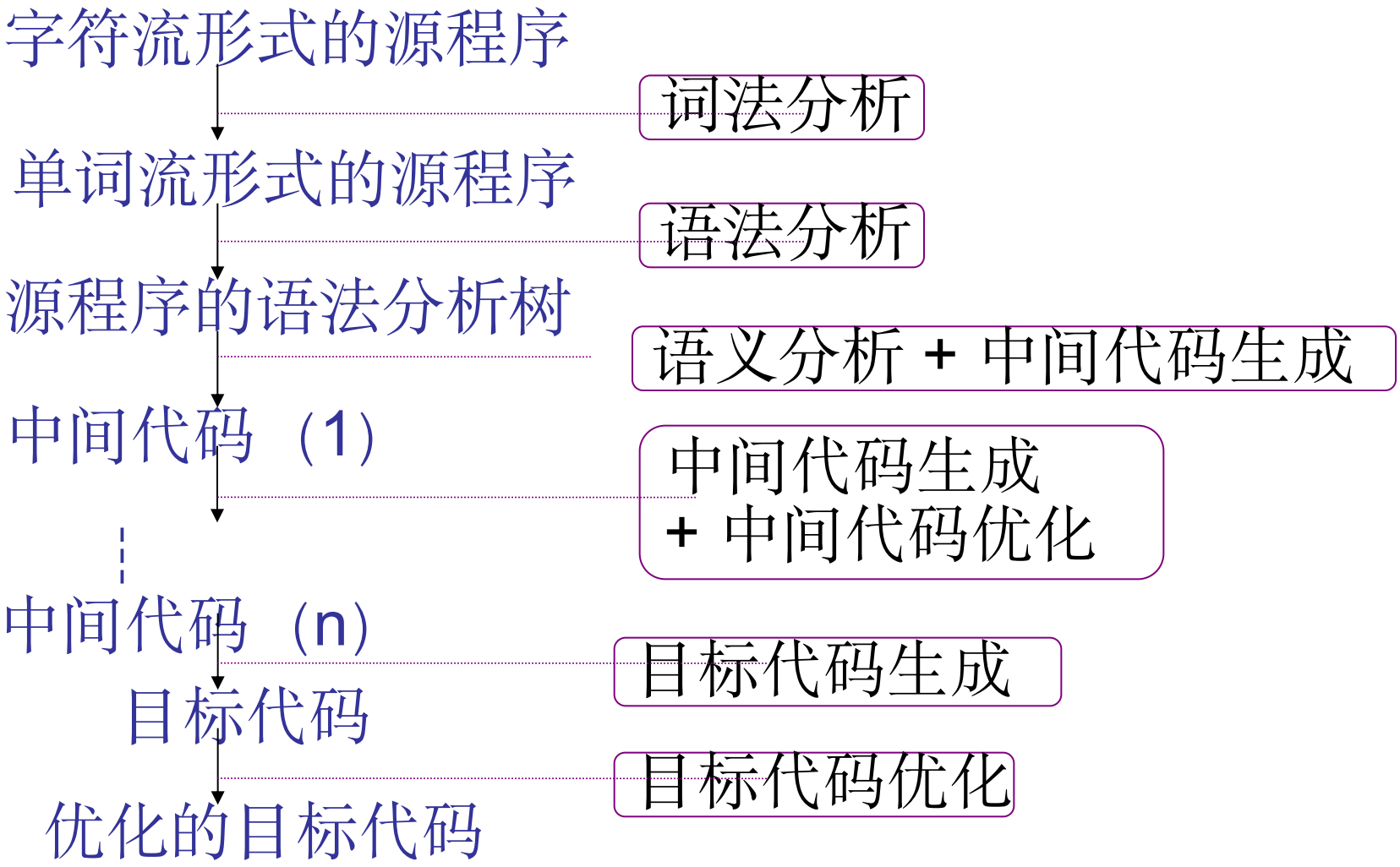
语义

```
char a = 65, b = 2;
int c;
c = a + b;
```

1.2 编译过程概述

- 所谓编译过程是指将高级语言程序翻译为等价的目标程序的过程。

英译中		编译过程	
We study English.			
↓	识别单词	词法分析	*分析和识别单词*
<名词,We> <动词,study> <名词,English> <.,句号>		↓	
↓	确定句子句型	语法分析	*识别出各种语法成分，并进行语法正确性检查*
SVO		↓	
↓	确定句子意义	语义分析	*对识别出的各种语法成分进行语义分析*
		↓	
S：我们 V：学习 O：英语		中间代码生成	*产生相应的中间代码*
		↓	
		代码优化	*目的是为了得到高质量的目标程序*
		↓	
↓	翻译成中文	目标代码生成	*由中间代码生成目标程序*
我们学习英语。			


字符流形式的源程序
词法分析
单词流形式的源程序
语法分析
源程序的语法分析树
语义分析 + 中间代码生成
中间代码（1）
中间代码生成
+ 中间代码优化
中间代码（n）
目标代码生成
目标代码
目标代码优化
优化的目标代码

举例

```
PROGRAM    m;
    VAR  sum,first,count:real;
BEGIN
     sum:=first+count*10
END.
```

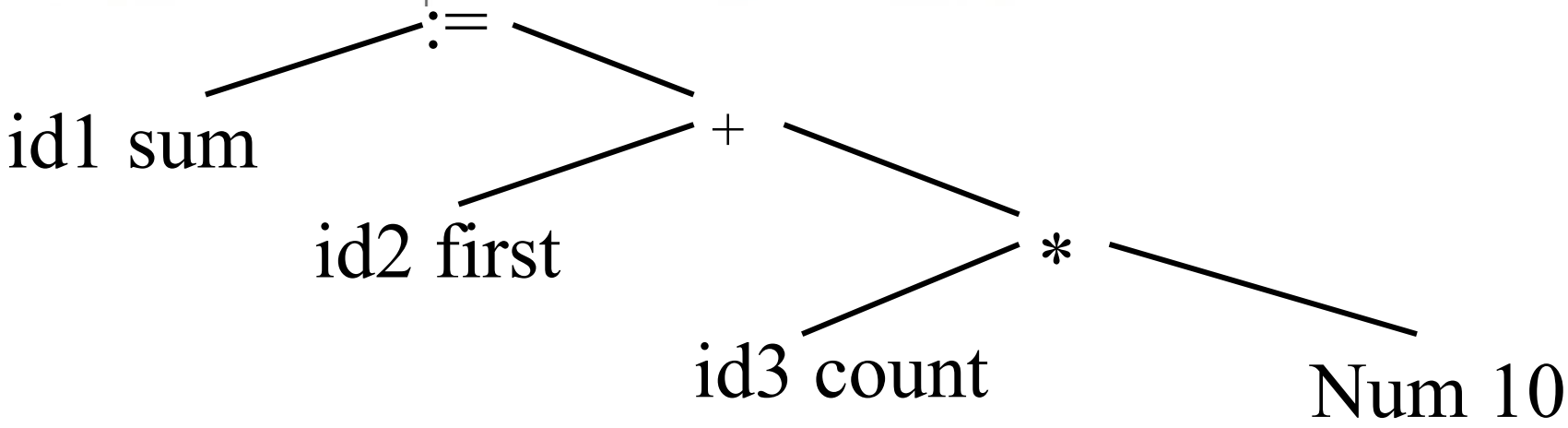
词法分析

- 从左到右一个字符一个字符地读入源程序，对构成源程序的字符流进行扫描和分解，从而识别出一个个单词（也称单词符号或符号）。

<保留字 PROGRAM>
<标识符 m>
<分隔符 ;>
<保留字 VAR>
<标识符 sum>
<分隔符 , >
<标识符 first>
<分隔符 , >
<标识符 count>
<分隔符 : >
<保留字 real>
<分隔符 ;>
<保留字, BEGIN>
<标识符 sum>
<赋值号 := >
<标识符 first>
<加号 +>
<标识符 count>
<乘号 *>
<整数 10>
<保留字 END>
<分隔符 . >

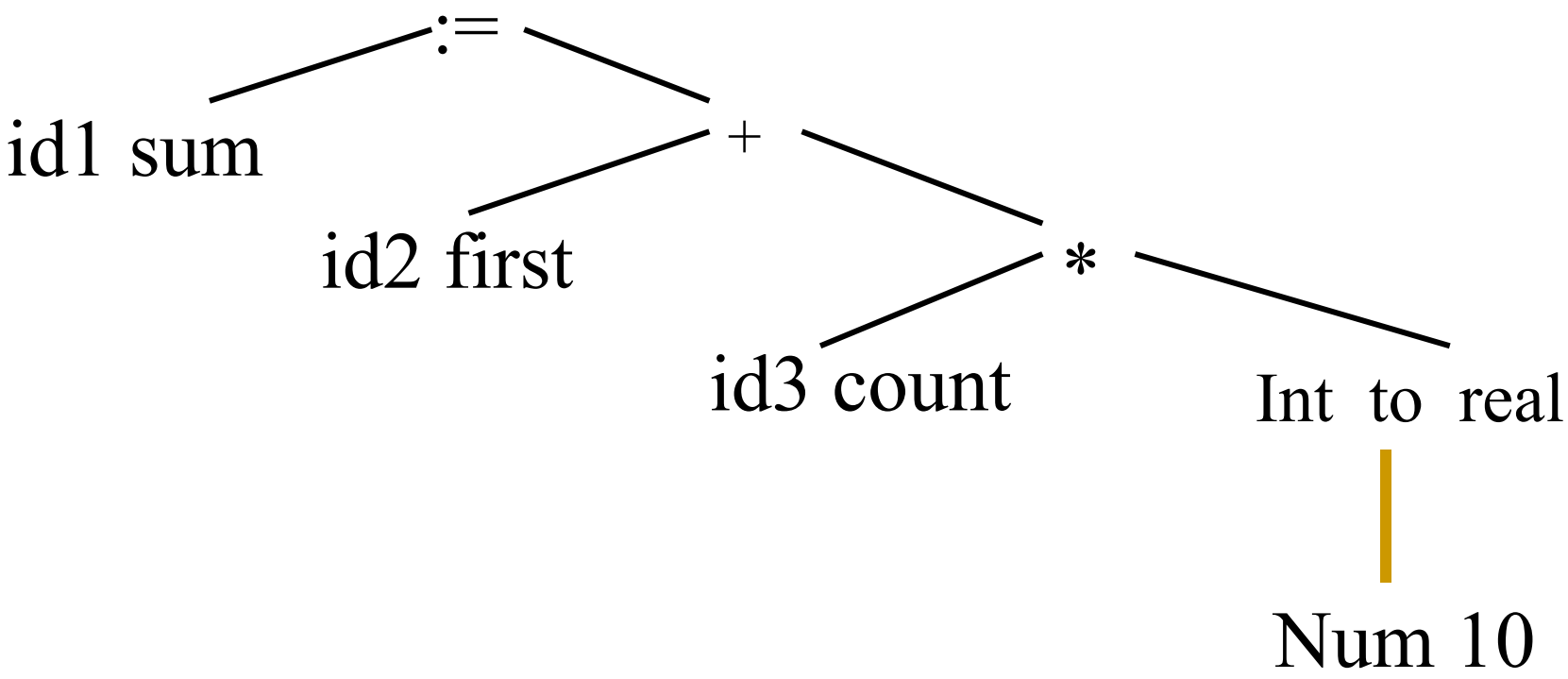
语法分析

<赋值语句>	::=<id>:=<表达式>
<复合语句>	::=**begin**<语句>{;<语句>} **end**
<空语句>	::=ε
<条件>	::=<表达式><关系运算符><表达式>|**odd** <表达式>
<表达式>	::=[+|-] <项>{<加减运算符><项>}
<项>	::=<因子>{<乘除运算符><因子>}
<因子>	::=<id>|<integer>|'('<表达式>')'



sum:=first+count*10

- 语义分析
  - 审查源程序有无语义错误，为代码生成阶段收集类型信息。



- 上下文相关性
- 类型匹配和类型转换
- 数组下标

中间代码生成

- “中间代码”是一种结构简单、含义明确的记号系统。很多编译程序采用了一种近似“三地址指令”的“四元式”。
- 四元式的形式为

  (运算符，运算对象1，运算对象2，结果)

id1:=id2+id3*10

(1)	(inttoreal	10	—	t_1)
(2)	(*	id3	t_1	t_2)
(3)	(+	id2	t_2	t_3)
(4)	(:=	t_3	—	id1)

图 1.7　中间代码

- 代码优化
  - 对产生的中间代码进行变换或进行改造，目的是使生成的目标代码更为高效。

id1:=id2+id3*10

```
(*    id3    10.0    t1)
(+    id2    t1      id1)
```

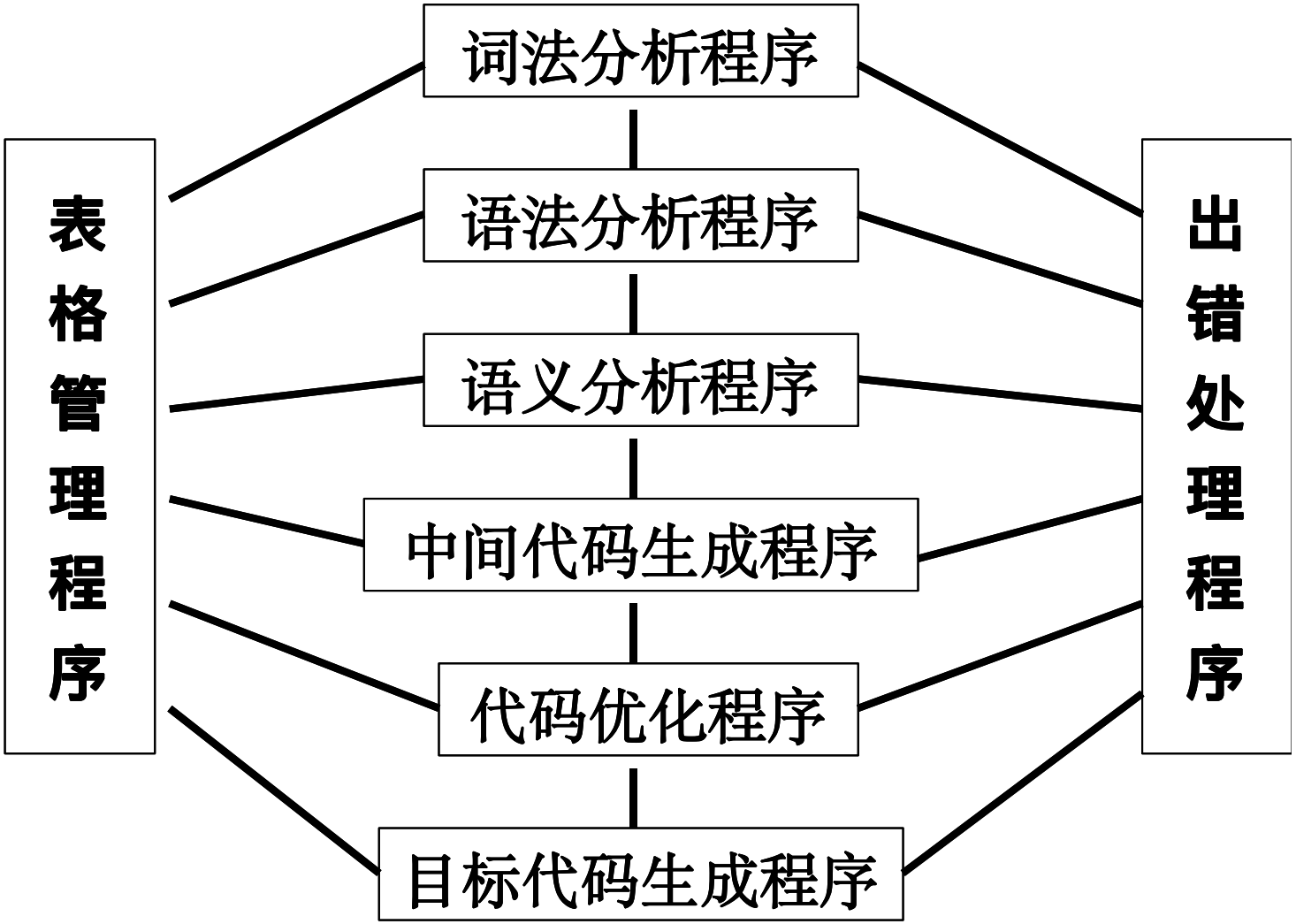
图 1.8　优化后的中间代码

目标代码生成

- 把中间代码变换成特定机器上的绝对指令代码或可重定位的指令代码或汇编代码。

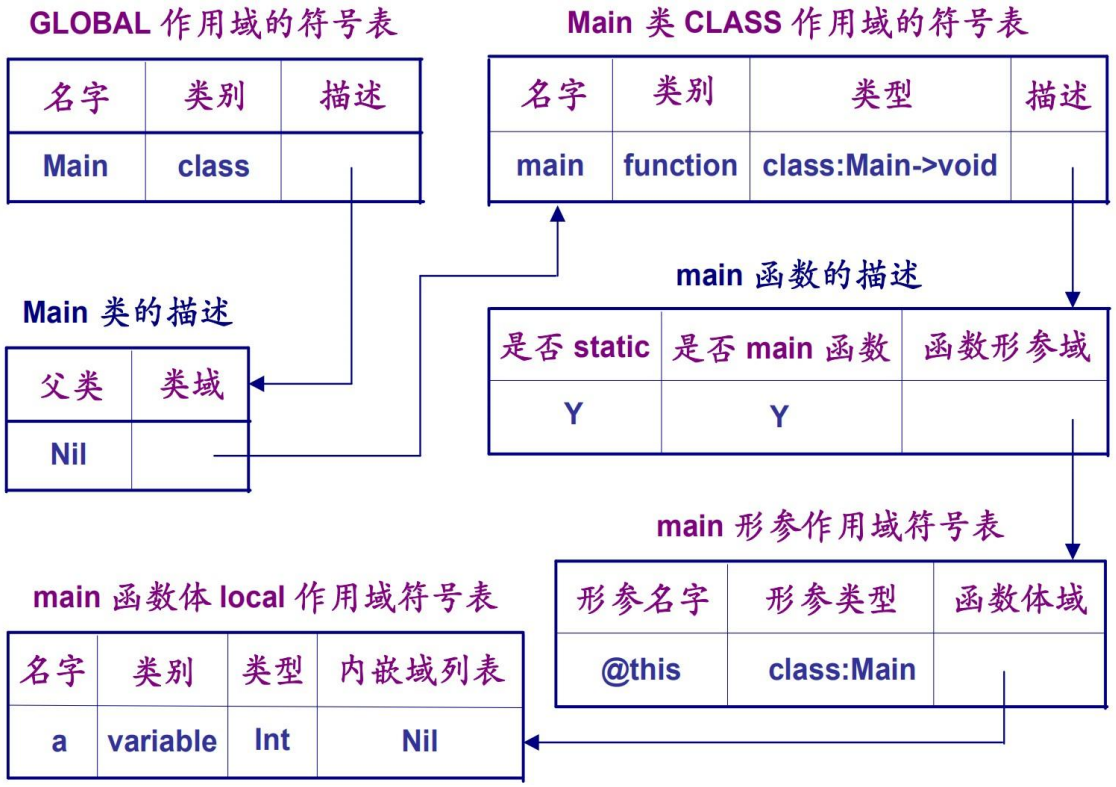
```
1.  mov     id3,     R2
2.  mul     #60.0,   R2
3.  mov     id2,     R1
4.  add     R2,      R1
5.  mov     R1,      id1
```

1.2.2 编译程序的结构

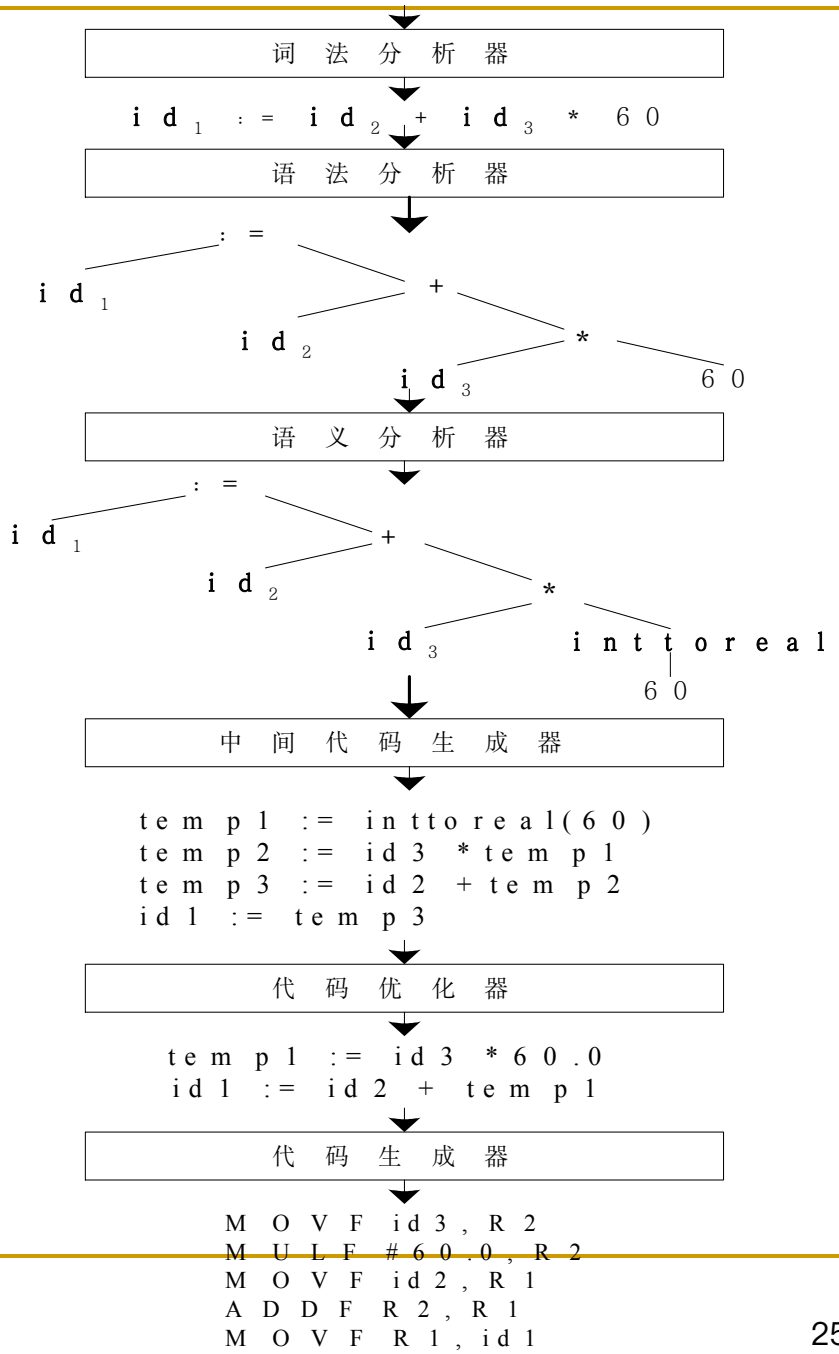

词法分析程序
语法分析程序
语义分析程序
中间代码生成程序
代码优化程序
目标代码生成程序
表格管理程序
出错处理程序

符号表

```
class Main {
 static void main()
 {
   int a;
   a="hello world";
   print(a);
  }
}
```



position := initial + rate * 60

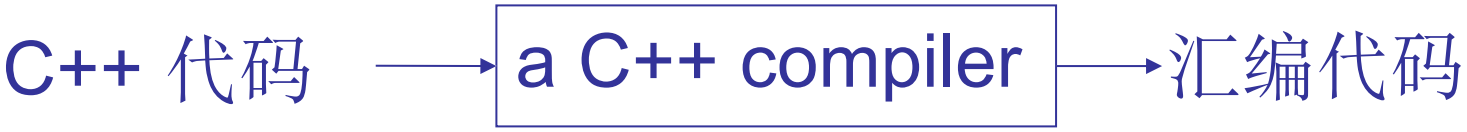


符 号 表

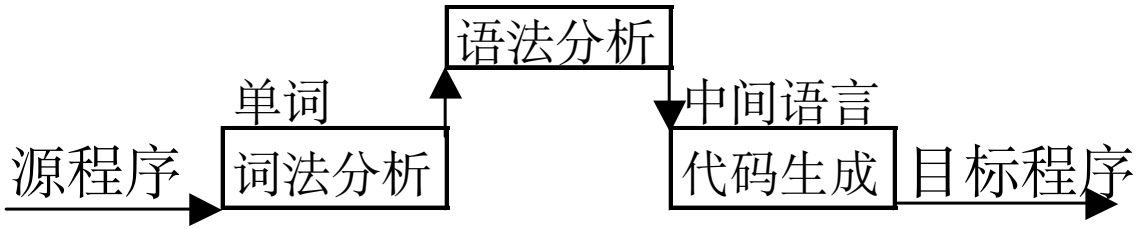
1	position	...
2	initial	...
3	rate	...
4		

1.2.3 编译阶段的组合

- 根据编译程序各部分功能，将编译程序分成前端和后端。通常将与源程序有关的编译部分（词法分析、语法分析、语义分析、中间代码生成、代码优化）称为前端；与目标机有关的部分（目标代码生成）称为后端。



- 编译程序按其完成规定任务的过程中对源程序或其等价的中间语言程序从头到尾扫视的次数分为一遍扫描和多遍扫描。



各部分相对独立性差
占用内存空间大
速度快

各部分相对独立性好
占用内存空间小
速度慢

表 1.1　PL/0 语言语法的 EBNF 描述

PL/0 语法单位	EBNF 描述
<程序>	::=<分程序>.
<分程序>	::=[<常量说明部分>][<变量说明部分>][<过程说明部分>]<语句>
<常量说明部分>	::=**const**<常量定义>{,<常量定义>};
<常量定义>	::=<id>=<integer>
<变量说明部分>	::=**var**<id>{,<id>};
<过程说明部分>	::=<过程首部><分程序>{;<过程说明部分>};
<过程首部>	::=**procedure** <id>;
<语句>	::=<赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语句>|<写语句>|<复合语句>|<空语句>
<赋值语句>	::=<id>:=<表达式>
<复合语句>	::=**begin**<语句>{;<语句>} **end**
<空语句>	::=ε
<条件>	::=<表达式><关系运算符><表达式>|**odd** <表达式>
<表达式>	::=[+|−]<项>{<加减运算符><项>}
<项>	::=<因子>{<乘除运算符><因子>}
<因子>	::=<id>|<integer>|'('<表达式>')'
<加减运算符>	::=+|−
<乘除运算符>	::=*|/
<关系运算符>	::==|#|<|<=|>|>=
<条件语句>	::=**if**<条件>**then** <语句>

e