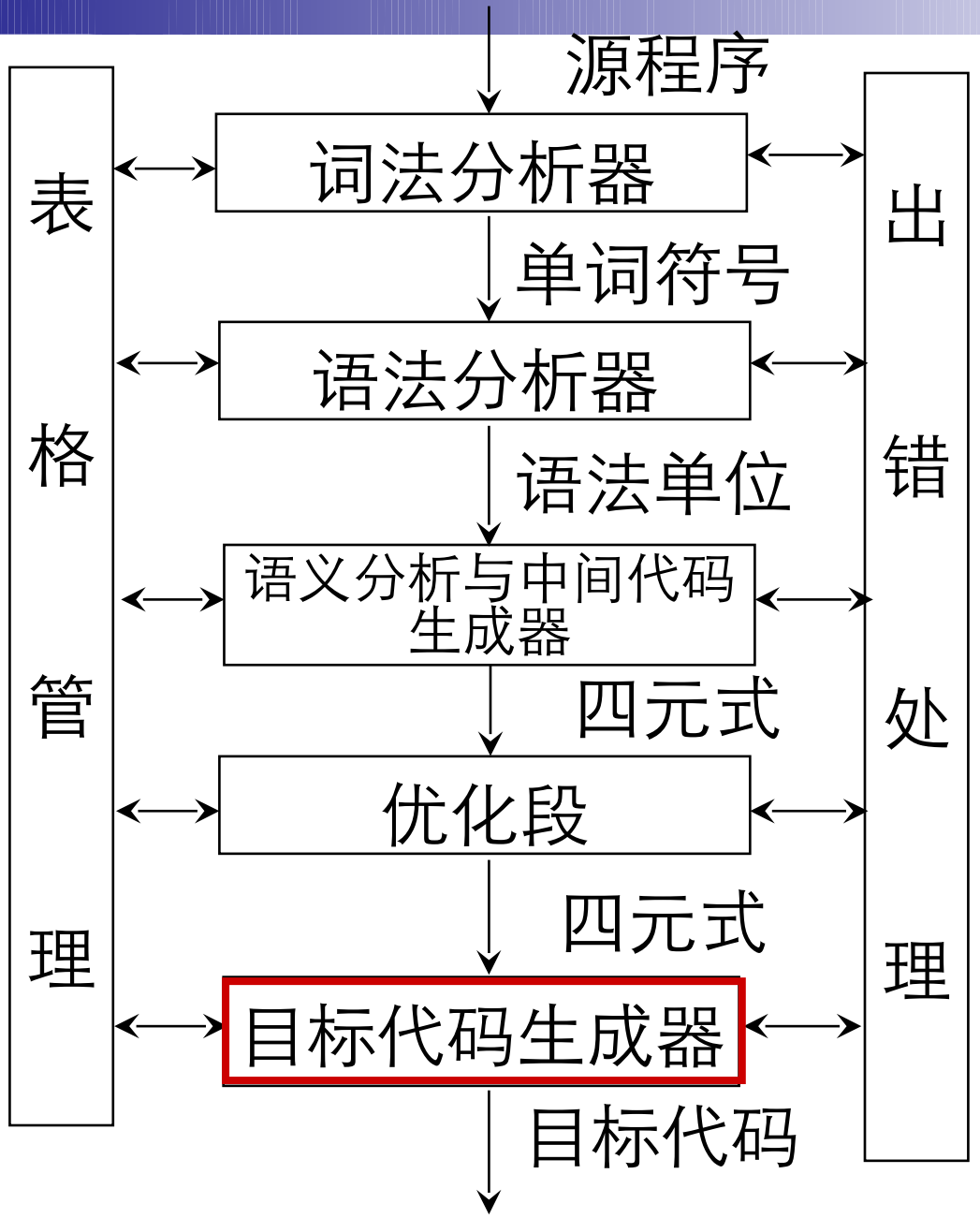
编译原理

第十一章 代码生成

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

第十一章 代码生成

- 基本问题
- 目标机器模型
- 一个简单代码生成器

第十一章 代码生成

- 基本问题
- 目标机器模型
- 一个简单代码生成器

代码生成

- 代码生成
  - 把语法分析后或优化后的中间代码变换成目标代码
- 目标代码的三种形式
  - 绝对指令代码：能够立即执行的机器语言代码，所有地址已经定位
  - 可重新定位指令代码：待装配的机器语言模块，执行时，由连接装配程序把它们和某些运行程序连接起来，转换成能执行的机器语言代码
  - 汇编指令代码：尚须经过汇编程序汇编，转换成可执行的机器语言代码

代码生成

- 代码生成着重考虑的问题
  - 如何使生成的目标代码较短
  - 如何充分利用计算机的寄存器，减少目标代码中访问存贮单元的次数
  - 如何充分利用计算机的指令系统的特点

11.1 基本问题

- 代码生成器的输入
  - 代码生成器的输入包括源程序的中间表示，以及符号表中的信息
  - 类型检查

- x:=y + i*j

  其中 x 、 y 为实型； i 、 j 为整型，产生的三地址代码为：

  T_1:=i int* j

  T_3:=inttoreal T_1

  T_2:=y real+ T_3

11.1 基本问题

- 目标程序
  - 绝对机器代码、可重定位机器语言、汇编语言
  - 采用汇编代码作为目标语言
- 指令选择
  - a:=a+1
    - INC a
    - LD R0, a
      ADD R0, #1
      ST R0, a

11.1 基本问题

- 寄存器分配
  - 在寄存器分配期间，为程序的某一点选择驻留在寄存器中的一组变量
  - 在随后的寄存器指派阶段，挑出变量将要驻留的具体寄存器
- 计算顺序选择

第十一章 代码生成

- 基本问题
- 目标机器模型
- 一个简单代码生成器

11.2 目标机器模型

- 考虑一个抽象的计算机模型
  - 具有多个通用寄存器，他们既可以作为累加器，也可以作为变址器
  - 运算必须在某个寄存器中进行
  - 含有四种类型的指令形式

类型	指令形式	意义（设 op 是二目运算符）
直接地址型	op R_i, M	(R_i) op (M) $\Rightarrow R_i$
寄存器型	op R_i, R_j	(R_i) op $(R_j) \Rightarrow R_i$
变址型	op R_i, c(R_j)	(R_i) op $((R_j)+c) \Rightarrow R_i$
间接型	op R_i, *M	(R_i) op $((M)) \Rightarrow R_i$
	op R_i, *R_j	(R_i) op $((R_j)) \Rightarrow R_i$
	op R_i, *c(R_j)	(R_i) op $((R_j)+c) \Rightarrow R_i$

op 包括一般计算机上常见的一些运算符，如

ADD（加）、SUB（减）、MUL（乘）、DIV（除）

如果 op 是一目运行符，则“op R_i, M” 的意义为：op （M）$\Rightarrow$ R_i，其余类型可类推。

指 令	意 义
LD R_i, B	把 B 单元的内容取到寄存器 R ， 即 (B)⇒ R_i
ST R_i, B	把寄存器 R_i 的内容存到 B 单元，即 (R_i)⇒ B
J X	无条件转向 X 单元
CMP A, B	比较 A 单元和 B 单元的值，根据比较情况把机器内部特征寄存器 CT 置成相应状态。 CT 占两个二进位。根据 A<B 或 A=B 或 A>B 分别置 CT 为 0 或 1 或 2 。
J < X	如 CT=0 转 X 单元
J≤ X	如 CT=0 或 CT=1 转 X 单元
J = X	如 CT=1 转 X 单元
J≠ X	如 CT≠1 转 X 单元
J > X	如 CT=2 转 X 单元
J≥ X	如 CT=2 或 CT=1 转 X 单元

第十一章 代码生成

- 基本问题
- 目标机器模型
- 一个简单代码生成器

11.3 一个简单代码生成器

- 不考虑代码的执行效率，目标代码生成是不难的，例如：

$$A:=(B+C)*D+E$$

翻译为四元式：

$T_1:=B+C$

$T_2:=T_1*D$

$T_3:=T_2+E$

$A:=T_3$

假设只有一个寄存器可供使用

- 四元式

T_1:=B+C

T_2:=T_1*D

T_3:=T_2+E

A:=T_3

- 目标代码：

```
LD    R0 , B
ADD   R0 , C
ST    R0 , T1
LD    R0 , T1
MUL   R0 , D
ST    R0 , T2
LD    R0 , T2
ADD   R0 , E
ST    R0 , T3
LD    R0 , T3
ST    R0 , A
```

- 假设 T1 , T2 , T3 在基本块之后不再引用

```
LD    R0 , B
ADD   R0 , C
MUL   R0 , D
ADD   R0 , E
ST    R0 , A
```

11.3 一个简单代码生成器

- 四元式的中间代码变换成目标代码
- 在一个基本块的范围内考虑如何充分利用寄存器
  - 尽可能留：在生成计算某变量值的目标代码时，尽可能让该变量保留在寄存器中
  - 尽可能用：后续的目标代码尽可能引用变量在寄存器中的值，而不访问内存
  - 及时腾空：在离开基本块时，把存在寄存器中的现行的值放到主存中

11.3.1 待用信息

- 如果在一个基本块内，四元式 i 对 A 定值，四元式 j 要引用 A 值，而从 i 到 j 之间没有 A 的其他定值，那么，我们称 j 是四元式 i 的变量 A 的待用信息，即下一个引用点

i: A:=B op C

…

j: D:=A op E

- 假设在变量的符号表登记项中含有记录待用信息和活跃信息的栏。

待用信息和活跃信息的表示

- (x, x) 表示变量的待用信息和活跃信息
  - 第 1 元
    - i 表示待用信息， ^ 表示非待用
  - 第 2 元
    - y 表示
- 在符号表
  - 表示后

变量名	初始状态→信息链（待用 / 活跃信息栏）
T	(^,y)→(3,y)→(^,^)
A	(^,^)→(2,y)→(1,y)
B	(^,^)→(1,y)
C	(^,^)→(2,y)
U	(^,^)→(4,y)→(3,y)→(^,^)
V	(^,^)→(4,y)→(^,^)

待用信息和活跃信息的表示

- (x, x) 表示变量的待用信息和活跃信息
  - 第 1 元
    - i 表示待用信息， ^ 表示非待用
  - 第 2 元
    - y 表示活跃， ^ 表示非活跃
- 在符号表中， (x ， x)→(x ， x)
  - 表示后面的符号对代替前面的符号对
- 不特别说明，所有说明变量在基本块出口之后均为非活跃变量

- 例：基本块
  1. T:=A-B
  2. U:=A-C
  3. V:=T+U
  4. W:=V+U
- 设 W 是基本块出口之后的活跃变量。

序号	四元式	左值	左操作数	右操作数
(1)	T:=A-B	(3,y)	(2,y)	(^,^)
(2)	U:=A-C	(3,y)	(^,^)	(^,^)
(3)	V:=T+U	(4,y)	(^,^)	(4,y)
(4)	W:=V+U	(^,y)	(^,^)	(^,^)

计算待用信息和活跃信息

- 计算待用信息和活跃信息的算法步骤

1. 开始时，把基本块中各变量的符号表登记项中的待用信息栏填为“非待用”，并根据该变量在基本块出口之后是不是活跃的，把其中的活跃信息栏填为“活跃”或“非活跃”；

变量名	待用 / 活跃信息栏
T	(^,^)→(3,y)
A	(^,^)→(2,y)
B	(^,^)→(1,y)

序号	四元式	左值	左操作数	右操作数
(1)	T:=A-B	(3,y)	(2,y)	(^,^)
(2	U:=A-C	(3,y)	(^,^)	(^,^)
(3)	V:=T+U	(4,y)	(^,^)	(4,y)
(4)	W:=V+U	(^,y)	(^,^)	(^,^)

- 2. 从基本块出口到入口由后向前依次处理各个四元式。对每一个四元式 i:A:=B op C，依次执行:

1) 把符号表中变量 A 的待用信息和活跃信息附加到四元式 i 上

2) 把符号表中 A 的待用信息和活跃信息分别置为“非待用”和“非活跃”

3) 把符号表中变量 B 和 C 的待用信息和活跃信息附加到四元式 i 上

4) 把符号表中 B 和 C 的待用信息均置为 i，活跃信息均置为“活跃”

例：基本块

1. T:=A-B
2. U:=A-C
3. V:=T+U
4. W:=V+U

设 W 是基本块出口之后的活跃变量。

建立待用信息链表与活跃变量信息链表如下：

附加在四元式上的待用 / 活跃信息表

序号	四元式	左值	左操作数	右操作数
(4)	W:=V+U	(^,y)	(^,^)	(^,^)
(3)	V:=T+U	(4,y)	(^,^)	(4,y)
(2)	U:=A-C	(3,y)	(^,^)	(^,^)
(1)	T:=A-B	(3,y)	(2,y)	(^,^)

变量名栏	初始状态→信息链 (待用 / 活跃信息
T	(^,^) → (3,y) → (^,^)
A	(^,^) → (2,y) → (1,y)
B	(^,^) → (1,y)
C	(^,^) → (2,y)
U	(^,^) → (4,y) → (3,y) → (^,^)
V	(^,^) → (4,y) → (^,^)
W	(^,y) → (^,^)

附加在四元式上的待用 / 活跃信息表

序号	四元式	左值	左操作数	右操作数
(4)	W:=V+U	(^,y)	(^,^)	(^,^)
(3)	V:=T+U	(4,y)	(^,^)	(4,y)
(2)	U:=A-C	(3,y)	(^,^)	(^,^)
(1)	T:=A-B	(3,y)	(2,y)	(^,^)

序号	四元式	左值	左操作数	右操作数
(1)	T:=A-B	(3,y)	(2,y)	(^,^)
(2)	U:=A-C	(3,y)	(^,^)	(^,^)
(3)	V:=T+U	(4,y)	(^,^)	(4,y)
(4)	W:=V+U	(^,y)	(^,^)	(^,^)

V	(^,^) → (4,y) → (^,^)
W	(^,y) → (^,^)

寄存器描述和变量地址描述

- 寄存器描述数组 RVALUE
  - 动态记录各寄存器的使用信息
  - RVALUE[R]={A,B}
- 变量地址描述数组 AVALUE
  - 动态记录各变量现行值的存放位置
  - AVALUE[A]={R1, R2, A}

寄存器描述和变量地址描述

- 寄存器的分配局限于基本块范围之内
  - 处理完基本块中所有四元式，对现行值在寄存器中的每个变量，如它在基本块之后是活跃的，则把它在寄存器中的值存放到它的主存单元中
- 对于四元式 A:=B
  - 如果 B 的现行值在某寄存器 R_i 中，则无须生成目标代码
  - 只须在 RVALUE(R_i) 中增加一个 A，（即把 R_i 同时分配给 B 和 A），并把 AVALUE(A) 改为 R_i

小结

- 目标代码形式
- 代码生成着重考虑的问题
- 一个简单代码生成器
  - 待用信息
  - 活跃信息
  - 寄存器描述信息
  - 变量地址描述信息