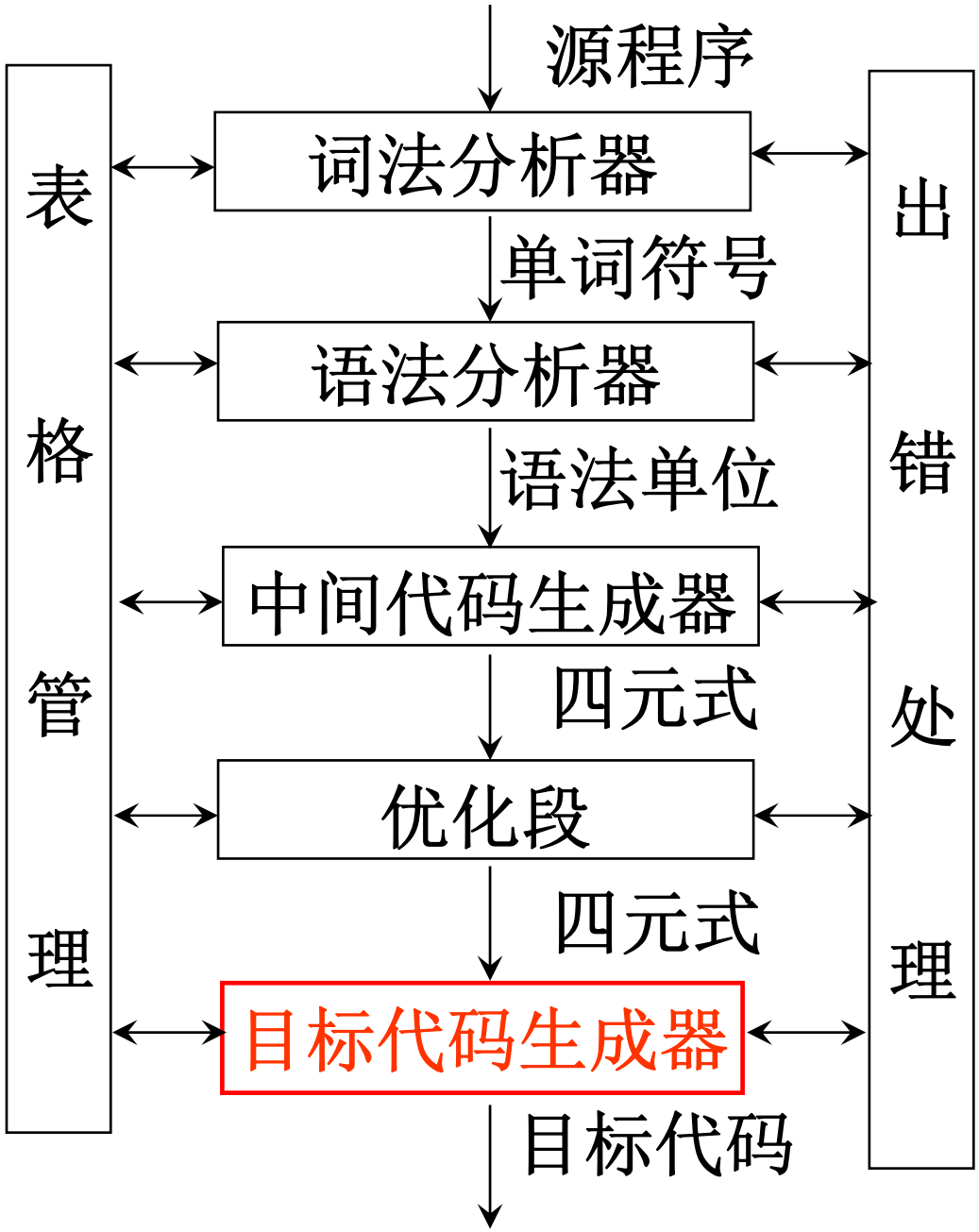
第十一章 目标代码生成

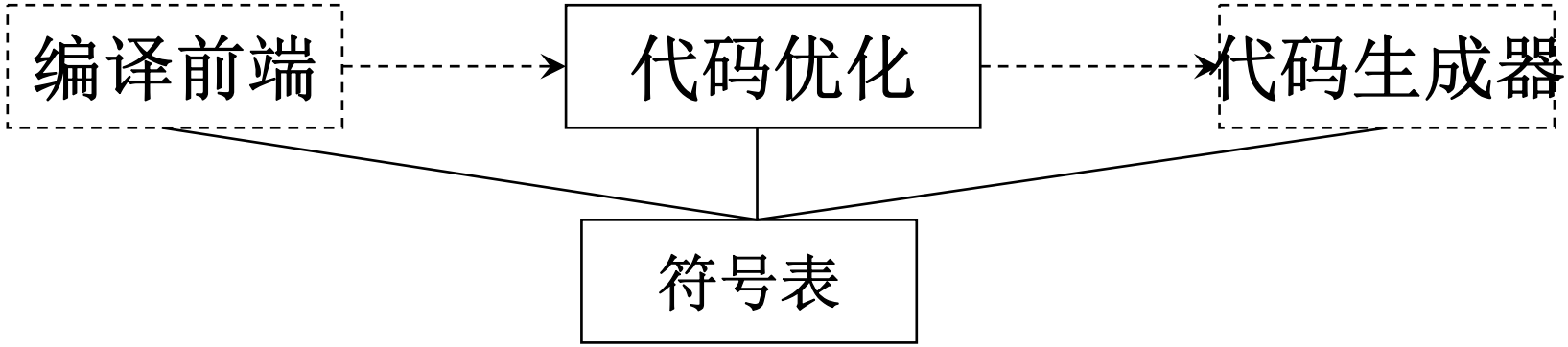

源程序
词法分析器
单词符号
语法分析器
语法单位
中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

内容线索

- **基本问题**
- **目标机器模型**
- **一个简单的代码生成器**

代码生成

- 代码生成器
  - 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
    - 利用符号表信息决定数据对象运行时地址
    - 类型检查



目标代码生成

- **代码生成**是把语义分析后或优化后的中间代码变换成目标代码。
- 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。

基本问题

- 目标代码一般有以下三种形式：
  - **能够立即执行的机器语言代码**，所有地址已经定位；
  - **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；
  - **汇编语言代码**。尚须经过汇编程序汇编，转换成可执行的机器语言代码。

基本问题

- 指令选择
  - 一致性和完整性
  - 指令速度和机器用语
  - a:=a+1
    - INC a　　/*实现最有效*/
    - LD R0, a　　/*将a放入寄存器R0*/
      ADD R0, #1　　/*1与R0相加*/
      ST R0, a　　/*R0的值存入a*/

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。
  - 最优寄存器指派是NP完全问题
- 计算顺序选择

内容线索

- ✓ 基本问题
- ■ **目标机器模型**
- ■ 一个简单的代码生成器

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) ⇒ R_i
寄存器型	op R_i, R_j	(R_i) op (R_j) ⇒ R_i
变址型	op R_i, c(R_j)	(R_i) op ((R_j)+c) ⇒ R_i
间接型	op R_i, *M op R_i, *R_j op R_i, *c(R_j)	(R_i) op ((M)) ⇒ R_i (R_i) op ((R_j)) ⇒ R_i (R_i) op (((R_j)+c)) ⇒ R_i

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

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

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（Ri）⇒ B
J　　X	无条件转向 X 单元
CMP A, B	把 A 单元和 B 单元的值进行比较，并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

内容线索

- ✓ 基本问题
- ✓ 目标机器模型
- ■ 一个简单的代码生成器

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T_1，T_2，T_3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
```

一个简单代码生成器

- 四元式的中间代码变换成目标代码，并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。

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

1　(x,x)表示变量的待用信息和活跃信息。其中i表示待用信息，y表示活跃，^表示非待用和非活跃；

2　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；

3　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。

- 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”；

2. 从基本块出口到基本块入口**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，依次执行下面的步骤：

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

设W和Y是基本块出口的活跃变量，请将相应变量的待用信息和活跃信息填入下表

四元式	左值	左操作数	右 操 作数
T:=A-B			
U:=A-C			
V:=T+U			
W:=V+U			
Y:=U-T			

寄存器描述和地址描述

- **寄存器描述数组RVALUE**
  - **动态记录各寄存器的使用信息**

    例：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**

    例：AVALUE[A]={R1, R2, A}

- 补充说明：
  - 因为寄存器的分配是局限于基本块范围之内的，一旦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，如果它在基本块之后是活跃的，则要把它存在寄存器中的值存放到它的主存单元中。
  - 要特别强调的是，对形如：A:=B的四元式，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。

代码生成算法

对每个四元式: i: A:=B op C，依次执行：

1. **以四元式: i: A:=B op C 为参数，调用函数过程GETREG(i: A:=B op C)，返回一个寄存器R，用作存放A的寄存器。**
2. **利用AVALUE[B]和AVALUE[C]，确定B和C现行值的存放位置B’和C’。如果其现行值在寄存器中，则把寄存器取作B’和C’**

3. 如果B’≠R，则生成目标代码：

LD R, B’

op R, C’

否则生成目标代码 op R, C’

如果B’或C’为R，则删除AVALUE[B]或AVALUE[C]中的R。

4. 令AVALUE[A]={R}, RVALUE[R]={A}。

5. 若B或C的现行值在基本块中不再被引用，也不是基本块出口之后的活跃变量，且其现行值在某寄存器R_k中，则删除RVALUE[R_k]中的B或C以及AVALUE[B]或AVALUE[C] 中的R_k，使得该寄存器不再为B或C占用。

- **寄存器分配：GETREG(i: A:=B op C) 返回一个用来存放A的值的寄存器**

1 尽可能用B独占的寄存器
2 尽可能用空闲寄存器
3 抢占用非空闲寄存器

1 如果B的现行值在某个寄存器R_i中，RVALUE[R_i]中只包含B，此外，或者B与A是同一个标识符，或者B的现行值在执行四元式A:=B op C之后不会再引用，则选取R_i为所需要的寄存器R，并转4；

2 如果有尚未分配的寄存器，则从中选取一个R_i为所需要的寄存器R，并转4；

1 尽可能用B所在的寄存器
2 尽可能用空闲寄存器
3 抢占用非空闲寄存器

3 从已分配的寄存器中选取一个R_i为所需要的寄存器R。最好使得R_i满足以下条件：

占用R_i的变量的值也同时存放在该变量的贮存单元中，或者在基本块中要在最远的将来才会引用到或不会引用到。

4. 要不要为R_i中的变量生成存数指令?

4. 要不要为R_i中的变量V生成存数指令?

(1) 如果V的地址描述数组AVALUE[V]说V还保存在R之外的其他地方，则不需要生成存数指令;

(2) 如果V是A，且不是B或C，则不需要生成存数指令;

(3) 如果V不会在此之后被使用，则不需要生成存数指令;

(4) 否则，生成 目标代码 ST R_i，V

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，只有R_0和R_1是可用寄存器，生成的目标代码和相应的RVALUE和AVALUE:

中间代码	目标代码	RVALUE	AVALUE
T:=A－B	LD R_0，A	R_0含有T	T在R_0中
	SUB R_0，B		
U:=A－C	LD R_1，A	R_0含有T	T在R_0中
	SUB R_1，C	R_1含有U	U在R_1中
V:=T＋U	ADD R_0，R_1	R_0含有V	V在R_0中
		R_1含有U	U在R_1中
W:=V＋U	ADD R_0，R_1	R_0含有W	W在R_0中
	ST R_0，W		

例：基本块

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

- 假设可用寄存器为R_0和R_1，设W和Y是基本块出口的活跃变量

四元式	左值	左操作数	右 操 作数
T:=A-B	**(3,Y)**	**(2,Y)**	**(^,^)**
U:=A-C	**(3,Y)**	**(^,^)**	**(^,^)**
V:=T+U	**(4,Y)**	**(5,^)**	**(4,Y)**
W:=V+U	**(^,Y)**	**(^,^)**	**(5,Y)**
Y:=U-T	**(^,Y)**	**(^,^)**	**(^,^)**

- 用简单代码生成算法生成其目标代码

1. T:=A-B	LD R_0, A
	SUB R_0, B
2. U:=A-C	LD R_1, A
	SUB R_1, C
3. V:=T+U	ST R_0, T
	ADD R_0, R_1
4. W:=V+U	ADD R_0, R_1
5. Y:=U-T	SUB R_1, T
	ST R_0, W
	ST R_1, Y

作业

- 1