



编译原理

2020年6月



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

编译原理

第八章

上海交通大学

张冬莱

Email: zhang-dm@cs.sjtu.edu.cn

2020年6月

第8章 代码生成



本章将介绍编译器中的最后一个阶段——代码生成器，它将源程序的中间表示作为输入，并产生等价有效的目标程序作为输出，如图8-1所示。其中：

- 代码优化阶段试图将中间表示转换成一种可以生成更有效率的目标代码的形式。
- 代码生成器输出的代码必须正确而且质量高。质量高的含义是它应该有效地利用目标机器的资源，此外，代码生成器本身也应该高效地运行。
- 理论上，产生最优代码的问题是无可判定的。在实践中，要选择能够产生好的（而不必是最优的）代码的启发式技术。

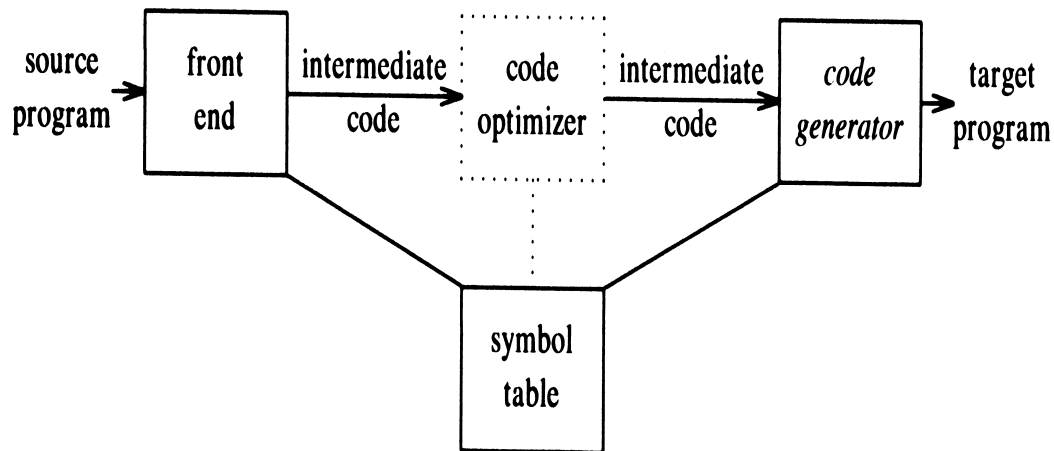


图8-1代码生成器在整个编译中的位置

8.1 代码生成器设计中的问题



虽然代码生成器的具体细节依赖于中间表示形式，目标语言和操作系统，但很多问题（如存储管理、指令选择、寄存器分配和计算次序等）几乎是所有的代码生成器所固有和公共的问题。代码生成器的主要设计目标：

- 目标代码正确
- 易于实现，测试和维护。



8.1.1 代码生成器的输入

代码生成器的输入包括：源程序的中间表示和符号表信息。

代码生成阶段可以认为其输入中没有错误的条件：

- 假定中间表示足够详细，这样，中间语言中名字的值可以表示为目标机器能够直接操作的量(位、整数、实数、指针等)。
- 假定已经完成了必要的类型检查，因此类型转换算符已插在需要的地方，而且明显的语义错误(如试图把浮点数作为数组下标)等都已经检测出来了。



8.1.2 目标程序



代码生成器的输出是目标程序,其输出的形式也是多种多样的,

包括:

- 绝对机器语言: 可以被放在内存中的固定地方并且立即被执行
- 可重定位的机器语言: 允许分别编译子程序。
- 汇编语言: 是可以使代码生成的过程变得容易一些。

8.1.3 指令选择



- 选择适当的目标机指令来实现中间表示(Intermediate Representation,IR)语句,例如: 三地址语句 $x = y + z$ 生成的目标代码为:
 - 1: LD R0, y /* 把y的值加载到寄存器R0中*/
 - 2: ADD R0, R0, z /* z加到R0上*/
 - 3: ST x, R0 /* 把R0的值保存到x中*/
- 例如: 但如上图所示, 目标代码中可能有冗余现象, 如a已经保存到R0中, 不需要在加载一次。

三地址语句序列

➤ $a = b + c$
➤ $d = a + e$

目标代码

➤ LD R0, b // $R0 = b$
➤ ADD R0, R0, c // $R0 = R0 + c$
➤ ST a, R0 // $a = R0$
➤ LD R0, a // $R0 = a$
➤ ADD R0, R0, e // $R0 = R0 + e$
➤ ST d, R0 // $d = R0$



8.1.4 寄存器分配与指派

操作数在寄存器中的指令通常要比操作数在内存中的指令短一些，执行也要快一些。因此，充分利用寄存器对生成好的代码尤其重要。寄存器的使用可以分成两个子问题：

- 1. 在寄存器分配期间，在程序的某一点选择要驻留在寄存器中的变量集。
- 2. 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。



8.2 目标语言



- 熟悉目标机器及其指令集是设计一个好的代码生成器的先决条件。
- 本章使用一个简单计算机的汇编代码作为目标语言。
- 这个计算机是很多寄存器机器的代表。

8.2.1 一个简单的目标机模型



一个简单的目标机模型是一个三地址机器模型，其中的包括了：

- 加载、保存、运算、跳转等操作
- 内存按字节寻址 n 个通用寄存器 $R0, R1, \dots, R_{n-1}$
- 假设所有的运算分量都是整数
- 指令之间可能有一个标号

目标机器的主要指令：

- 加载指令 $LD \text{ dst, addr}$
- 保存指令 $ST \text{ x, r}$
- 运算指令 $OP \text{ dst, src1, src2}$
- 无条件跳转指令 $BR \text{ L}$
- 条件跳转指令 $Bcondr, L$

8.2.2 寻址模式和指令开销



目标机器的寻址模式：

- 指令MOV R0, R1: 将寄存器R0的内容复制到寄存器R1中。这条指令的开销是1, 因为它仅占内存中的一个字。
- 存储指令MOV R5, M: 将寄存器R5的内容复制到内存单元M中。这条指令的开销是2, 因为内存单元M的地址存放在该指令之后的一个字中。
- 指令ADD #1, R3: 将寄存器R3的内容增加1。这条指令的开销是2, 因为常数1必须出现在指令后面的下一个字中。
- 指令SUB 4(R0), *12(R1): 将值 $\text{contents}(\text{contents}(12 + \text{contents}(R1))) - \text{contents}(4 + \text{contents}(R0))$ 存入目的地址 *12(R1)中。这条指令的开销是3, 因为常数4和12要存放在指令之后的下两个字中。

8.2.2 指令开销



我们将主要介绍下面几种策略：

- 应急模式恢复策略。最容易实现。当发现错误时，语法分析器开始抛弃输入记号，每次抛弃一个记号，直到发现某个指定的同步记号为止。比较简单，不会陷入死循环。对于一个语句中出现错误较少时比较合适。
- 短语级恢复策略。发现错误时，语法分析器对剩余的输入字符串作局部纠正，即用一个能使语法分析器继续工作的字符串来代替剩余输入的前缀。可能引起死循环。被用于自顶向下语法分析中。主要缺点是难以应付实际错误出现在诊断点之前的情况。



8.4 基本块和流图



- 三地址语句的一种图形表示叫做流图。
- 流图中的节点表示计算，边表示控制流。
- 利用流图可以作为从中间代码收集信息的工具。
- 某些寄存器分配算法使用流图来寻找消耗程序大部分运行时间的内循环。



8.4.1 基本块




下面的算法可用于把三地址语句序列划分成基本块：

- 划分成基本块算法：
- 输入：一个三地址语句序列。
- 输出：一个基本块列表，其中每个三地址语句仅在一个块中。
- 方法：
 1. 首先确定入口语句（即基本块的第一个语句）的集合。所用规则如下：
 - (i) 第一个语句是入口语句。
 - (ii) 任何能由条件转移语句或无条件转移语句转移到的语句都是入口语句。
 - (iii) 紧跟在转移语句或条件转移后面的语句是入口语句。
 2. 对于每个入口语句，其基本块由它和由每个入口语句构造所在的基本块：
 - ① 由一入口语句到转移或停止语句(含转移或停止语句)之间的语句序列。
 - ② 由一入口语句到下一入口语句(不含下一入口语句)之间的语句序列。
 3. 凡未被归入基本块的语句，为程序控制流图不能到达的语句，因此为不可能执行的语句，故可从程序中删除。

例8.1

- 考虑下图中的源代码片段，它计算两个长度为20的向量a和b的点积。在我们的目标机器上完成该计算的三地址语句序列如后图所示。
- 语句(1) (3) 是一个入口语句，因为最后一条语句可以跳转到它。由规则(iii)，跟在语句(12)之后的语句是一个入口语句（注意，计算点积的三地址码仅是程序片断）。这样，语句(1)和(2)构成一个基本块，从语句(3)开始的余下的语句形成第二个基本块



```
begin
  prod := 0;
  i := 1;
  do begin
    prod := prod + a[i] * b[i];
    i := i + 1
  end
  while i <= 20
end
```

计算点积的程序

```
(1)  prod := 0
(2)  i := 1
(3)  t1 := 4 * i
(4)  t2 := a [ t1 ]      /* compute a[i] */
(5)  t3 := 4 * i
(6)  t4 := b [ t3 ]      /* compute b[i] */
(7)  t5 := t2 * t4
(8)  t6 := prod + t5
(9)  prod := t6
(10) t7 := i + 1
(11) i := t7
(12) if i <= 20 goto (3)
```

8.4.2 后续使用信息(下次引用信息)



- 在基本块B中，变量A在i点的值，j点引用，并且 $i \rightarrow j$ 的通路上没有A的其他定值和引用，则j为i处A的下一个引用点，即下次引用信息。
- 基本块内i处一个变量的活跃信息是指基本块出口之后该变量是否还要被引用，而待用信息则是指基本块内的下一引用点，这些是寄存器分配所需的信息。因此在为基本块的每一语句生成目标代码时，应先求出该语句中变量的待用信息和活跃信息。

后续使用信息(待用信息)的计算



为计算变量的待用信息，在变量符号表中设待用信息栏和活跃信息栏，然后执行下列步骤：

- (1) 开始时，把基本块中各变量的符号表的待用信息栏初始化为“非待用”，活跃信息栏按该变量在基本块出口后是否活跃而初始化为“活跃”或“非活跃”。
- (2) 从出口语句到入口语句反向扫描每个语句(如：P: $x:=y \text{ op } z$)，依次执行：
 - 将符号表中x变量的待用信息和活跃信息附加到语句P上；然后将x在符号表中的信息置为“非待用”、“非活跃”。
 - 将符号表中y，z变量的待用信息和活跃信息附加到语句P上；然后将符号表y、z的待用置为P，活跃栏置为“活跃”。

例8.2： 设W为基本块出口的活跃变量

$$\begin{aligned} (1) \quad t^3_+ &:= a^2_+ - b^0 \\ (2) \quad u^4_+ &:= a^3_+ + c^0 \\ (3) \quad v^4_+ &:= a^0_- - t^0 \\ (4) \quad w^0_+ &:= v^0_- + u^0 \end{aligned}$$
[illegible]

8.4.3 流图



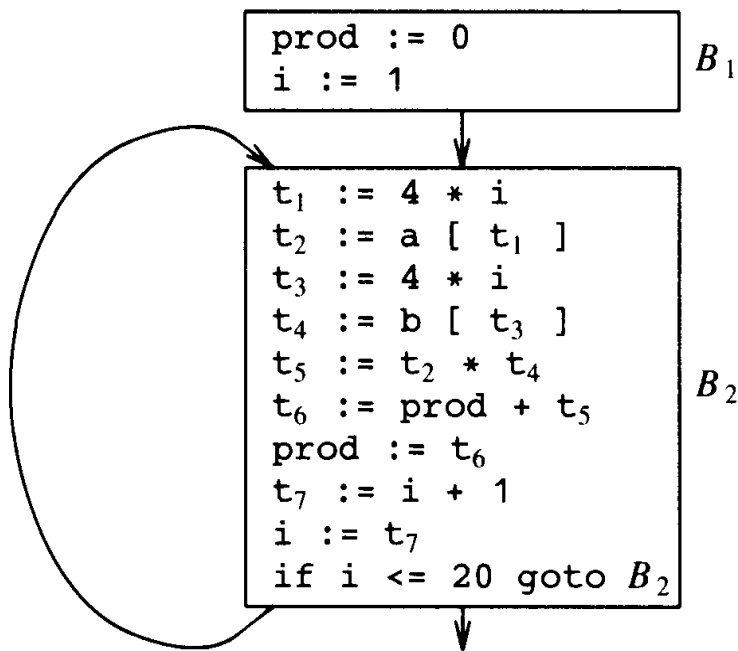
通过构造称为流图的有向图，我们可以把控制流信息加到组成程序的基本块集合中。流图的节点是基本块。有一个特殊的节点称为首节点，这个基本块的入口语句是程序的第一个语句。如果在某个执行序列中 B2 跟随在 B1 之后，则从 B1 到 B2 有一条有向边，即如果：

- 1. 从 B1 的最后一条语句有条件或无条件转移到 B2 的第一个语句；或者
 - 2. 按程序的次序，B2 紧跟在 B1 之后，并且 B1 不是结束于无条件转移
- 则我们说 B1 是 B2 的前驱，而 B2 是 B1 的后继。

8.4.4 流图的表示



- 计算点积的三地址码程序的流图如后图所示。B₁ 是首节点，注意，最后一条语句中，转移到语句(3)的语句已由等价的转移到 B₂ 块开始的语句所代替。



计算点积的三地址码程序的流图

8.4.5 循环



- 在流图中，什么是循环？怎样找出所有的循环？
- 循环是流图中满足下列条件的一簇节点：
 - 1. 簇中所有节点是强连通的，即从循环中任一节点到另一节点都有一条长度大于等于1的路径，路径上的所有节点都在这簇节点中。
 - 2. 这种节点簇有惟一的入口。从循环外的节点到达循环中任一节点的惟一方式是首先通过入口。
- 不包含其他循环的循环叫做内循环。

循环的查找



下面给出的方法是通过分析流图中结点间的控制关系来查找流图中的循环：

- 定义：从流图的首结点出发到达结点 n 的任一通路都必需经过结点 d ，则称 d 是 n 的必经结点，记为 $d \text{ DOM } n$ 。

根据定义，容易看出：

- (1) 每个结点是它本身的必经结点，即 $n \text{ DOM } n$ 。
- (2) 流图的首结点 n_0 ，是流图中任一结点 n 的必经结点， $n_0 \text{ DOM } n$ 。
- (3) 循环 L 的入口结点 d ，是循环内任一结点的必经结点。如果 d 不是首结点 n_0 ，即 $n_0 \notin L$ ， n_0 在循环外，则由 n_0 到循环中任一结点 n 的任一通路，必经过 d 而进入循环，故 d 是 n 的必经结点。
- (4) 一个结点的必经结点可能不止一个，如对循环内除入口结点 n 而言，它至少有三个必经结点：首结点 n_0 ，循环入口结点 d ，及结点 n 自身。我们将流图中结点 n 的所有必经结点称为 n 的必经结点集，用 $D(n)$ 表示。

必经结点集的性质及求解



如果将DOM看成结点集N上的一种二元关系，那么它具有如下性质：

- (1) 自反性。对任一 $n \in N$ ， $n \text{ DOM } n$ ；
- (2) 反对称性。若 $m, n \in N$ ， $m \text{ DOM } n$ 且 $n \text{ DOM } m$ 则必有 $m=n$ ；
- (3) 传递性。若 $l, m, n \in N$ ，而 $l \text{ DOM } m$ 且 $m \text{ DOM } n$ ，则必有 $l \text{ DOM } n$ 。

这是迭代求解过程，其初值除首结点 $D(n_0)=n\{0\}$ 外，其余结点 n ， $D(n)=N$ 。每次迭代中如果有一个结点 n 的新才 $D(n)$ 不相同，表明迭代未收敛，这由布尔变量change来标志。集合可用位向量表示，此时集合运算 \cup 、 \cap 就可用 \wedge 、 \vee 来代替了。

```
D(n0)={n0};  
for(n∈N-{n0})  
  { D(n)=N; }  
  chang=true;  
  while(change)  
  {change=false; }  
  for(n∈N-{n0})  
  {  
    ;  
    if (new != D(n))  
    {change=true; D(n)=new; }  
  }  
图求必经结点集D(n)的算法
```

例8.3



- 右图的流图，其必经结点已如前图所示，它有如下四条回边：

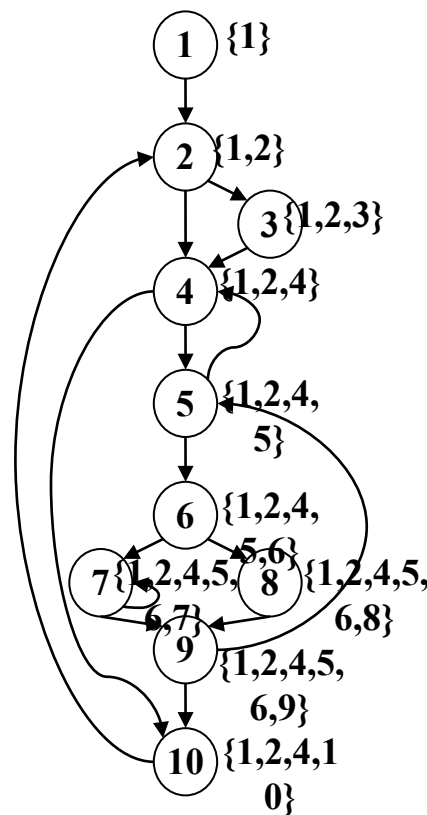
$5 \rightarrow 4$ ，因 $4 \in D(5) = \{1, 2, 4, 5\}$

$9 \rightarrow 5$ ，因 $5 \in D(9) = \{1, 2, 4, 5, 6, 9\}$

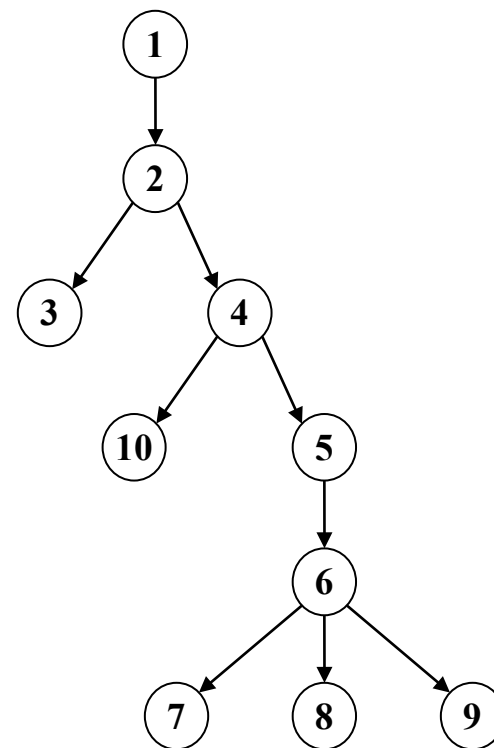
$10 \rightarrow 2$ ，因 $2 \in D(10) = \{1, 2, 4, 10\}$

$7 \rightarrow 7$ ， $7 \in D(7)$

- 定义：若 $n \rightarrow d$ 是流图 $G=(N, E, n_0)$ 的一条回边， M 是流图中有通路到达 n 而该通路不经过 d 的结点集，则 $Loop = \{n, d\} \cup M$ ，构成了 G 的一个子图，称为由回边 $n \rightarrow d$ 组成的自然循环。



必经结点集



必经结点树

8.5 基本块的优化



- 基本块内的优化类型

- 1. 合并已知量

对于 $A := \text{op } B$ 或 $A := B \text{ op } C$ ，其中 B 及 C 均为常数，则编译时即可计算出 $\text{op } B$ 或 $B \text{ op } C$ 的值，作为 A 的值，而不必生成相应的代码。

- 2. 删除公共子表达式

也称为删除多余运算。如例题8.4中，语句③ $t2 := R + r$

和⑥ $t4 := R + r$ 就是公共子表达式 $R + r$ ，故语句⑥ $t4 := R + r$ 可变换成 $t4 := t2$ ，这样就避免了多余运算。

基本块的dag表示及构造



用一个有向无环图dag(directed acyclic graph)来表示一个基本块，方法见第六章6.1.为了描述计算过程，我们在无环路有向图的结点上给出如下标记或附加标记：

如果待处理的语句形如 $x:=y \text{ op } z$ ，其处理的步骤为：

- (1)在已建立的dag图中寻找能代表 y 、 z 当前值的结点。
- (2)若代表 y 、 z 当前值的结点至少有一个标记为非常数，则执行步骤(3)，否则执行下列步骤：
 - 执行 $y \text{ op } z$ ，令其结果常数为 x_0 。
 - 如果， y 或 z 是执行当前语句 $x:=y \text{ op } z$ 时新建立的结点，则删除它。
 - 如果已建立的dag子图中没有标记为 x_0 的常数叶结点，则建立新的结点 n ，它的标记为 x_0 。
 - 如果已建立子图有标记为 x_0 的叶结点，则令该结点为 n 。
 - 执行步骤(4)



(3) 在已建立的dag子图中寻找这样的结点：它标记为 op ，它的左子结点为代表 y 当前值的结点它的右子结点为代表 z 当前值的结点。也有两种可能：

- 在已建立的dag子图中没有这样的结点，则建立一新的内部结点 n ，它以代表 y 、 z 当前值的结点作为它的左右子结点，令它标记为 op 。

- 在已建立子图中存在这样的结点，不妨令它为 n 。

(4) 对于在步骤(2)或(3)中找到的或新建立的结点 n ，应将 x 加入到该结点的附加标记集中，但在此之前如果 x (不是 x_0)已出现在某个结点的附加标记集中，则应先将 x 从这个附加标记集中删除。这表明此时 x 已重新定值， x 的当前值已不是原来的那个 x 了。

(5) 处理下一语句，直至结束。

例8.4



与下列基本块相应的dag的构造过程如图8.2。

- ① $pi := 3.14$
- ② $t1 := 2 * pi$
- ③ $t2 := R + r$
- ④ $A := t1 * t2$
- ⑤ $t3 := 2 * pi$
- ⑥ $t4 := R + r$
- ⑦ $t5 := t3 * t4$
- ⑧ $t6 := R - r$
- ⑨ $t7 := t5 * t6$
- ⑩ $A := t7 - A$

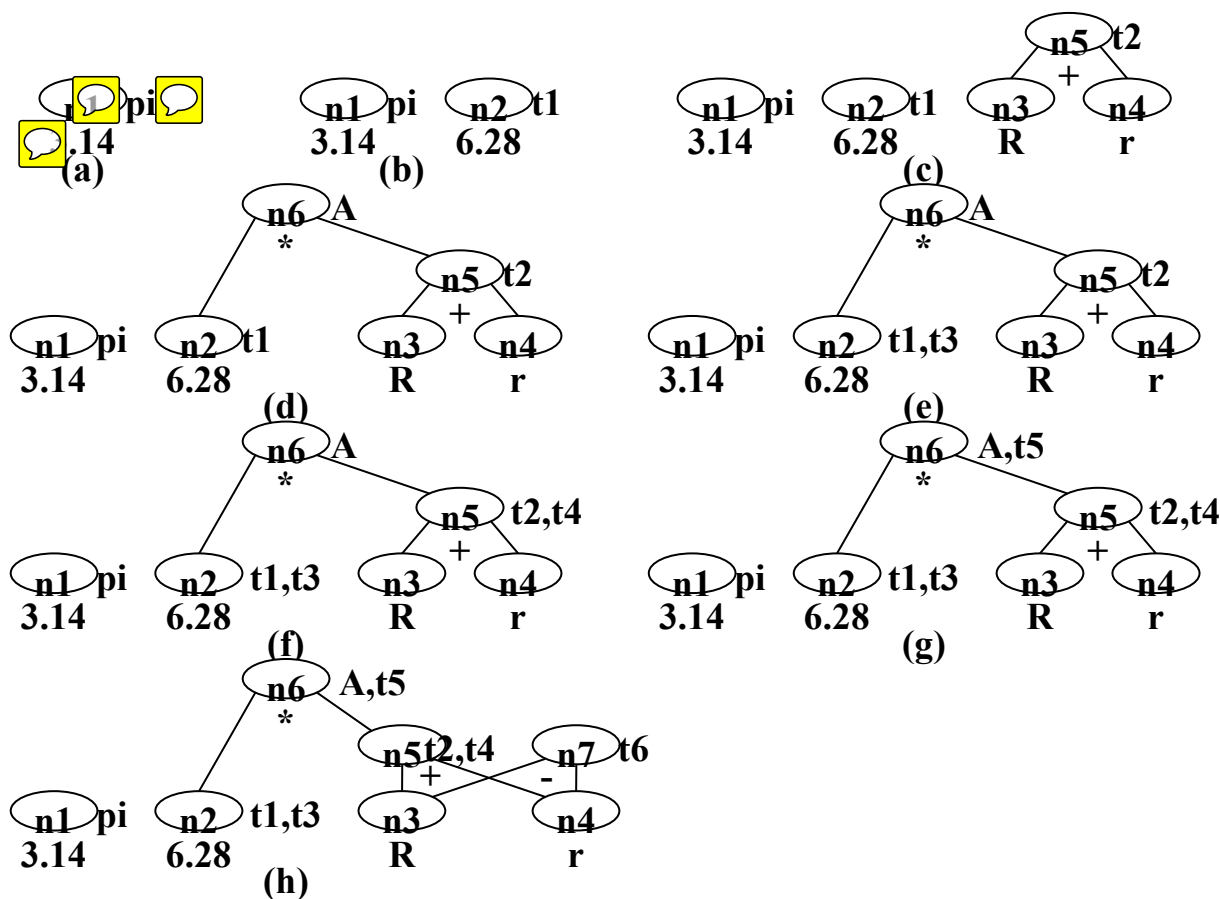


图8.4的dag按结点建立次序重构基本块，则可得经上述优化后的基本块。

- ① $pi:=3.14$
- ② $t1:=6.28$
- ③ $t3:=6.28$
- ④ $t2:=R+r$
- ⑤ $t4:=t2$
- ⑥ $t5:=6.28*t2$
- ⑦ $t6:=R-r$
- ⑧ $t7:=t5*t6$
- ⑨ $A:=t7-t5$

如果 pi 及 $t1-t7$ 出基本块不活跃(它们只是在计算表达式时使用的临时变量)，则重构的基本块可进一步优化为：

- ① $t1:=R+r$
- ② $t5:=6.28*t2$
- ③ $t6:=R-r$
- ④ $t7:=t5*t6$
- ⑤ $A:=t7-t5$

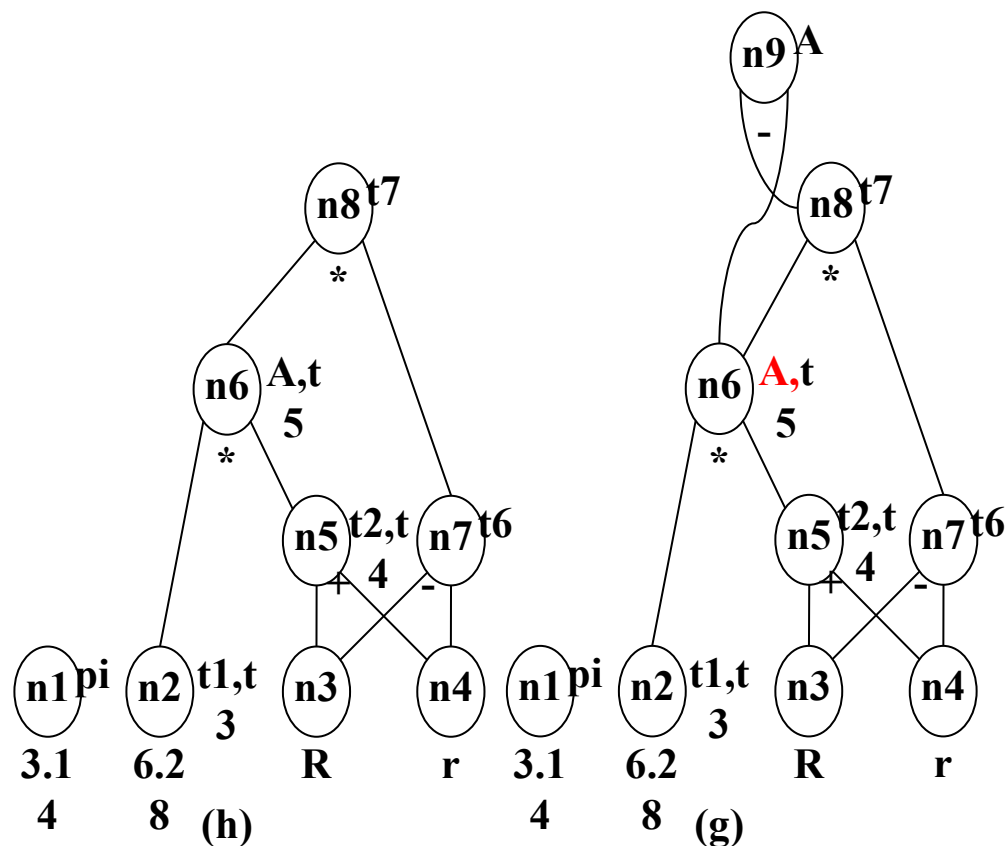


图8.4 dag的构造过程

8.6 一个简单的代码生成器



本节中的代码生成策略为三地址语句序列生成目标代码，它依次考虑每条语句，记住该语句当前是否有操作数在寄存器中，如果有的话，尽量利用这一点。为简单起见，我们假设三地址语句的每种算符都有对应的目标机器算符。还假定将计算结果尽可能长时间地保留在寄存器中，只有在下面两种情况下才将其存入内存：

- (a) 如果此寄存器要用于其他计算。
- (b) 正好在过程调用、转移语句之前。
- 条件(b)暗示在基本块的结尾，必须将所有的东西都保存起来。必须这样做的原因是，离开一个基本块后，可能进入几个不同的基本块中的一个，或者进入一个还可以从其他块进入的基本块。在这两种情况下，没有额外的工作，就认为不管控制怎样到达某基本块，该块引用的数据总是处于相同的寄存器中是不妥的。因此，为避免可能的错误，这个简单的代码生成算法在穿越基本块或调用过程时，存储所有的东西。

▪

8.6.1 寄存器描述符和地址描述符



对每个形如 $x = y \text{ op } z$ 的三地址指令 l ，执行如下动作：

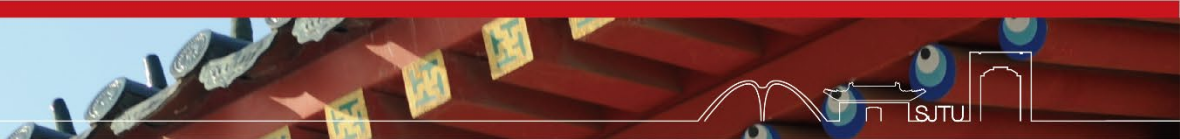
- 调用函数`getreg(l)`来为 x, y, z 选择寄存器，把这些寄存器称为 R_x, R_y, R_z
- 如果 R_y 中存放的不是 y ，则生成指令“LD R_y, y' ”。 y' 是存放 y 的内存位置之
- 类似的，如果 R_z 中存放的不是 z ，
- 生成指令“LD R_z, z' ”生成目标指令“OP R_x, R_y, R_z ”

寄存器描述符(registerdescriptor):

- 记录每个寄存器当前存放的是哪些变量的值

地址描述符(addressdescriptor):

- 记录运行时每个名字的当前值存放在哪个或哪些位置该位置可能是寄存器、栈单元、内存地址或者是它们的某个集合这些信息可以存放在该变量名对应的符号表条目中



基本块的收尾处理

- 对于一个在基本块的出口处可能活跃的量 x ，如果它的地址描述符表明它的值没有存放在 x 的内存位置上，则生成指令“ST x, R ”(R 是在基本块结尾处存放 x 值的寄存器)

管理寄存器和地址描述符

- 当代码生成算法生成加载、保存和其他指令时，它必须同时更新寄存器和地址描述符

对于指令“LD R, x ”

- 修改 R 的寄存器描述符，使之只包含 x
- 修改 x 的地址描述符，把 R 作为新增位置加入到 x 的位置集合中
- 从任何不同于 x 的地址描述符中删除 R

对于指令“OP R_x, R_y, R_z ”

- 修改 R_x 的寄存器描述符，使之只包含 x
- 从任何不同于 R_x 的寄存器描述符中删除 x
- 修改 x 的地址描述符，使之只包含位置 R_x
- 从任何不同于 x 的地址描述符中删除 R_x

对于指令“ST x, R ”

- 修改 x 的地址描述符，使之包含自己的内存位置

对于复制语句 $x=y$ ，如果需要生成加载指令“LD R_y, y' ”则

- 修改 R_y 的寄存器描述符，使之只包含 y
- 修改 y 的地址描述符，把 R_y 作为新增位置加入到 y 的位置集合中
- 从任何不同于 y 的变量的地址描述符中删除 R_y
- 修改 R_y 的寄存器描述符，使之也包含 x
- 修改 x 的地址描述符，使之只包含 R_y

例8.5

$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$

LD R1, a
LD R2, b
SUB R2, R1, R2

$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$

LD R3, c
SUB R1, R1, R3

$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$

ADD R3, R2, R1

$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$

LD R2, d

$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$

ADD R1, R3, R1

R1	R2	R3	a	b	c	d	t	u	v
a	t		a, R1	b, R2	c	d	R2		

R1	R2	R3	a	b	c	d	t	u	v
u	t	c	a, R1	b	c, R3	d	R2	R1	

R1	R2	R3	a	b	c	d	t	u	v
u	t	v	a	b	c, R3	d	R2	R1	R3

R1	R2	R3	a	b	c	d	t	u	v
u	d, a	v	R2	b	c	d, R2	R2	R1	R3

R1	R2	R3	a	b	c	d	t	u	v
d	d, a	v	R2	b	c	R1		R1	R3



$t = a - b$
 $u = a - c$
 $v = t + u$
 $a = d$
 $d = v + u$
exit

ST a, R2
ST d, R1

R1	R2	R3	a	b	c	d	t	u	v
d	a	v	R2, a	b	c	R1, d			R3

第八章 总结



主要内容：

- 目标语言与指令选择
- 基本块划分与流图构造
- 后续使用信息(下次引用信息)的求解
- 循环体查找
- 基本块DAG优化
- 目标代码生成

第八章 作业



1.请对右列程序在划分基本块的基础上构造程序流图。

2.请对B1基本块内的代码分别应用dag进行局部优化；
写出优化后的代码序列；
如果只有L是在基本块后活跃的，
写出相应的优化代码序列。

3.对题1构造的流图 分别：

- (a) 求各结点的必经结点集；
- (b) 求流图中的回边；
- (c) 求由回边组成的循环；

4.将 $x = a / (b + c) - d * (e + f)$ 生成三地址代码后生成目标代码，设有任意多可用寄存器，乘法和除法助记符为mul和div

(1) i:=1	(8) J:=2*J
(2) if I>5 goto(4)	(9) if J≥8 goto(12)
(3) J:=j	(10) goto(14)
(4) if J<6 goto(1)	(11) if I=4 goto(4)
(5) i:=i*2	(12) I:=I-1
(6) if I<4 goto(11)	(13) if J=7 goto(11)
(7)J:=J/2	(14) halt

B1: F:=1
C:=F+E
D:=F+3
B:=A*A
G:=B-D
H:=E
I:=H*G
J:=D/4
K:=J+G
L:=H
L:=I-L

谢谢!



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

上海交通大学