山东大学
SHANDONG UNIVERSITY

编译原理

第十一章 目标代码生成

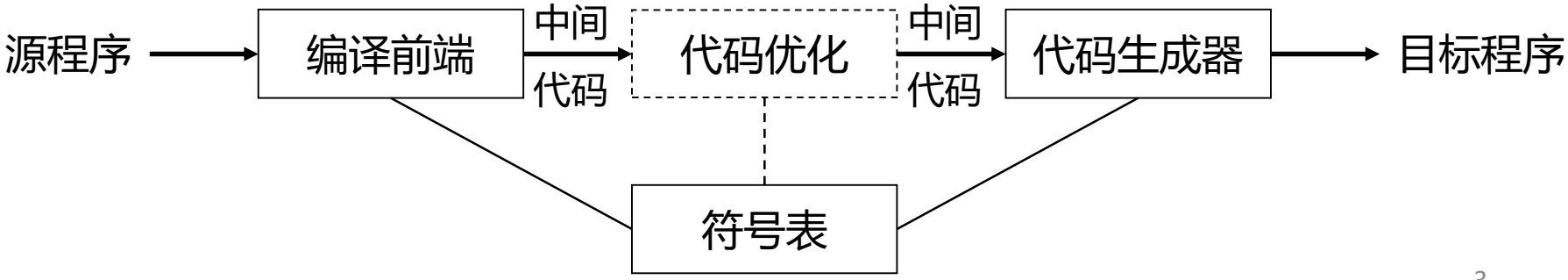
授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

第十一章 目标代码生成

- 目标代码生成：以源程序的中间代码作为输入，产生等价的目标程序作为输出；目标代码有三种形式
  - 能够立即执行的机器语言代码，所有地址均已定位。
  - 待装配的机器语言模块。
  - 汇编语言代码，需经过汇编程序汇编，转换为可执行的机器语言代码。
- 代码生成要着重考虑两个问题：
  - 如何使生成的目标代码较短；
  - 如何充分利用寄存器，减少目标代码中访问存储单元的次数。



第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.1 基本问题

- 代码生成器的输入
  - 中间语言的选择，本章采用三地址码，但其中许多技术可以用于其它中间表示
  - 代码生成器利用符号表中的信息，决定中间代码中名字所指示的数据对象的运行时地址，它是可再定位地址或绝对地址。
  - 假定已做过类型检查，输入没有错误。
- 目标程序
  - 本章采用汇编语言作为目标语言。
- 指令选择
  - 指令集的一致性和完全性是重要因素；
  - 生成代码都质量取决于它的速度和大小。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

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
  - 计算完成的顺序会影响目标代码都有效性。

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

11.2 目标机器模型

基本指令

- 直接地址型：$op\ R_i, M$ // 单目：$op\ (M) \Rightarrow R_i$，双目：$(R_i)\ op\ (M) \Rightarrow R_i$
- 寄存器型：$op\ R_i, R_j$ // 单目：$op\ (R_j) \Rightarrow R_i$，双目：$(R_i)\ op\ (R_j) \Rightarrow R_i$
- 变址型：$op\ R_i, c[R_j]$

  // 单目：$op\ (c + (R_j)) \Rightarrow R_i$，双目：$(R_i)\ op\ (c + (R_j)) \Rightarrow R_i$
- 间接型：// x86需要LEA取地址，以下表示不支持

  $op\ R_i,* M$ // 单目：$op\ ((M)) \Rightarrow R_i$，双目：$(R_i)\ op\ ((M)) \Rightarrow R_i$

  $op\ R_i,* R_j$ // 单目：$op\ ((R_j)) \Rightarrow R_i$，双目：$(R_i)\ op\ ((R_j)) \Rightarrow R_i$

  $op\ R_i,* c[R_j]$// 单目：$op\ ((c + (R_j))) \Rightarrow R_i$，双目：$(R_i)\ op\ ((c + (R_j))) \Rightarrow R_i$
- 运算符（操作码）op包括常见的运算，如果ADD、SUB、MUL、DIV等。
- 立即数前面加#。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

乘法指令

- 乘法指令
  - MUL reg/mem；无符号乘法；如果乘积的高半部分不为零，则 MUL 会把进位标志位和溢出标志位置 1。
  - IMUL reg/mem；有符号乘法；如果乘积的高半部分不是其低半部分的符号扩展，则进位标志位和溢出标志位置 1。
- 乘法指令只有一个操作数，为乘数，另外一个操作数被乘数在一个固定寄存器中，乘积也放在固定的寄存器中
  - 乘法指令操作数为reg8/mem8，被乘数在AL中，乘积存入AX。
  - 乘法指令操作数为reg16/mem16，被乘数在AX中，乘积存入DX:AX；即高16位在DX中，低16位在AX中。
  - 乘法指令操作数为reg32/mem32，被乘数在EAX中，乘积存入EDX:EAX；即高32位在EDX中，低32位在EAX中。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

除法指令

除法指令

- DIV reg/mem；无符号除法。
- IDIV reg/mem；有符号除法。

被除数、商和余数使用固定寄存器规则

- 除法指令操作数为reg8/mem8，被除数放入AX，商放入AL，余数放入AH。
- 除法指令操作数为reg16/mem16，被除数放入DX:AX，商放入AX，余数放入DX
- 除法指令操作数为reg32/mem32，被除数放入EDX:EAX，商放入EAX，余数放入EDX。

符号扩展指令

- CBW；将AL的符号位扩展到AH，调用IDiv reg8/mem8前使用。
- CWD；将AX的符号位扩展到DX，调用IDiv reg16/mem16前使用。
- CDQ：将EAX的符号位扩展到EDX，调用IDiv reg32/mem32前使用。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.2 目标机器模型

存取指令

- $MOV\ R_i, B$ // 把B单元的内容加载到寄存器R_i，即$(B) \Rightarrow R_i$
- $MOV\ B, R_i$ // 把寄存器R_i的内容存储到B单元，即$(R_i) \Rightarrow B$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.2 目标机器模型

比较与跳转指令

- $JMP\ X$ // 无条件跳转到X单元
- $CMP\ A,B$ // 把A单元和B单元的内容比较，根据比较情况把机器内部特征寄存器CT置成相应状态，0,1,2分别表示小于、等于、大于。

跳转条件	无符号跳转	有符号跳转	字母说明
相等跳转	JE		E: Equal
不等跳转	JNE		N: Not
大于跳转	JA / JNBE	JG / JNLE	A: Above
大于等于跳转	JAE / JNB	JGE / JNL	G: Greater than
小于跳转	JB / JNAE	JL / JNGE	B: Below
小于等于跳转	JBE / JNA	JLE / JNG	L: Less than

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.3 一个简单的代码生成器

【例11.1】$A=(B+C)*D+E$，中间代码如下

$T_1=B+C$

$T_2=T_1*D$

$A=T_2+E$

对$x=y+z$，可以简单翻译为：

$MOV\ EAX,y$

$ADD\ EAX,z$

$MOV\ x,EAX$

目标代码

(1) $MOV\ EAX,B$

(2) $ADD\ EAX,C$

(3) $MOV\ T_1,EAX$

(4) $MOV\ EAX,T_1$

(5) $MUL\ D$

(6) $MOV\ T_2,EAX$

(7) $MOV\ EAX,T_2$

(8) $ADD\ EAX,E$

(9) $MOV\ A,EAX$

11.3 一个简单的代码生成器

$T_1 = B + C$

$T_2 = T_1 * D$

$A = T_2 + E$

目标代码

(1) $MOV\ EAX, B$

(2) $ADD\ EAX, C$

(3) $MOV\ T_1, EAX$

(4) $MOV\ EAX, T_1$

(5) $MUL\ D$

(6) $MOV\ T_2, EAX$

(7) $MOV\ EAX, T_2$

(8) $ADD\ EAX, E$

(9) $MOV\ A, EAX$

- 问题
  - 指令（4）（7）在前一条指令存储后马上取出，寄存器内容未变化，因此是多余的；
  - 临时变量T_1和T_2是生成中间代码引入的，基本块后不活跃，因此（3）（6）多余。

优化后的目标代码

(1) $MOV\ EAX, B$

(2) $ADD\ EAX, C$

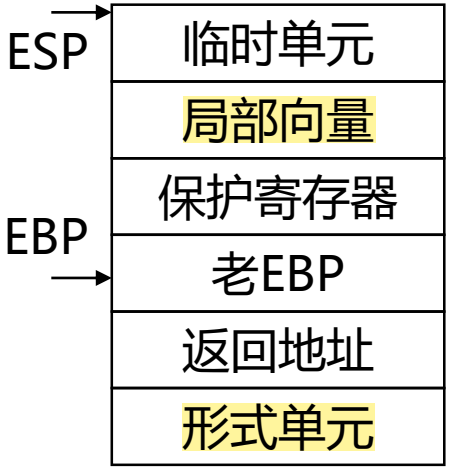
(3) $MUL\ D$

(4) $ADD\ EAX, E$

(5) $MOV\ A, EAX$

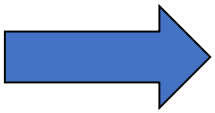
11.3 一个简单的代码生成器

- 真实指令
  - 形参：$EBP + (offset + 8)$
  - 变量（如果保留n个寄存器）：$EBP - (offset + 4(n+1))$



名字	类别	类型	大小	偏移量
x	形参	integer	4	0
y	形参	integer	4	4
a	变量	integer	4	0
b	变量	integer	4	4
var	变量	integer	4	8
$1	临时变量	integer	4	12
$2	临时变量	integer	4	16

(1) $MOV\ EAX, x$
(2) $ADD\ EAX, y$
(3) $MUL\ b$
(4) $ADD\ EAX, \$1$
(5) $MOV\ \$2, EAX$



(1) $MOV\ EAX, [EBP + 8]$
(2) $ADD\ EAX, [EBP + 12]$
(3) $MUL\ [EBP - 40]$
(4) $ADD\ EAX, [EBP - 48]$
(5) $MOV\ [EBP - 52], EAX$

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.3.1 待用信息

- 一个基本块范围内考虑如何充分利用寄存器的问题
  - 当生成计算某变量值的目标代码时，尽可能的将该变量的值保存在寄存器中，直到该寄存器必须用来存放别的变量值，或达到基本块的出口；
  - 后续的目标代码尽可能的引用保存在寄存器的值，而不访问主存。
- 具体做法
  - 把还要引用的变量值尽可能保存在寄存器中；
  - 把基本块内不再被引用的变量所占用的寄存器及早释放。
- 为此，当翻译$A = B\ op\ C$时，需要知道：
  - A, B, C是否还会在基本块内被引用，即活跃信息；
  - 在哪些中间代码中被引用，即待用信息。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.3.1 待用信息

获得待用信息和活跃信息的基本思路

- 从基本块的出口从后向前扫描每个中间代码，对每个变量建立相应的待用信息链和活跃变量信息链；
- 如果没有进行过数据流分析，且临时变量不可跨基本块使用，则把基本块中所有临时变量均看作基本块出口之后的非活跃变量，而把所有非临时变量看作基本块出口之后的活跃变量；
- 如果某些临时变量可以跨基本块引用，那么也把它们看作基本块出口之后的活跃变量。

符号

- 符号(×,×)表示待用及活跃信息；
- 待用信息i表示下一个引用点；
- 活跃信息用Y表示活跃。

a活跃，待用10 {

a不活跃 {

a活跃，待用27 {

$$\begin{array}{l}\cdots\\(10)\ x=a+b\\\cdots\\(18)\ a=\cdots\\\cdots\\(27)\ y=a*b\end{array}$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

获取待用信息的算法

- 为每个变量建立待用和活跃信息链，每条中间代码的变量可附加该信息

(1) 初始化，把基本块中各变量的符号表中，待用信息栏填为“非待用”；并根据变量在基本块出口之后是不是活跃，填写活跃信息栏。

(2) 从基本块出口到基本块入口由后向前依次处理各个中间代码，对每个中间代码 (i) $A = B\ op\ C$

① 把符号表中A的待用信息和活跃信息附加到中间代码(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B和C的待用信息和活跃信息附加到中间代码(i)上；

④ 把符号表中B和C的待用信息置为i，活跃信息置为“活跃”。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.2】考察基本块，其中W是出口活跃变量，计算待用信息和活跃信息。

(1) $T = A - B$　　(2) $U = A - C$　　(3) $V = T + U$　　(4) $W = V + U$

变量名	待用信息及活跃信息
T	$(-,-)$ $\rightarrow (3,Y)$ $\rightarrow (-,-)$
A	$(-,-)$ $\rightarrow (2,Y)$ $\rightarrow (1,Y)$
B	$(-,-)$ $\rightarrow (1,Y)$
C	$(-,-)$ $\rightarrow (2,Y)$
U	$(-,-)$ $\rightarrow (4,Y)$ $\rightarrow (3,Y)$ $\rightarrow (-,-)$
V	$(-,-)$ $\rightarrow (4,Y)$ $\rightarrow (-,-)$
W	$(-,Y)$ $\rightarrow (-,-)$

序号	中间代码	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	$W = V + U$	$(-,Y)$	$(-,-)$	$(-,-)$

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

11.3.2 寄存器描述和地址描述

- 寄存器信息
  - 空闲？分配给某个变量？分配给几个变量（复写时出现这种情况）？
  - 建立一个编译时用的寄存器描述数组$Rvalue$，动态地记录各寄存器的上述信息
- 变量信息
  - 如果变量存在寄存器中，自然希望使用寄存器中的值，而不是主存中的值。
  - 建立一个变量地址描述数组$Avalue$，动态地记录各变量现行值的存放位置：是寄存器中，还是某主存单元，还是既在寄存器又在主存单元。

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

基本块代码生成算法

- 对每个中间代码：$(i)\ A = B\ op\ C$

$(1) R = getR(i)$。

(2) 根据$Avalue[B]$和$Avalue[C]$，确定变量B和C的存放位置B'和C'；如果现行值在寄存器，则把寄存器取做B'和C'。

(3) 如果$B' \neq R$，则生成目标代码$MOV\ R, B'; op\ R, C'$；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$Avalue[A] = \{R\}$，并令$Rvalue[R] = \{A\}$。

(5) 如果B和C在基本块不再引用，在基本块出口后不再活跃，且现行值在某个寄存器R_i，则删除$Rvalue[R_i]$中的B或C，以及$Avalue[B]$或$Avalue[C]$中的R_i。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 获得存放A的寄存器$getR(i)$，设第i个四元式为$A = B\ op\ C$

(1) 如果B存放在某个寄存器R_i，$Rvalue[R_i]$只包含B，同时，或者B与A是同一标识符，或者中间代码i中B的信息为$(-,-)$（即后面不再引用），则选取R_i为所需寄存器，转(4)④。

(2) 如果(1)失败，若有空闲寄存器R_i，选择其作为所需寄存器，转(4) ④ 。

(3) 若(2)也失败，需要从已分配寄存器中选择R_i：占用该R_i的变量也保存在主存中，或者在最远的地方被引用（即待用信息值最大）。

(4) 对$Rvalue[R_i]$中的每个变量M，如果$M \neq A$，或者$M = A = C \neq B \wedge B \notin Rvalue[R_i]$，则：

① 如果$M \notin Avalue[M]$，生成目标代码$MOV\ M, R_i$；

② 如果$M = B$，或者$M = C \wedge B \in Rvalue[R_i]$，则令$Avalue[M] = \{M, R\}$，否则令$Avalue[M] = \{M\}$；

③ 删除$Rvalue[R_i]$中的M；

④ 给出R，返回。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.3】基本块生成目标代码，假设有寄存器R_0, R_1。

(1) $T = A - B$　　(2) $U = A - C$　　(3) $V = T + U$　　(4) $W = V + U$

序号	中间代码	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	$W = V + U$	$(-, Y)$	$(-, -)$	$(-, -)$

中间代码	目标代码	Rvalue	Avalue
$T = A - B$	$MOV\ R_0, A$ $SUB\ R_0, B$	$Rv(R_0) = \{T\}$	$Av(T) = \{R_0\}$
$U = A - C$	$MOV\ R_1, A$ $SUB\ R_1, C$	$Rv(R_0) = \{T\}$ $Rv(R_1) = \{U\}$	$Av(T) = \{R_0\}$ $Av(U) = \{R_1\}$
$V = T + U$	$ADD\ R_0, R_1$	$Rv(R_0) = \{V\}$ $Rv(R_1) = \{U\}$	$Av(V) = \{R_0\}$ $Av(U) = \{R_1\}$
$W = V + U$	$ADD\ R_0, R_1$	$Rv(R_0) = \{W\}$	$Av(W) = \{R_0\}$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

各中间代码对应的目标代码

- $A = B\ op\ C$

  $MOV\ R_i, B$

  $op\ R_i, C$

  - 其中R_i是新分配给A的寄存器；
  - 如果B和/或C的现行值在寄存器中，则目标中B和/或C用寄存器表示。但如果C在R_i中，则C要用其主存单元表示；
  - 如果B的现行值在R_i中，则不生成第一条目标代码。

- $A = op\ B$

  $MOV\ R_i, B$

  $op\ R_i$

  - 其中R_i是新分配给A的寄存器；
  - 如果B的现行值在R_i中，则不生成第一条目标代码。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

各中间代码对应的目标代码

- $A = B$

  $MOV\ R_i, B$

  - 其中R_i是新分配给A的寄存器；
  - 如果B的现行值在R_i中，则不生成目标代码。

- $A = B[I]$

  $MOV\ R_j, I$

  $MOV\ R_i, B[R_j]$

  - 其中R_i是新分配给A的寄存器；
  - 如果I的现行值在某个R_j中，则不生成第一条目标代码，否则R_j是分配给I的寄存器。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

各中间代码对应的目标代码

- $A[I] = B$

  $MOV\ R_i, B$

  $MOV\ R_j, I$

  $MOV\ A[R_j], R_i$

  - 如果B的现行值在R_i中，则不生成第一条目标代码；
  - 如果I的现行值在某个R_j中，则不生成第二条目标代码，否则R_j是分配给I的寄存器。

- $goto\ X$

  $JMP\ X'$

  - X'是标号为X的中间代码的目标代码的首地址。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

各中间代码对应的目标代码

- $if\ A\ \theta\ B\ goto\ X$

  $MOV\ R_i, A$

  $CMP\ R_i, B$

  $j\theta\ X'$

  - X'是标号为X的中间代码的目标代码的首地址；
  - 如果A的现行值在R_i中，则不生成第一条目标代码；
  - 如果B的现行值在某个R_k中，则目标代码中的B就是R_k；
  - θ指$<, \leq, =, \neq, >, \geq$。

- $A =* p$

  $LEA\ R_j, p$

  $MOV\ R_i, [R_j]$

  - 其中R_i是新分配给$* p$的寄存器。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

各中间代码对应的目标代码

- $*p = A$

  $MOV\ R_i, A$

  $LEA\ R_j, p$

  $MOV\ [R_j], R_i$

  - 其中R_i是新分配给A的寄存器；
  - 如果A的现行值在R_i中，则不生成第一条目标代码。

- 如果寄存器中的某变量在基本块出口之后是活跃的，则需要用MOV指令将其存储到主存单元中
  - 利用$Rvalue$确定哪些变量的现行值在寄存器中；
  - 利用$Avalue$确定哪些变量的现行值不在主存中；
  - 利用活跃变量信息确定哪些变量是活跃的。

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

11.4 寄存器分配

- 基本思想
  - 在循环中，寄存器不是平均分配，而是从可用寄存器中分出几个，固定分配给几个变量单独使用。
- 执行代价：每条指令的执行代价 = 每条指令访问主存单元次数 + 1
  - $op\ R_i, R_j$　　执行代价为1
  - $op\ R_i, M$　　执行代价为2
  - $op\ R_i, [R_j]$　　执行代价为2
  - $op\ R_i, [M]$　　执行代价为3（x86需要LEA和MOV两条指令，代价4）
- 可以计算：如果循环中把某个固定寄存器分配给该变量，执行代价能节省多少
  - 根据计算结果，把可用的几个寄存器，固定分配给节省执行代价多的几个变量

11.4 寄存器分配

- 固定分配寄存器，相对于原简单代码生成算法，节省的执行代价计算如下
  - 原代码生成算法中，仅当变量在基本块中被定值时，其值才存放在寄存器中。因此固定分配寄存器后，在该变量被定值前，每引用一次，就减少一次主存访问，执行代价就减少1。（如Add R0, M变为Add R0, Rx）
  - 原代码生成算法中，如果在基本块中被定值且在基本块出口之后是活跃的，那么出基本块时要把它存储到主存中。固定分配后，出基本块时无需再转存到主存，因此执行代价节省2。（Mov M, Rx的指令代价为2）

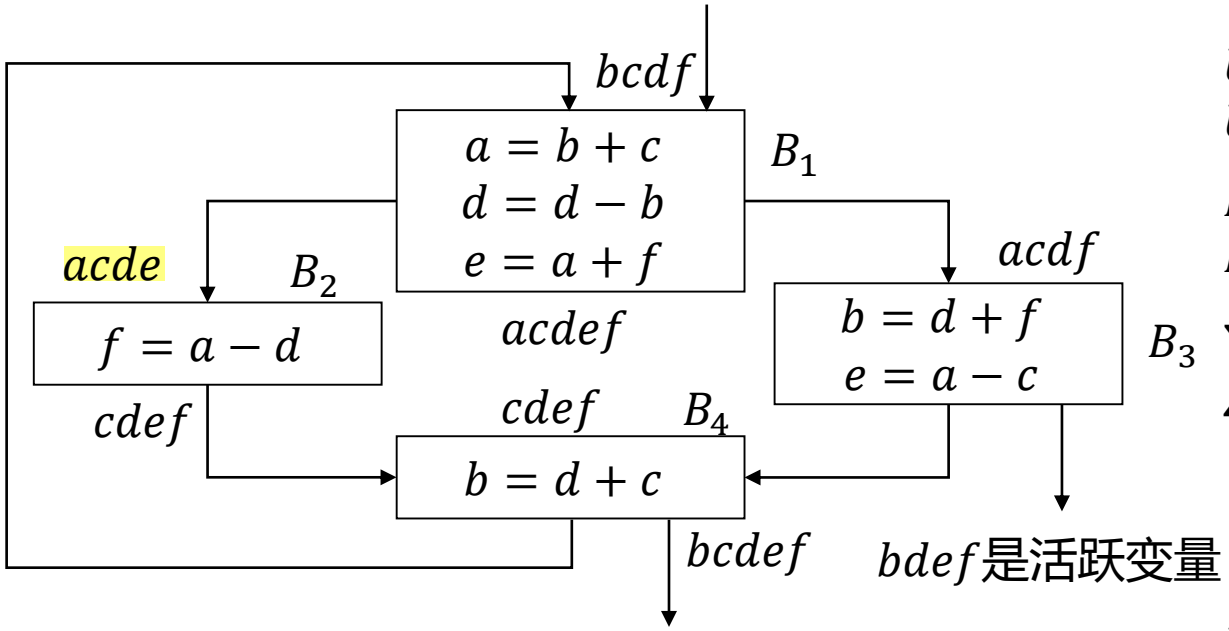
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

11.4 寄存器分配

- 循环L中代价节省公式：$\sum_{B\in L}[Use(M,B)+2Live(M,B)]$
  - $Use(M,B)$：基本块B中对M定值前引用M的次数。
  - $Live(M,B)=\begin{cases}1,\text{如果M在基本块B中被定值且在B的出口后是活跃的}\\0,\text{其它情况}\end{cases}$
- 忽略的因素
  - 如果M在循环入口前是活跃的，循环入口需要取到固定寄存器，执行代价加2；如果B是循环出口基本块，C是循环外B的后继基本块，如果C入口前M活跃，则出口时需要将M存入主存，执行代价加2；但这两处只执行一次，相对循环次数可以忽略。
  - 循环一次，各基本块不一定都执行到，该因素也忽略。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.4】某程序的最内层循环，假定R_0, R_1, R_2固定分配给3个变量使用。



$Use(a, B_1) = 0, Use(a, B_2) = 1$
$Use(a, B_3) = 1, Use(a, B_4) = 0$

$Live(a, B_1) = 1, Live(a, B_2) = 0$
$Live(a, B_3) = 0, Live(a, B_4) = 0$

$$\sum_a \ldots = 1 + 1 + 2 \times 1 = 4$$

$Use(b, B_1) = 2, Use(b, B_2) = 0$
$Use(b, B_3) = 0, Use(b, B_4) = 0$

$Live(b, B_1) = 0, Live(b, B_2) = 0$
$Live(b, B_3) = 1, Live(b, B_4) = 1$

$$\sum_b \ldots = 2 + 2 \times (1 + 1) = 6$$

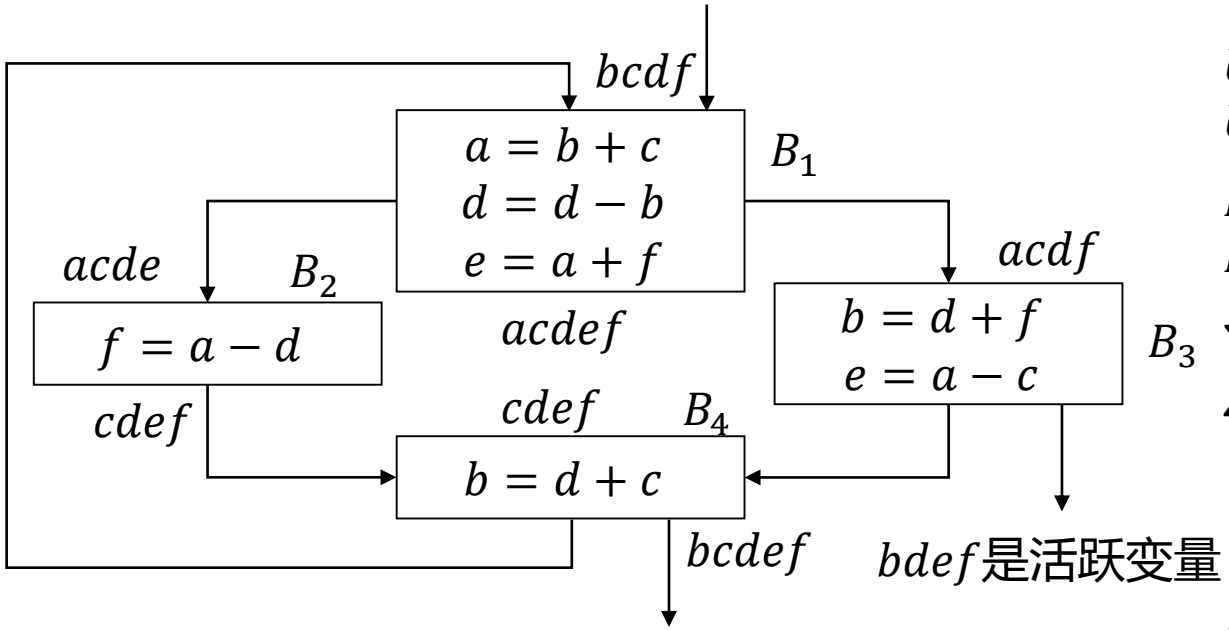
$Use(c, B_1) = 1, Use(c, B_2) = 0$
$Use(c, B_3) = 1, Use(c, B_4) = 1$

$Live(c, B_1) = 0, Live(c, B_2) = 0$
$Live(c, B_3) = 0, Live(c, B_4) = 0$

$$\sum_c \ldots = 1 + 1 + 1 + 2 \times 0 = 3$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.4】某程序的最内层循环，假定R_0, R_1, R_2固定分配给3个变量使用。



$Use(d, B_1) = 1, Use(d, B_2) = 1$
$Use(d, B_3) = 1, Use(d, B_4) = 1$

$Live(d, B_1) = 1, Live(d, B_2) = 0$
$Live(d, B_3) = 0, Live(d, B_4) = 0$

$\sum_d \ldots = 1 \times 4 + 2 \times 1 = 6$

$Use(f, B_1) = 1, Use(f, B_2) = 0$
$Use(f, B_3) = 1, Use(f, B_4) = 0$

$Live(f, B_1) = 0, Live(f, B_2) = 1$
$Live(f, B_3) = 0, Live(f, B_4) = 0$

$\sum_f \ldots = 1 + 1 + 2 \times 1 = 4$

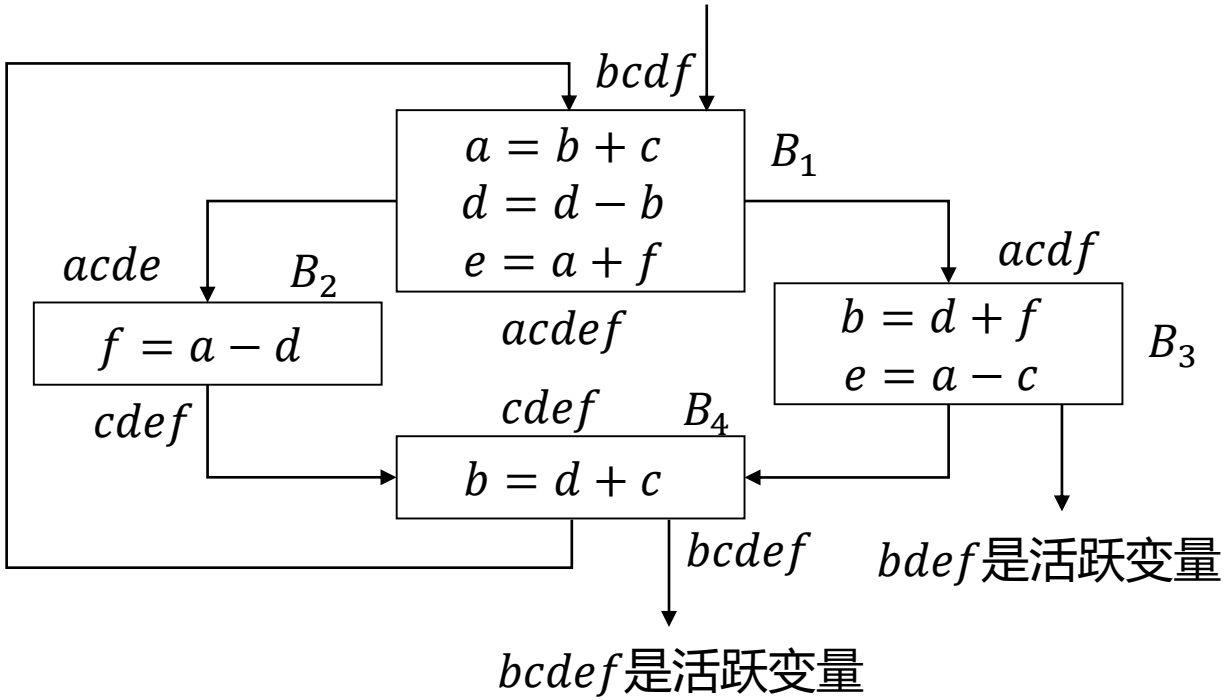
$Use(e, B_1) = 0, Use(e, B_2) = 0$
$Use(e, B_3) = 0, Use(e, B_4) = 0$

$Live(e, B_1) = 1, Live(e, B_2) = 0$
$Live(e, B_3) = 1, Live(e, B_4) = 0$

$\sum_e \ldots = 0 + 2 \times (1 + 1) = 4$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.4】某程序的最内层循环，假定R_0, R_1, R_2固定分配给3个变量使用。



$\sum_a \ldots = 4, \sum_b \ldots = 6, \sum_c \ldots = 3, \sum_d \ldots = 6, \sum_e \ldots = 4, \sum_f \ldots = 4$

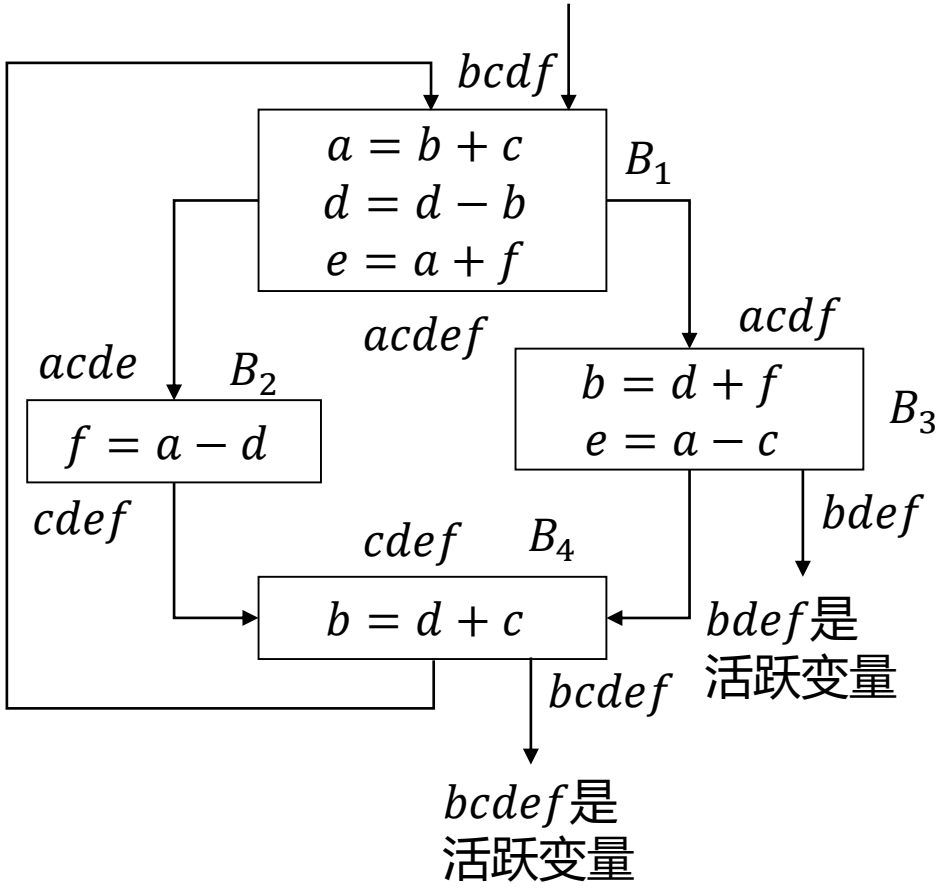
可以选择b, d，以及a, e, f中的一个。

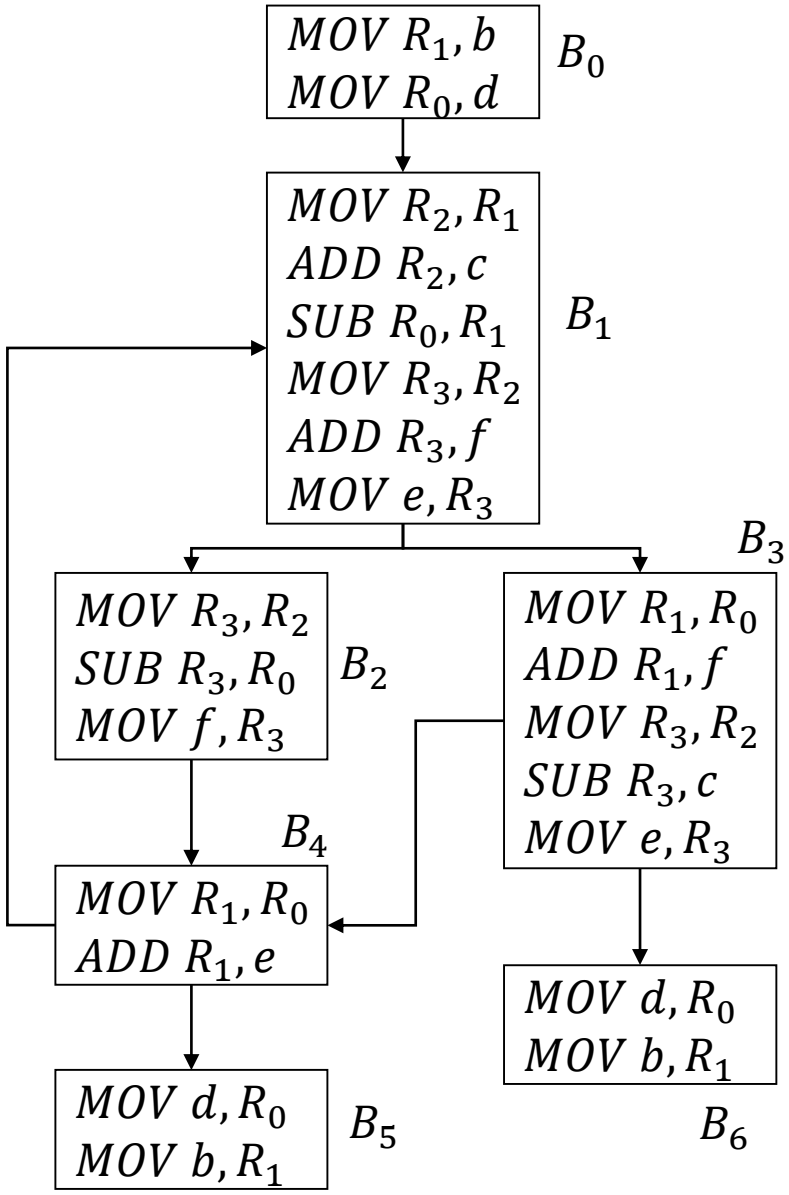
寄存器分配目标代码生成

- 循环中，如果涉及已固定分配寄存器的变量，则采用分配给的寄存器
  - 但对$A = B\ op\ C$，如果$A = C \neq B$，且寄存器R固定分配给A，但B的值不在R中，那么当$M \notin Avalue[C]$时，先生成目标代码$MOV\ C, R$，再认为C在主存中生成$A = B\ op\ C$的目标代码。
- 如果其中某变量在循环入口之前是活跃的，那么循环入口之前要生成把它们值分别取到相应寄存器的目标代码。
- 如果其中某变量在循环出口之后是活跃的，那么循环出口后面，要分别生成代码，把它们在寄存器中的值放入主存单元。
- 在循环中每个基本块的出口，对未固定分配到寄存器的变量，仍按以前算法生成目标代码，把它们在寄存器的值存入主存单元。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.5】R_0, R_1, R_2固定分配给d, b, a使用。





第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

基本思想

- 计算$A = B\ op\ C$时，如果计算完右边左对象B，紧接着计算A，就可以及时利用寄存器中的信息。
  - 考虑先算DAG的右子树，再算左子树，紧跟着左子树算父结点。

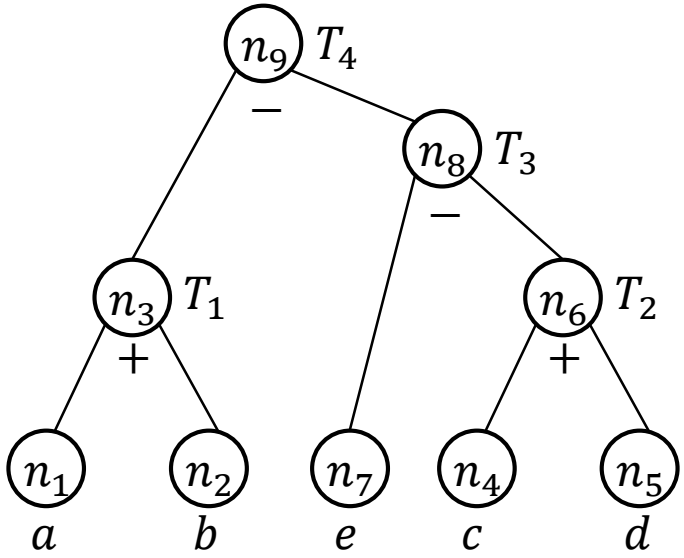
【例11.5】利用DAG调整语句顺序

$T_1 = a + b$

$T_2 = c + d$

$T_3 = e - T_2$

$T_4 = T_1 - T_3$



(1) $T_2 = c + d$
(2) $T_3 = e - T_2$
(3) $T_1 = a + b$
(4) $T_4 = T_1 - T_3$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

基本思想

- 原程序（假设2个寄存器）

$T_1 = a + b$

$T_2 = c + d$

$T_3 = e - T_2$

$T_4 = T_1 - T_3$

- 调序后（假设2个寄存器）

$T_2 = c + d$

$T_3 = e - T_2$

$T_1 = a + b$

$T_4 = T_1 - T_3$

(1) $MOV\ R_0, a$
(2) $ADD\ R_0, b$
(3) $MOV\ R_1, c$
(4) $ADD\ R_1, d$
(5) $MOV\ T_1, R_0$
(6) $MOV\ R_0, e$
(7) $SUB\ R_0, R_1$
(8) $MOV\ R_1, T_1$
(9) $SUB\ R_1, R_0$
(10) $MOV\ T_4, R_1$

(1) $MOV\ R_0, c$
(2) $ADD\ R_0, d$
(3) $MOV\ R_1, e$
(4) $SUB\ R_1, R_0$
(5) $MOV\ R_0, a$
(6) $ADD\ R_0, b$
(7) $SUB\ R_0, R_1$
(8) $MOV\ T_4, R_0$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

DAG结点计算顺序

- 设DAG有N个内部结点，线性表$T[N]$记录计算顺序，初始为空值。

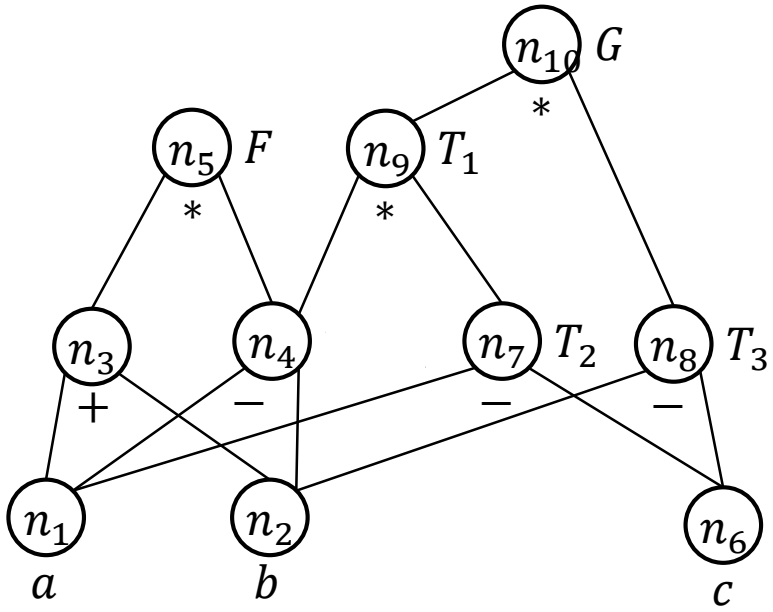
```
i = N;                  // 从后往前填充
while 存在未列入T的内部结点 {
    选取一个未列入T但其父结点均列入T，或者没有父结点的内部结点n;
    T[i--] = n;
    while n的最左子结点m不为叶结点，且其全部父结点均已列入T {
        T[i--] = m; n = m;
    }
}
```

- 最后的$T[1], T[2], \ldots, T[N]$即为结点计算顺序。
- 如果叶结点上有附加信息，可以先计算。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.6】利用DAG调整语句顺序

(1) $T_1 = a + b$
(2) $T_2 = a - b$
(3) $F = T_1 * T_2$
(4) $T_1 = a - b$
(5) $T_2 = a - c$
(6) $T_3 = b - c$
(7) $T_1 = T_1 * T_2$
(8) $G = T_1 * T_3$

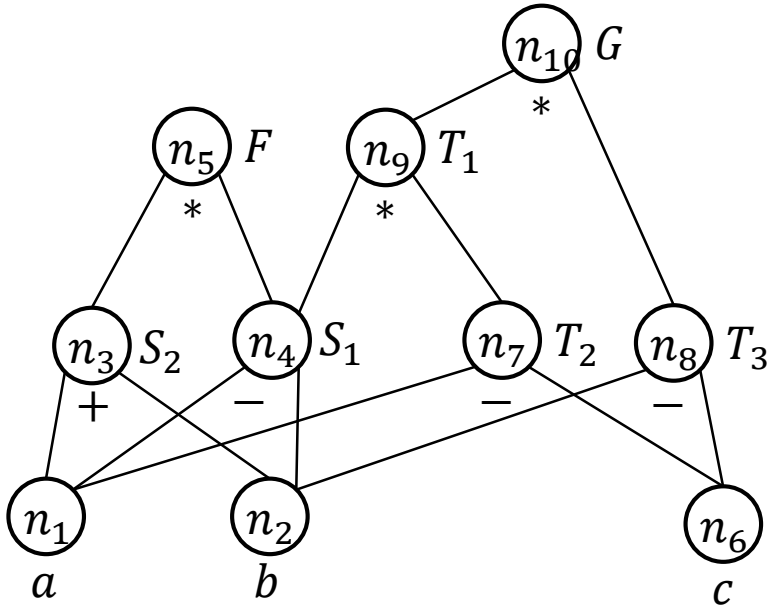


山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

【例11.6】利用DAG调整语句顺序

(1) $T_1 = a + b$
(2) $T_2 = a - b$
(3) $F = T_1 * T_2$
(4) $T_1 = a - b$
(5) $T_2 = a - c$
(6) $T_3 = b - c$
(7) $T_1 = T_1 * T_2$
(8) $G = T_1 * T_3$

1	2	3	4	5	6	7
n_8	n_7	n_4	n_9	n_{10}	n_3	n_5



(1) $T_3 = b - c$
(2) $T_2 = a - c$
(3) $S_1 = a - b$
(4) $T_1 = S_1 * T_2$
(5) $G = T_1 * T_3$
(6) $S_2 = a + b$
(7) $F = S_2 * S_1$

第十一章 目标代码生成

- 11.1 基本问题
- 11.2 目标机器模型
- 11.3 一个简单的代码生成器
  - 11.3.1 待用信息
  - 11.3.2 寄存器描述和地址描述
  - 11.3.3 代码生成算法
- 11.4 寄存器分配
- 11.5 DAG的目标代码
- 11.6 窥孔优化

11.6 窥孔优化

- 窥孔优化(peephole optimization)：通过考察一小段目标指令（称为窥孔），把这些指令替换为更短和更快的一段指令，从而提高目标代码质量。
  - 窥孔是目标程序中的一个可移动的小窗口。
  - 窥孔代码不一定是相邻的，尽管有的实现有这样的要求。
  - 窥孔优化的一个特点是，优化后所产生的结果可能会给后面的优化提供进一步的机会，为了最大优化效果，有时需对目标代码进行若干遍处理。
- 冗余存取

(1) $MOV\ A, R_0$

(2) $MOV\ R_0, A$

  - (2)可删除；
  - ✓ 如果(2)带有标号，则不能保证(2)一定紧接着(1)执行，此时不能删除；
  - ✓ 如果(1)(2)在同一个基本块，这种变换一定是安全的。

11.6 窥孔优化

不可达代码

- 无条件转移指令之后的无标号指令应该删除；
- 这种操作可以重复。

- c语言代码

$\#define\ debug\ 0$

……

$if\ (debug)$ {打印调试信息}

- 中间代码

$(j\ =, debug, 1, L_1)$

$(j, -, -, L_2)$

L_1: 打印调试信息

L_2: ……

- 初步优化后

$(j\ \neq, debug, 1, L_2)$

L_1: 打印调试信息

L_2: ……

- 现在条件$0 \neq 1$恒真，相当于

$(j, -, -, L_2)$

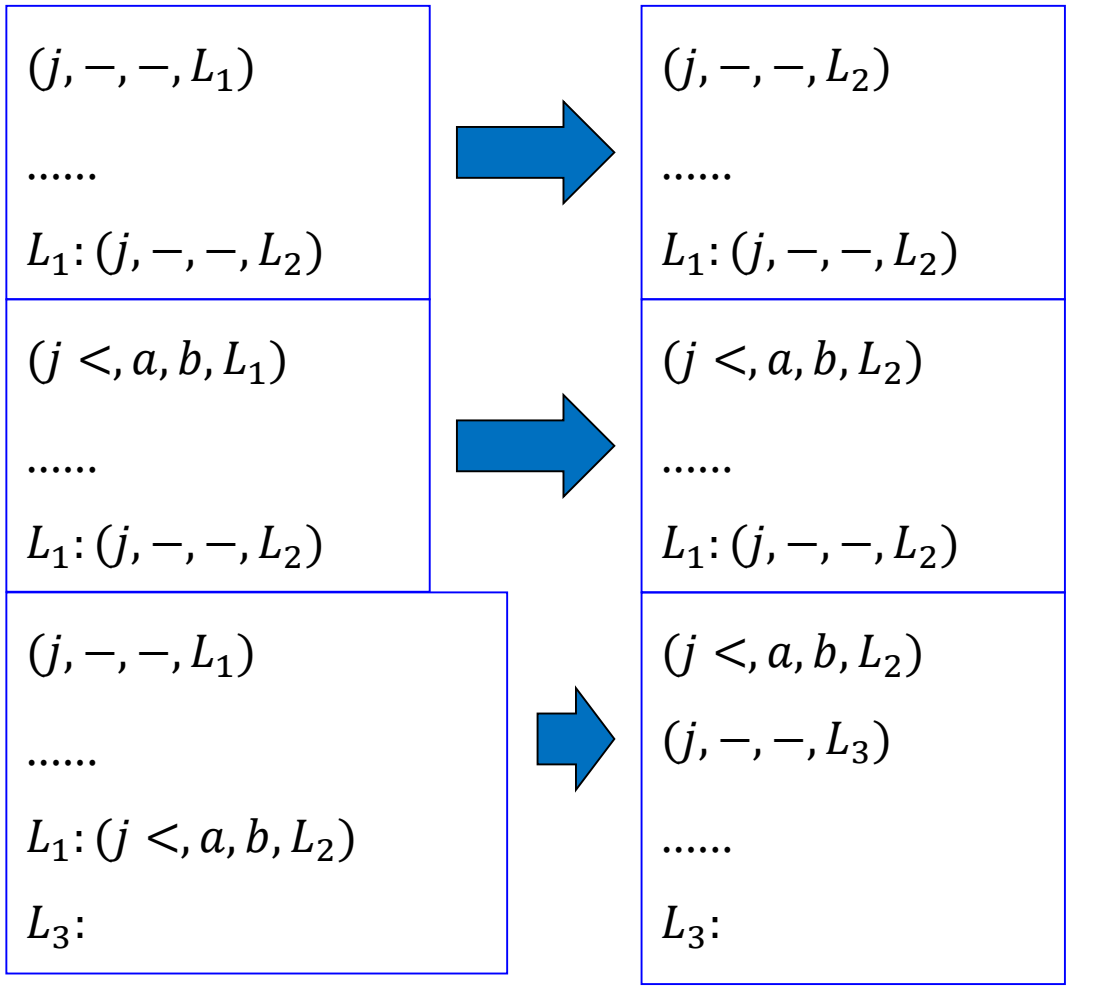
打印调试信息 // 可删

L_2: ……

11.6 窥孔优化

控制流优化

- 中间代码生成算法可能会产生不必要的连续跳转。



- 如果没有别的语句跳转到L_1，且L_1紧跟在一个无条件跳转语句之后，则可删除。
- 同上处理。
- 替换后指令条数相同，但后者可能跳过无条件跳转，而前者总要执行无条件跳转。

11.6 窥孔优化

- 强度削弱
  - $MUL\ R,\#2$ 可替换为：$\ll\ R,\#1$
  - $MUL\ R,\#4$ 可替换为：$\ll\ R,\#2$
- 删除无用操作
  - $ADD\ R,0$
  - $MUL\ R,\#1$

第十、十一章作业

【作业11-1】有如下基本块代码：

$$(1)T_1 = x + y \quad (2)T_2 = x - y \quad (3)u = T_1 * T_2 \quad (4)T_1 = x - y$$

$$(5)T_2 = x + y \quad (6)T_3 = x * y \quad (7)T_1 = T_2 * T_1 \quad (8)v = T_1 * T_3$$

（1）构造DAG图；

（2）写出优化后的代码；

（3）写出DAG目标代码优化后的中间代码；

（4）假设所有局部变量在基本块出口处都不活跃，所有非局部变量在基本块出口处都活跃；有两个寄存器R_0和R_1，写出目标代码。

山东大学
SHANDONG UNIVERSITY

第十一章 目标代码生成

The End

谢谢

授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn