代码生成器的主要任务

- 指令选择
  - 选择适当的目标机指令来实现中间表示(*IR*)语句
  - 例：
    - 三地址语句
      - $x = y + z$
    - 目标代码
      - *LD R0, y* /* 把*y*的值加载到寄存器*R0*中 */
      - *ADD R0, R0 , z* /* *z*加到*R0*上 */
      - *ST x , R0* /* 把*R0*的值保存到*x*中 */

代码生成器的主要任务

- 指令选择
  - 选择适当的目标机指令来实现中间表示(*IR*)语句
  - 例：

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
```

代码生成器的主要任务

- 指令选择
  - 选择适当的目标机指令来实现中间表示(*IR*)语句
- 寄存器分配和指派
  - 把哪个值放在哪个寄存器中
- 指令排序
  - 按照什么顺序来安排指令的执行

一个简单的目标机模型

- 三地址机器模型
  - 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, …, Rn-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 dst, addr*
  - *LD r, x*
  - *LD r_1, r_2*
- 保存指令　　*ST x, r*
- 运算指令　　*OP dst, src1, src2*
- 无条件跳转指令　*BR L*
- 条件跳转指令　　*Bcond r, L*
  - 例: *BLTZ r, L*

寻址模式

- 变量名*a*
  - 例：*LD R1 , a*
    - *R1 = contents (a)*

寻址模式

- 变量名*a*
- *a*(*r*)
  - *a*是一个变量，*r*是一个寄存器
  - 例：*LD R1 , a(R2)*
    - *R1 = contents (a + contents(R2))*

寻址模式

- 变量名*a*
- *a*(*r*)
- *c*(*r*)
  - *c*是一个整数
  - 例： *LD R1 , 100 (R2)*
    - *R1 = contents (contents(R2) + 100)*

寻址模式

- 变量名*a*
- *a*(*r*)
- *c*(*r*)
- **r*
  - 在寄存器*r*的内容所表示的位置上存放的内存位置
  - 例：*LD R1 , * R2*
    - *R1 = conents (contents (contents (R2)))*

寻址模式

- 变量名*a*
- *a*(*r*)
- *c*(*r*)
- **r*
- **c*(*r*)
  - 在寄存器*r*中内容加上*c*后所表示的位置上存放的内存位置
  - 例：*LD R1 , *100(R2)*
    - *R1 = conents (contents (contents(R2) + 100))*

寻址模式

- 变量名*a*
- *a*(*r*)
- *c*(*r*)
- **r*
- **c*(*r*)
- #*c*
  - 例：*LD R1，#100*
    - *R1 = 100*

运算语句的目标代码

- 三地址语句
  - $x = y - z$
- 目标代码
  - *LD R1 , y* // *R1 = y*
  - *LD R2 , z* // *R2 = z*
  - *SUB R1 , R1 , R2* // *R1 = R1 - R2*
  - *ST x , R1* // *x = R1*

尽可能避免使用上面的全部四个指令，如果
- ✓ 所需的运算分量已经在寄存器中了
- ✓ 运算结果不需要存放回内存

数组寻址语句的目标代码

- 三地址语句
  - $b = a[i]$
  - a是一个实数数组，每个实数占8个字节
- 目标代码
  - *LD R1 , i // R1 = i*
  - *MUL R1 , R1, 8 // R1=R1 * 8*
  - *LD R2 , a(R1) // R2=contents (a + contents(R1))*
  - *ST b , R2 // b = R2*

数组寻址语句的目标代码

- 三地址语句
  - $a[j] = c$
  - a是一个实数数组，每个实数占8个字节
- 目标代码
  - *LD R1 , c* // *R1 = c*
  - *LD R2 , j* // *R2 = j*
  - *MUL R2 , R2 , 8* // *R2 = R2 * 8*
  - *ST a(R2) , R1* // *contents(a+contents(R2))=R1*

指针存取语句的目标代码

- 三地址语句
  - *x = *p*
- 目标代码
  - *LD R1, p* // *R1 = p*
  - *LD R2, 0 (R1)* // *R2 = contents (0 + contents (R1))*
  - *ST x , R2* // *x = R2*

指针存取语句的目标代码

- 三地址语句
  - *p = y
- 目标代码
  - *LD R1, p // R1 = p*
  - *LD R2, y // R2 = y*
  - *ST 0(R1), R2 //contents (0 + contents (R1)) = R2*

条件跳转语句的目标代码

- 三地址语句
  - *if x < y goto L*
- 目标代码
  - *LD　　R1 , x　　// R1 = x*
  - *LD　　R2 , y　　// R2 = y*
  - *SUB　R1 , R1 , R2　　// R1=R1 - R2*
  - *BLTZ　R1 , M　　// if R1 < 0 jump to M*

*M*是标号为*L*的三地址指令所产生的目标代码中的第一个指令的标号

过程调用和返回的目标代码

静态存储分配

- 三地址语句
  - *call callee*
- 目标代码
  - *ST callee.staticArea, #here + 20*
  - *BR callee.codeArea*

*callee*的活动记录在静态区中的起始位置

*callee*的目标代码在代码区中的起始位置

- 三地址语句
  - *return*
- 目标代码
  - *BR *callee.staticArea*

过程调用和返回的目标代码

栈式存储分配

- 三地址语句
  - *call callee*
- 目标代码
  - *ADD SP, SP, #caller.recordsize*
  - *ST 0(SP), #here + 16*
  - *BR callee.codeArea*

- 三地址语句
  - *return*
- 目标代码
  - 被调用过程
    - *BR *0(SP)*
  - 调用过程
    - *SUB SP, SP, #caller.recordsize*

三地址语句的目标代码生成

- 对每个形如$x = y\ \mathrm{op}\ z$的三地址指令I，执行如下动作
  - 调用函数$getreg(I)$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的内存位置之一
  - 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“$OP\ R_x, R_y, R_z$”

寄存器描述符和地址描述符

- 寄存器描述符 (*register descriptor*)
  - 记录每个寄存器当前存放的是哪些变量的值
- 地址描述符 (*address descriptor*)
  - 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变量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*

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*”
  - 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

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*”
  - 对于指令“*OP* R_x, R_y, R_z”
  - 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

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*”
  - 对于指令“*OP* R_x, R_y, R_z”
  - 对于指令“*ST x , R*”
  - 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

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

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

R1	R2	R3
a	t	

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

例

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

```
LD   R3 , c
SUB R1 , R1 , R3
```

R1	R2	R3
u	t	c

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

例

$t = a - b$

$u = a - c$

$v = t + u$ → `ADD R3 , R2 , R1`

$a = d$

$d = v + u$

R1	R2	R3
u	t	v

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

例

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$ → LD R2 , d

$d = v + u$

R1	R2	R3
u	d , a	v

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

例

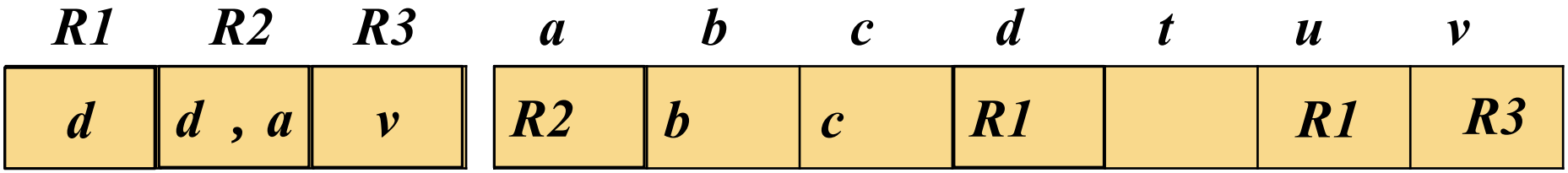
$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$ → `ADD R1 , R3 , R1`



例

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

exit

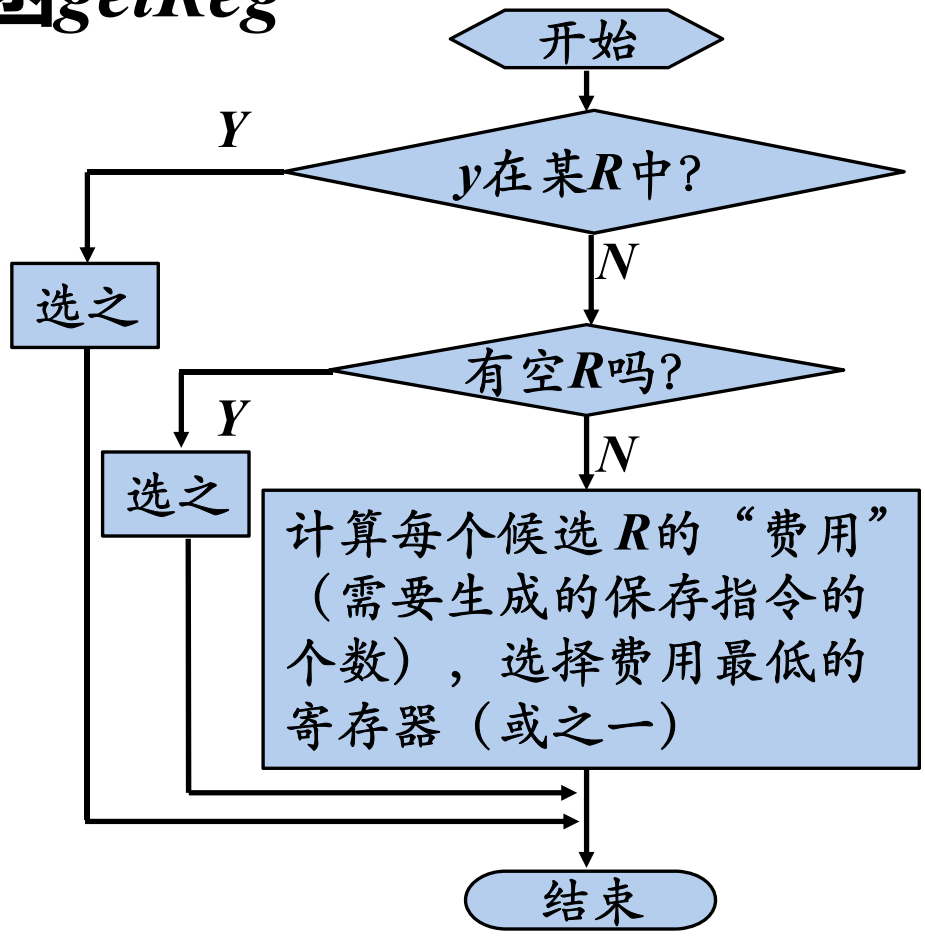
```
ST   a , R2
ST   d , R1
```

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

寄存器选择函*getReg*

$x = y$ op z

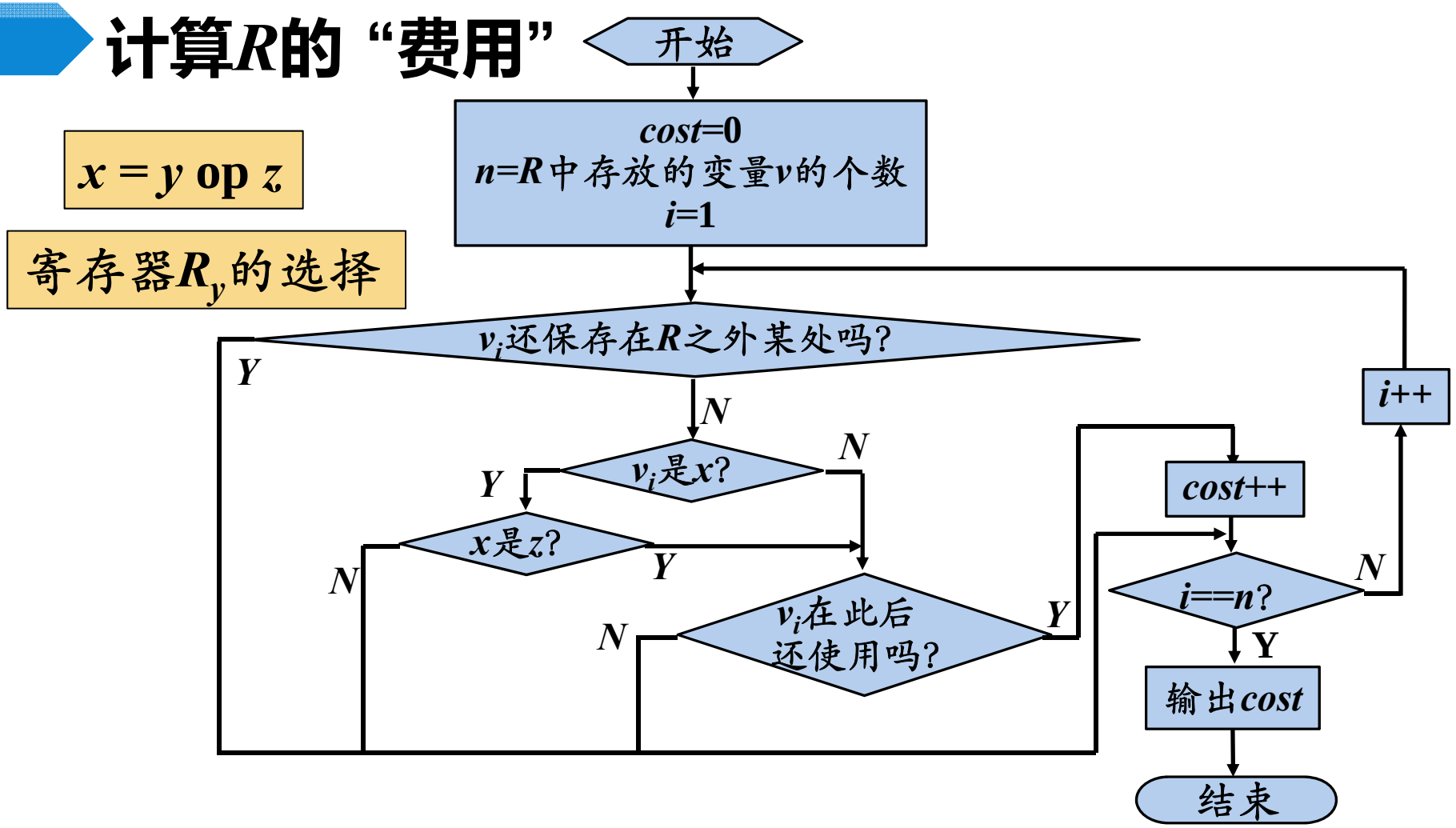
寄存器R_y的选择



计算*R*的“费用”

$x = y$ op z

寄存器R_y的选择



寄存器R_x的选择

$x = y$ op z

- 选择方法与R_y类似，区别之处在于
  - 因为x的一个新值正在被计算，因此只存放了x的值的寄存器对R_x来说总是可接受的，即使x就是y或z之一(因为我们的机器指令允许一个指令中的两个寄存器相同)
  - 如果y在指令I之后不再使用，且(在必要时加载y之后)R_y仅仅保存了y的值，那么，R_y同时也可以用作R_x。对z和R_z也有类似选择

当I是复制指令x=y时，选择好R_y后，令$R_x = R_y$

窥孔优化

- 窥孔(*peephole*)是程序上的一个小的滑动窗口
- 窥孔优化是指在优化的时候，检查目标指令的一个滑动窗口(即窥孔)，并且只要有可能就在窥孔内用更快或更短的指令来替换窗口中的指令序列
- 也可以在中间代码生成之后直接应用窥孔优化来提高中间表示形式的质量

具有窥孔优化特点的程序变换的例子

- 冗余指令删除
- 控制流优化
- 代数优化
- 机器特有指令的使用

冗余指令删除

- 消除冗余的加载和保存指令
  - 例

三地址指令序列

- *a=b+c*
- *d=a+e*

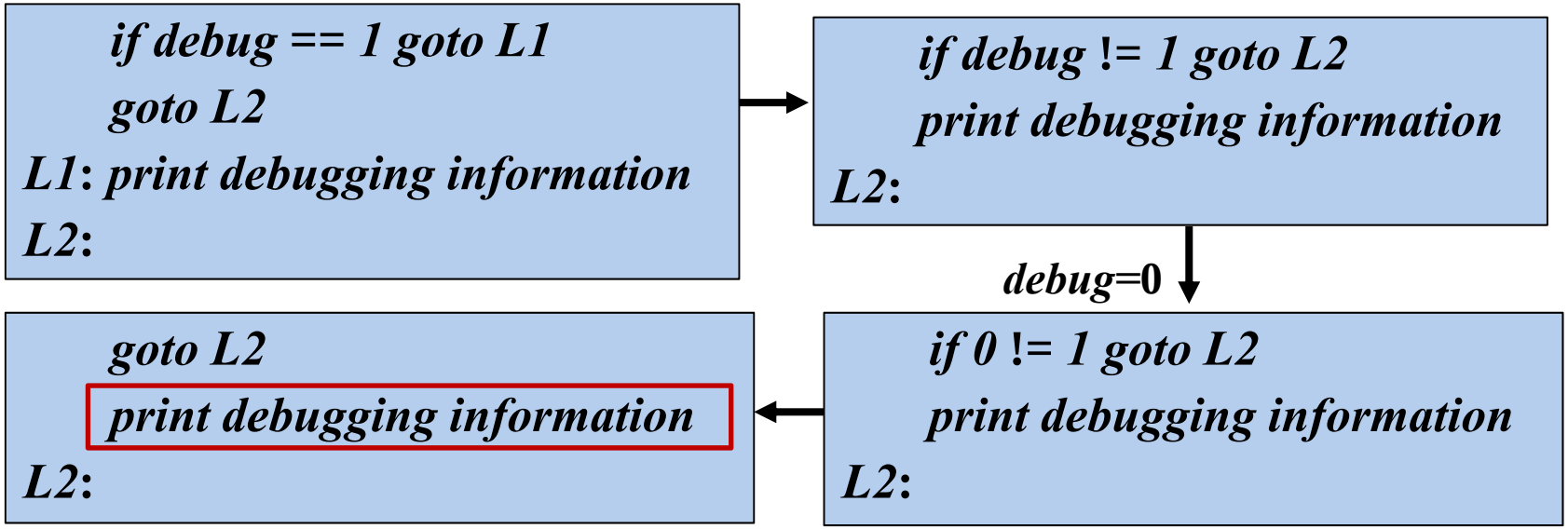
目标代码

- *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*

如果第四条指令有标号，则不可以删除

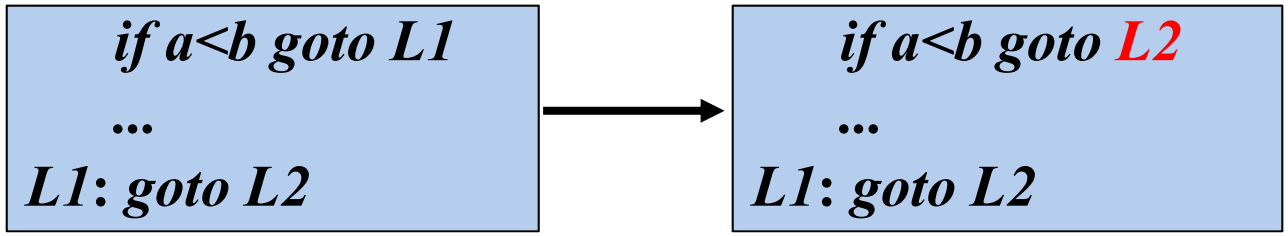
冗余指令删除

- 消除冗余的加载和保存指令
- 消除不可达代码
  - 一个紧跟在无条件跳转之后的不带标号的指令可以被删除
    - 例



控制流优化

- 在代码中出现跳转到跳转指令的指令时，某些条件下可以使用一个跳转指令来代替
  - 例



如果不再有跳转到*L1*的指令，并且语句*L1: goto L2*之前是一个无条件跳转指令，则可以删除该语句

代数优化

- 代数恒等式
  - 消除窥孔中类似于$x=x+0$或$x=x*1$的运算指令
- 强度削弱
  - 对于乘数(除数)是2的幂的定点数乘法(除法)，用移位运算实现代价比较低
  - 除数为常量的浮点数除法可以通过乘数为该常量倒数的乘法来求近似值

特殊指令的使用

- 充分利用目标系统的某些高效的特殊指令来提高代码效率
  - 例如：*INC*指令可以用来替代加1的操作