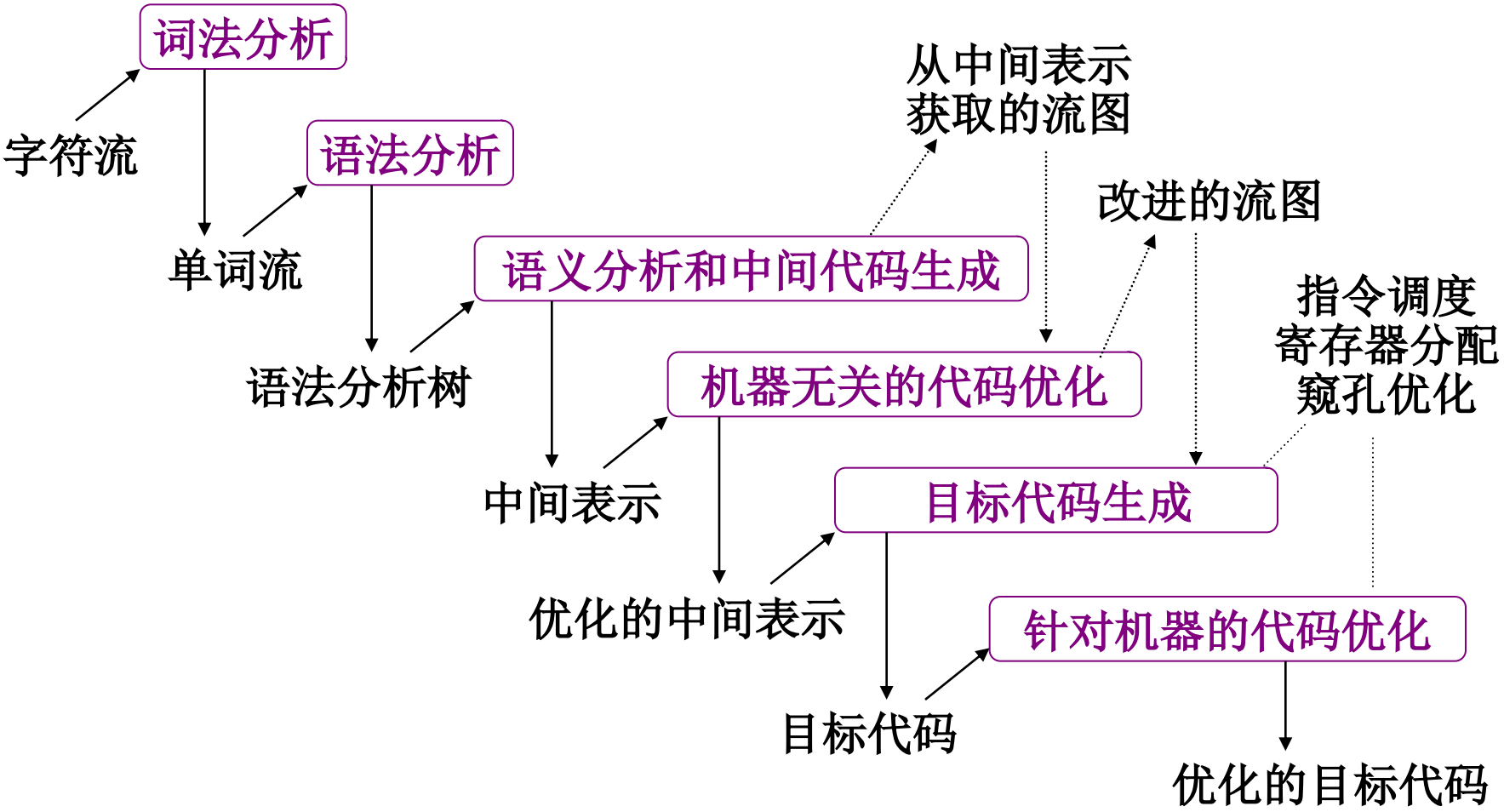
第十章　代码优化和目标代码生成

代码优化及目标代码生成基础

- ✧二者在编译程序中的逻辑位置
- ✧基本块、流图和循环
- ✧数据流分析基础
- ✧基本块的 *DAG* 表示（局部优化技术）
- ✧目标代码生成技术
- ✧代码优化技术

目标代码生成及代码优化基础

✧二者在编译程序中的逻辑位置



10.1 基本块、流图和循环

✧10.1.1 基本块（*basic block*）

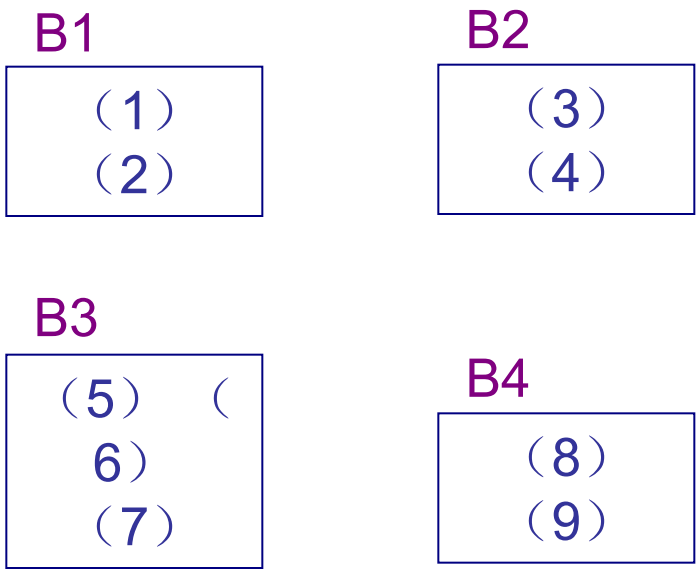
- 概念
  - 程序中一个顺序执行的语句序列
  - 只有一个入口语句和一个出口语句
  - 除入口语句外其他语句均不可以带标号
  - 除出口语句外其他语句均不可能是转移或停语句
- 入口语句
  - 程序的第一个语句；或者
  - 条件转移语句或无条件转移语句的转移目标语句；或者
  - 紧跟在条件转移语句后面的语句

划分基本块的算法

- 步骤
  - 求出 TAC 程序之中各个基本块的入口语句
  - 对每一入口语句，构造其所属的基本块。它是由该语句到下一入口语句（不包括下一入口语句），或到一转移语句（包括该转移语句），或到一停语句（包括该停语句）之间的语句序列组成的
  - 凡未被纳入某一基本块的语句，都是程序中控制流程无法到达的语句，因而也是不会被执行到的语句，可以把它们删除

划分基本块举例

– **举例** 右边程序可划分成 4 个基本块




```
*(1)  read  x
 (2)  read y
*(3)  r:=x mod y
 (4)  if r=0 goto (8)
*(5)  x:=y
 (6)  y:=r
 (7)  goto(3)
*(8)  write y
 (9)  halt
```

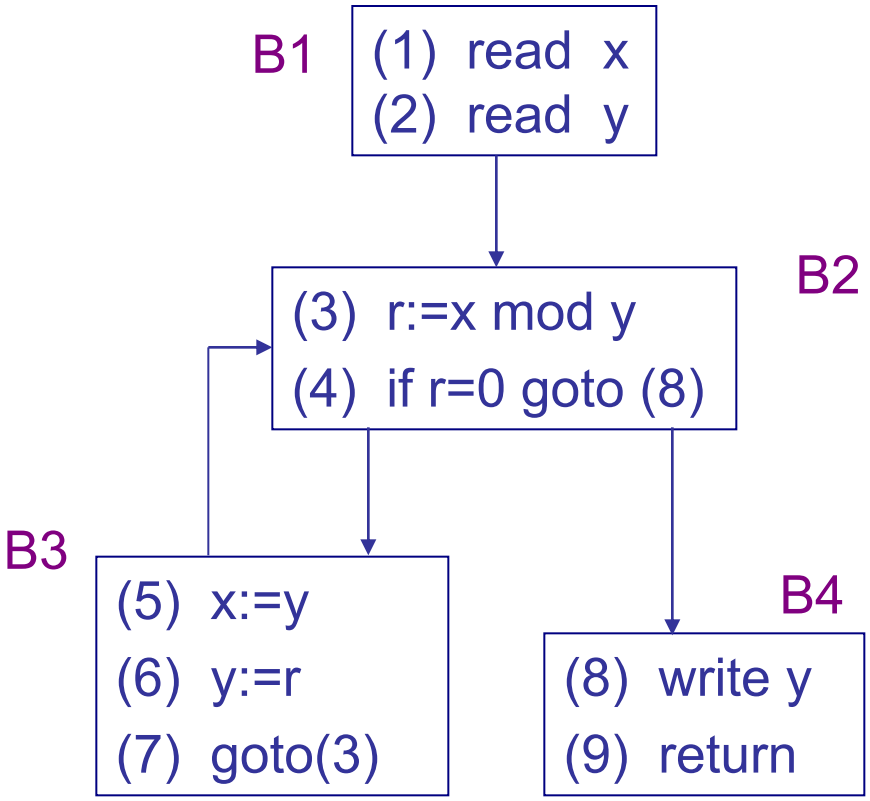
10.1.2 流图（*flow graph*）

- **概念** 可以为构成程序的基本块增加控制流信息，方法是构造一个有向图，称之为流图或控制流图（*CFG，Control-Flow Graph*）

  流图以基本块集为结点集；第一个结点为含有程序第一条语句的基本块；从基本块 *i* 到基本块 *j* 之间存在有向边，当且仅当

  - 基本块 *j* 在程序的位置紧跟在 *i* 后,且 *i* 的出口语句不是转移 *(*可为条件转移*)*语句、停语句或者返回语句；或者
  - *i* 的出口是 *goto(S)* 或 *if goto(S)*, 而 *(S)* 是 *j* 的入口语句

流图举例



*(1) read x
(2) read y
*(3) r:=x mod y
(4) if r=0 goto (8)
*(5) x:=y
(6) y:=r
(7) goto(3)
*(8) write y
(9) return

10.1.3 循环（*loop*）

- **支配结点集**（*dominators*）

  如果从流图的首结点出发,到达 *n* 的任意通路都要经过 *m*，则称 *m* 支配 *n*, 或 *m* 是 *n* 的支配结点，记为 *m DOM n*
  (∀*a*. *a DOM a*)

  结点*n* 的所有支配结点的集合, 称为结点 *n* 的支配结点集, 记为*D(n)*.

支配结点集举例

D(1)={1}

D(2)={1, 2}

D(3)={1, 2, 3}

D(4)={1, 2, 4}

D(5)={1, 2, 4, 5}

D(6)={1, 2, 4, 6}

D(7)={1, 2, 4, 7}

自然循环

假设 $n \rightarrow d$ 是流图中的一条有向边，如果 *d DOM n* 则称 $n \rightarrow d$ 是流图中的一条回边（*back edge*）

有向边 $n \rightarrow d$ 是回边，它对应的自然循环是由结点 *d*，结点 *n* 以及有通路到达 *n* 而该通路不经过 *d* 的所有结点组成，并且 *d* 是该循环的唯一入口结点

同时，因 *d* 是 *n* 的支配结点，所以 *d* 必可达该循环中任意结点

注：流图中的任何结点都是从首结点可达的

自然循环举例

有向边 $n \to d$ 是回边，对应的自然循环是由结点 d，结点 n 以及有通路到达 n 而该通路不经过 d 的所有结点组成，并且 d 是该循环的唯一入口结点

对应回边 $6 \to 6$：

{ 6 }

对应回边 $7 \to 4$：

{ 4，5，6，7 }

对应回边 $4 \to 2$：

{ 2，3，4，5，6，7}

10.3 代码优化技术

- 依优化范围划分的代码优化技术
  - 窥孔优化（*peephole optimization*）

    局部的几条指令范围内的优化
  - 局部优化

    基本块范围内的优化
  - 全局优化

    流图范围内的优化
  - 过程间优化

    整个程序范围内的优化

- 依优化对象划分
  - 目标代码优化

    面向目标代码
  - 中间代码优化

    面向程序的中间表示
  - 源级优化

    面向源程序

10.3.1窥孔优化（*peephole optimization*）

- **工作方式** 在目标指令序列上滑动一个包含几条指令的窗口（称为窥孔），发现其中不够优化的指令序列，用一段更短或更有效的指令序列来替代它，使整个代码得到改进

1. 删除冗余的“取”和“存”

– 举例

指令序列

（1） MOV R0，a
（2） MOV a，R0

可优化为

（1） MOV R0，a

2. 合并已知量

- 举例

  代码序列

  （1） r2:=3*2

  可优化为

  （1） r2:=6

3. 常量传播

– 举例

代码序列

（1） r2:=4
（2） r3:=r1+r2

可优化为

（1） r2:=4
（2） r3:=r1+ 4

注：虽然条数未少，但若是知道r2不再活跃时，可删除（1）

4. 代数化简

- 举例

  代码序列

  （1） x:=x+0
  （2） ……
  （n） y:=y*1

  中的（1），（n）可在窥孔优化时删除

5. 控制流优化

– 举例

代码序列

```
        goto L1
        ……
L1:     goto L2
```

可替换为

```
        goto L2
        ……
L1:     goto L2
```

6. 死代码删除

- 举例

  代码序列

```
debug := false
if (debug) print …
……
```

  可替换为

```
debug := false
……
```

7. 强度削弱

- 举例

  x:=2.0*f　可替换为　x:=f+f

  x:=f/ 2.0 可替换为　x:=f*0.5

8. 使用目标机惯用指令

- 举例

  某个操作数与*1* 相加，通常用“加*1*”指令，而不是用“加”指令

  某个定点数乘以*2*，可以采用“左移”指令；而除以*2*，则可以采用“右移”指令

  …

10.3.2 局部优化

- 在一个基本块内的优化
  - 常量传播
  - 常量合并
  - 删除公共子表达式
  - 复写传播
  - 删除无用赋值
  - 代数化简

基本块的 DAG 表示

- *DAG*

  DAG 指有向无圈图（*Directed Acyclic Graph*）

- 基本块的 *DAG* 是在结点上带有标记的 *DAG*

  叶结点 代表名字的初值，以唯一的标识符（变量名字或常数）标记（为避免混乱，用x_0表示变量名字x的初值）

  内部结点 用运算符号标记

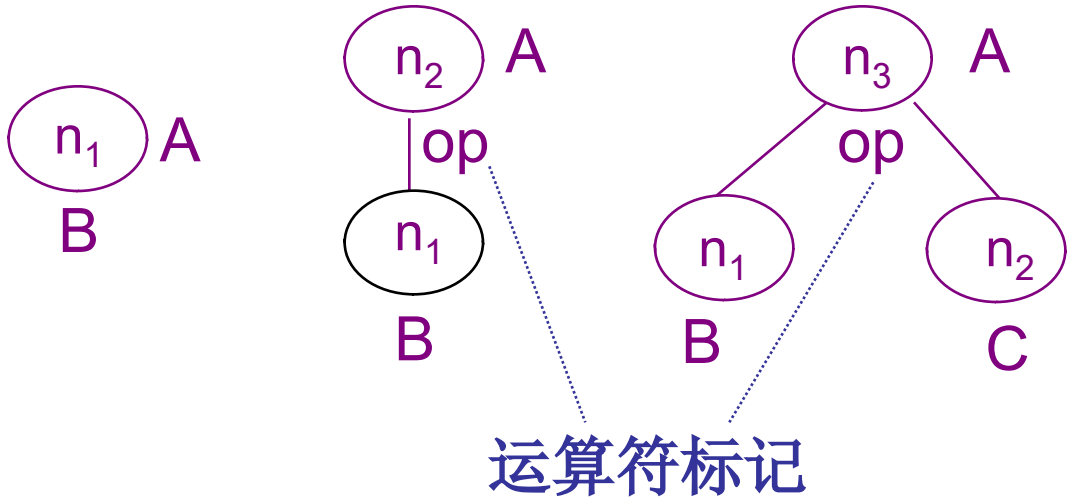
  所有结点都可有一个附加的变量名字表

基本块 *DAG* 表示的构造

– 仅考虑三种形式的TAC语句

*TAC*语句　　*A*:=*B*　　*A*:=op *B*　　*A*:= *B* op *C*

DAG 子图



基本块 *DAG* 表示的构造

设 *x:=y op z*, *x:=op y*, *x:=y* 分别为第1、2、3种 *TAC* 语句

设函数 *node(name)* 返回最近创建的关联于 *name* 的结点

DAG 为空. 对基本块的每一 *TAC* 语句，依次进行下列步骤：

对第 1 种语句 *x:=y op z* ：

若无*node(y)* 建叶结点*node(y)* ；无**node(z)**建叶结点**node(z)**。

若 **node(y)** 和 **node(z)**都是常数叶结点，执行 **y op z**得到新常数为**p**. 若 **node(p)**无定义，则构造一个用 **p** 做标记的叶结点 **n**. 若 **node(y)**或 **node(z)**是处理当前语句时新构造出来的结点，则删除它. 置**node(p)=n**. （合并已知量）

若 **node(y)** 或 **node(z)**不是常数叶结点，检查是否存在某个标记为 **op** 的结点，其左孩子是 **node(y)** ，右孩子是**node(z)** ？若无，则创建这样的结点. 令该结点为 **n**. （删除多余运算）

基本块 DAG 表示的构造

对于第 2 种语句 *x:=op y* ：

若 node(y) 是常数叶结点，执行op y得到新常数p. 若node(p)无定义，构造p 做标记的叶结点n. 若node(y)是处理当前语句时新构造出来的结点，删除它. 置node(p)=n. （合并已知量）

若 node(y) 不是常数叶结点，则检查是否存在某个标记为 op 的结点，其唯一的孩子是 node(y)？若无，则创建这样的结点. 无论有无，都令该结点为 n.（删除多余运算）

对于第 3 种语句 *x:= y*

令 node(y) 为 n；从 node(x)的附加标识符表中将 x 删除，将x添加到结点 n 的附加变量名字表中.（删除无用赋值）

✧基本块 *DAG* 表示的构造

- 举例

T0:=3.14 ⇦

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例

T0:=3.14

T1:=2*T0 ⇦

T2:=R+r

A:=T1*T2

B:=A

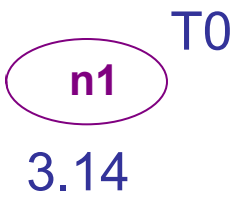
T3:=2*T0

T4:=R+r

T5:=T3*T4

T6:=R-r

B:=T5*T6



基本块的 DAG 表示

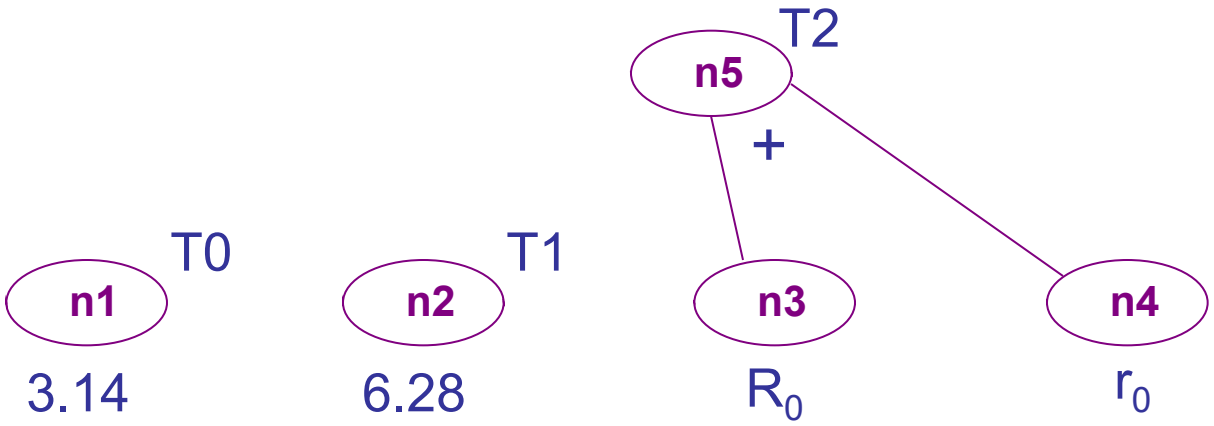
✧基本块 DAG 表示的构造

- 举例

T0:=3.14
T1:=2*T0
T2:=R+r
A:=T1*T2
B:=A
T3:=2*T0
T4:=R+r
T5:=T3*T4
T6:=R-r
B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

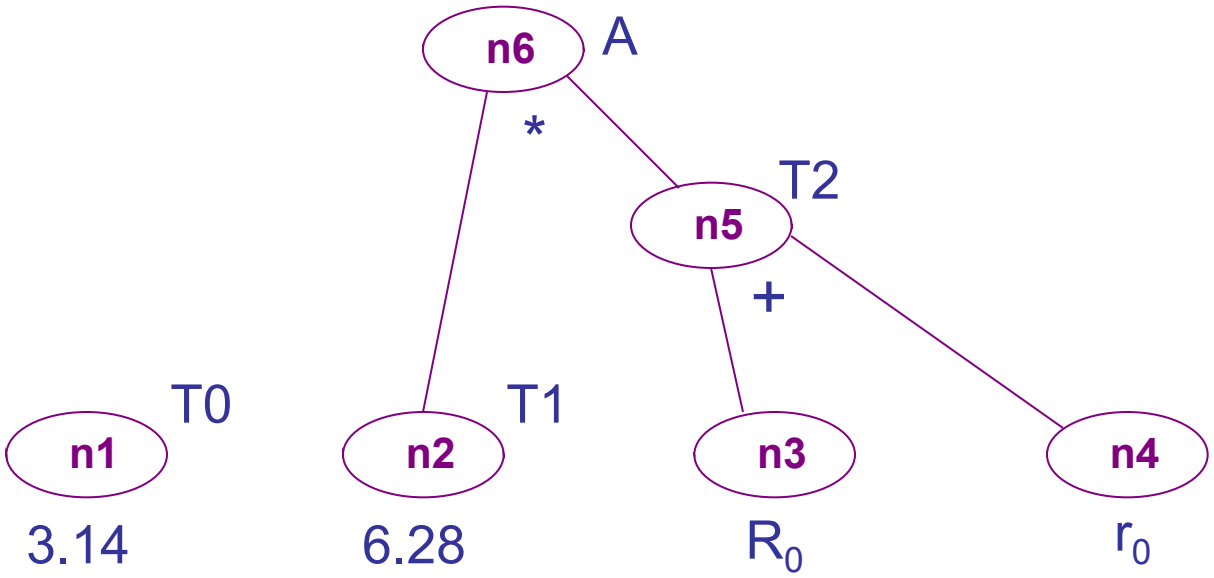
T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

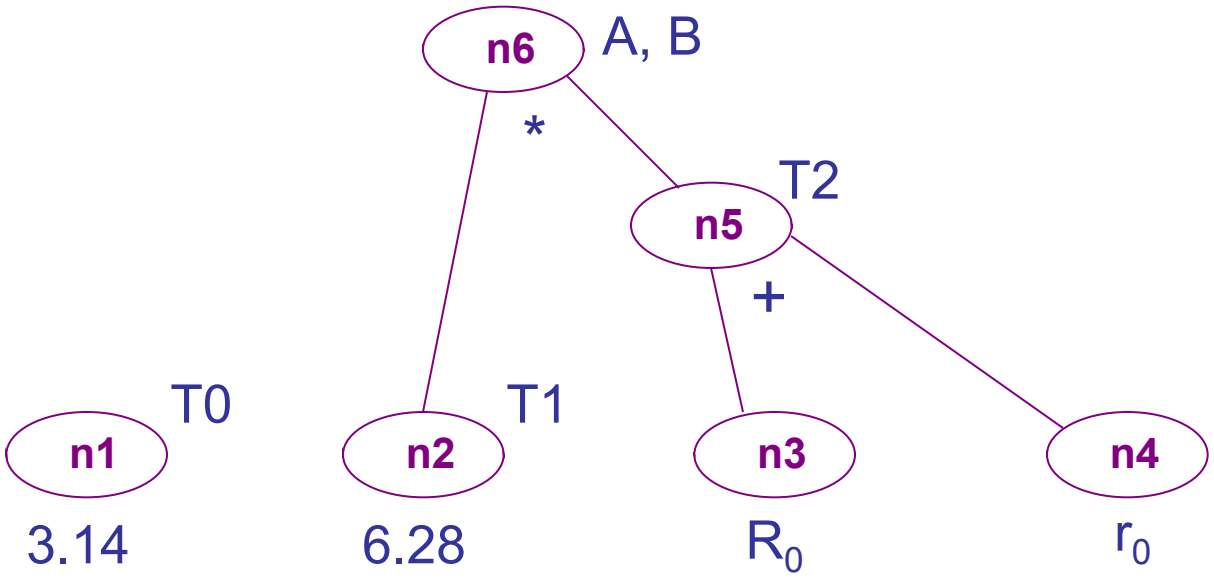
T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

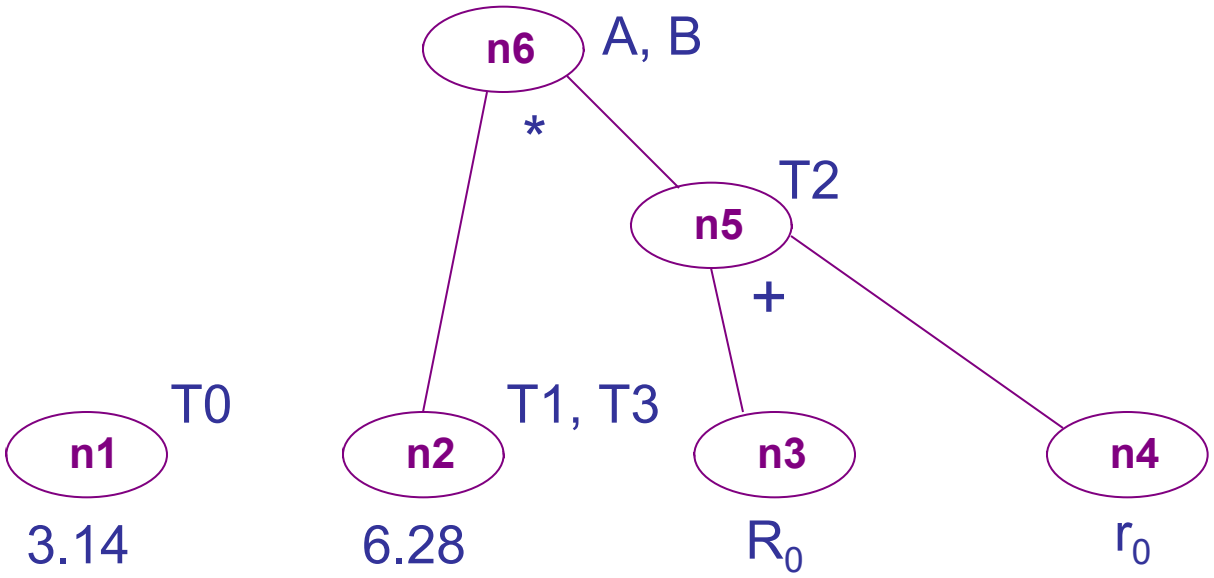
– 举例



T0:=3.14
T1:=2*T0
T2:=R+r
A:=T1*T2
B:=A
T3:=2*T0
T4:=R+r
T5:=T3*T4
T6:=R-r
B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r ⇦

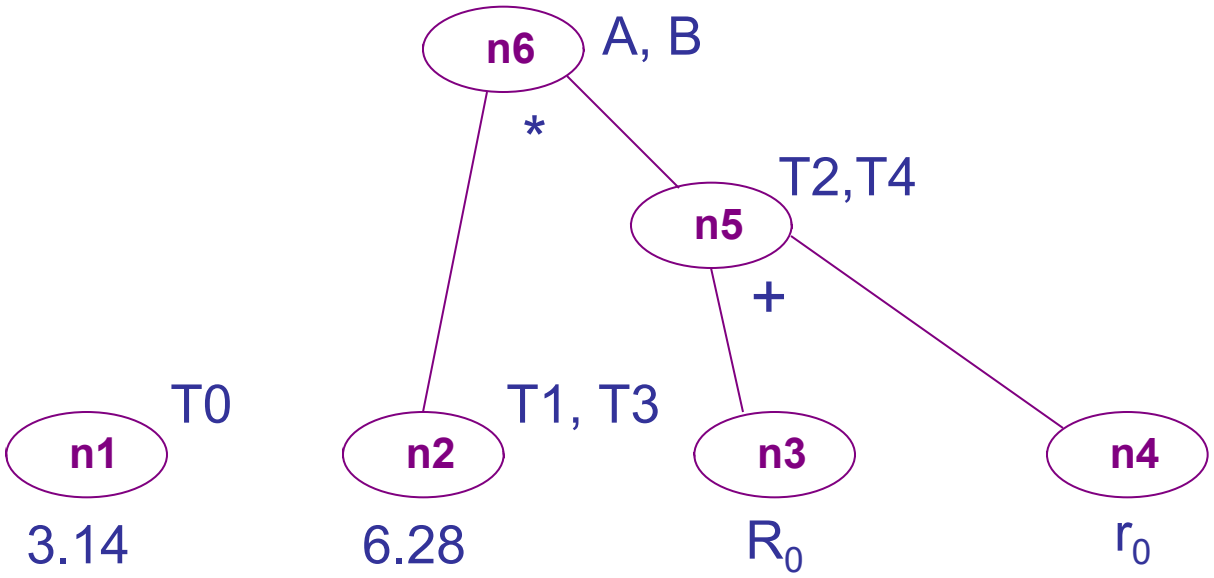
T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

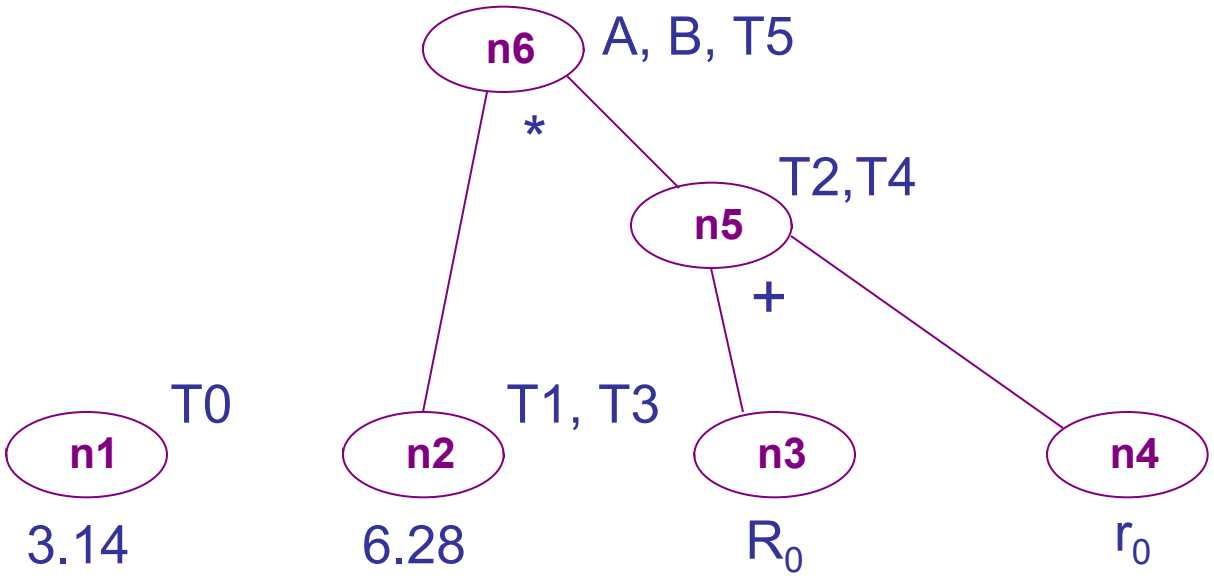
T5:=T3*T4 ⇦

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

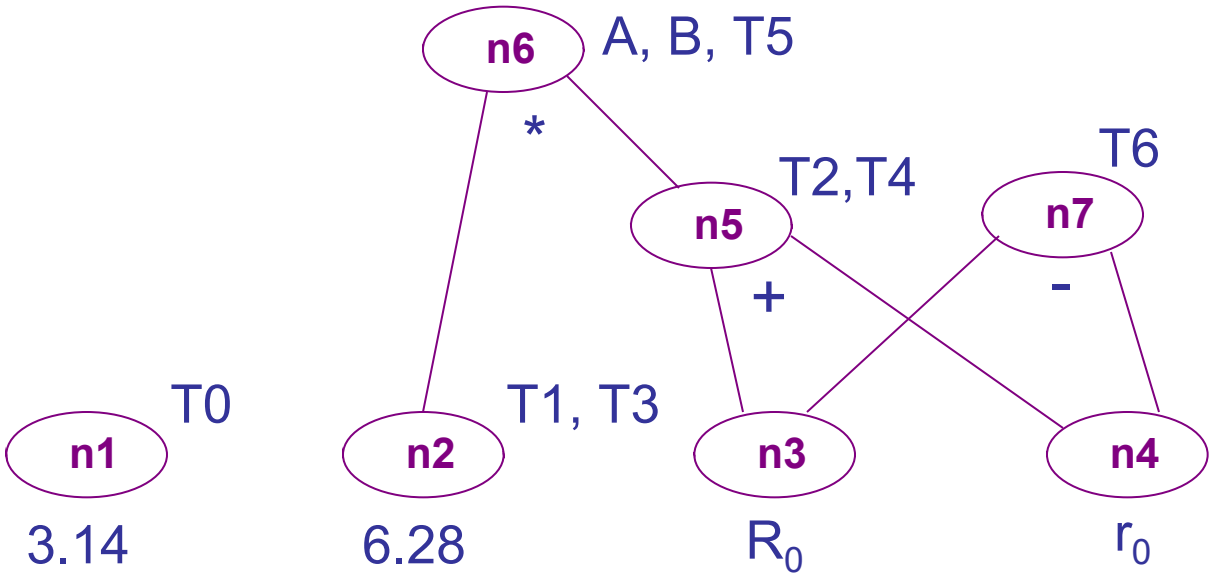
T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

– 举例



T0:=3.14

T1:=2*T0

T2:=R+r

A:=T1*T2

B:=A

T3:=2*T0

T4:=R+r

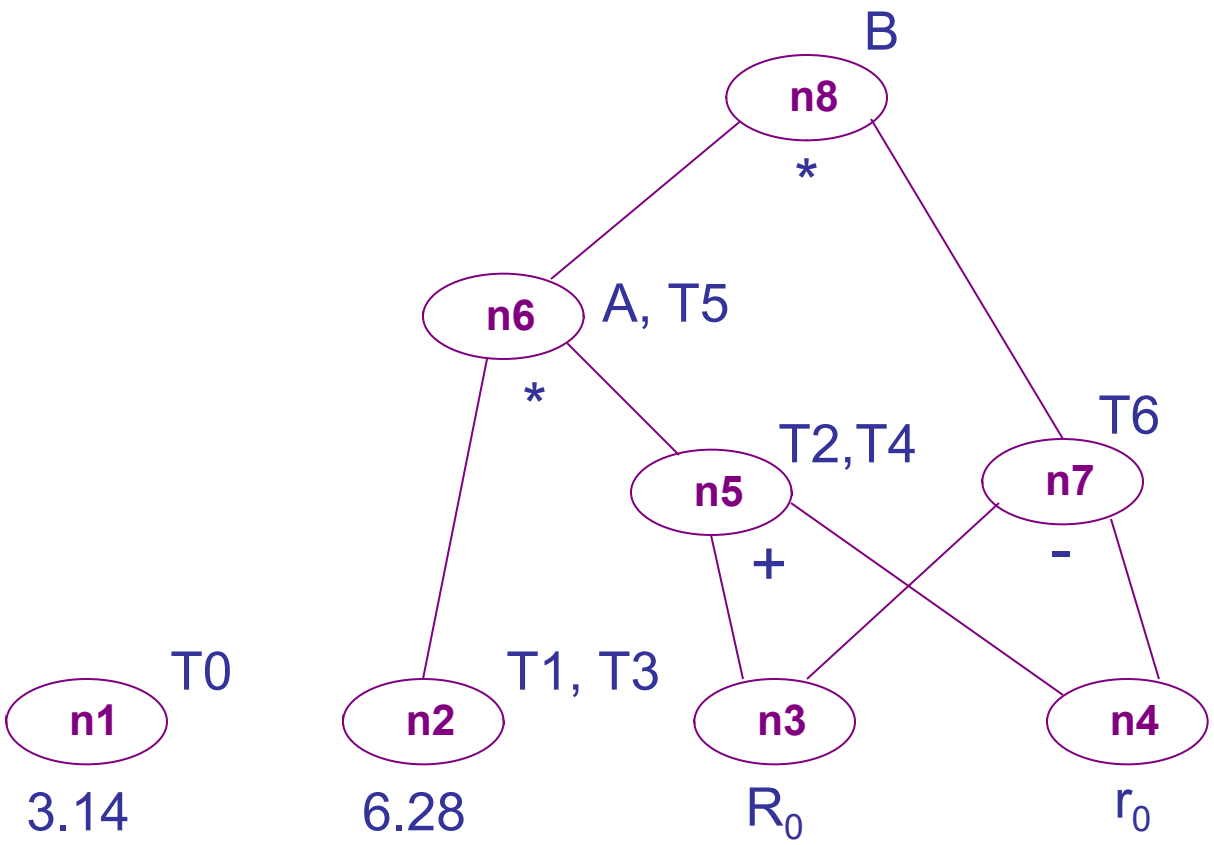
T5:=T3*T4

T6:=R-r

B:=T5*T6

✧基本块 *DAG* 表示的构造

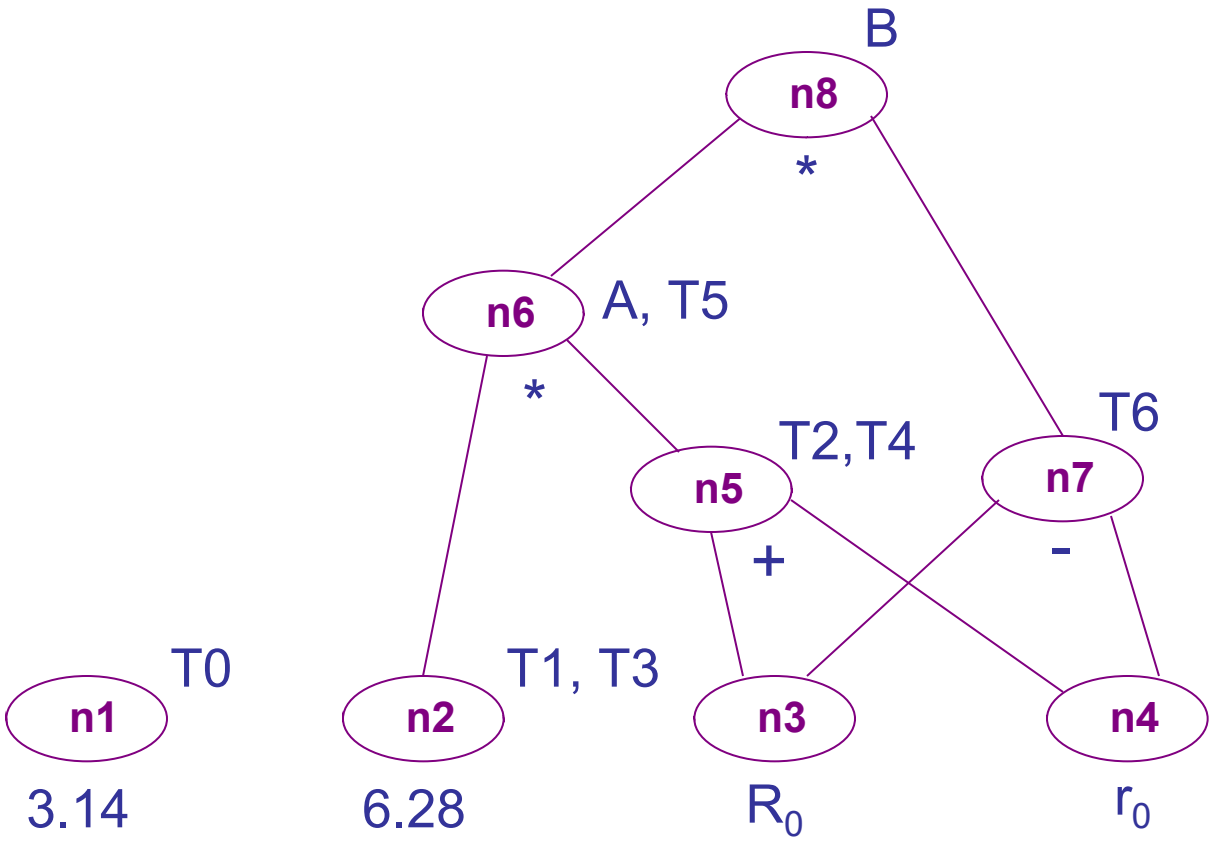
– 举例



T0:=3.14
T1:=2*T0
T2:=R+r
A:=T1*T2
B:=A
T3:=2*T0
T4:=R+r
T5:=T3*T4
T6:=R-r
B:=T5*T6

✧从基本块的 *DAG* 表示可得到等价的基本块

– 举例：从下图的 *DAG* 可得到右边的新的基本块（经拓扑排序及添加适当的复写语句）



T0:=3.14

T1:=6.28

T3:=6.28

T2:=R+r

T4:=T2

A:=6.28*T2

T5:=A

T6:=R-r

B:=A*T6

✧从基本块的 *DAG* 表示可得到等价的基本块

– 比较变换前后的基本块

```
T0:=3.14
T1:=2*T0
T2:=R+r
A:=T1*T2
B:=A
T3:=2*T0
T4:=R+r
T5:=T3*T4
T6:=R-r
B:=T5*T6
```

```
T0:=3.14
T1:=6.28
T3:=6.28
T2:=R+r
T4:=T2
A:=6.28*T2
T5:=A
T6:=R-r
B:=A*T6
```

所作的优化

合并已知量

删除多余运算

删除无用赋值

复写传播

10.3.3 循环优化（*loop optimization*）

– 举例

- 代码外提（*code motion*）

  while （i< limit/2）{…}

  等价于 t:=limit/2;
  while （i< t）{…}

- 循环不变量（*loop-invariant*）代码可以外提

如对于循环内部的语句 x:=y+z，若 y 和 z 的定值点都在循环外，则x:=y+z 为循环不变量

– 举例

- 归纳变量（*induction variable*）相关的优化

  归纳变量是在循环的顺序迭代中取得一系列值的变量

  常见的归纳变量如循环下标及循环体内显式增量和减量的变量

  通常可以针对归纳变量可以进行如下优化：

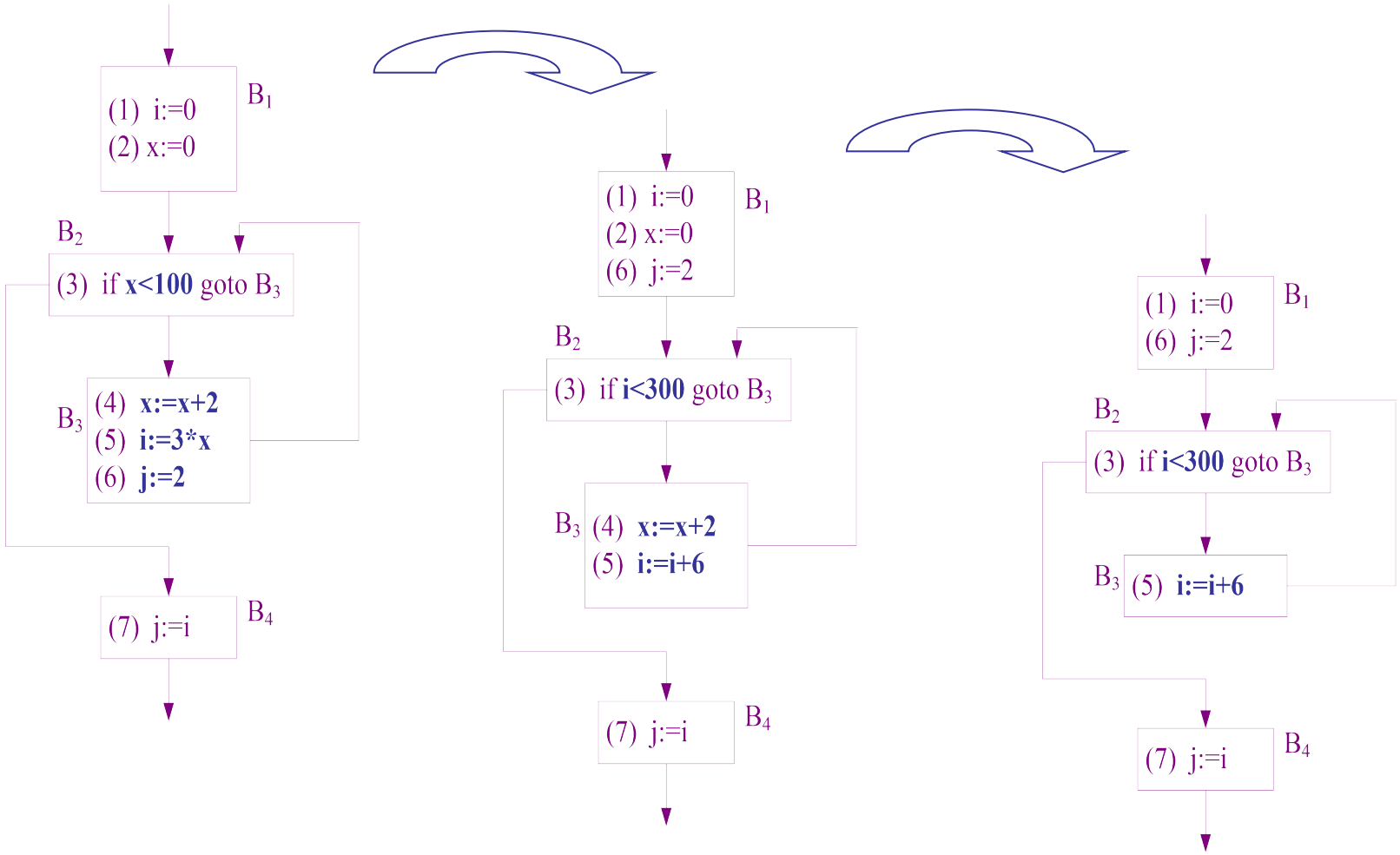
  1）削弱归纳变量的计算强度

  2）因常常可以有冗余的归纳变量，可以只在寄存器中保存个别归纳变量，而不是全部. 特别是经强度削弱后，往往可以删除某些归纳变量

代码优化技术

✧ 循环优化举例

代码外提
强度削弱
删除归纳变量



✧10.3.4 全局优化

- 优化技术

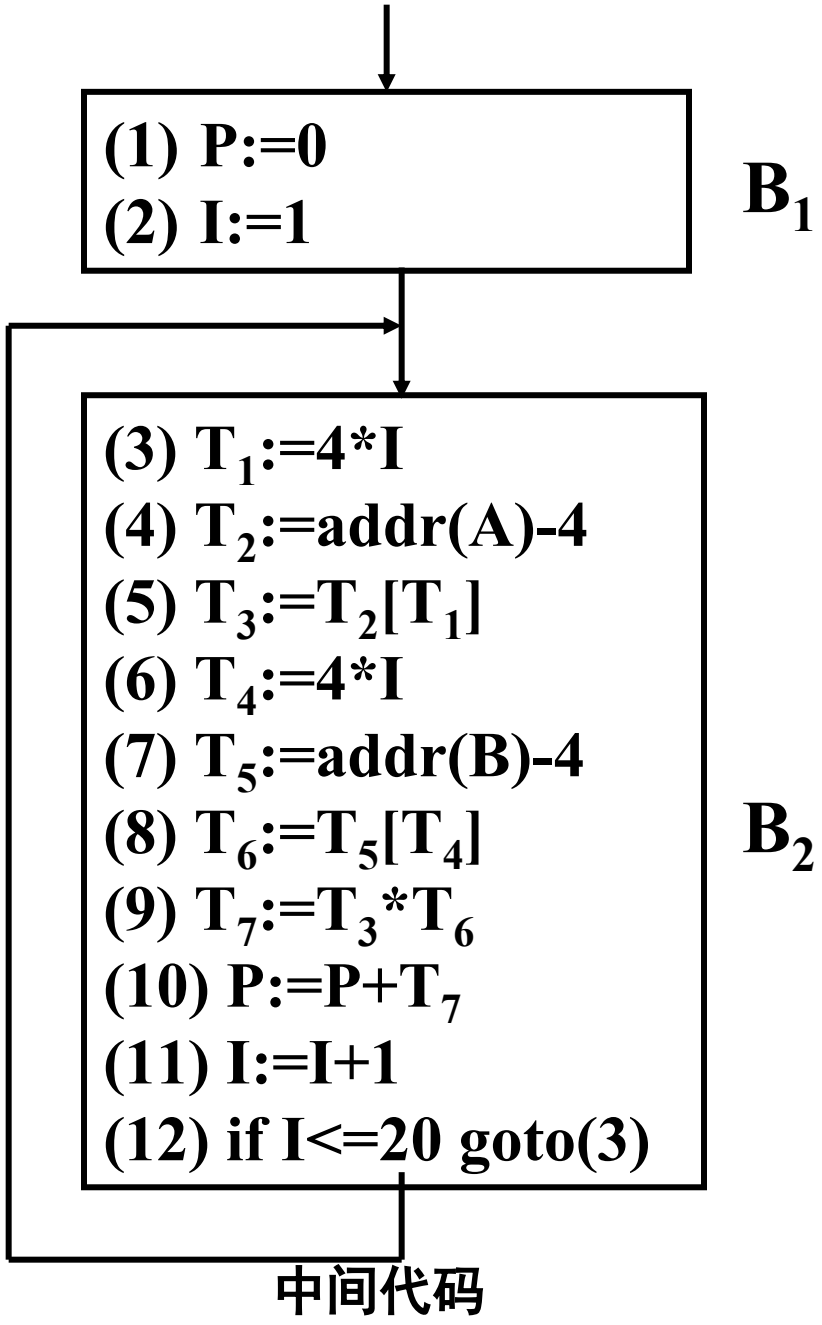
1. 删除多余运算
2. 循环不变代码外提
3. 强度削弱
4. 变换循环控制条件
5. 合并已知量与复写传播
6. 删除无用赋值

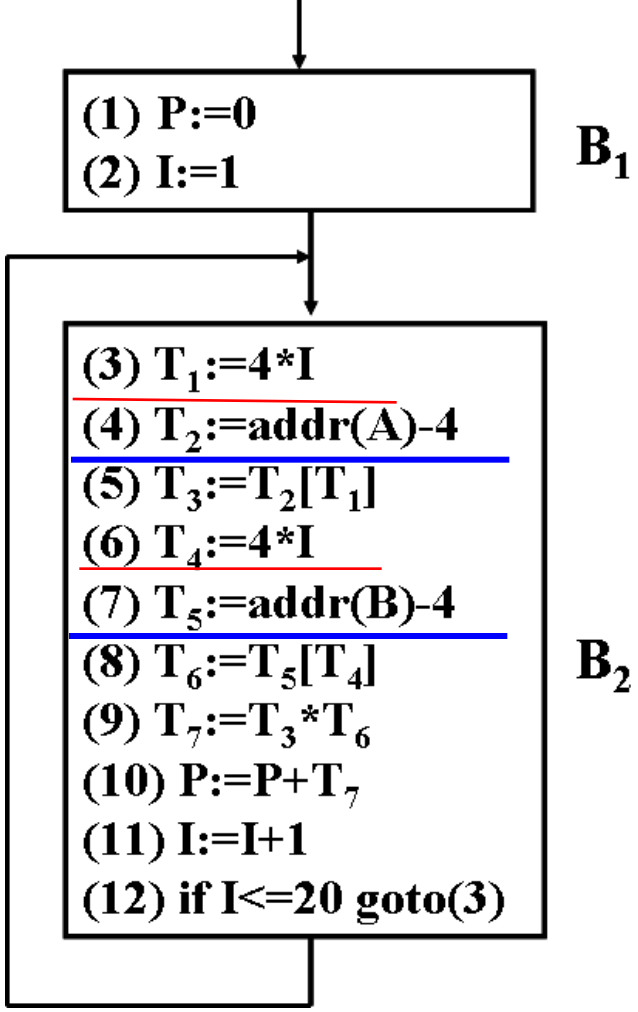
P:=0

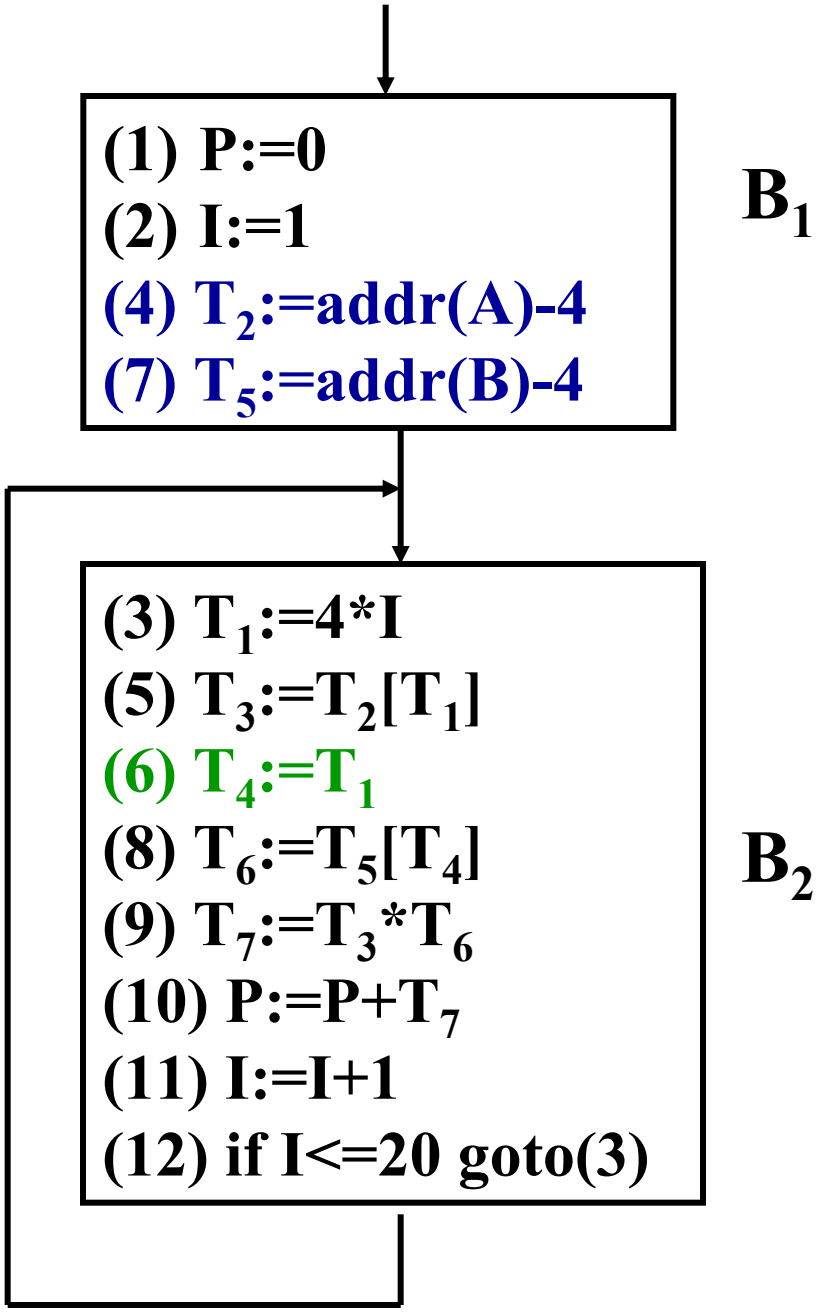
for I:=1 to 20 do

P:=P+A[I]*B[I]

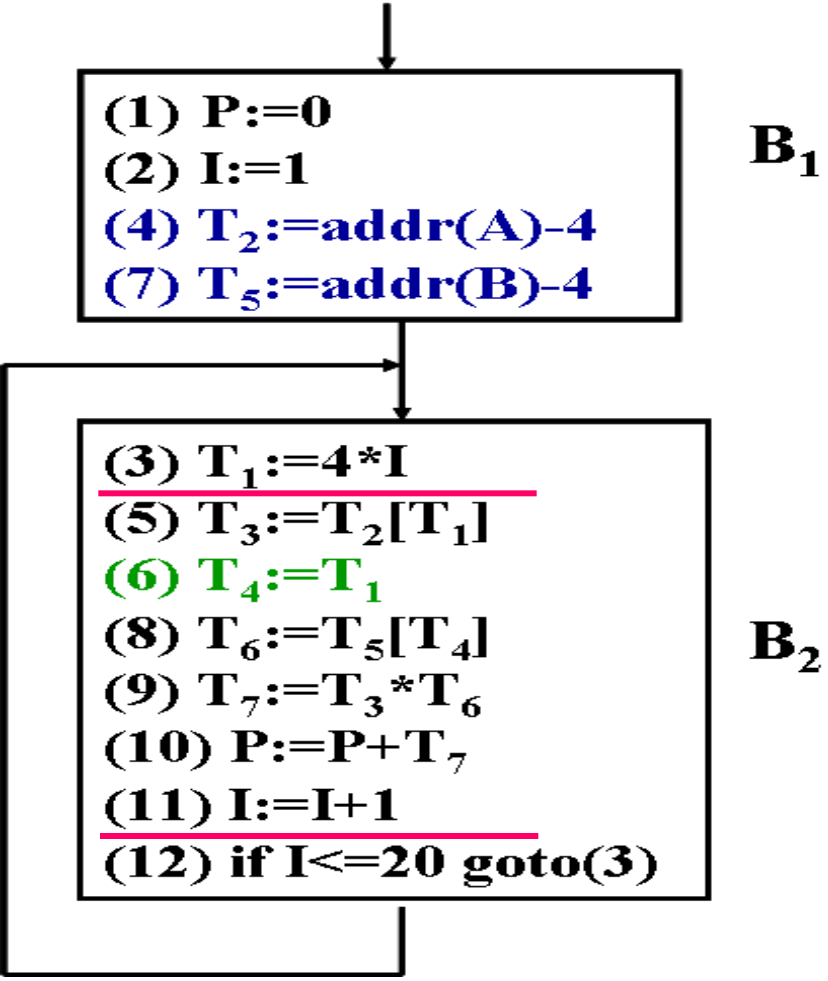
数组下标从1开始。



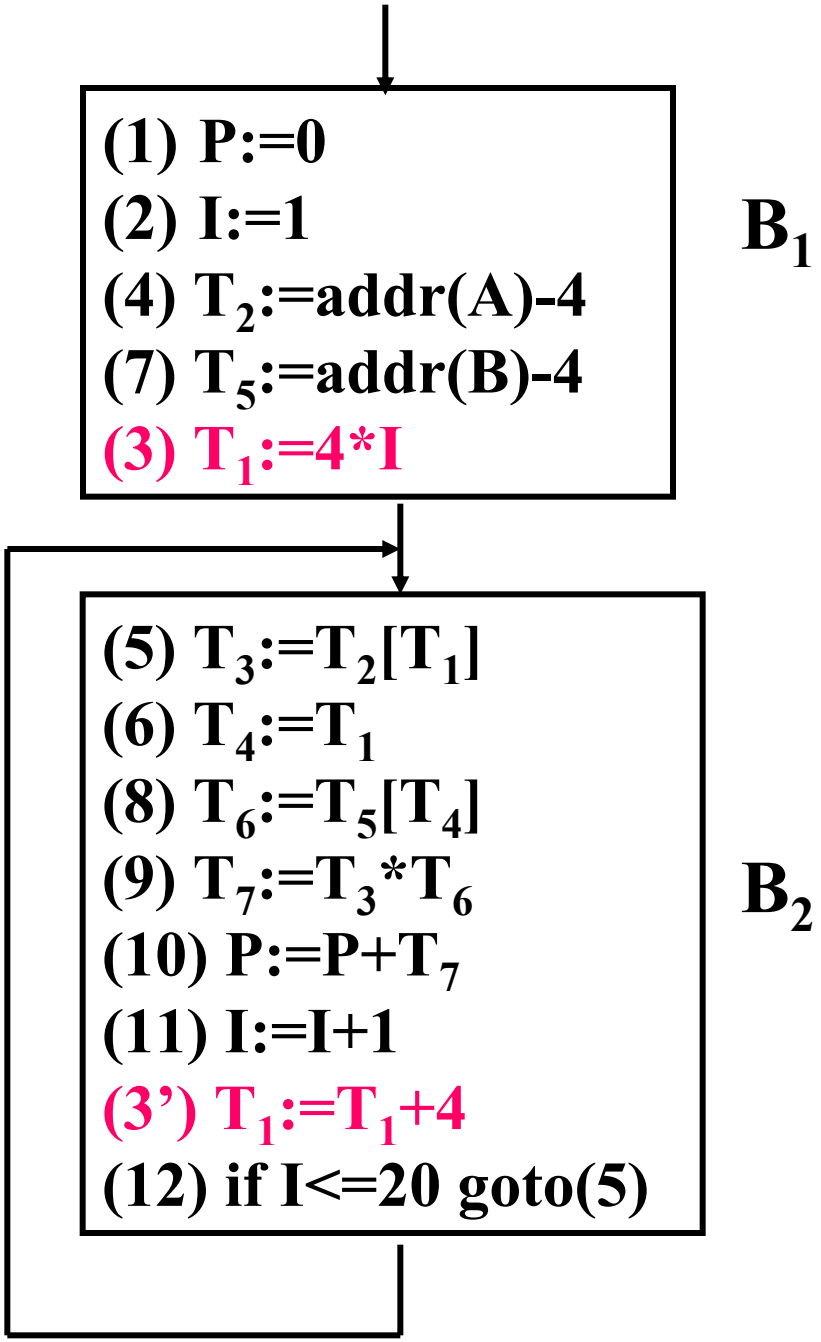


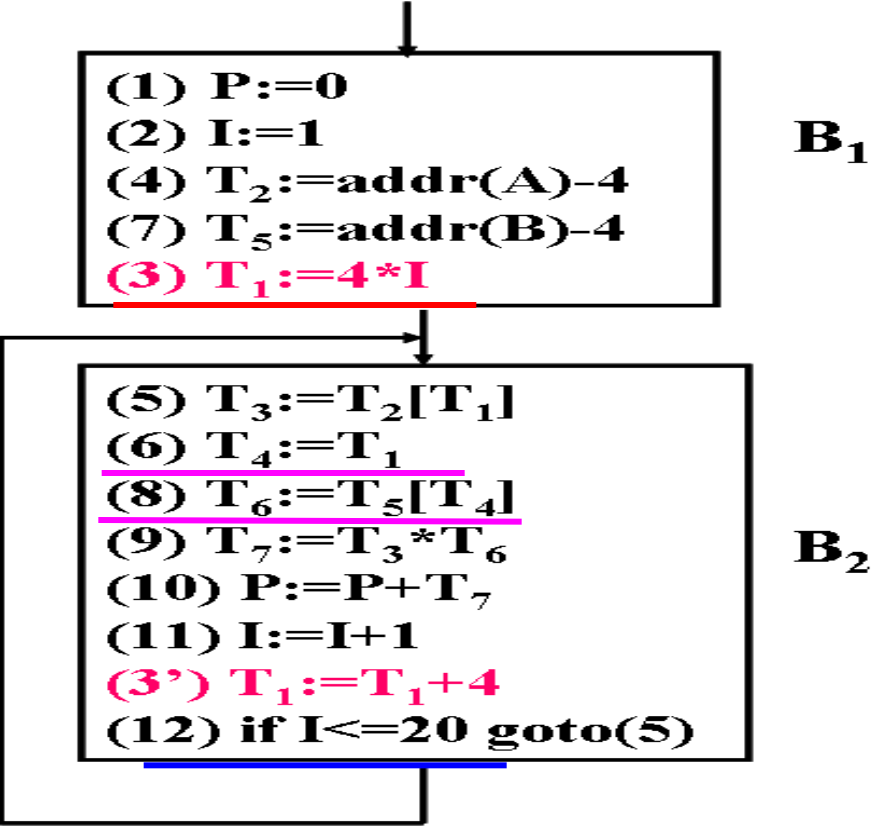


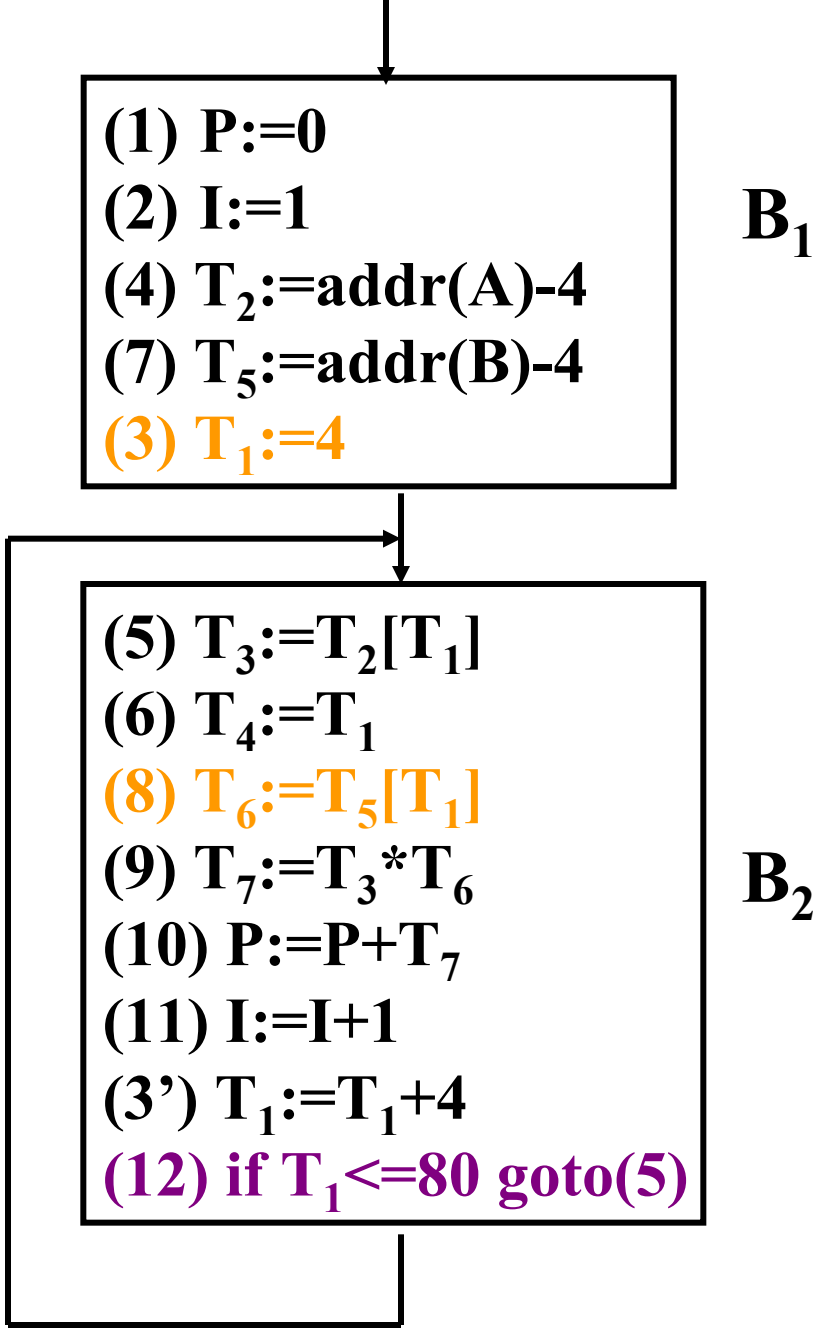
- 删除多余运算(删除公共子表达式)：相同的运算只计算第一次，后面的运算改为赋值
- 代码外提：把循环不变的运算，即其结果独立于循环次数的表达式，提到循环的前面。



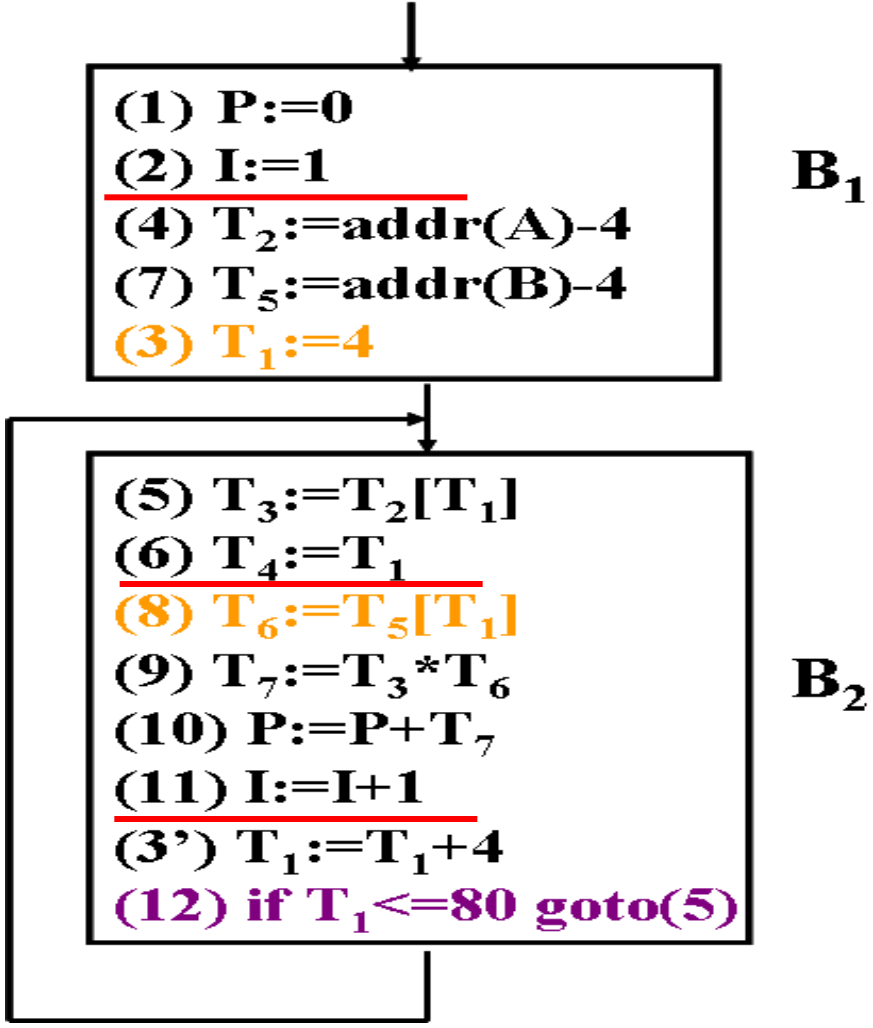
- 强度削弱：把强度大的运算换算成强度小的运算

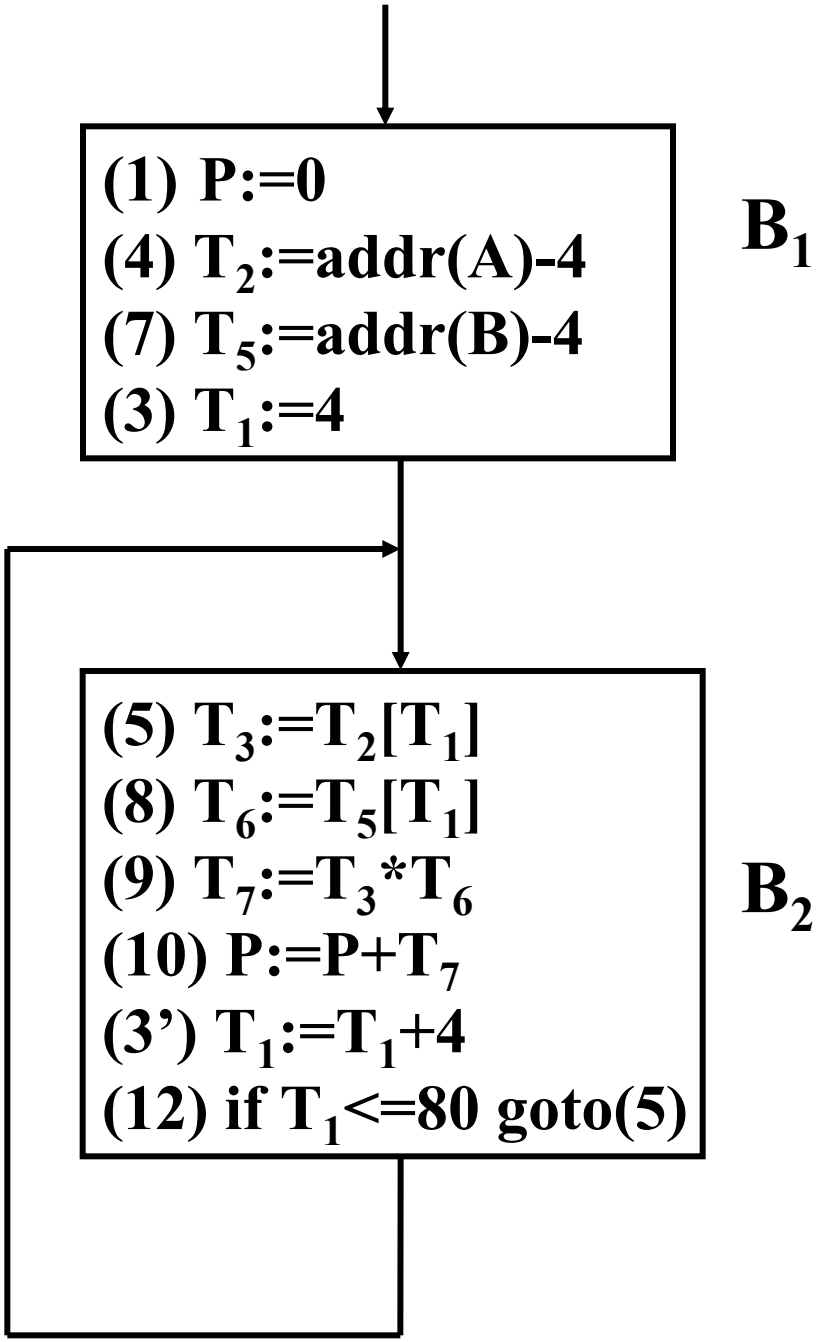






- 变换循环控制条件
- 合并已知量与复写传播
  - 编译时求出已知量的值，称为合并已知量
  - 如果变量b是变量a的拷贝，那么变量a可以替换变量b，称为复写传播





- 删除无用赋值：
  将没有使用的临时变量及其赋值计算删除

作业

2题：（1）（2）

4题