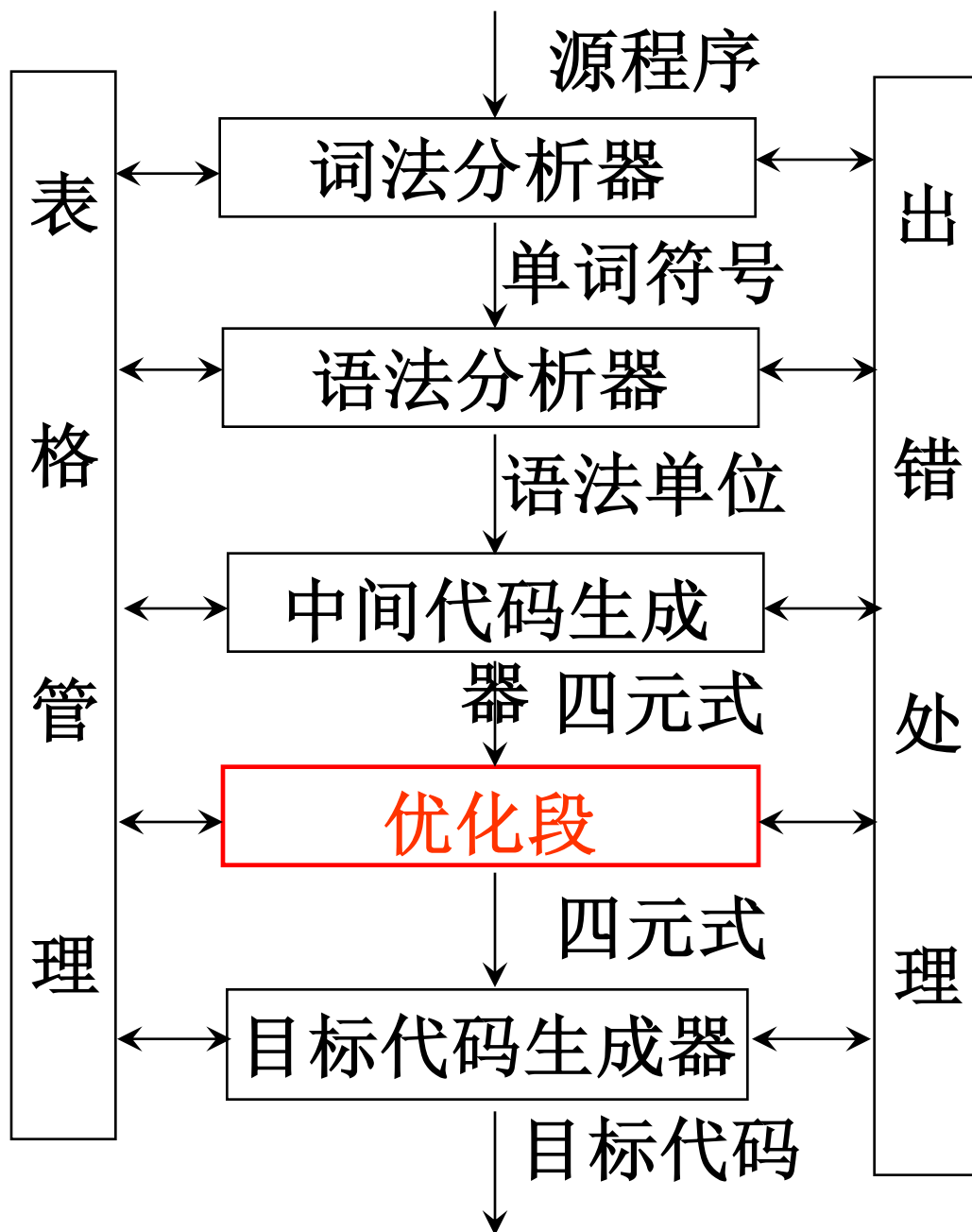




# 第十章 优化

授课人：高珍



# 问题的提出

## ■ 编译程序的作用

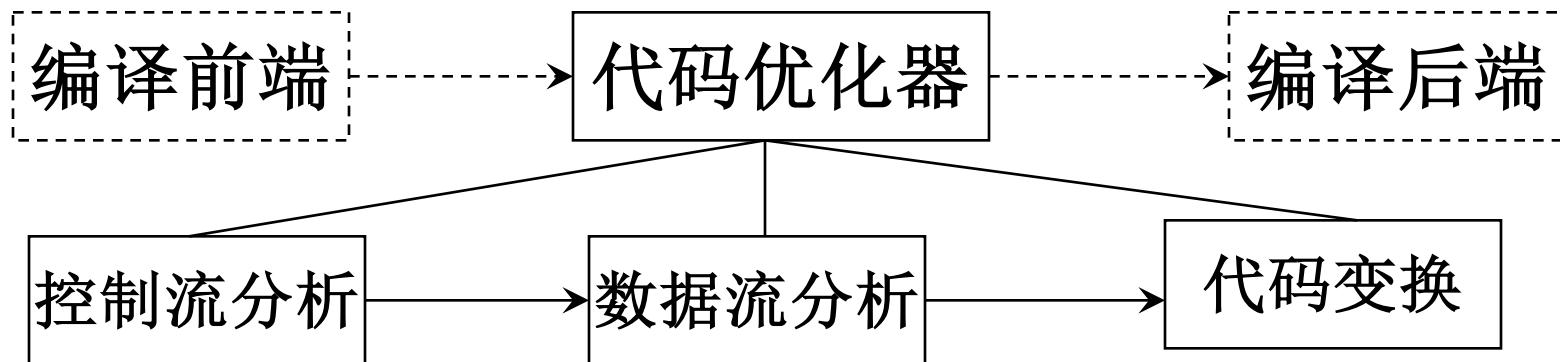
- 使计算机的使用方式从用机器语言编程发展到用高级语言编程。是计算机发展史上的一次飞跃。

## ■ 早期编译程序的不足

- 目标程序质量差
  - 占用的空间大
  - 运行的时间长

# 优化

- **优化**：对程序进行各种等价变换，使得从变换后的程序出发，能生成更有效的目标代码。



# 内容线索

- 概述
- 局部优化
- 循环优化

# 概述

- 优化的目的是为了产生更高效的代码。由优化编译程序提供的对代码的各种变换必须遵循一定的原则：
  - **等价**原则：经过优化后不应改变程序运行的结果；
  - **有效**原则：使优化后所产生的目标代码运行时间较短，占用的存储空间较小；
  - **合算**原则：应尽可能以较低的代价取得较好的优化效果。

# 概述

## ■ 优化的三个不同级别：

- 局部优化
- 循环优化
- 全局优化

## ■ 优化的种类：

- 删除多余运算(或称删除公用子表达式)
- 复写传播
- 删除无用赋值
- 代码外提
- 强度削弱
- 变换循环控制条件
- 合并已知量

```

void quicksort (m, n);
int m, n;
{
    int i, j;
    int v, x;
    if (n<=m) return;
    /* fragment begins here*/
    i=m-1; j=n; v=a [n];
    while (1) {
        do i=i+1; while (a [i]<v);
        do j=j-1; while (a [j]>v);
        if (i>=j) break;
        x=a [i]; a[i]=a [j]; a[j]=x;
    }
    x=a[i]; a[i]=a [n]; a [n]=x;
    /*fragment ends here*/
    quicksort (m, j); quicksort (i+1, n);
}

```



```
i=m-1; j=n; v=a [n];
```

```
while (1) {
```

```
do i=i+1; while (a [i]<v);
```

```
do j=j-1; while (a [j]>v);
```

```
if (i>=j) break;
```

```
x=a [i]; a[i]=a [j]; a[j]=x;
```

```
}
```

```
x=a[i]; a[i]=a [n]; a [n]=x;
```

**B<sub>1</sub>**

```
i:=m-1  
j:=n  
T1:=4*n  
v:=a[T1]
```

**B<sub>2</sub>**

```
i:=i+1  
T2:=4*i  
T3:=a[T2]  
if T3<v goto B2
```

**B<sub>3</sub>**

```
j:=j-1  
T4:=4*j  
T5:=a[T4]  
if T5>v goto B3
```

**B<sub>4</sub>**

```
if i>=j goto B6
```

**B<sub>5</sub>**

```
T6:=4*i  
x:=a [T6]  
T7:=4*i  
T8:=4*j  
T9:=a [T8]  
a [T7]=T9  
T10:= 4*j  
a [T10]=x  
goto B2
```

**B<sub>6</sub>**

```
T11:=4*i  
x:=a [T11]  
T12:=4*i  
T13:=4*n  
T14:=a [T13]  
a [T12]=T14  
T15:= 4*n  
a [T15]=x
```

中间代码程序段

# 优化技术示例

## ■ 优化的种类：

- 删除多余运算(或称删除公用子表达式)
- 复写传播
- 删除无用赋值
- 强度削弱
- 删除归纳变量

# 删除公共子表达式

B<sub>5</sub>:

T<sub>6</sub> := 4 \* i

x := a[T<sub>6</sub>]

T<sub>7</sub> := 4 \* i

T<sub>8</sub> := 4 \* j

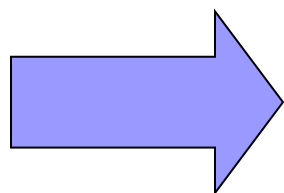
T<sub>9</sub> := a[T<sub>8</sub>]

a[T<sub>7</sub>] = T<sub>9</sub>

T<sub>10</sub> := 4 \* j

a[T<sub>10</sub>] = x

goto B<sub>2</sub>



B<sub>5</sub>:

T<sub>6</sub> := 4 \* i

x := a[T<sub>6</sub>]

T<sub>7</sub> := T<sub>6</sub>

T<sub>8</sub> := 4 \* j

T<sub>9</sub> := a[T<sub>8</sub>]

a[T<sub>7</sub>] = T<sub>9</sub>

T<sub>10</sub> := T<sub>8</sub>

a[T<sub>10</sub>] = x

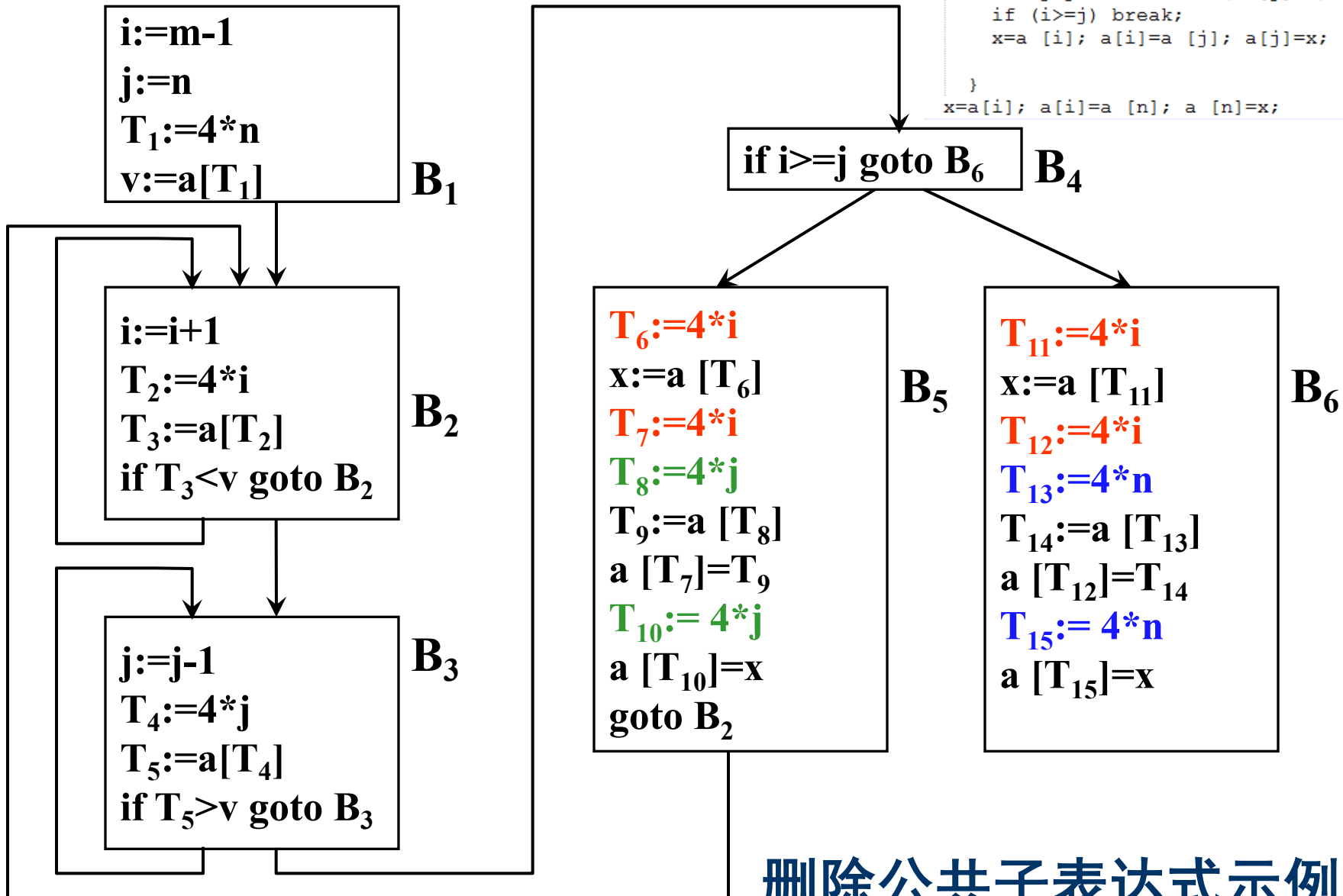
goto B<sub>2</sub>

```
i=m-1; j=n; v=a [n];  
while (1) {  
    do i=i+1; while (a [i]<v);  
    do j=j-1; while (a [j]>v);  
    if (i>=j) break;  
    x=a [i]; a[i]=a [j]; a[j]=x;  
}  
x=a[i]; a[i]=a [n]; a [n]=x;
```

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```

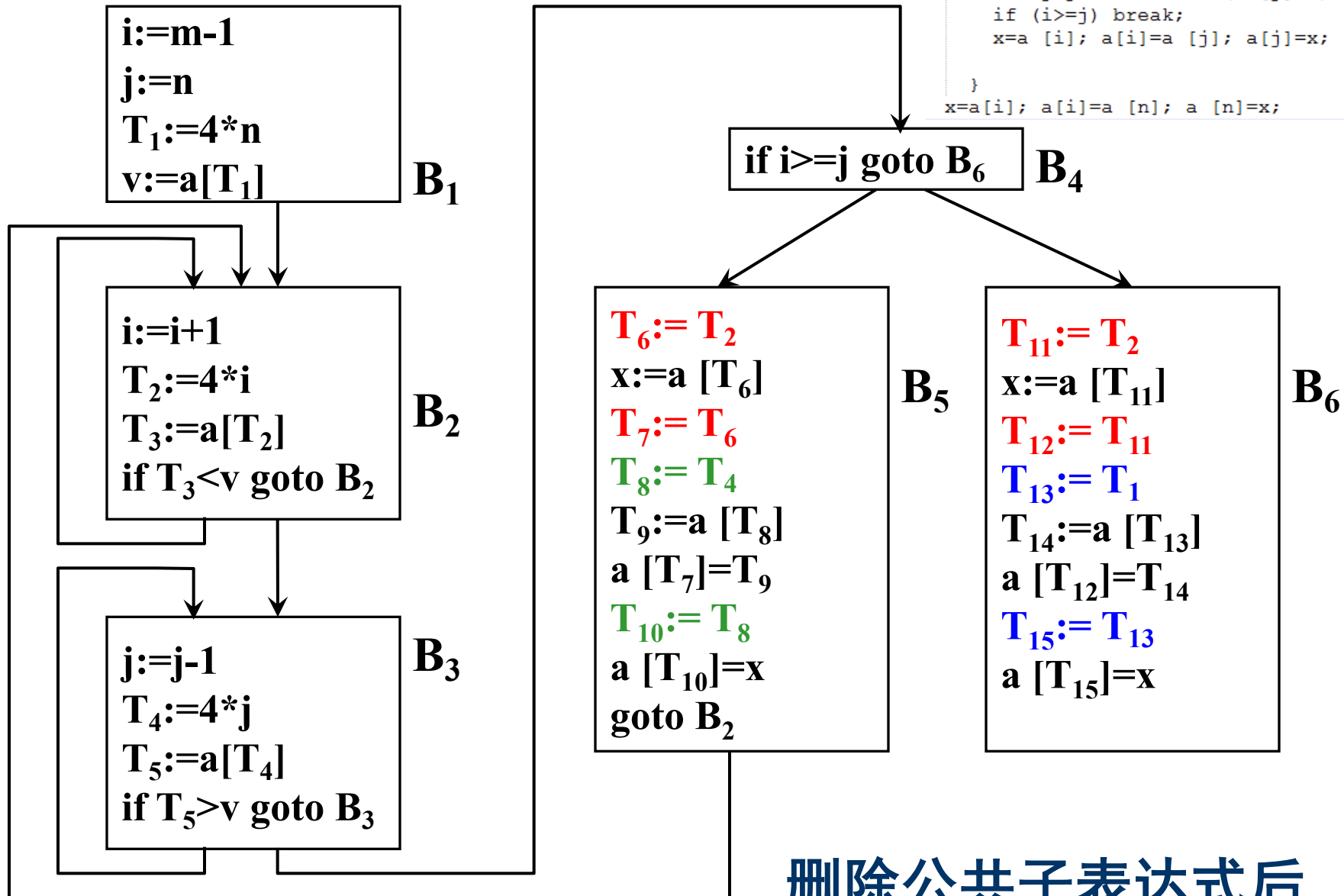


删除公共子表达式示例

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```



删除公共子表达式后  
(删除多余运算)

# 优化技术示例

## ■ 优化的种类：

- 删除多余运算(或称删除公用子表达式)
- **复写传播**
- 删除无用赋值
- 强度削弱
- 删除归纳变量

# 复写传播

```
i=m-1; j=n; v=a [n];  
while (1) {  
    do i=i+1; while (a [i]<v);  
    do j=j-1; while (a [j]>v);  
    if (i>=j) break;  
    x=a [i]; a[i]=a [j]; a[j]=x;  
}  
x=a[i]; a[i]=a [n]; a [n]=x;
```

B<sub>5</sub>:

T<sub>6</sub> := T<sub>2</sub>

x := a[T<sub>6</sub>]

T<sub>7</sub> := T<sub>6</sub>

T<sub>8</sub> := T<sub>4</sub>

T<sub>9</sub> := a[T<sub>8</sub>]

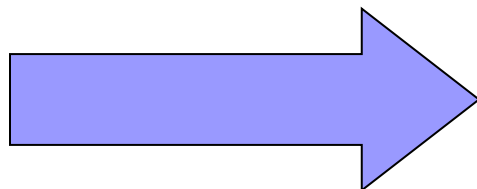
a[T<sub>7</sub>] = T<sub>9</sub>

T<sub>10</sub> := T<sub>8</sub>

a[T<sub>10</sub>] = x

goto B<sub>2</sub>

T<sub>6</sub> := T<sub>2</sub>  
x := a[T<sub>6</sub>]



没有改变  
T<sub>6</sub>的值

B<sub>5</sub>:

T<sub>6</sub> := T<sub>2</sub>

x := a[T<sub>2</sub>]

T<sub>7</sub> := T<sub>2</sub>

T<sub>8</sub> := T<sub>4</sub>

T<sub>9</sub> := a[T<sub>4</sub>]

a[T<sub>2</sub>] = T<sub>9</sub>

T<sub>10</sub> := T<sub>4</sub>

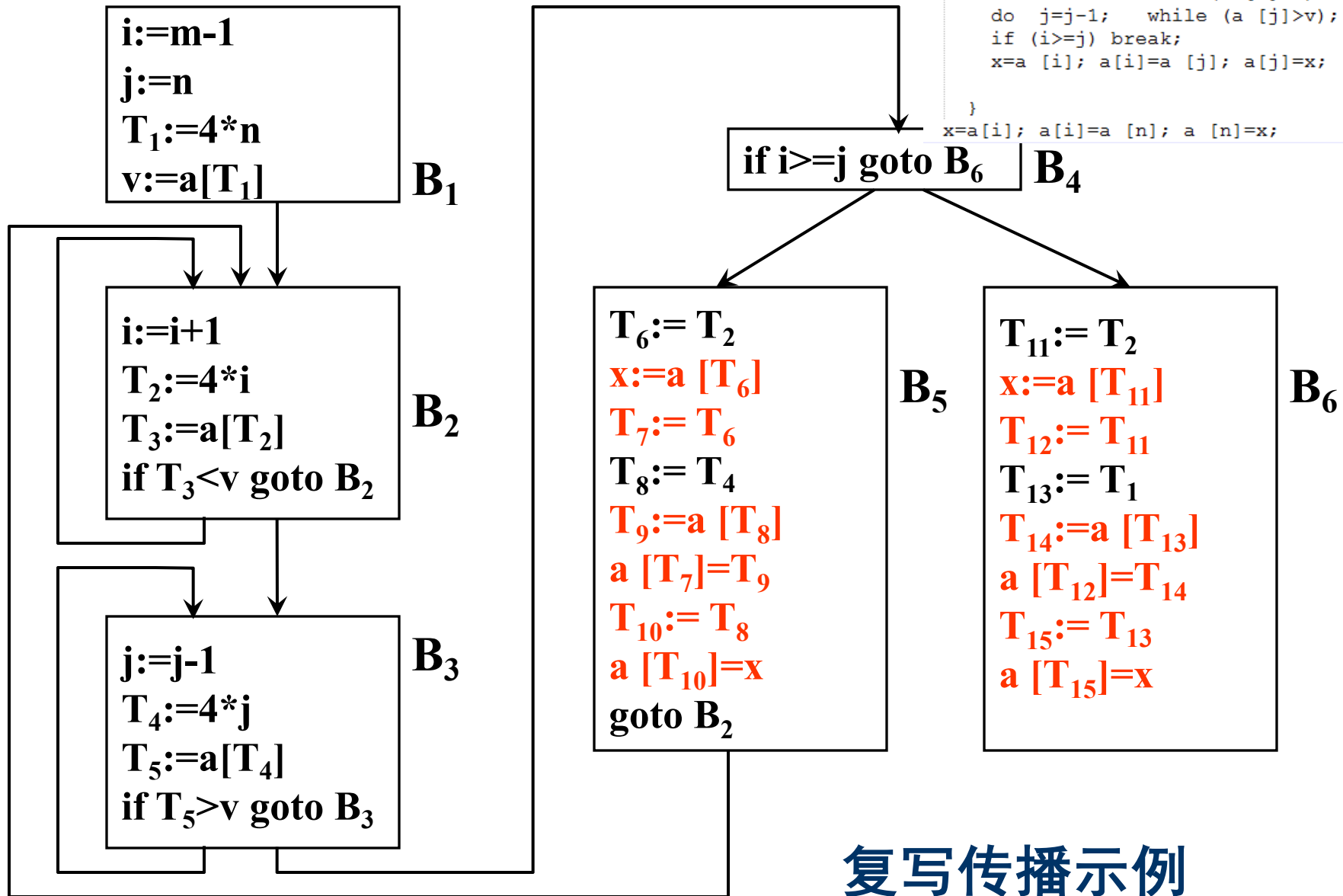
a[T<sub>4</sub>] = x

goto B<sub>2</sub>

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```



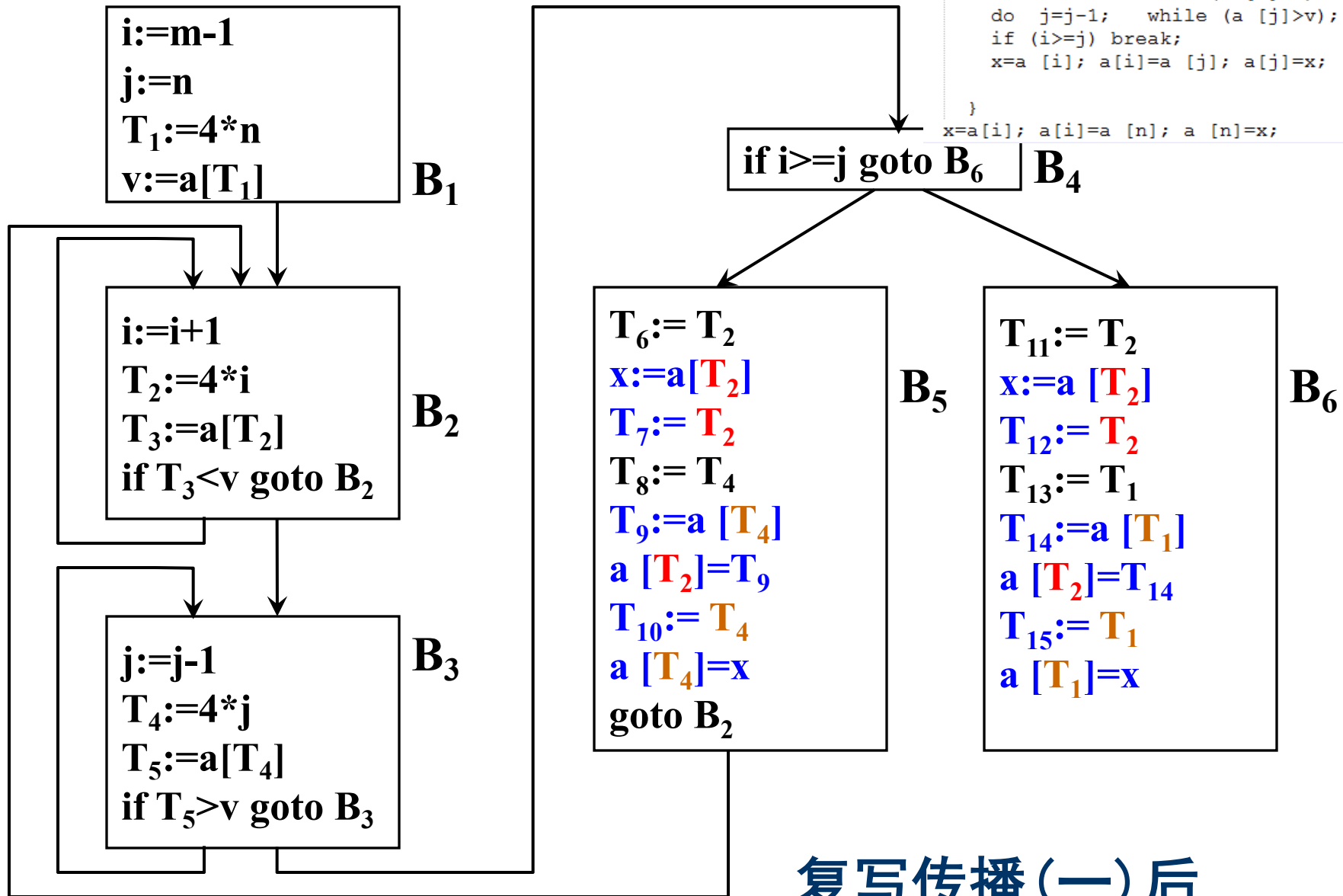
复写传播示例



```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```

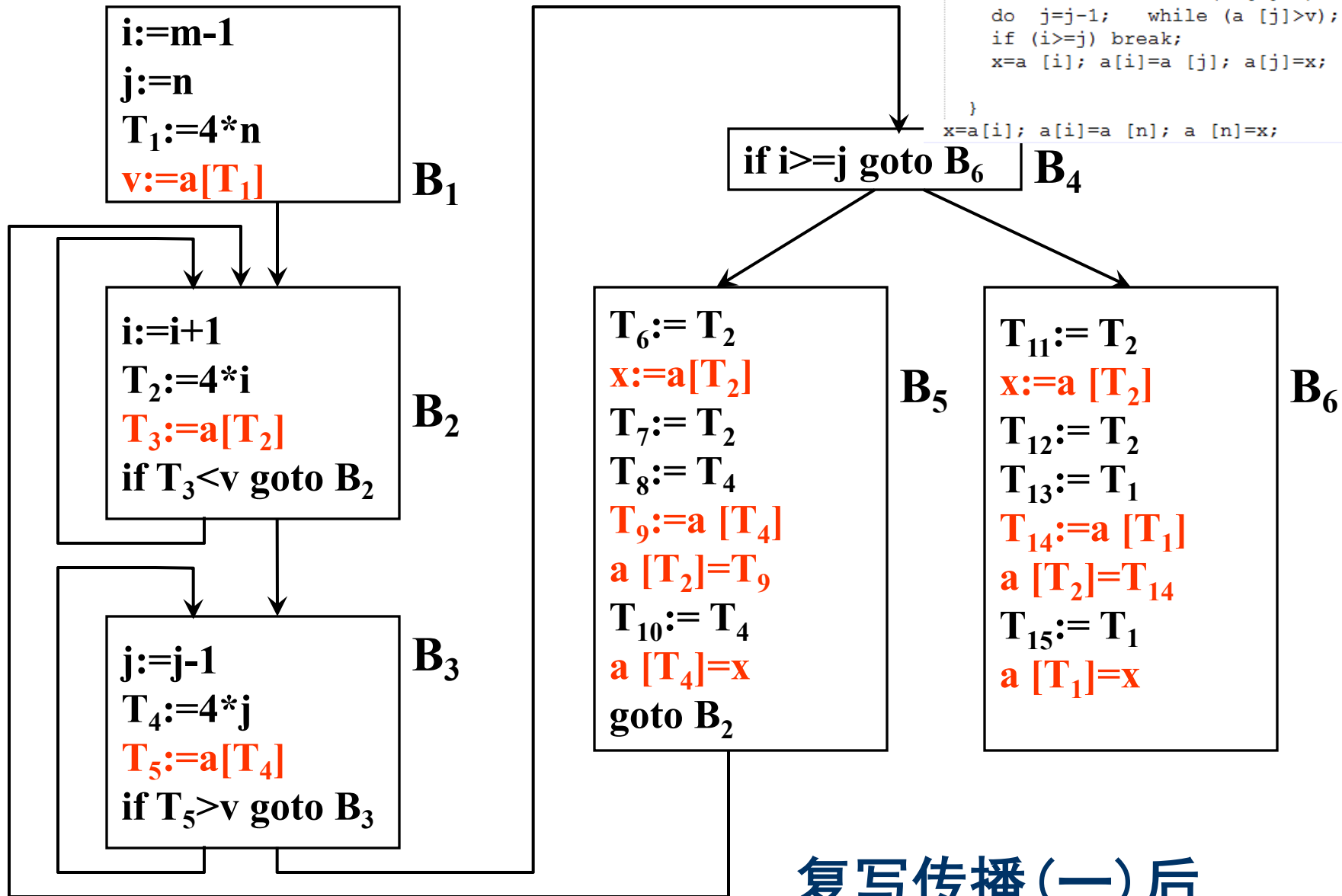


复写传播(一)后

```

i:=m-1; j:=n; v=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```

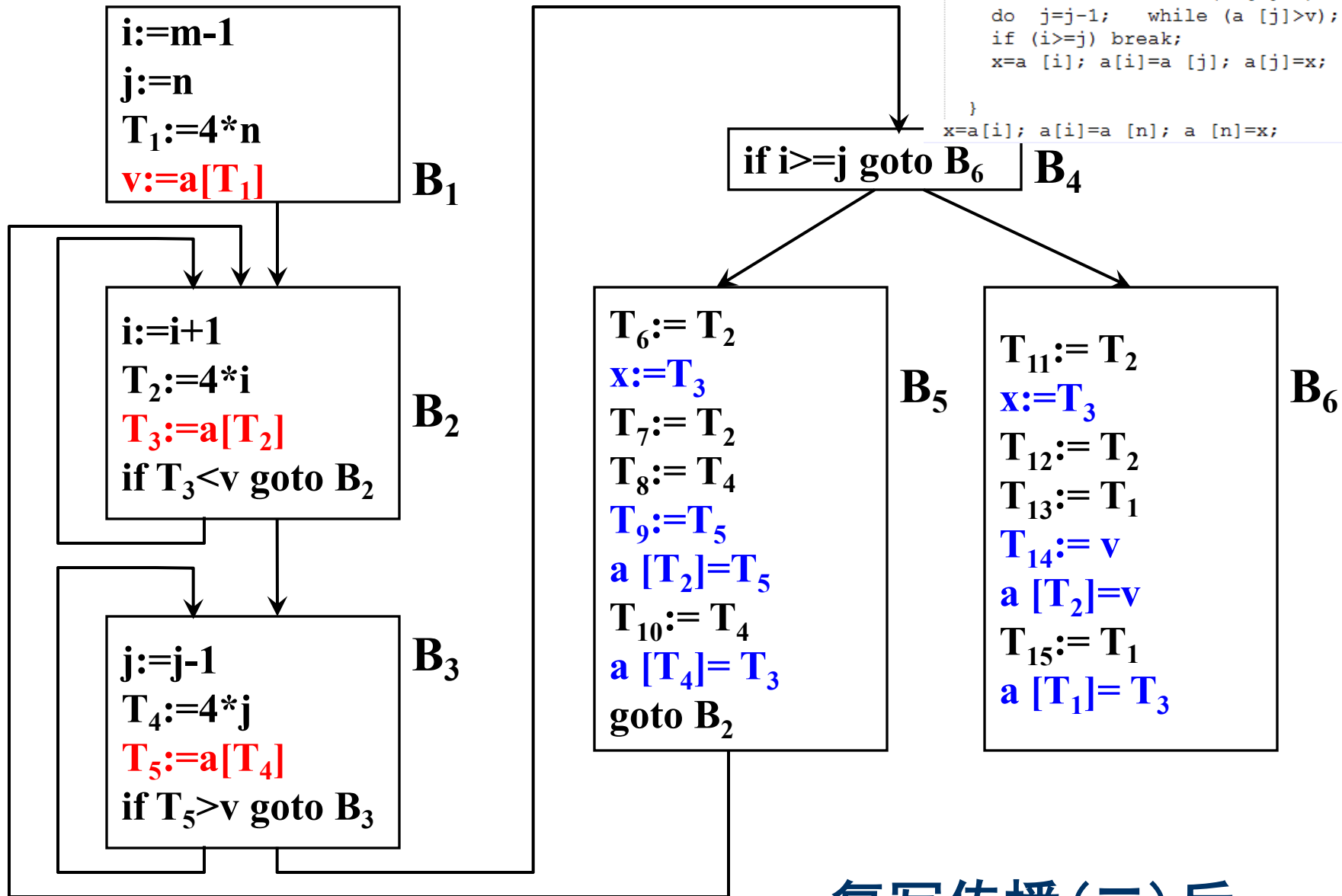


复写传播(一)后  
观察a[T<sub>1</sub>] a[T<sub>2</sub>] a[T<sub>4</sub>]

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```



复写传播(二)后

# 优化技术示例

## ■ 优化的种类：

- 删除多余运算(或称删除公用子表达式)
- 复写传播
- **删除无用赋值**
- 强度削弱
- 删除归纳变量

# 删除无用代码

```
i=m-1; j=n; v=a [n];  
while (1) {  
    do i=i+1; while (a [i]<v);  
    do j=j-1; while (a [j]>v);  
    if (i>=j) break;  
    x=a [i]; a[i]=a [j]; a[j]=x;  
}  
x=a[i]; a[i]=a [n]; a [n]=x;
```

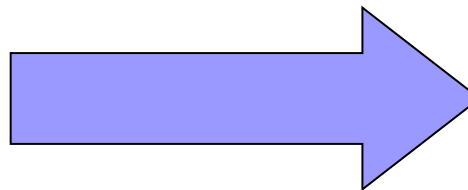
$B_5$

←

$T_6 := 4 * i$   
 $x := a [T_6]$   
 $T_7 := 4 * i$   
 $T_8 := 4 * j$   
 $T_9 := a [T_8]$   
 $a [T_7] = T_9$   
 $T_{10} := 4 * j$   
 $a [T_{10}] = x$   
goto  $B_2$

$B_5$

$T_6 := T_2$   
 $x := T_3$   
 $T_7 := T_2$   
 $T_8 := T_4$   
 $T_9 := T_5$   
 $a [T_2] = T_5$   
 $T_{10} := T_4$   
 $a [T_4] = T_3$   
goto  $B_2$



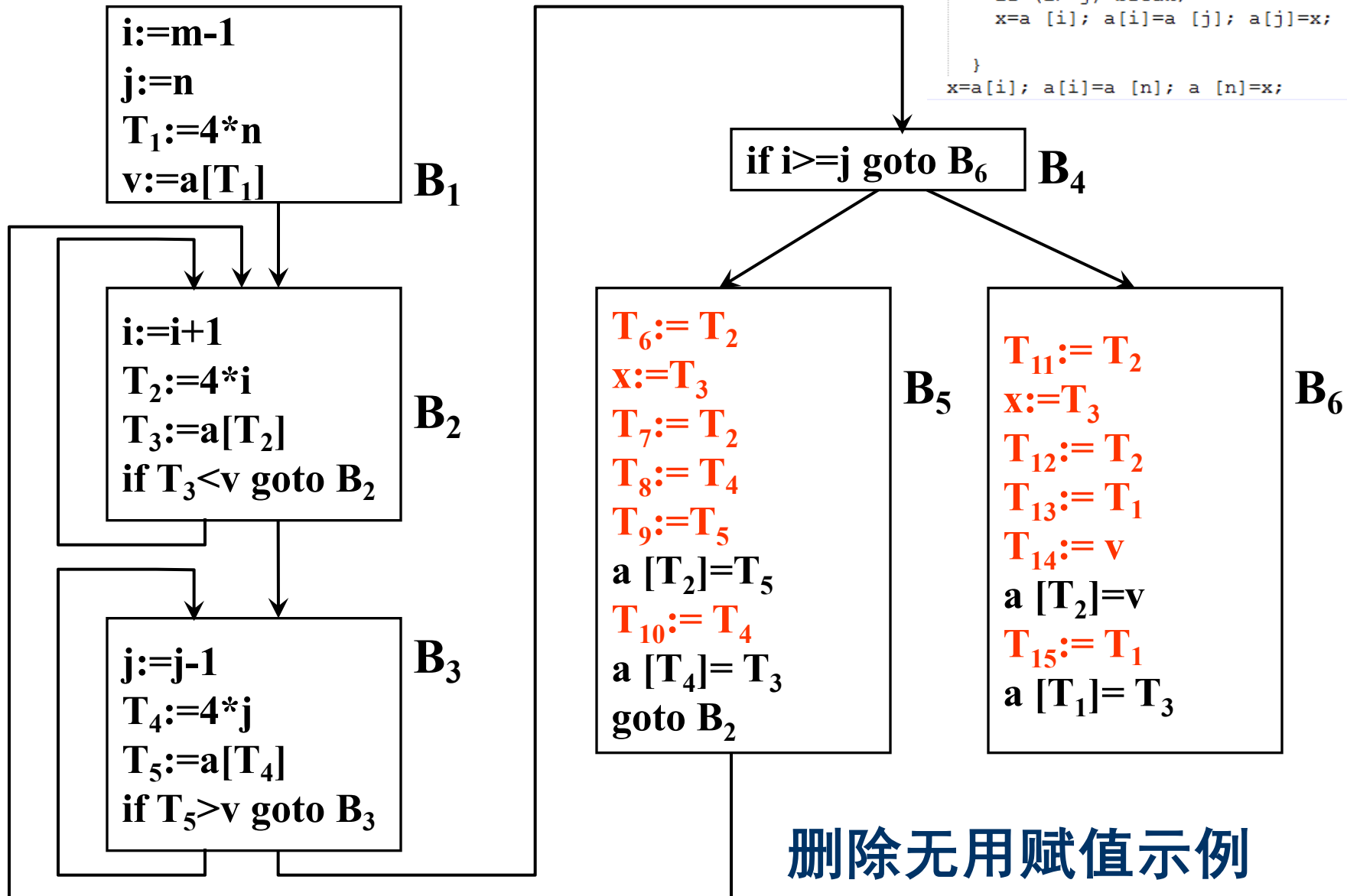
$B_5$

$a [T_2] = T_5$   
 $a [T_4] = T_3$   
goto  $B_2$

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```

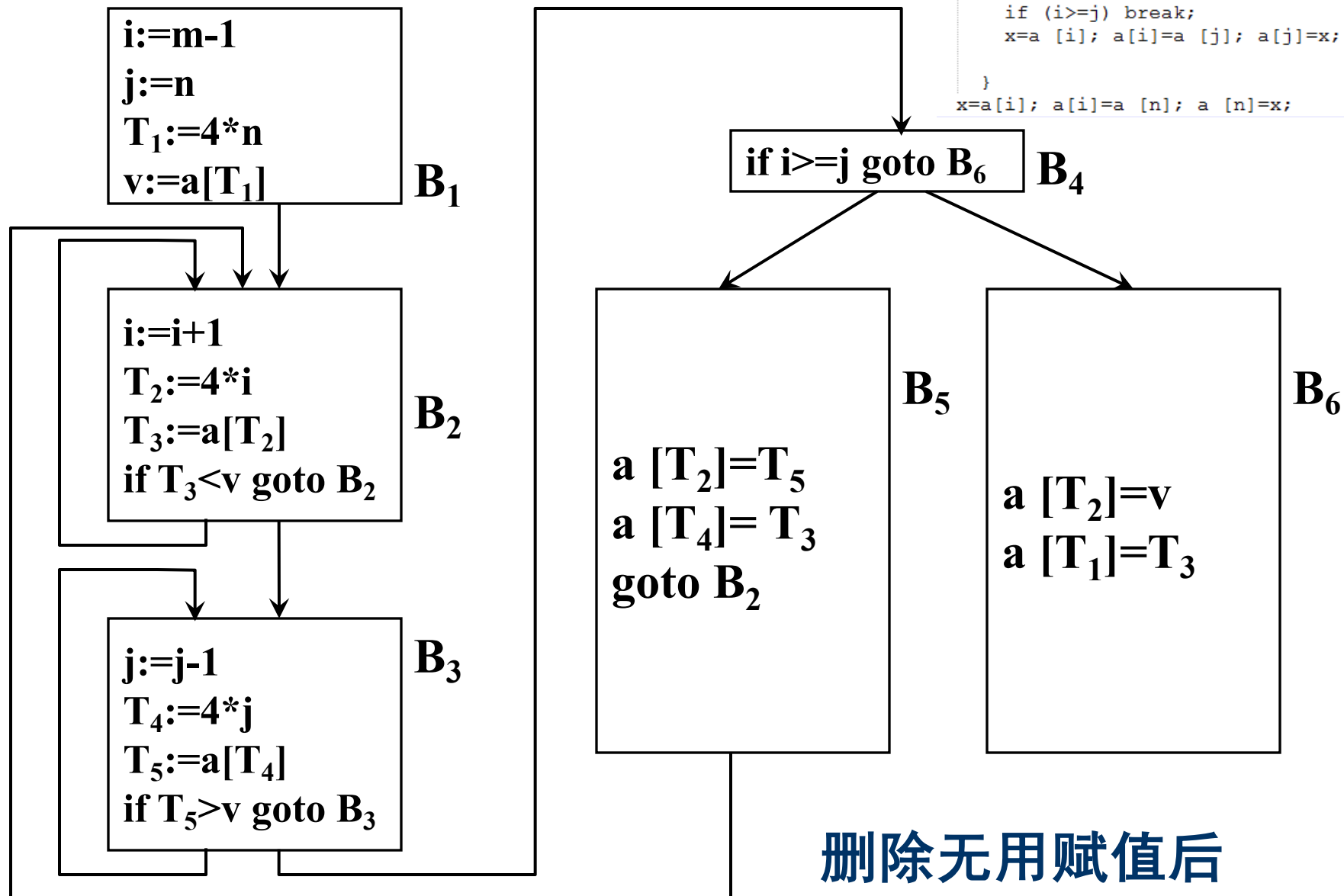


**删除无用赋值示例**

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```



删除无用赋值后

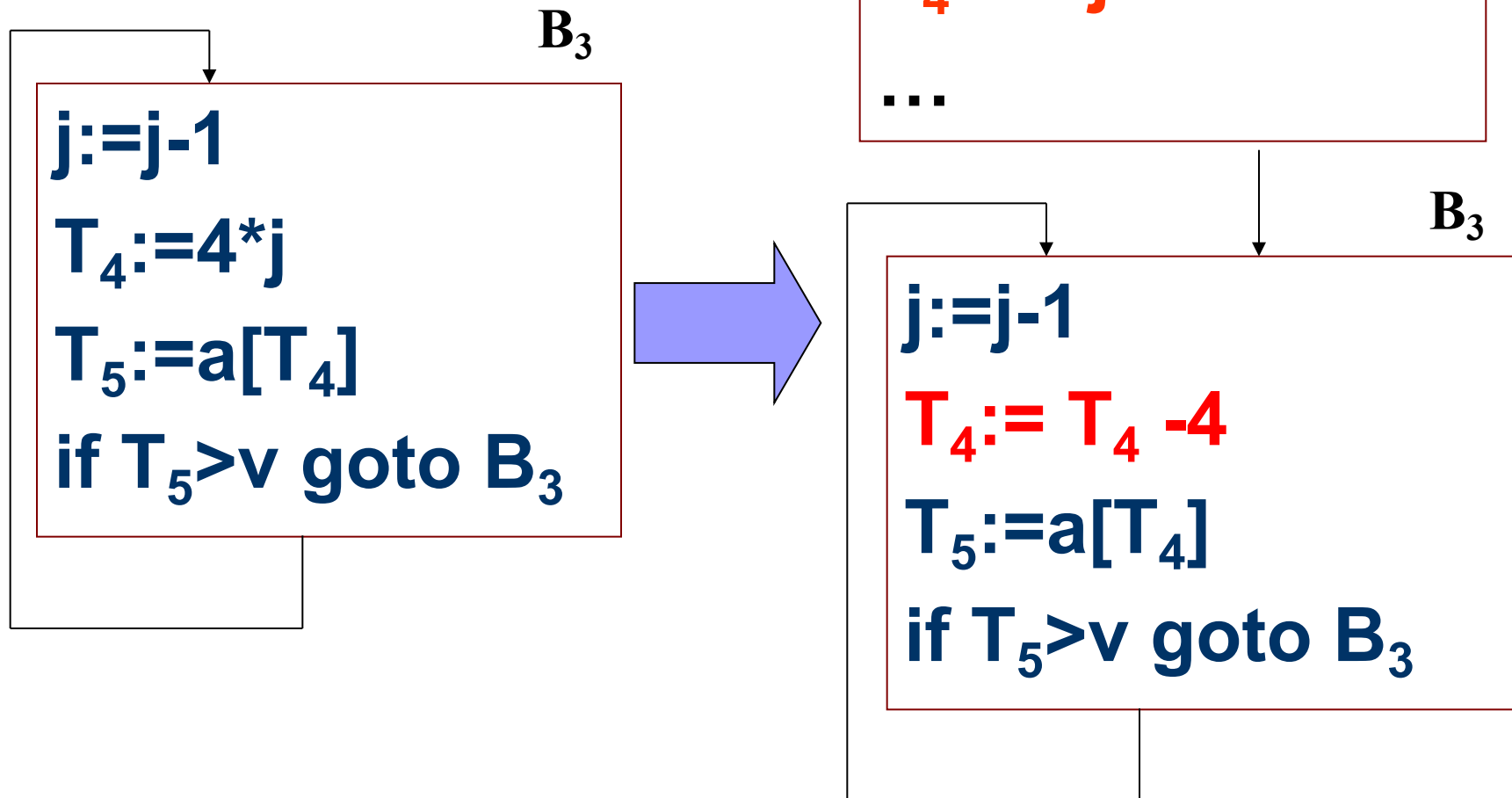
# 优化技术示例

## ■ 优化的种类：

- 删除多余运算(或称删除公用子表达式)
- 复写传播
- 删除无用赋值
- **强度削弱**
- 删除归纳变量



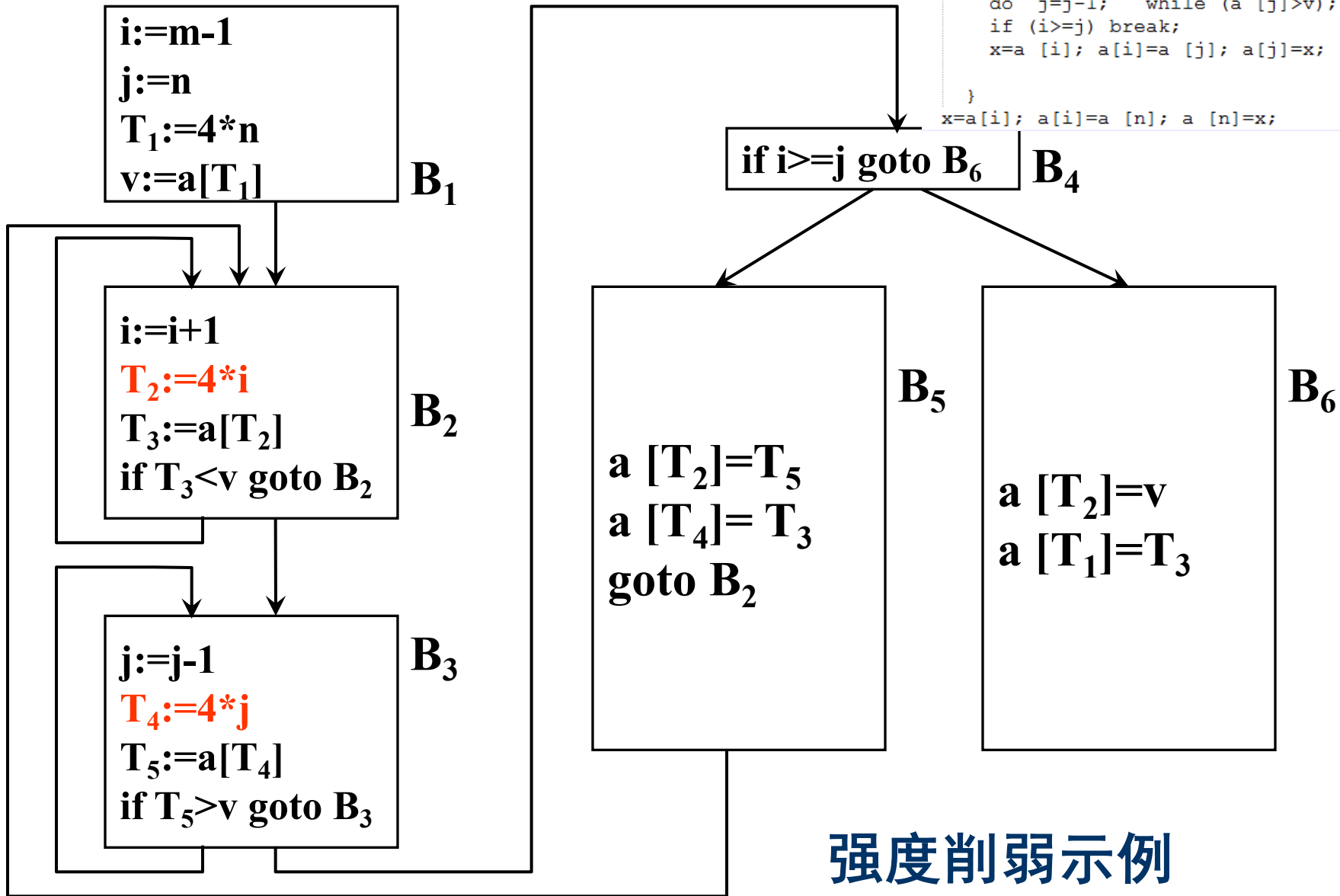
## 强度削弱



```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```

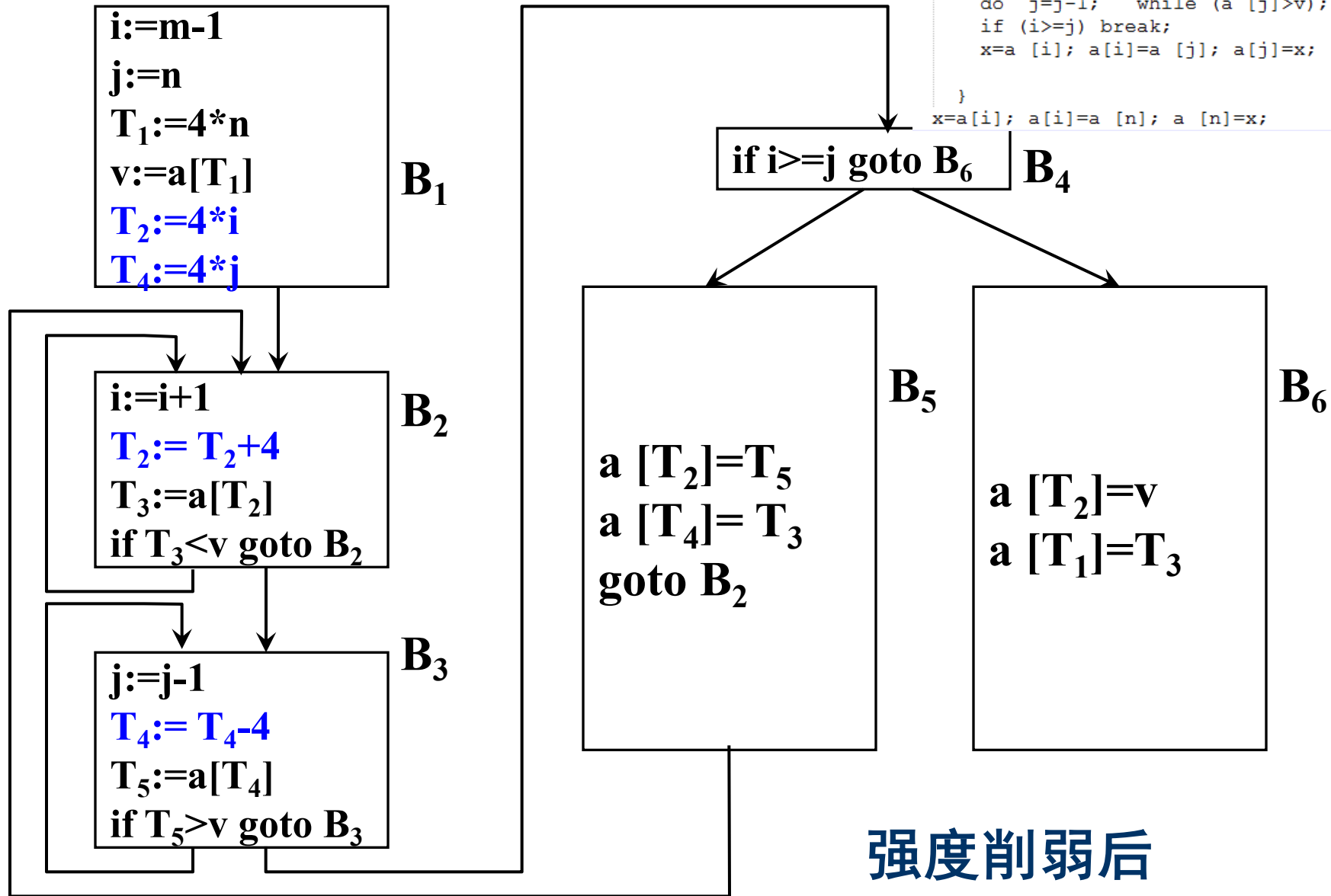


强度削弱示例

```

i:=m-1; j:=n; v:=a [n];
while (1) {
  do i=i+1; while (a [i]<v);
  do j=j-1; while (a [j]>v);
  if (i>=j) break;
  x=a [i]; a[i]=a [j]; a[j]=x;
}
x=a[i]; a[i]=a [n]; a [n]=x;

```



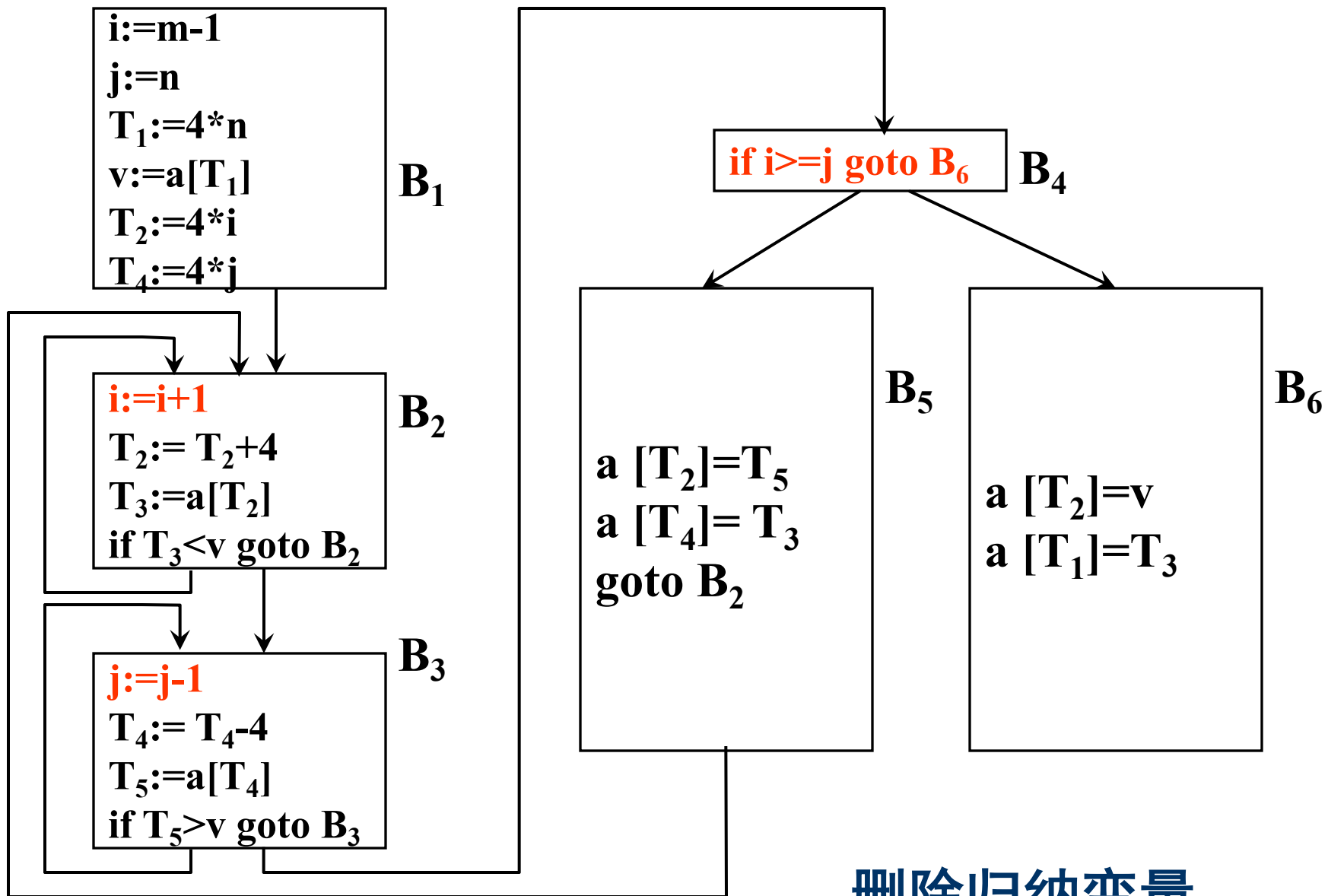
# 优化技术示例

## ■ 优化的种类：

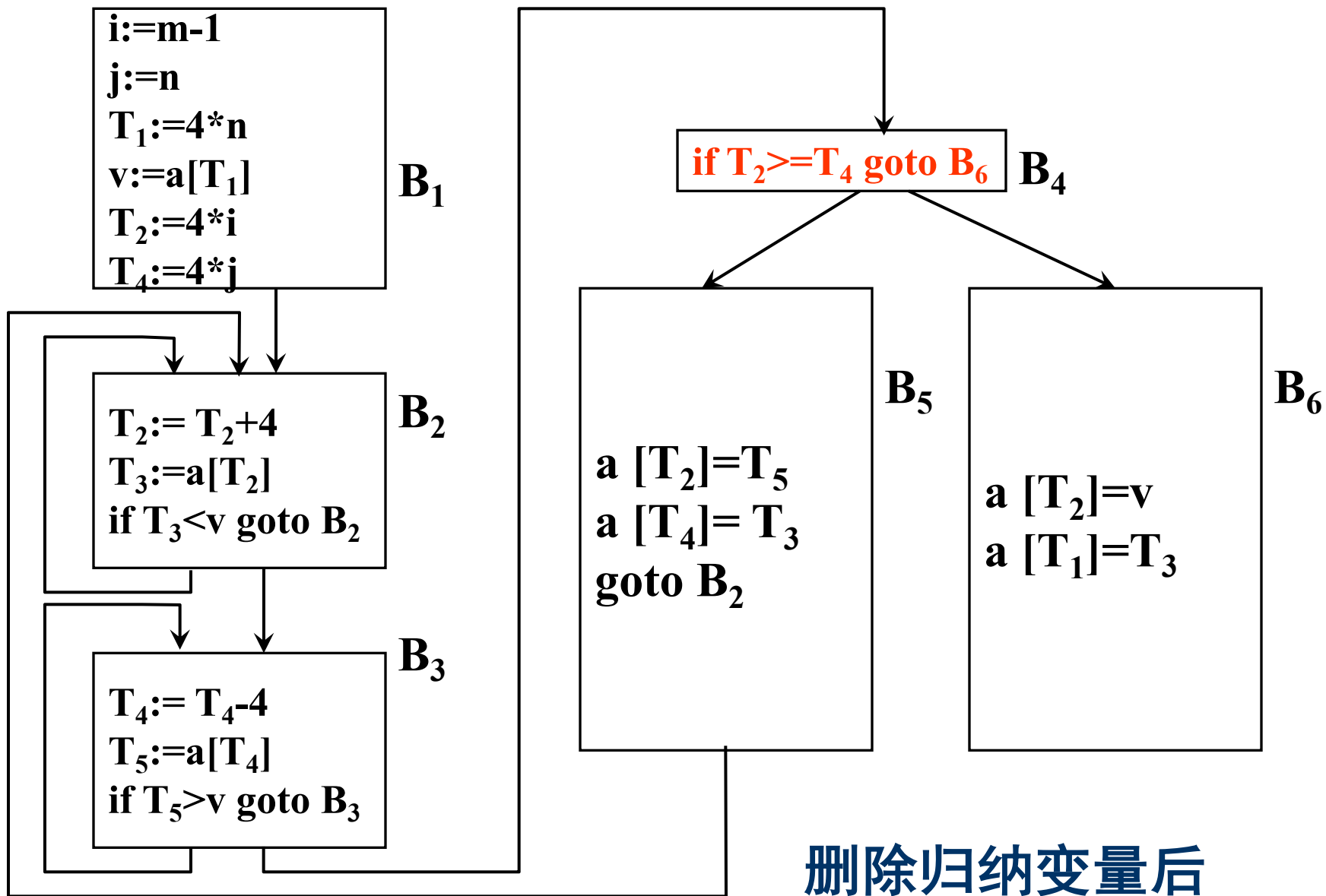
- 删除多余运算(或称删除公用子表达式)
- 复写传播
- 删除无用赋值
- 强度削弱
- 删除归纳变量

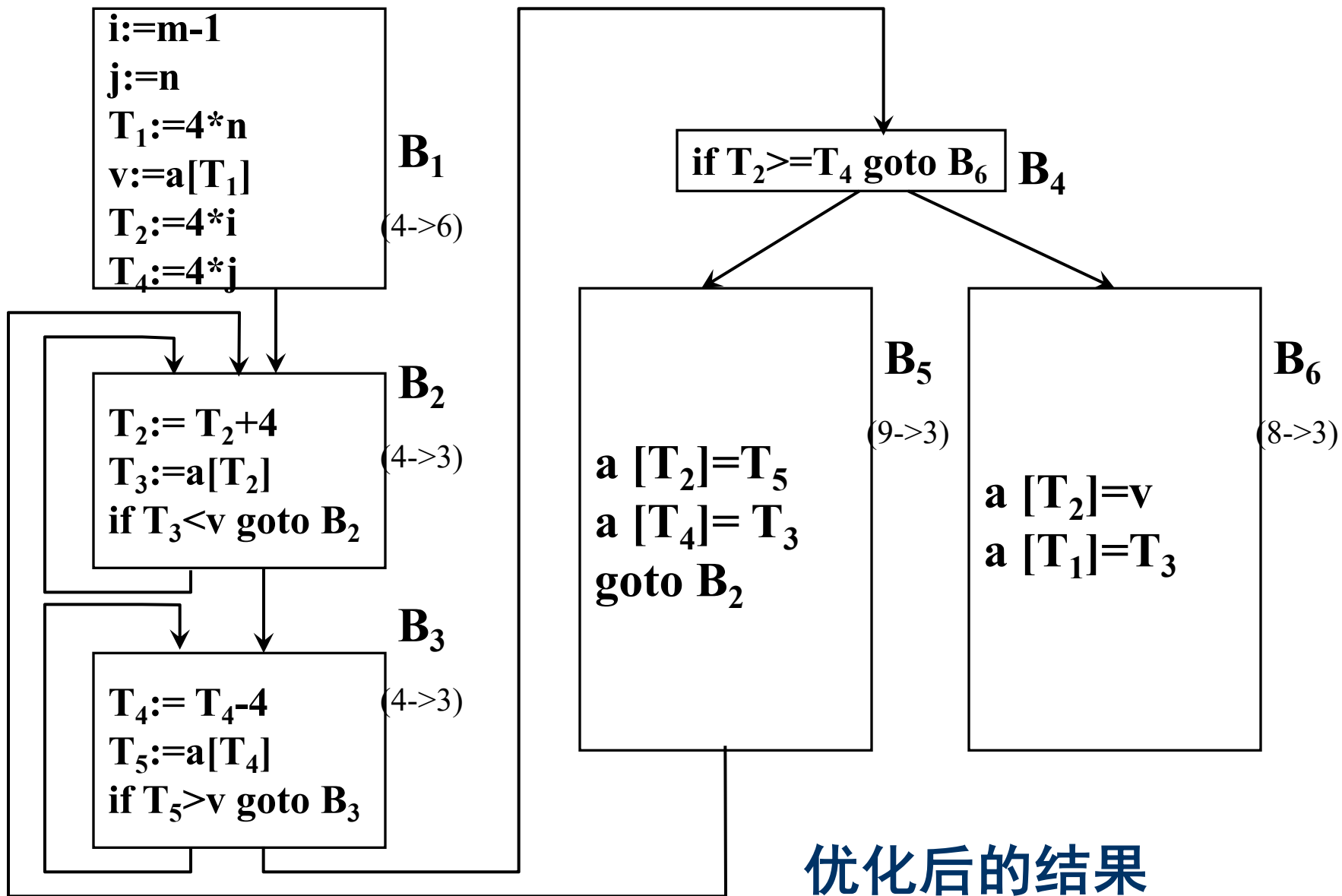
# 删除归纳变量

- $B_2$ 中的 $i$ 与 $T_2$ 的值保持着线性关系
- $B_3$ 中的 $j$ 与 $T_4$ 的值保持着线性关系
- 此种变量称之为归纳变量
- 对该类变量也可进行优化



删除归纳变量







# 内容线索

## ✓ 概述

- 删除多余运算(或称删除公用子表达式)
- 复写传播
- 删除无用赋值
- 代码外提
- 强度削弱
- 变换循环控制条件
- 合并已知量

## ✓ 局部优化

## ✓ 循环优化

# 问题

- 什么是基本块？
- 怎样划分基本块？
- 在基本块中可以进行哪些优化？
- 怎样进行局部优化？

# 基本块及相关概念

- **基本块**：指程序中一顺序执行语句序列，其中只有一个入口和一个出口。入口就是其中第一个语句，出口就是其中最后一个语句。
- 如果一条三地址语句为 $x:=y+z$ ，则称对x**定值**并**引用**y和z。
- 在一个基本块中的一个名字，所谓在程序中的某个给定点是**活跃**的，是指如果在程序中(包括在本基本块或在其它基本块中)它的值在该点以后被引用。

# 局部优化

- 局限于基本块范围内的优化称为**基本块内的优化**，或称**局部优化**。
- 在一个基本块内通常可以实行下面的优化：
  - 删除公共子表达式
  - 删除无用赋值
  - 合并已知量
  - 临时变量改名
  - 交换语句的位置
  - 代数变换

# 划分基本块

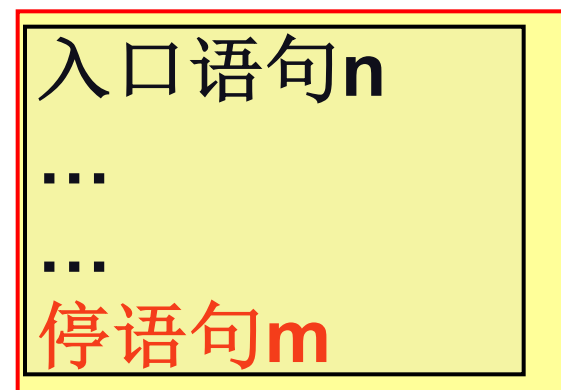
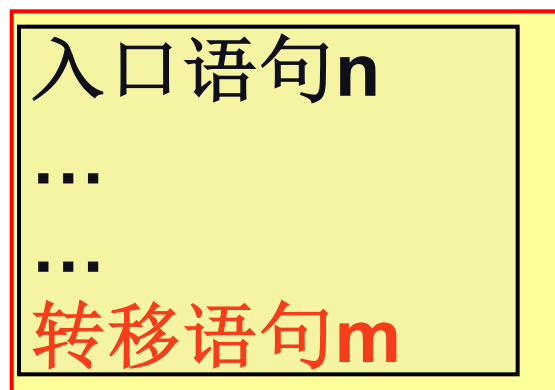
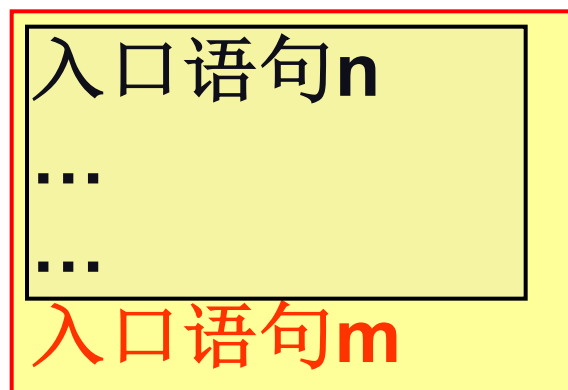
## ■ 划分四元式程序为基本块的算法：

1. 求出四元式程序中各个基本块的入口语句：
  - 1) 程序第一个语句，或
  - 2) 能由条件转移语句或无条件转移语句转移到的语句，或
  - 3) 紧跟在条件转移语句后面的语句。

# 划分基本块

## ■ 划分四元式程序为基本块的算法：

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到**下一入口语句**（不包括该入口语句）或到**一转移语句**（包括该转移语句）或**一停语句**（包括该停语句）之间的语句序列组成的。



3. 凡未被纳入某一基本块中的语句，可以从程序中删除。

## 例. 划分基本块

(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

1. 求出四元式程序中各个基本块的入口语句:

- 1) 程序第一个语句, 或
- 2) 能由条件转移语句或无条件转移语句转移到的语句, 或
- 3) 紧跟在条件转移语句后面的语句。

## 例. 划分基本块

(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**

1. 求出四元式程序中各个基本块的入口语句:

- 1) 程序第一个语句, 或
- 2) 能由条件转移语句或无条件转移语句转移到的语句, 或
- 3) 紧跟在条件转移语句后面的语句。



## 例. 划分基本块

(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**

1. 求出四元式程序中各个基本块的入口语句:

- 1) 程序第一个语句, 或
- 2) 能由条件转移语句或无条件转移语句转移到的语句, 或
- 3) 紧跟在条件转移语句后面的语句。

## 例. 划分基本块

(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**

1. 求出四元式程序中各个基本块的入口语句:

- 1) 程序第一个语句, 或
- 2) 能由条件转移语句或无条件转移语句转移到的语句, 或
- 3) 紧跟在条件转移语句后面的语句。

## 例. 划分基本块

(1) read X

(2) read Y

(3)  $R := X \bmod Y$

(4) if  $R=0$  goto (8)

(5)  $X := Y$

(6)  $Y := R$

(7) goto (3)

(8) write Y

(9) halt

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句)、或一停语句(包括该停语句)之间的语句序列组成的。

问题：语句(8)和(9)是否会被执行？

例. 划分基本块

(1)	<b>read X</b>
(2)	read Y
(3)	<b>R:=X mod Y</b>
(4)	if R=0 goto (10)
(5)	<b>X:=Y</b>
(6)	Y:=R
(7)	goto (3)
(8)	X:=Y+1
(9)	Y:=X-1
(10)	<b>write Y</b>
(11)	halt

# 随堂作业 P306 2

划分基本块，并作出其程序流图

```
(1)      read A, B
(2)      F:=1
(3)      C:=A*A
(4)      D:=B*B
(5)      if C<D goto L1
(6)      E:=A*A
(7)      F:=F+1
(8)      E:=E+F
(9)      write E
(10)     halt
(11) L1: E:=B*B
(12)     F:=F+2
(13)     E:=E+F
(14)     write E
(15)     if E>100 goto L2
(16)     halt
(17) L2: F:=F-1
(18)     goto L1
```

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句。

## 划分基本块，并作出其程序流图

```
(1)      read A, B
(2)      F:=1
(3)      C:=A*A
(4)      D:=B*B
(5)      if C<D goto L1
(6)      E:=A*A
(7)      F:=F+1
(8)      E:=E+F
(9)      write E
(10)     halt
(11) L1: E:=B*B
(12)     F:=F+2
(13)     E:=E+F
(14)     write E
(15)     if E>100 goto L2
(16)     halt
(17) L2: F:=F-1
(18)     goto L1
```

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句。

## 划分基本块，并作出其程序流图

```
(1)      read A, B
(2)      F:=1
(3)      C:=A*A
(4)      D:=B*B
(5)      if C<D goto L1
(6)      E:=A*A
(7)      F:=F+1
(8)      E:=E+F
(9)      write E
(10)     halt
(11) L1: E:=B*B
(12)     F:=F+2
(13)     E:=E+F
(14)     write E
(15)     if E>100 goto L2
(16)     halt
(17) L2: F:=F-1
(18)     goto L1
```

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句。

## 划分基本块，并作出其程序流图

```
(1)      read A, B
(2)      F:=1
(3)      C:=A*A
(4)      D:=B*B
(5)      if C<D goto L1
(6)      E:=A*A
(7)      F:=F+1
(8)      E:=E+F
(9)      write E
(10)     halt
(11) L1: E:=B*B
(12)     F:=F+2
(13)     E:=E+F
(14)     write E
(15)     if E>100 goto L2
(16)     halt
(17) L2: F:=F-1
(18)     goto L1
```

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句。



## 划分基本块，并作出其程序流图

(1)	<b>read A, B</b>	<b>B1</b>
(2)	F:=1	
(3)	C:=A*A	
(4)	D:=B*B	
(5)	if C<D goto L1	

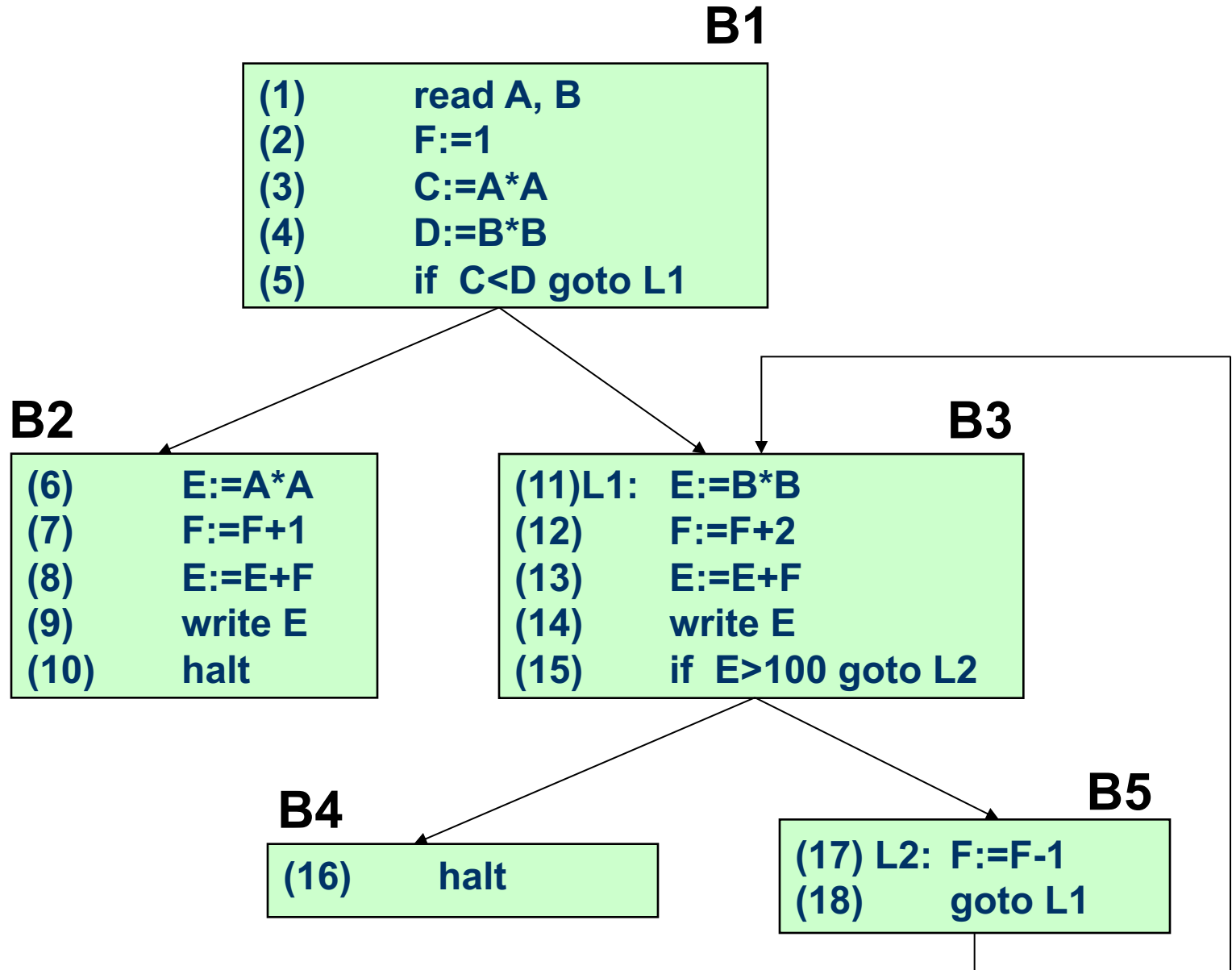
(6)	<b>E:=A*A</b>	<b>B2</b>
(7)	F:=F+1	
(8)	E:=E+F	
(9)	write E	
(10)	halt	

(11)	<b>L1: E:=B*B</b>	<b>B3</b>
(12)	F:=F+2	
(13)	E:=E+F	
(14)	write E	
(15)	if E>100 goto L2	

(16)	<b>halt</b>	<b>B4</b>
------	-------------	-----------

(17)	<b>L2: F:=F-1</b>	<b>B5</b>
(18)	goto L1	

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句)、或一停语句(包括该停语句)之间的语句序列组成的。



# 优化措施

## ■ 在一个基本块内通常可以实行下面的优化：

- ☐ 删除公共子表达式
- ☐ 删除无用赋值
- ☐ 合并已知量
- ☐ 临时变量改名
- ☐ 交换语句的位置
- ☐ 代数变换

# 优化措施

- 合并已知量

$T_1 := 2 \quad \dots \quad T_2 := 4 * T_1$  变换成  $T_2 := 8$

- 临时变量改名

$T := b + c$  其中T是一个临时变量名。改成:  $S := b + c$

- 交换语句的位置

$T_1 := b + c$

$T_2 := x + y$

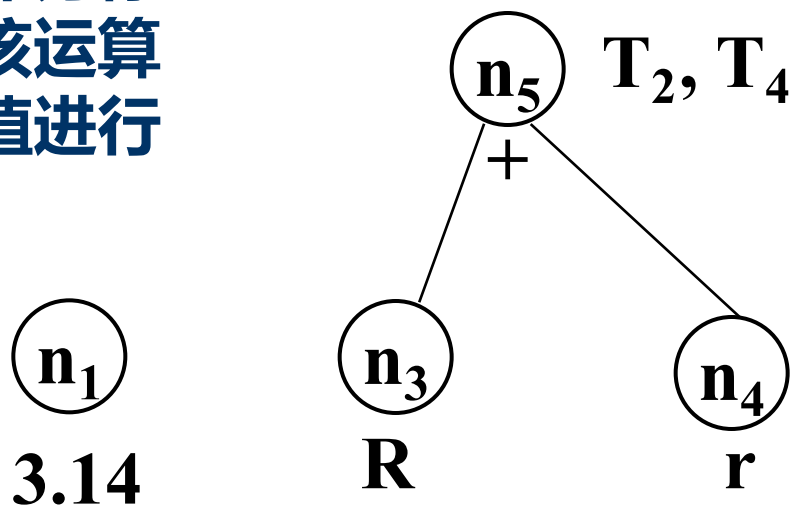
- 代数变换

(1) 删除  $x := x + 0$  或  $x := x * 1$

(2)  $x := y ** 2$  (调用函数完成) 变换成  $x := y * y$

# 基本块的DAG表示

- 基本块的DAG是一种带有下述标记或附加信息的DAG
  - 图的叶结点以一标识符或常数作为标记，表示该结点代表该变量或常数的值；
  - 图的内部结点以一运算符作为标记，表示该结点代表应用该运算符对其后继结点所代表的值进行运算的结果；
  - 图中各个结点上可能附加一个或多个标识符(称附加标识符)表示这些变量具有该结点所代表的值。



# 基本块的DAG表示及应用

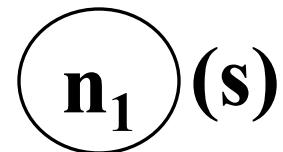
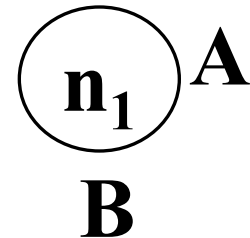
## ■ 与各四元式相对应的DAG结点形式:

### 四元式

0型:  $A := B$   
( $:=$ ,  $B$ ,  $-$ ,  $A$ )

0型: goto (s)  
(j,  $-$ ,  $-$ , (s))

### DAG 图

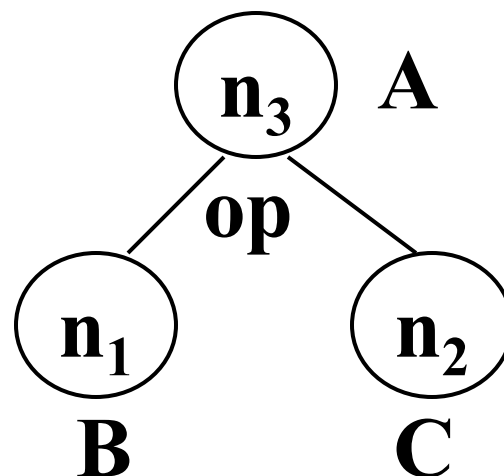
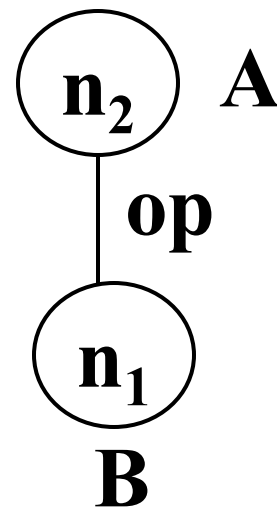


# 四元式

1型:  $A := \text{op } B$   
(op, B, -, A)

2型:  $A := B \text{ op } C$   
(op, B, C, A)

# DAG 图

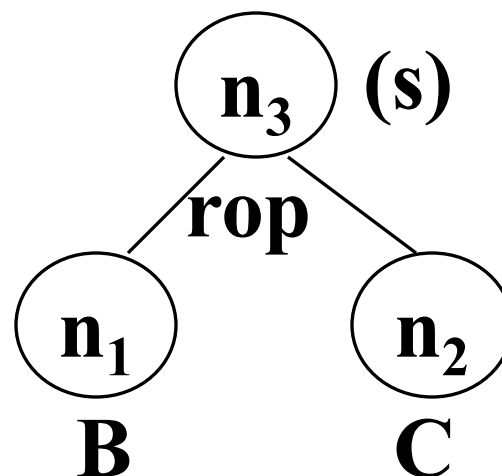
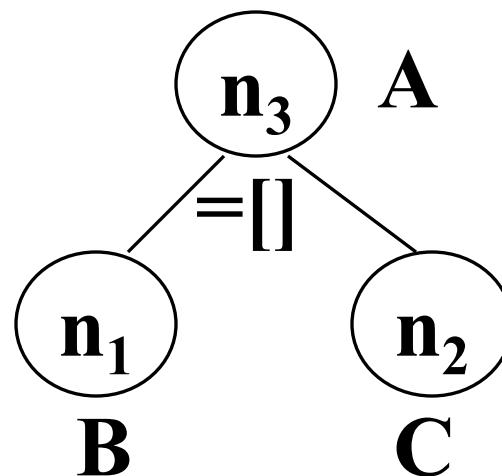


## 四元式

2型:  $A := B[C]$   
( $=[]$ ,  $B$ ,  $C$ ,  $A$ )

2型: if  $B$  rop  $C$  goto  $(s)$   
(jrop,  $B$ ,  $C$ ,  $(s)$ )

## DAG 图

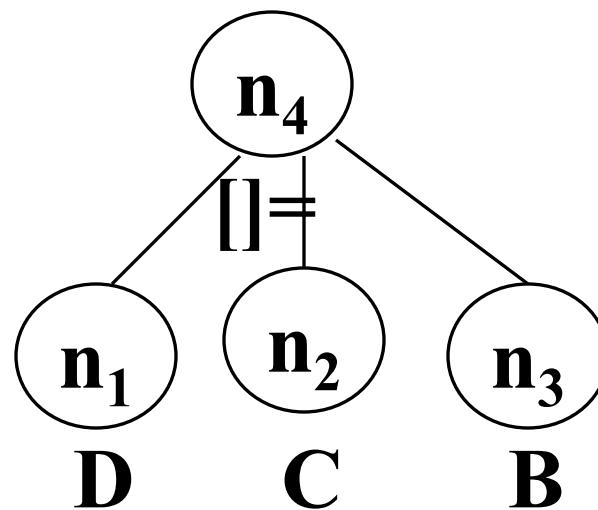




# 四元式

3型:  $D[C] := B$   
( $[] =$ ,  $B$ ,  $-$ ,  
 $D[C]$ )

# DAG 图



## 例. 试构造以下基本块G的DAG

(1)  $T_0 := 3.14$

(2)  $T_1 := 2 * T_0$

(3)  $T_2 := R + r$

(4)  $A := T_1 * T_2$

(5)  $B := A$

(6)  $T_3 := 2 * T_0$

(7)  $T_4 := R + r$

(8)  $T_5 := T_3 * T_4$

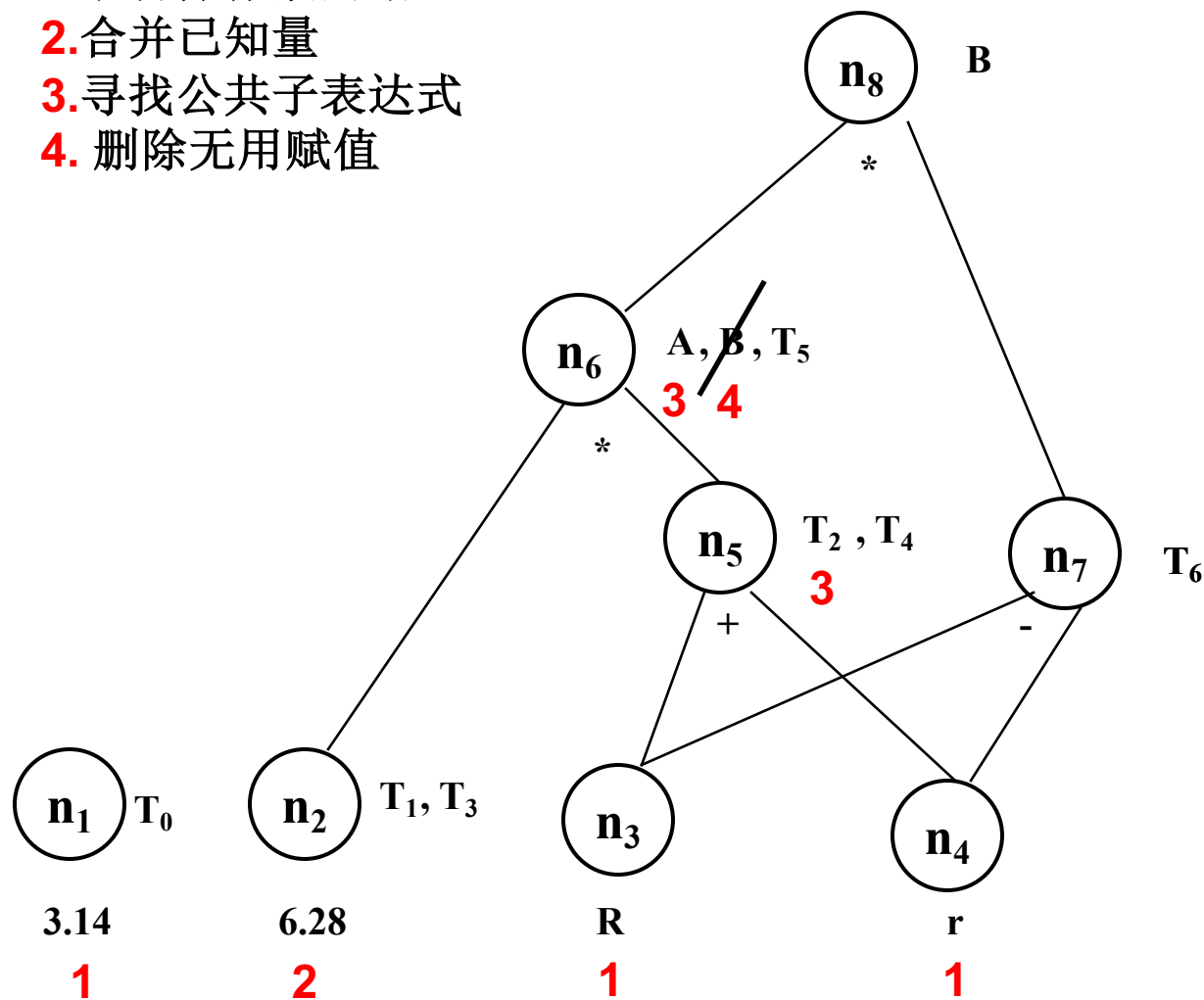
(9)  $T_6 := R - r$

(10)  $B := T_5 * T_6$

- (1)  $T_0 := 3.14$
- (2)  $T_1 := 2 * T_0$
- (3)  $T_2 := R + r$
- (4)  $A := T_1 * T_2$
- (5)  $B := A$
- (6)  $T_3 := 2 * T_0$
- (7)  $T_4 := R + r$
- (8)  $T_5 := T_3 * T_4$
- (9)  $T_6 := R - r$
- (10)  $B := T_5 * T_6$

## 构建DAG算法步骤及示例

1. 准备操作数的结点
2. 合并已知量
3. 寻找公共子表达式
4. 删除无用赋值

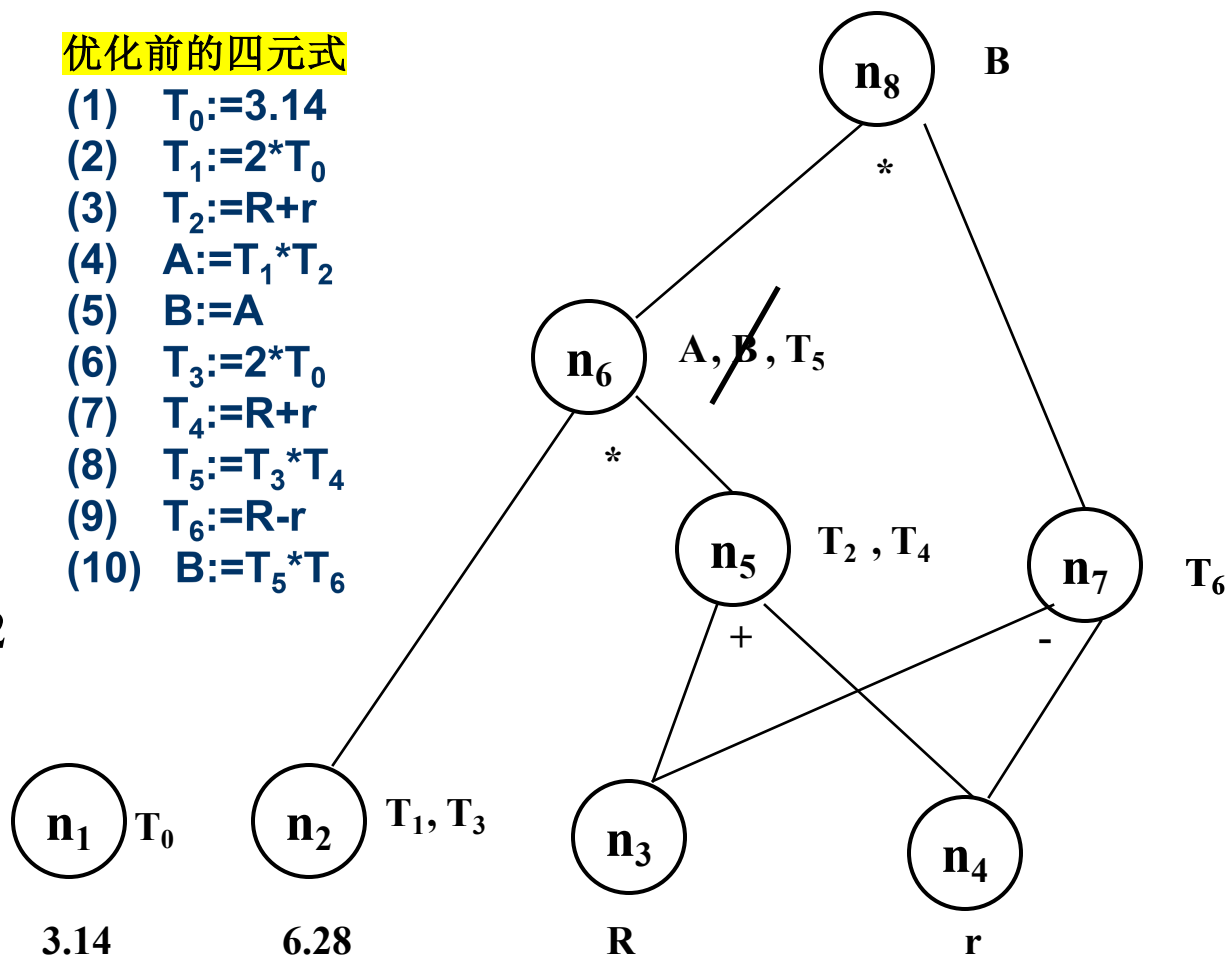


## □ 优化后的四元式

- (1)  $T_0 := 3.14$
- (2)  $T_1 := 6.28$
- (3)  $T_3 := 6.28$
- (4)  $T_2 := R + r$
- (5)  $T_4 := T_2$
- (6)  $A := 6.28 * T_2$
- (7)  $T_5 := A$
- (8)  $T_6 := R - r$
- (9)  $B := A * T_6$

### 优化前的四元式

- (1)  $T_0 := 3.14$
- (2)  $T_1 := 2 * T_0$
- (3)  $T_2 := R + r$
- (4)  $A := T_1 * T_2$
- (5)  $B := A$
- (6)  $T_3 := 2 * T_0$
- (7)  $T_4 := R + r$
- (8)  $T_5 := T_3 * T_4$
- (9)  $T_6 := R - r$
- (10)  $B := T_5 * T_6$



# 优化后的四元式——若只有A和B是出基本块之后活跃的(数据流分析)

(1)  $T_2 := R + r$

(2)  $A := 6.28 * T_2$

(3)  $T_6 := R - r$

(4)  $B := A * T_6$

(1)  $S_1 := R + r$

(2)  $A := 6.28 * S_1$

(3)  $S_2 := R - r$

(4)  $B := A * S_2$

(1)  $T_0 := 3.14$

(2)  $T_1 := 6.28$

(3)  $T_3 := 6.28$

(4)  $T_2 := R + r$

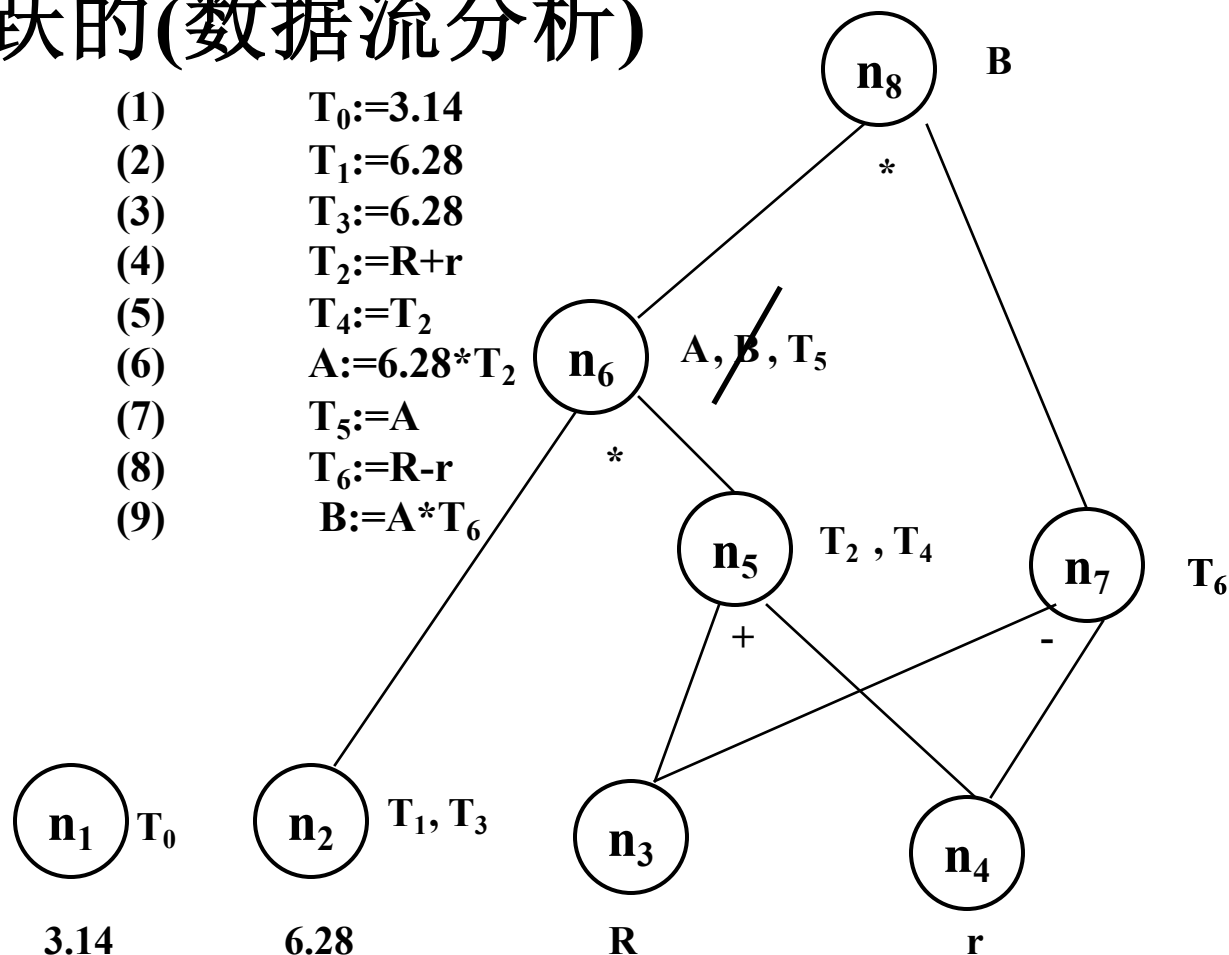
(5)  $T_4 := T_2$

(6)  $A := 6.28 * T_2$

(7)  $T_5 := A$

(8)  $T_6 := R - r$

(9)  $B := A * T_6$



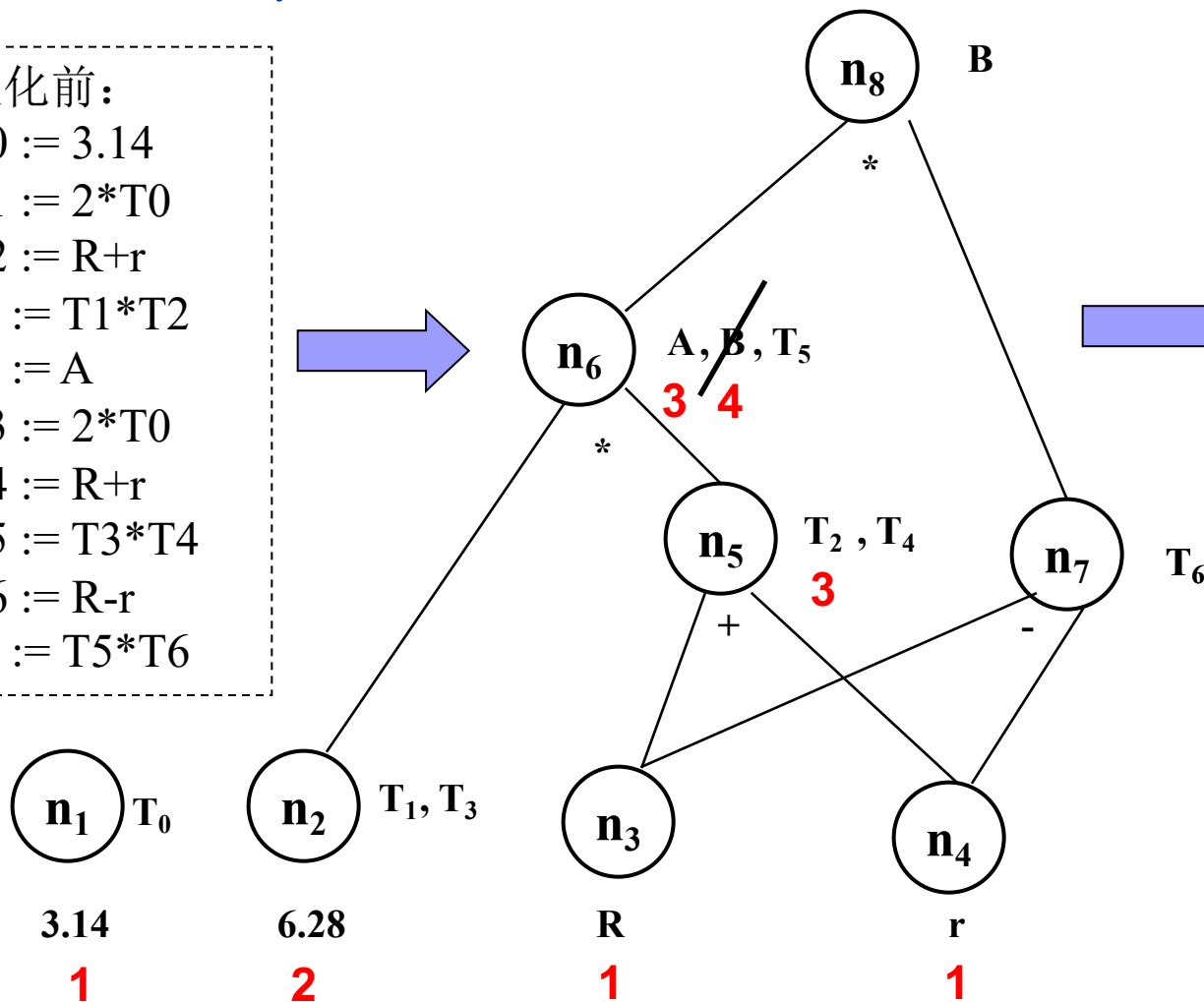
# DAG算法效果

优化前:

```
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
```

假设: 若只有A  
和B是出基本块  
之后活跃的

优化后:  
S1:=R+r  
A:=6.28\*S1  
S2:=R-r  
B:=A\*S2



# 基本块的DAG构造算法

- 对基本块中每一四元式，依次执行以下步骤：
  1. 准备操作数的结点
  2. 合并已知量
  3. 寻找公共子表达式
  4. 删除无用赋值

# 基本块的DAG构造算法

对于不同类型代码：

(0型)  $A := B$  (1型)  $A := op\ B$  (2型)  $A := B\ op\ C$  /  $A := B[c]$

## 1. 准备操作数的结点

- 如果  $NODE(B)$  无定义，则构造一标记为  $B$  的叶结点并定义  $NODE(B)$  为这个结点；
  - 如果当前四元式是0型则记  $NODE(B)$  的值为  $n$ ，转4。
  - 如果当前四元式是1型，则转2(1)
  - 如果当前四元式是2型，则(i)如果  $NODE(C)$  无定义，则构造一标记为  $C$  的叶结点并定义  $NODE(C)$  为这个结点；(ii)转2(2)。



# 基本块的DAG构造算法

## 2.合并已知量（如果操作数都是常数，完成计算）

- (1) 如果 $\text{NODE}(B)$ 是标记为常数的叶结点，则转2(3)；否则，转3(1)。
- (2) 如果 $\text{NODE}(B)$ 和 $\text{NODE}(C)$ 都是标记为常数的叶结点，则转2(4)；否则，转3(2)。
- (3) 执行 $\text{op } B$  (即合并已知量)。令得到的新常数为 $P$ 。如果 $\text{NODE}(B)$ 是处理当前四元式时新构造出来的结点，则删除它。如果 $\text{NODE}(P)$ 无定义，则构造一用 $P$ 作标记的叶结点 $n$ 。置 $\text{NODE}(P)=n$ ，转4。
- (4) 执行 $B \text{ op } C$  (即合并已知量)。令得到的新常数为 $P$ 。如果 $\text{NODE}(B)$ 或 $\text{NODE}(C)$ 是处理当前四元式时新构造出来的结点，则删除它。如果 $\text{NODE}(P)$ 无定义，则构造一用 $P$ 作标记的叶结点 $n$ 。置 $\text{NODE}(P)=n$ ，转4。

# 基本块的DAG构造算法

## 3.寻找公共子表达式（操作数不全是常数，完成计算）

- (1) 检查DAG中是否已有一结点，其唯一后继为NODE(B)且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。
- (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

# 基本块的DAG构造算法

## 4. 删除无用赋值（完成赋值）

如果 $\text{NODE}(A)$ 无定义，则把 $A$ 附加在结点 $n$ 上并令 $\text{NODE}(A)=n$ ；否则，先把 $A$ 从 $\text{NODE}(A)$ 结点上的附加标识符集中删除（注意，如果 $\text{NODE}(A)$ 是叶结点，则其 $A$ 标记不删除）。把 $A$ 附加到新结点 $n$ 上并置 $\text{NODE}(A)=n$ 。转处理下一四元式。

# DAG 的优化

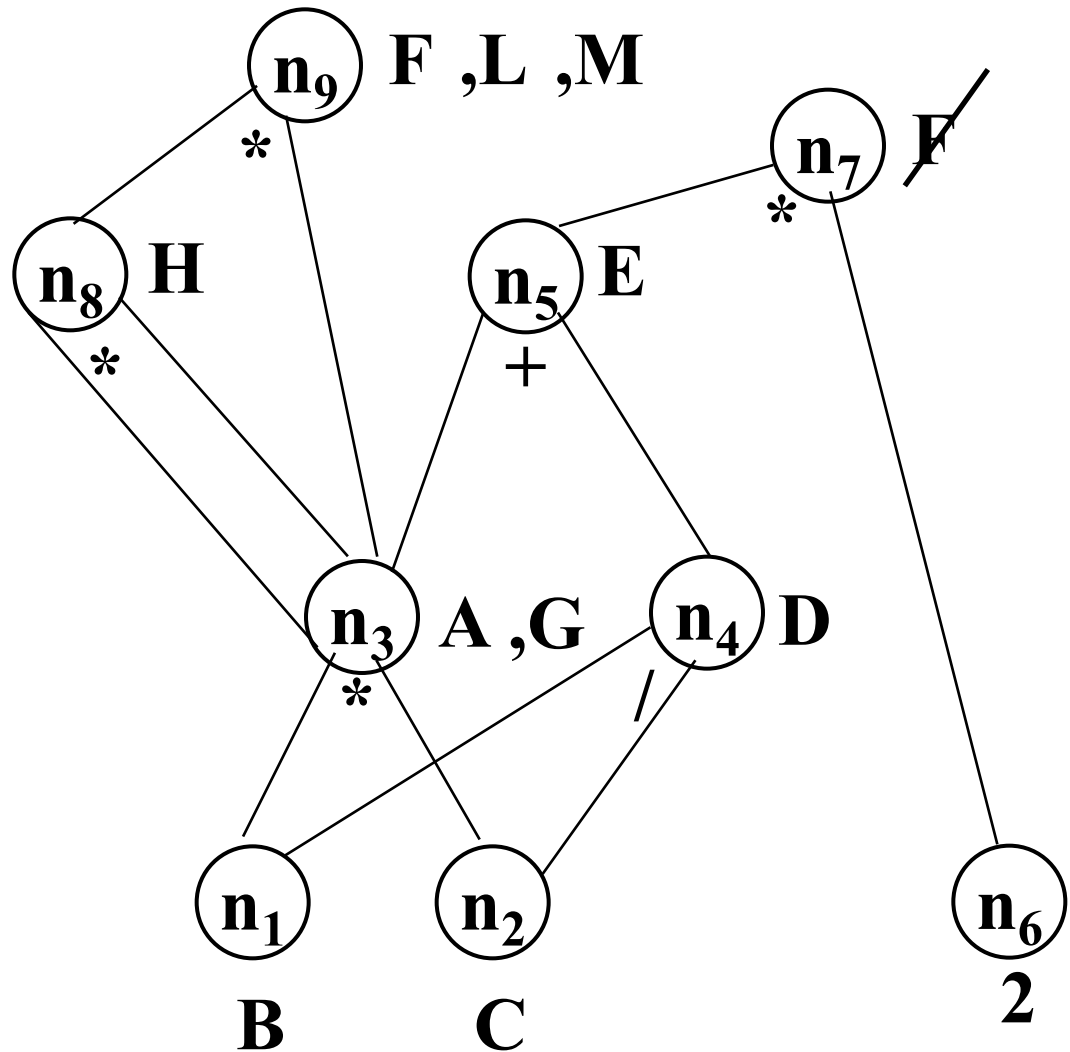
- 合并已知量
- 寻找公共子表达式
- 删除无用赋值
- 在基本块外被定值并在基本块内被引用的所有标识符，就是作为叶子结点上标记的那些标识符。
- 在基本块内被定值并且该值在基本块后面可以被引用的所有标识符，就是DAG各结点上的那些附加标识符。

# 随堂练习 P306 3( $B_1$ )

试构造以下基本块G的DAG

- (1)  $A := B * C$
- (2)  $D := B / C$
- (3)  $E := A + D$
- (4)  $F := 2 * E$
- (5)  $G := B * C$
- (6)  $H := G * G$
- (7)  $F := H * G$
- (8)  $L := F$
- (9)  $M := L$

- (1)  $A := B * C$
- (2)  $D := B / C$
- (3)  $E := A + D$
- (4)  $F := 2 * E$
- (5)  $G := B * C$
- (6)  $H := G * G$
- (7)  $F := H * G$
- (8)  $L := F$
- (9)  $M := L$



关注G, L, M

(1)  $A := B * C$

(2)  $D := B / C$

(3)  $E := A + D$

~~(4)  $F := 2 * E$~~

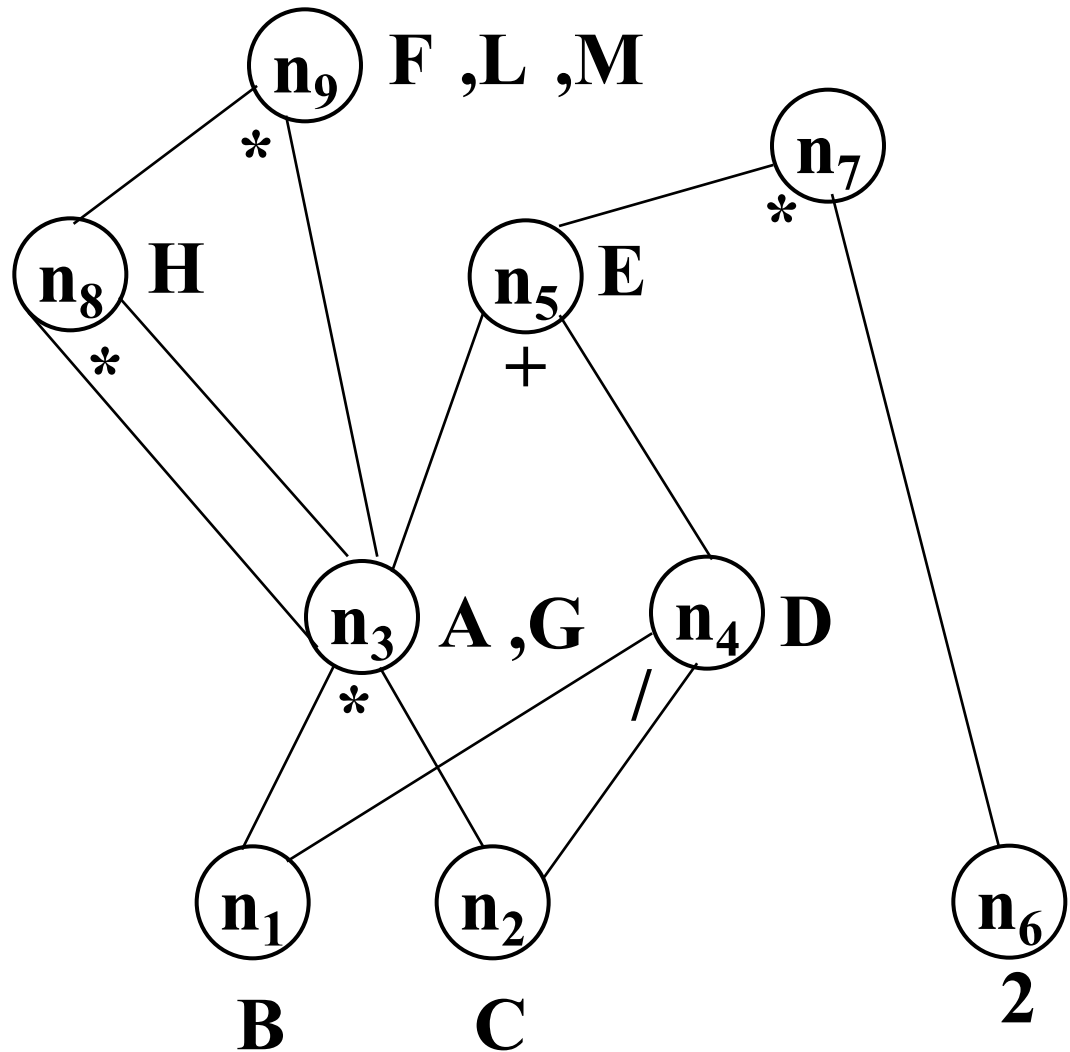
(5)  $G := A$

(6)  $H := G * G$

(7)  $F := H * G$

(8)  $L := F$

(9)  $M := L$



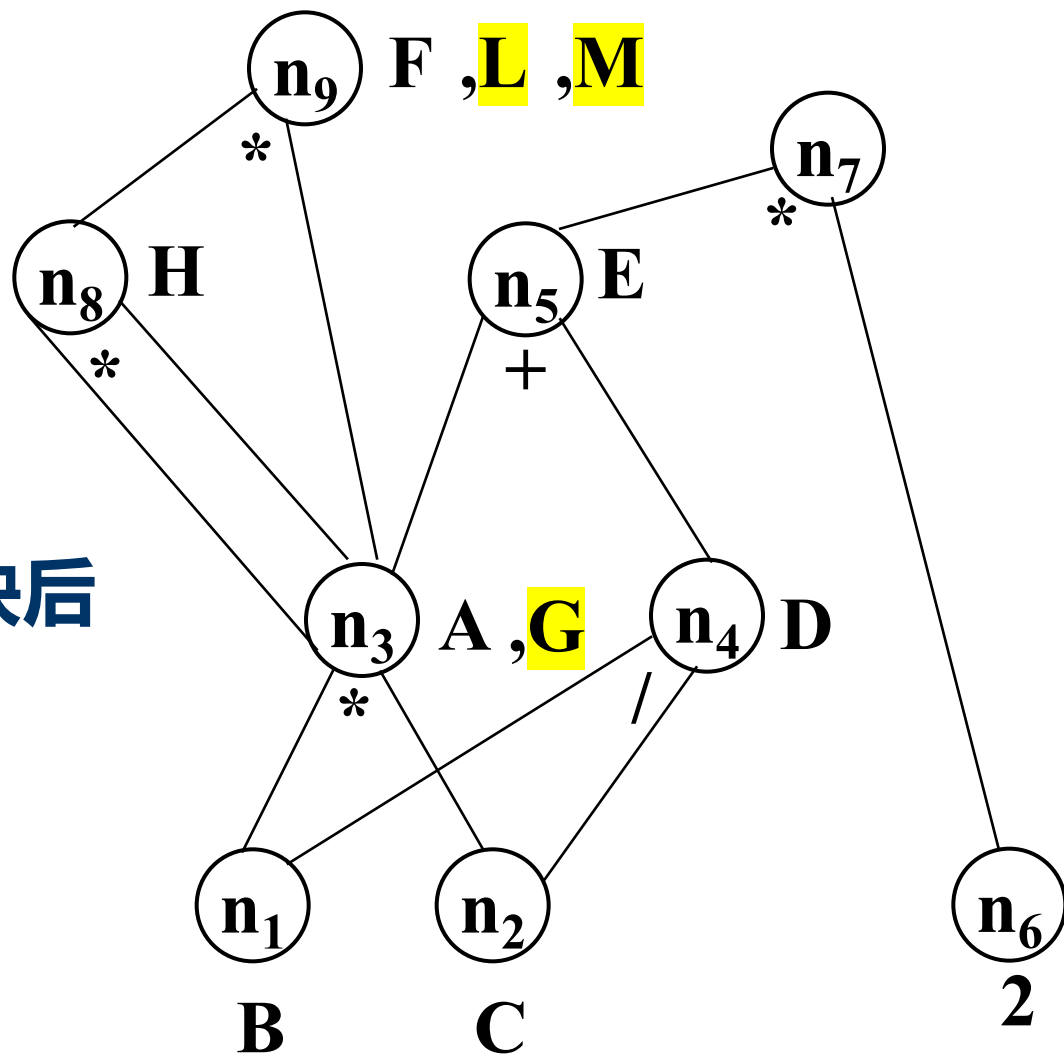
假设只有G, L, M在基本块后还要被引用

(1)  $G := B * C$

(2)  $H := G * G$

(3)  $L := H * G$

(4)  $M := L$



假设只有L在基本块后还要被引用

(1)  $G := B * C$

(2)  $H := G * G$

(3)  $L := H * G$



# 小测试

- 课后练习： P306 3(B2)
- CANVAS测试：
  - 基于DAG的优化

# 内容线索

- ✓ 概述
- ✓ 局部优化
- 循环优化

# 循环优化

- 对循环中的代码，可以实行：
  - 代码外提
  - 强度削弱
  - 删除归纳变量（变换循环控制条件）

# 代码外提

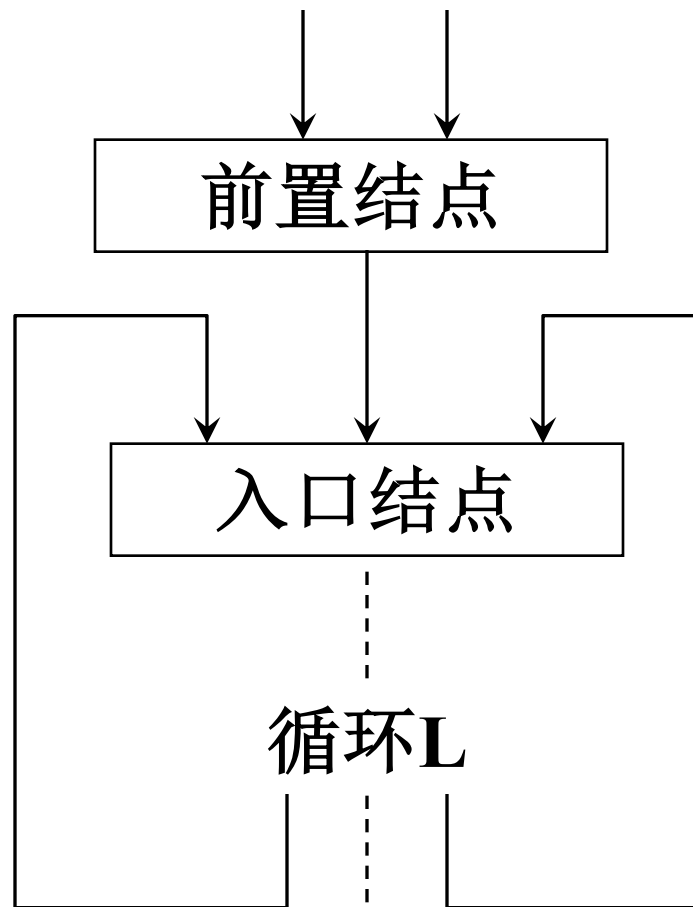
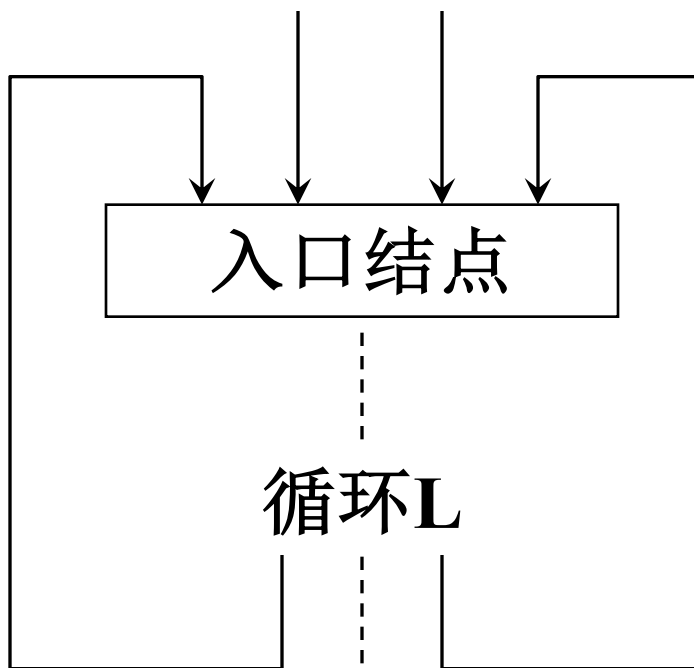
- **循环不变运算**: 对四元式 $A := B \text{ op } C$ , 若B和C是常数, 或者到达它们的B和C的定值点都在循环外。
- 变量A在某点d的**定值到达**另一点u (或称变量A的定值点d到达另一点u), 是指流图中从d有一通路到达u且该通路上没有A的其它定值。

$d : A := B \text{ op } C$

...

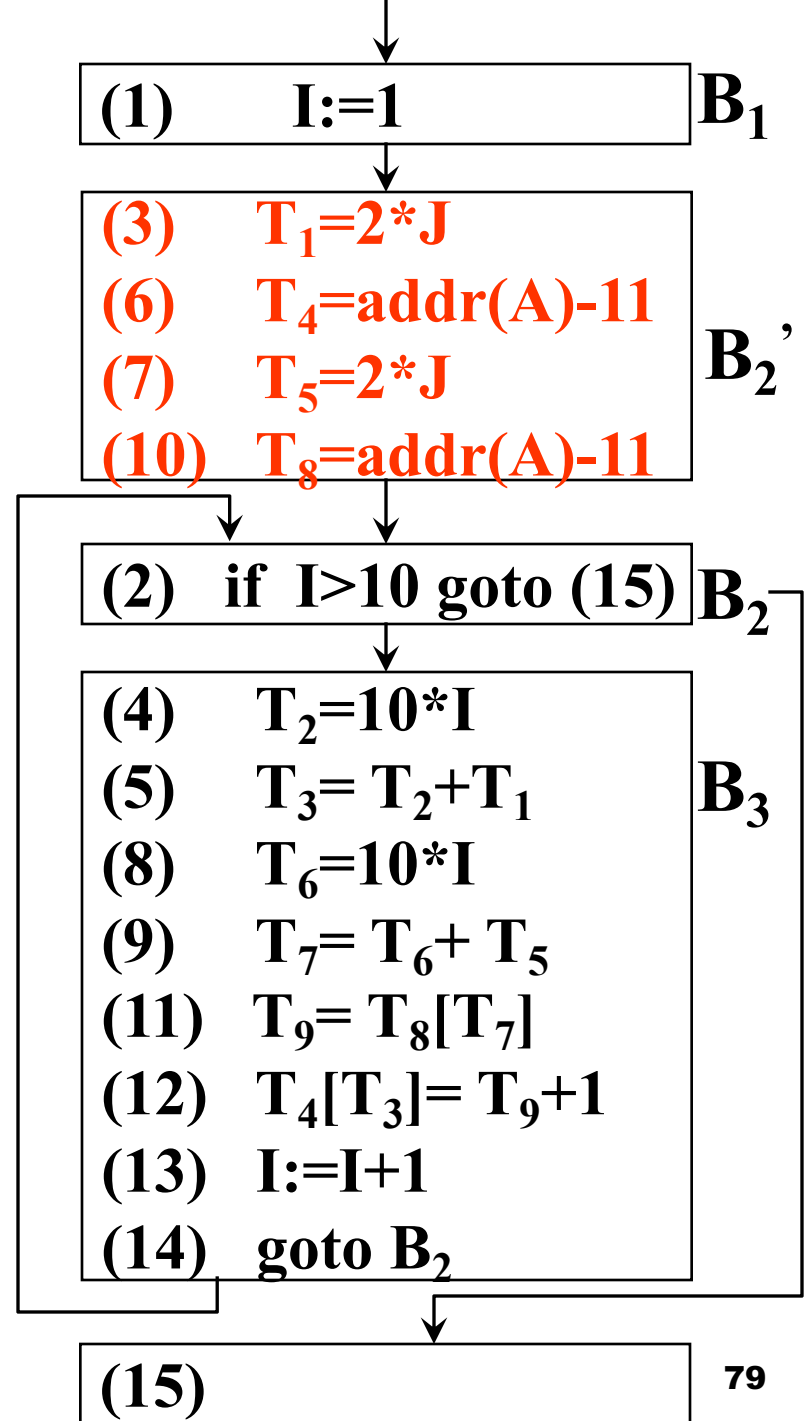
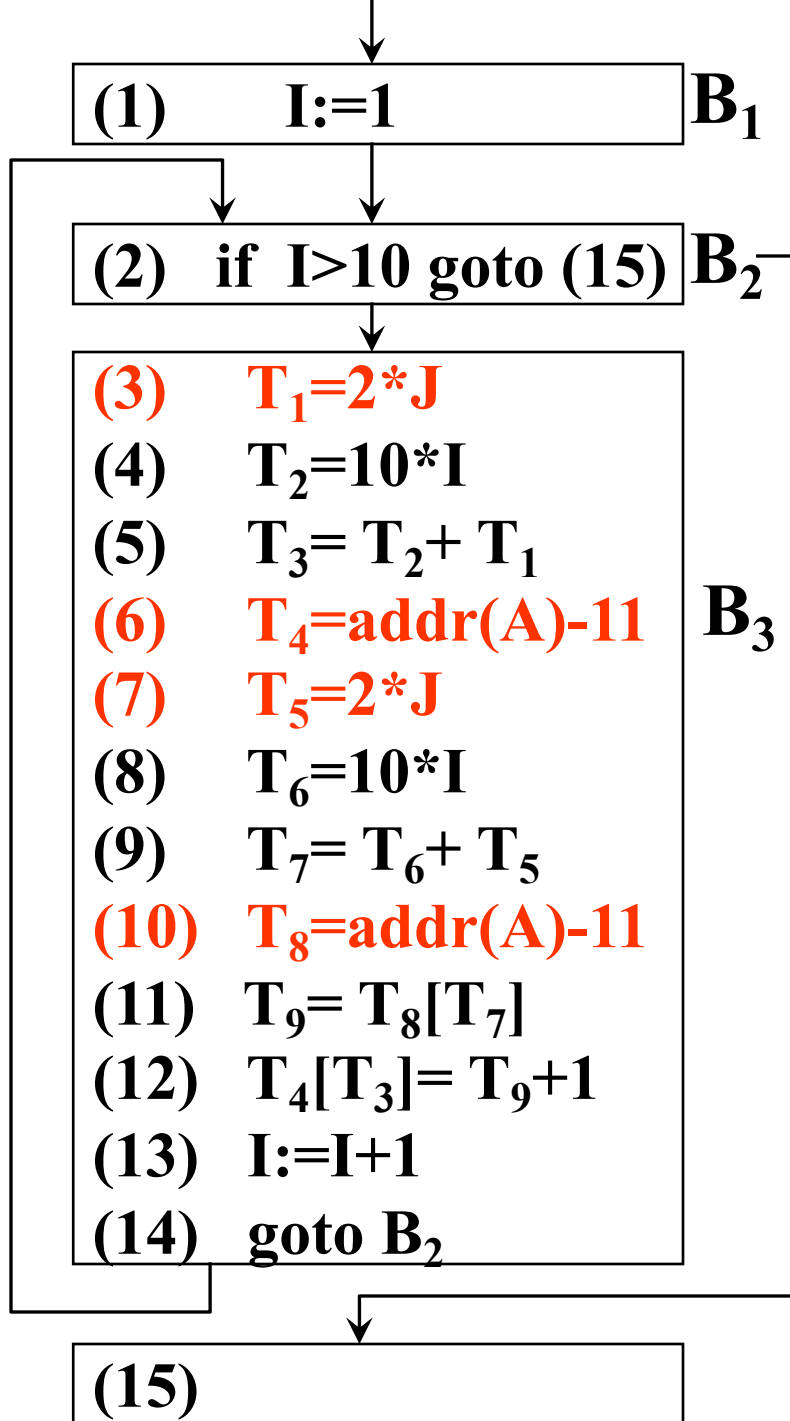
$u : D := A \text{ op } E$

- 把循环不变运算提到循环体外。



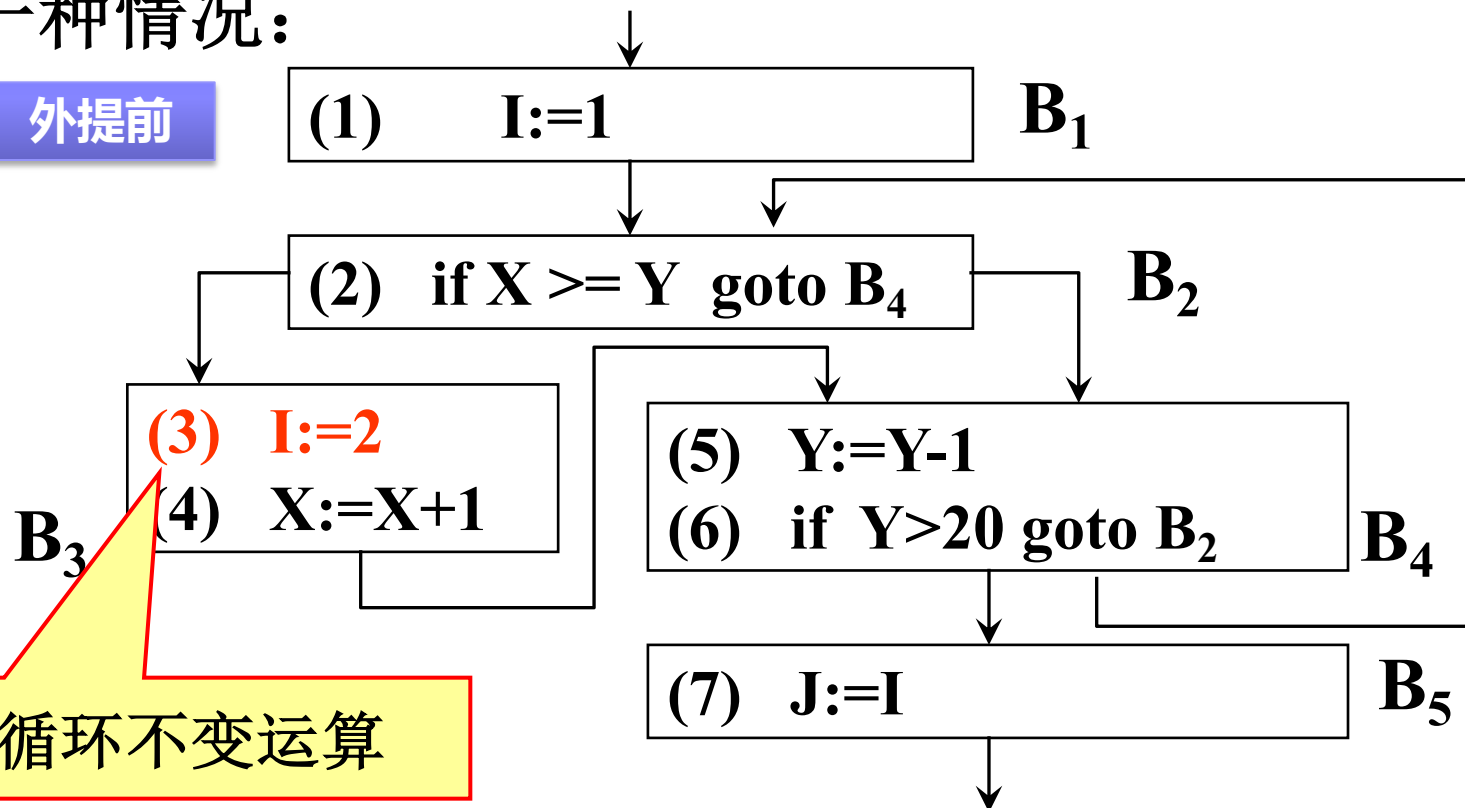


**for  $l:=1$  to 10 do  $A[l, 2*J] := A[l, 2*J] + 1$**



## 第一种情况:

外提前

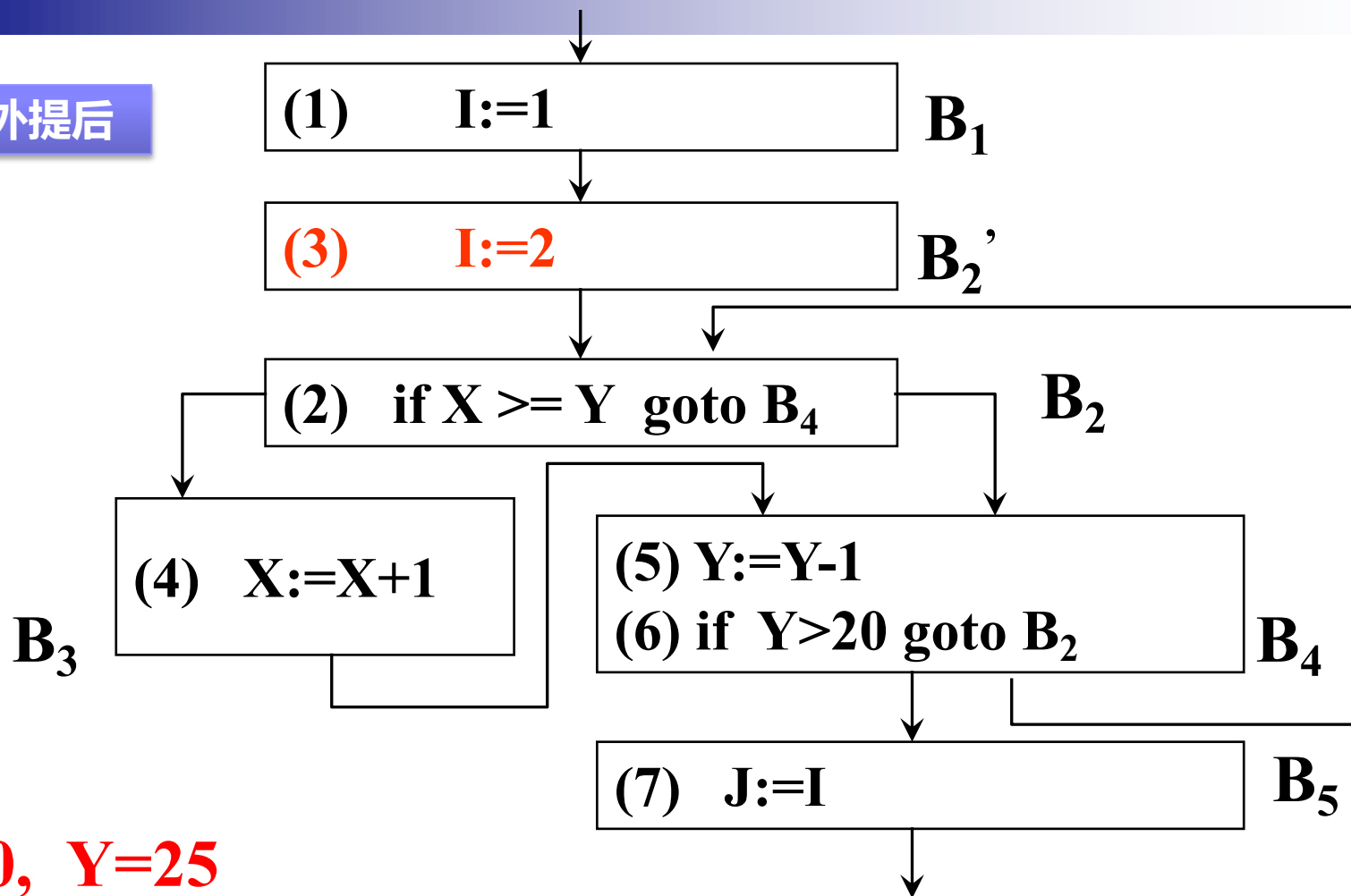


**X=30, Y=25**

**$B_1 \rightarrow B_2 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow \cdots \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$**   
**J=1, I=1**



外提后

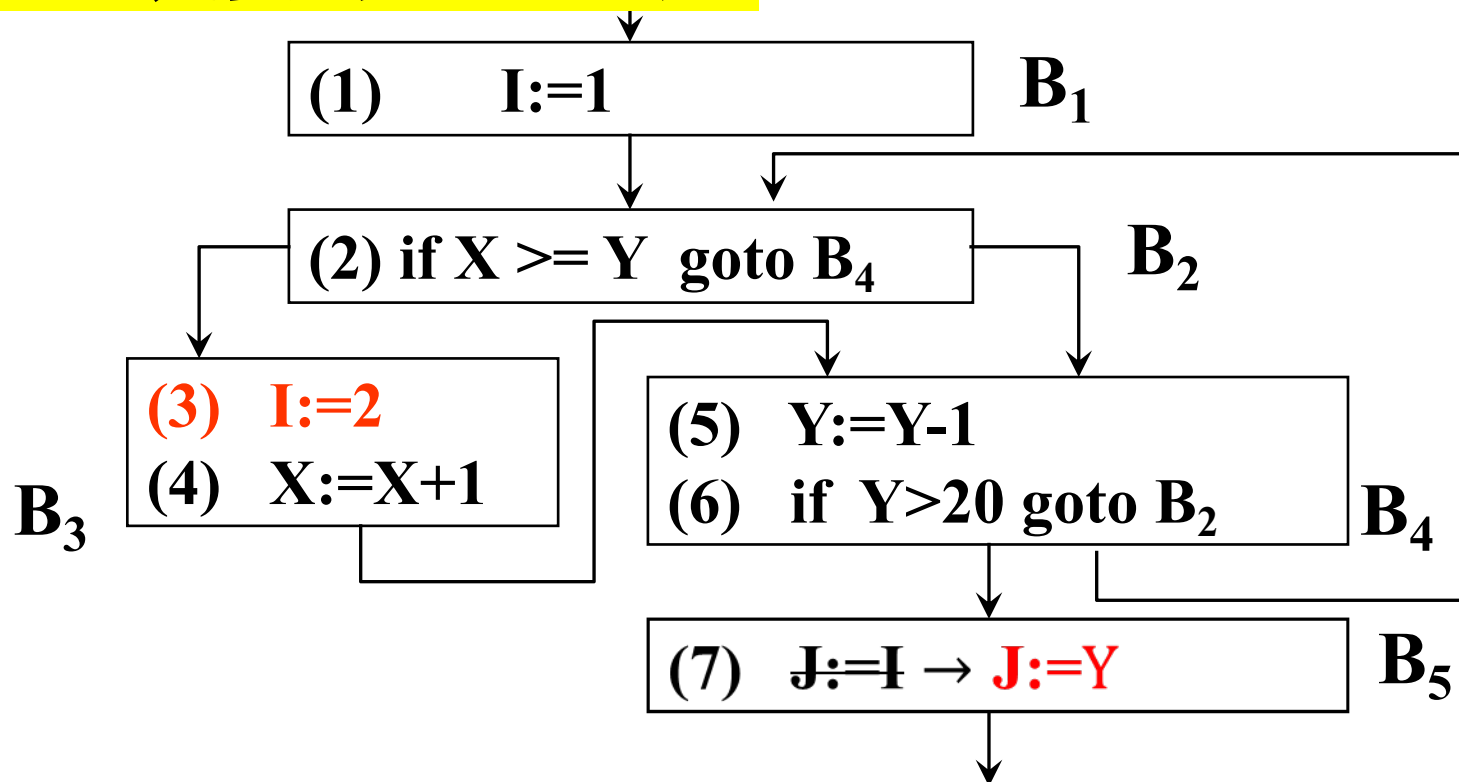


**X=30, Y=25**

**$B_1 \rightarrow B_2 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow \dots \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$**   
**J=2, I=2**

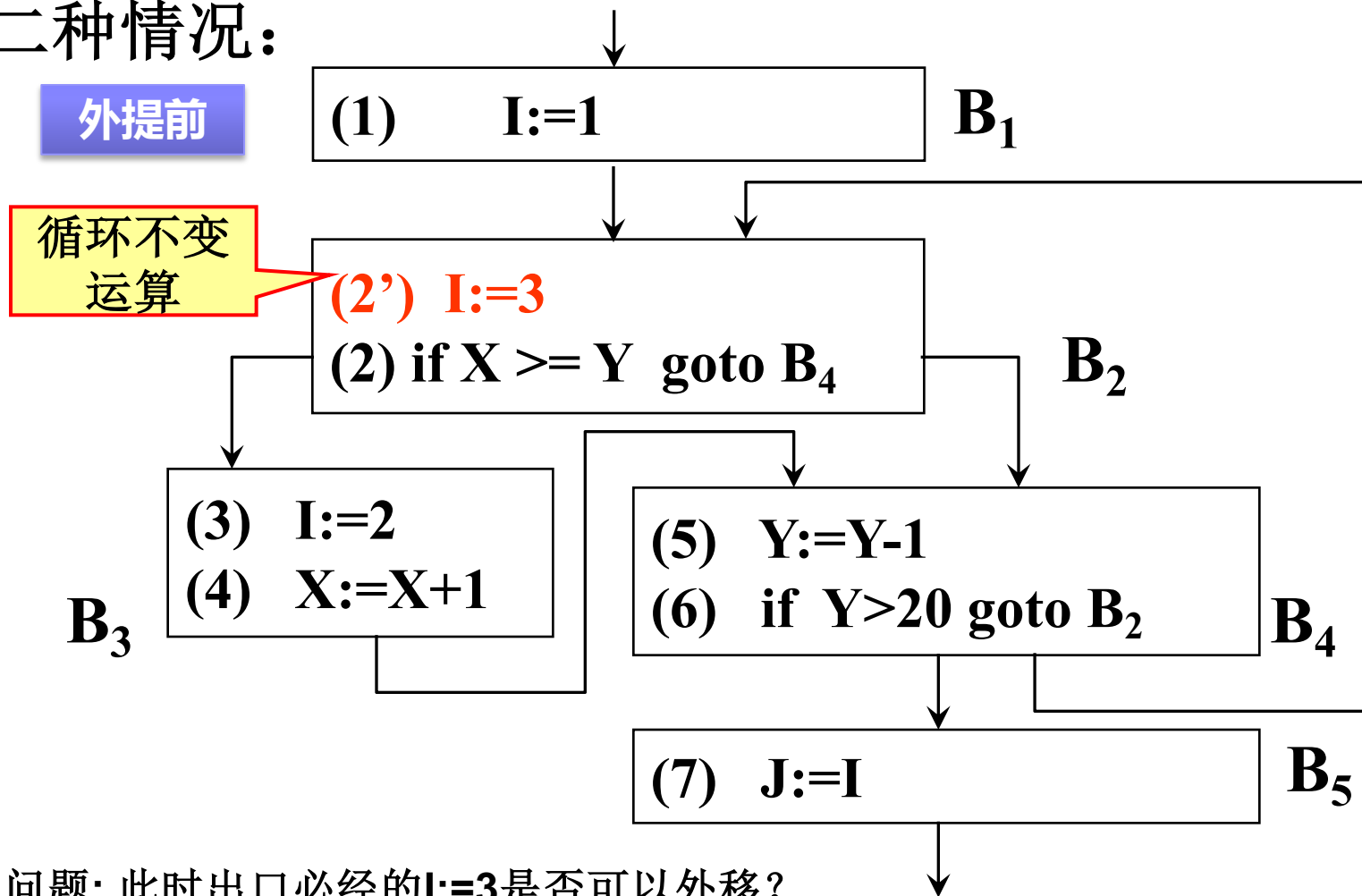
**代码S(A:=B OP C)外提条件 (1) : 不变运算所在  
的结点是L所有出口结点的必经结点.**

## 讨论：一定要出口必经吗？



**代码外提条件：出循环后不再引用I的值，即使不在必经结点内也可以考虑外移。**

## 第二种情况:



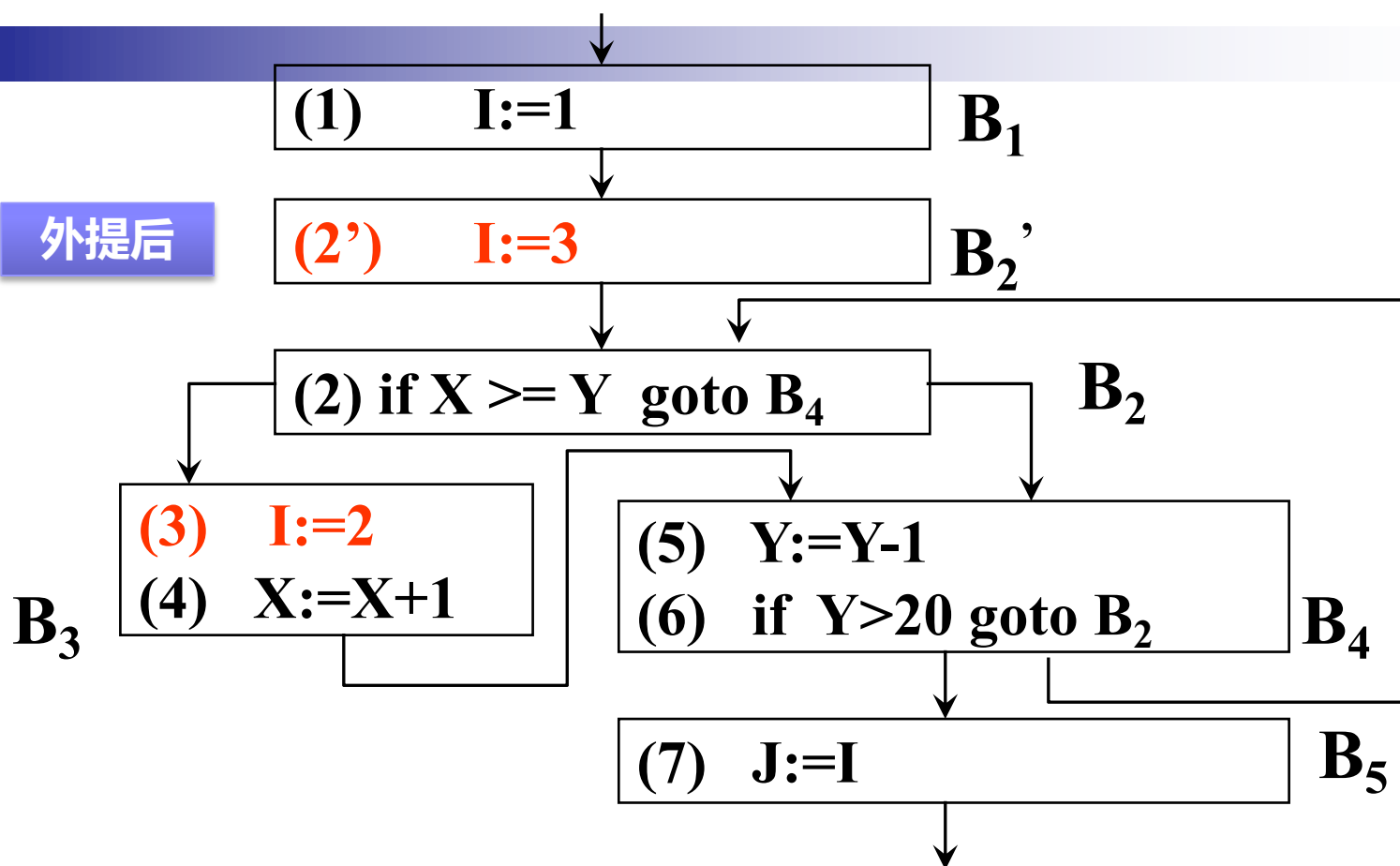
问题: 此时出口必经的 $I:=3$ 是否可以外移?

考虑:  $X=21, Y=22$

$B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

$I=3, J=3$

外提后

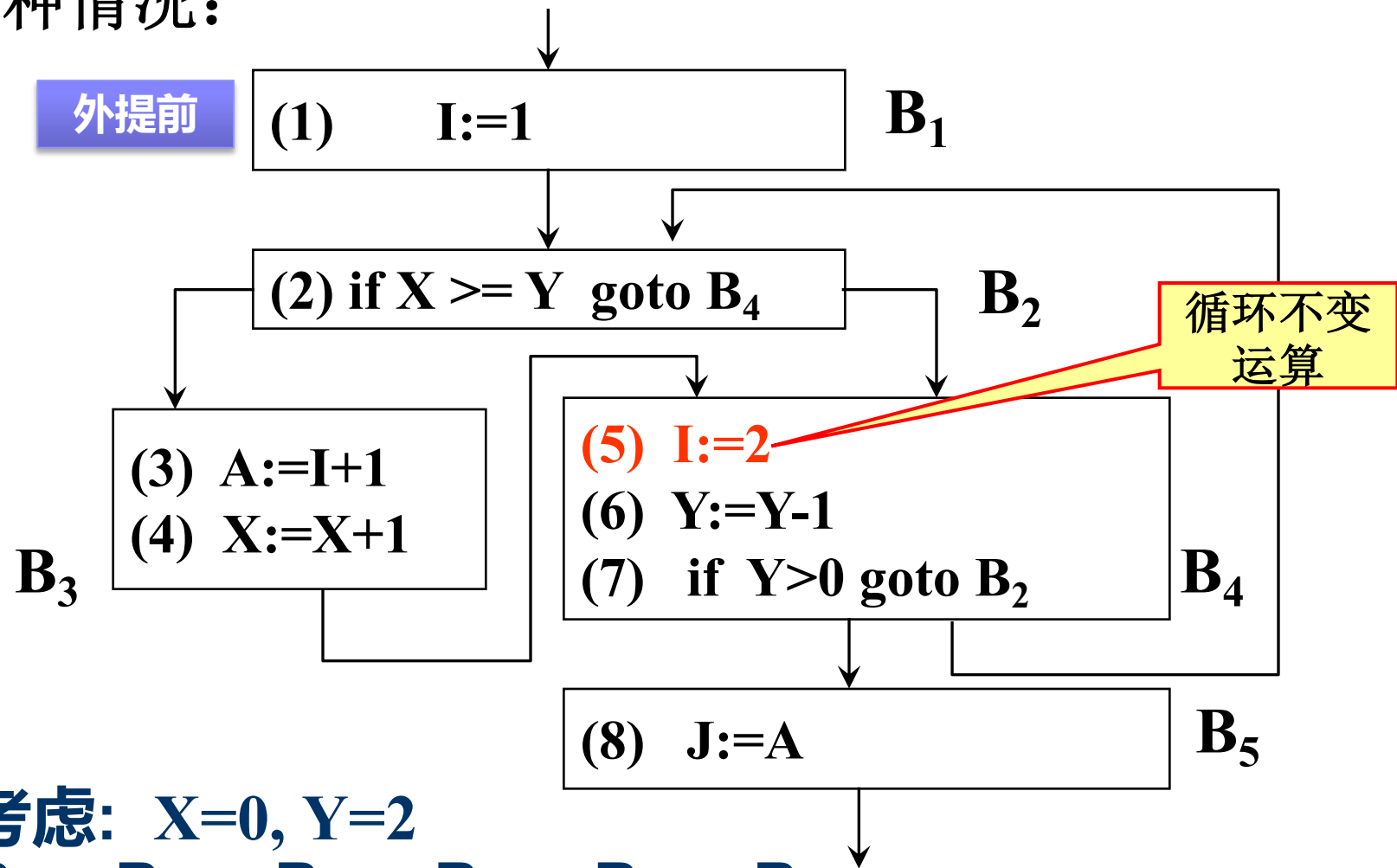


考虑:  $B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

$I=2, J=2$

代码  $S(A:=B \text{ OP } C)$  外提条件 (2) : **A在循环中其他地方未再定值, 才能把循环不变运算  $A:=B \text{ op } C$  外提;**

# 第三种情况:

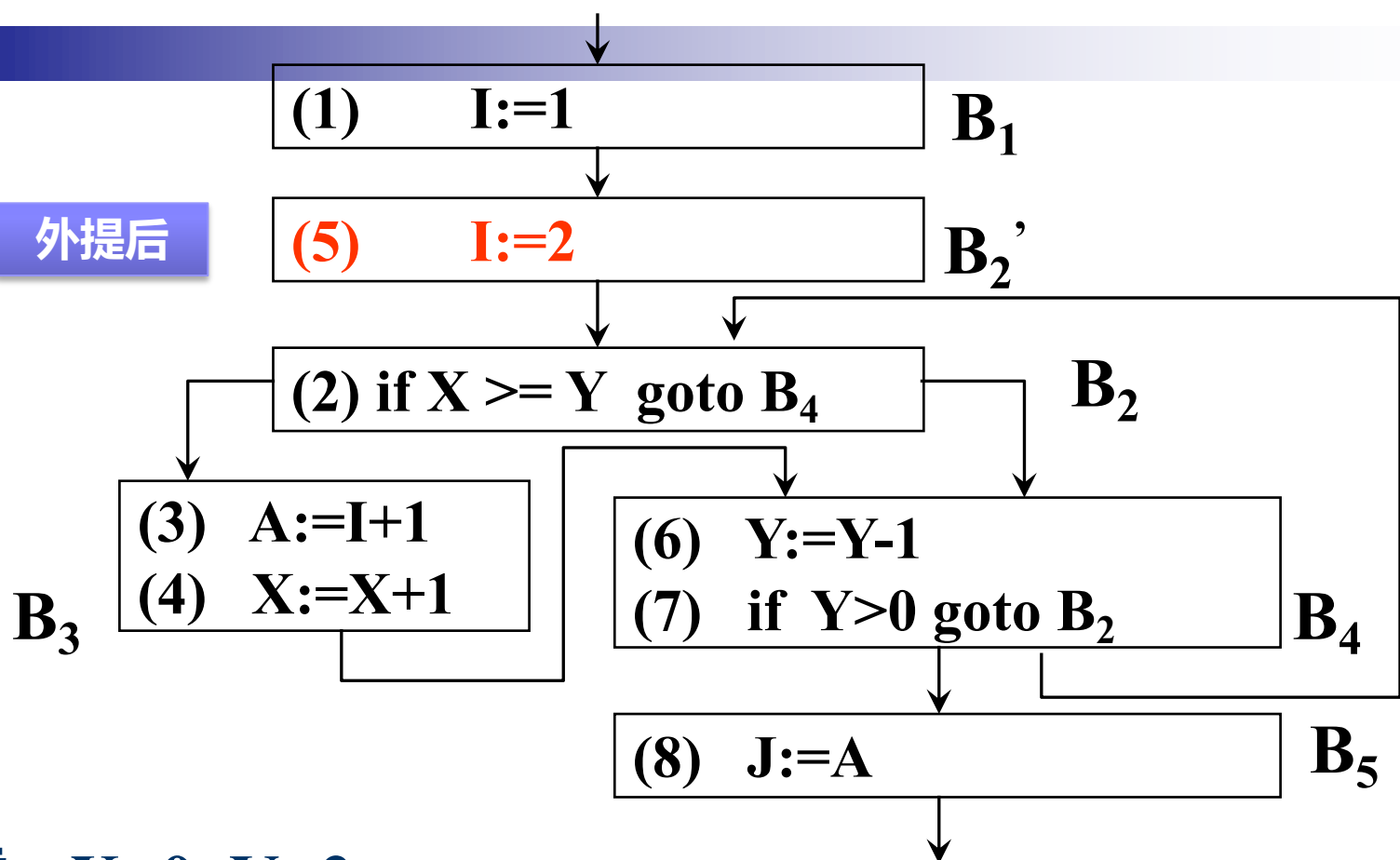


考虑:  $X=0, Y=2$

$B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

$A=2, J=2$

外提后



考虑:  $X=0, Y=2$

$B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

$A=3, J=3$

**S(A:=B OP C)外提条件 (3): 循环中所有A的引用点  
只有S中的A的定值才能到达。**

# 查找循环L的不变运算的算法

- 步骤1: 依次查看L中各基本块的每个四元式, 如果它的每个运算对象或为常数, 或者定值点在L外, 则将此四元式标记为"不变运算";
- 步骤2: 重复第3步直至没有新的四元式被标记为"不变运算"为止;
- 步骤3: 依次查看尚未被标记为"不变运算"的四元式, 如果它的每个运算对象或为常数, 或定值点在L之外, 或只有一个到达-定值点且该点上的四元式已被标记为"不变运算", 则把被查看的四元式标记为"不变运算"。

# 代码外提算法

1. 求出L的所有不变运算
2. 对每个不变运算s:  $A := B \text{ op } C$  或  $A := \text{op } B$  或  $A := B$  检查是否满足条件(1)或(2)
  - (1)
    - (i) s所在的结点是L所有出口结点的必经结点;
    - (ii) A在L中其他地方未再定值;
    - (iii) L中所有A的引用点只有s中的A的定值才能到达。



(2) A在离开L后不再是活跃的，并且条件(1)的(ii)和(iii)成立。所谓**A在离开L后不是活跃的**是指，A在L的任何出口结点的后继结点入口处不是活跃的。

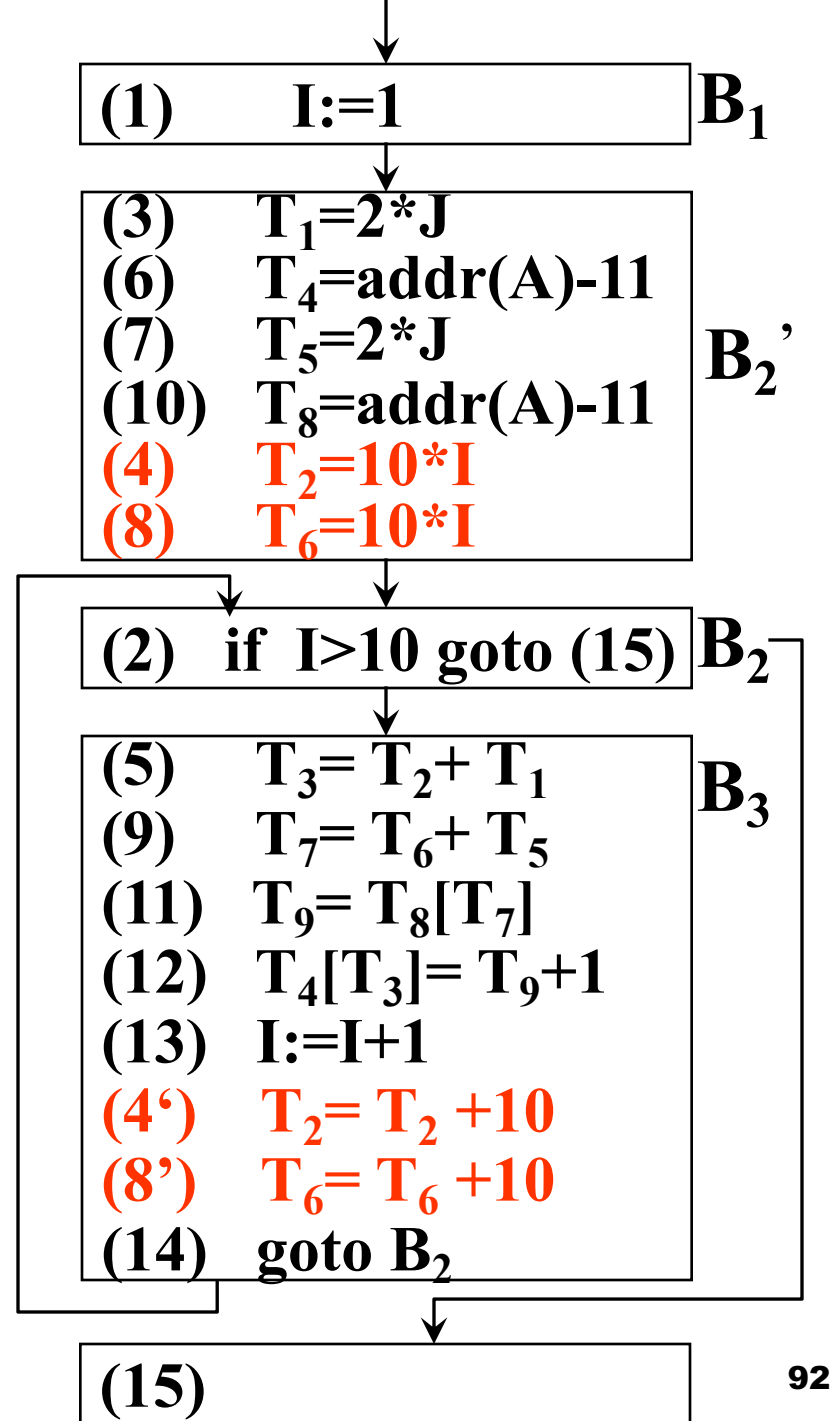
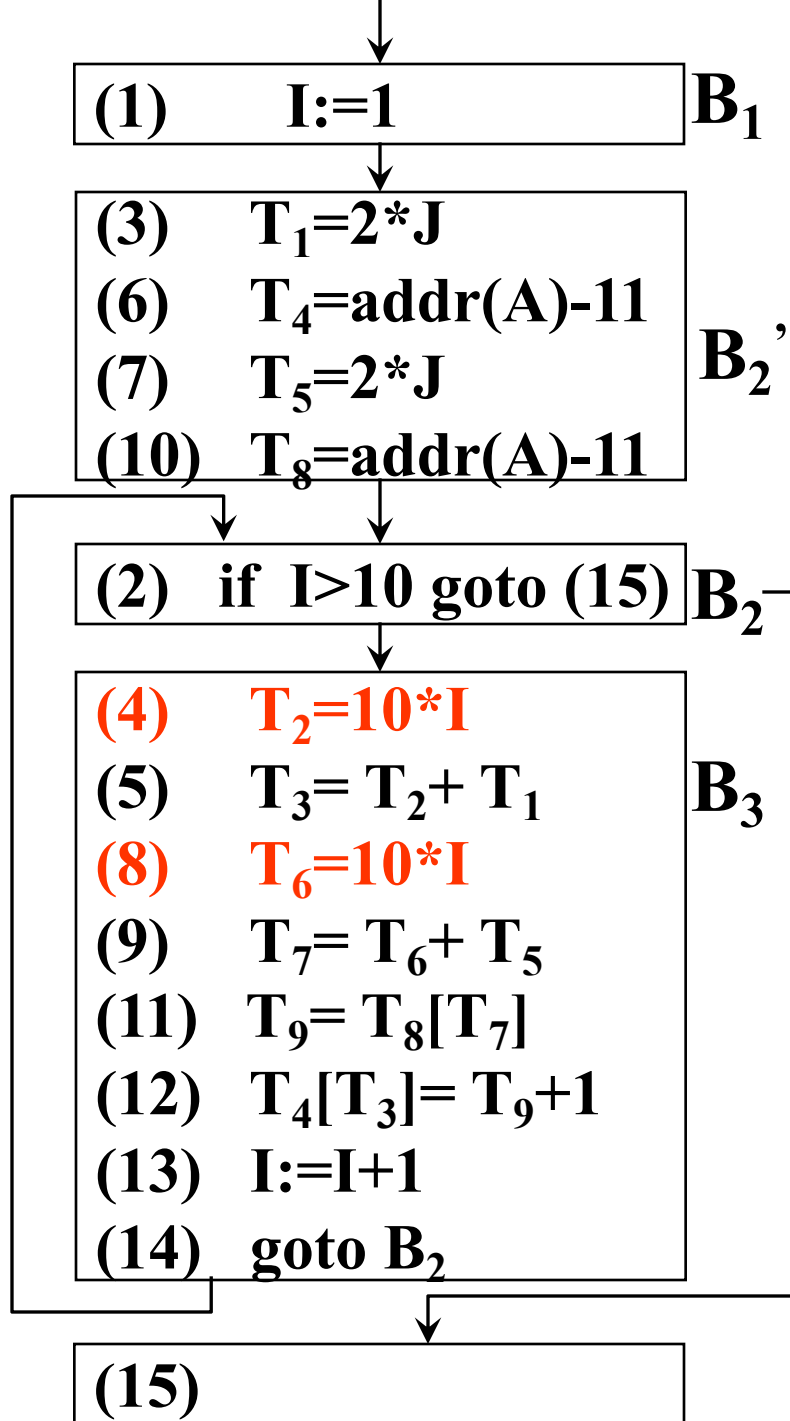
3.按步骤1所找出的不变运算的次序，依次把符合条件2的条件(1)或(2)的不变运算s外提到L的前置结点中。但是，如果s的运算对象(B或C)是在L中定值的，那么，只有当这些定值四元式都已外提到前置结点中时，才能把s也外提到前置结点中。

# 循环优化

- 对循环中的代码，可以实行：
  - 代码外提
  - 强度削弱
  - 删除归纳变量（变换循环控制条件）

# 强度削弱

- 把程序中执行时间较长的运算转换为执行时间较短的运算。
  - 如把循环中的乘法运算用递归加法运算替换



# 循环优化

- 对循环中的代码，可以实行：
  - 代码外提
  - 强度削弱
  - 删除归纳变量（变换循环控制条件）

# 删除归纳变量

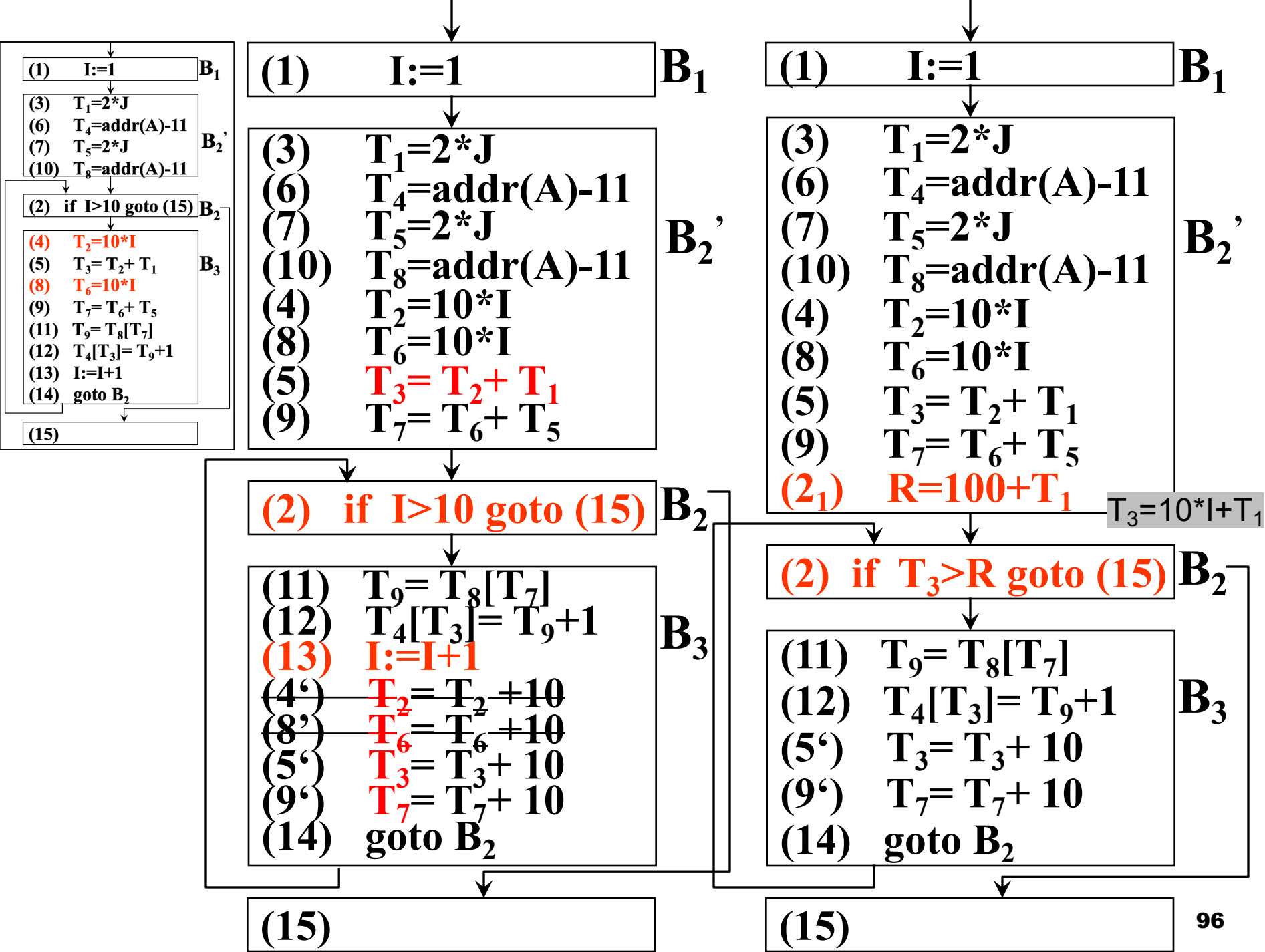
- 如果循环中对变量 $I$ 只有唯一的形如 $I := I \pm C$ 的赋值，且其中 $C$ 为循环不变量，则称 $I$ 为循环中的**基本归纳变量**。
- 如果 $I$ 是循环中一基本归纳变量， $J$ 在循环中的定值总是可化归为 $I$ 的同一线性函数，也即 $J = C_1 * I \pm C_2$ ，其中 $C_1$ 和 $C_2$ 都是循环不变量，则称 $J$ 是**归纳变量**，并称它与 $I$ **同族**。
- 一个基本归纳变量也是一归纳变量。

# 强度削弱和删除归纳变量的统一算法

1. 利用循环不变运算信息，找出循环中所有基本归纳变量。
2. 找出所有其它归纳变量A，并找出A与已知基本归纳变量X的同族线性函数关系  $F_A(X)$ 。
3. 对2中找出的每一归纳变量A，进行强度削弱。
4. 删除对归纳变量的无用赋值。
5. 删除基本归纳变量: 如果基本归纳变量B在循环出口之后不是活跃的，并且在循环中，除在其自身的递归赋值中被引用外，只在形如

if B rop Y goto L

中被引用，则可选取一与B同族的归纳变量M来替换B进行条件控制。最后删除循环中对B的递归赋值的代码。





## 总结分析

- 强度削弱主要是针对与基本归纳变量有线性关系的归纳变量的赋值运算进行;
- 经过强度削弱后, 循环中可能出现一些新的无用赋值, 可以删除;
- 对于基础归纳变量, 也可以进一步删除;
- 对于削弱下标变量地址计算的强度非常有效。

# 小测试

- 课后练习：P307 5.
- CANVAS：循环优化

**Dank u**

Dutch

**Merci**

French

**Спасибо**

Russian

**Gracias**

Spanish

شكراً

Arabic

감사합니다

Korean

धन्यवाद

Hindi

תודה רבה

Hebrew

**Tack så mycket**

Swedish

**Obrigado**

Brazilian  
Portuguese

**Dankon**

Esperanto

***Thank You !***

谢谢

Chinese

ありがとうございます

Japanese

**Trugarez**

Breton

**Danke**

German

**Tak**

Danish

**Grazie**

Italian

நன்றி

Tamil

**děkuji**

Czech

ขอบคุณ

Thai

**go raibh maith agat**

Gaelic