# 第十一章 代码优化

LOGO

广东工业大学计算机学院

# 本课内容

- 11.1 优化技术简介
- 11.2 局部优化
- 11.3 控制流分析和循环优化

# 代码优化概念、目的与原则

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，我们通常称这种变换为优化。

优化的目的是为了产生更高效的代码。

优化的原则：

(1) 等价原则

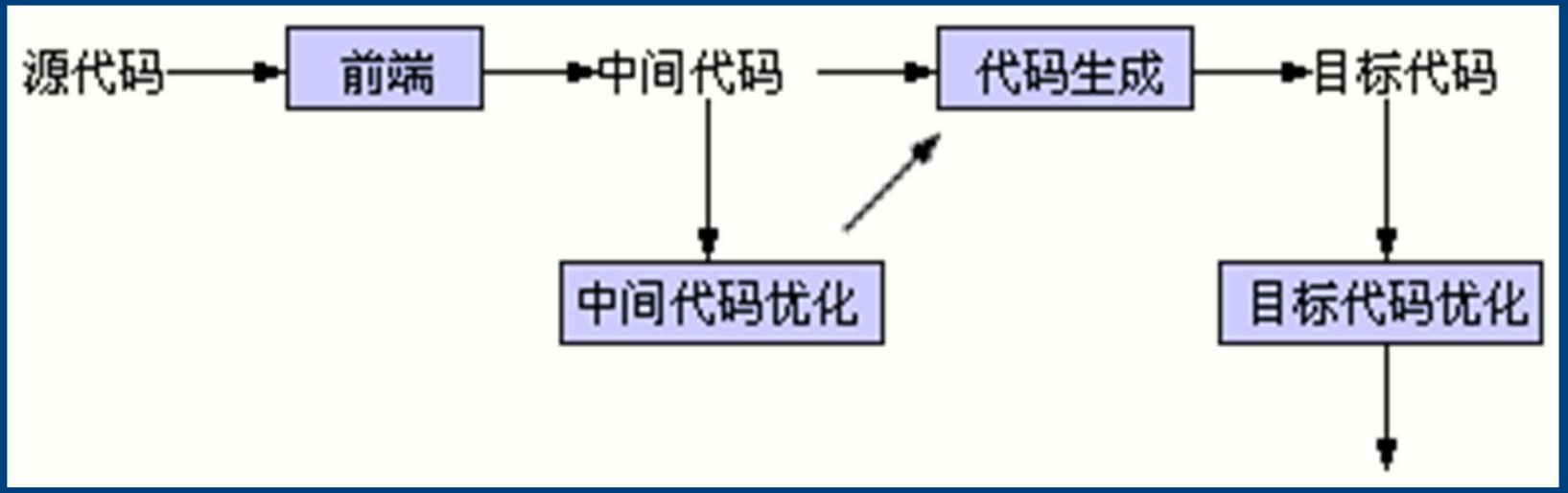
(2) 有效原则

(3) 合算原则

应尽可能以较低的代价取得较好的优化效果。

# 优化工作阶段

- 一般，优化工作阶段可在中间代码生成之后和（或）目标代码生成之后进行。




- 中间代码的优化是对中间代码进行等价变换。
- 目标代码的优化在目标代码生成之后进行，这一类优化在很大程度上依赖于具体的机器，我们不做详细讨论。

# 代码优化分类

## 2. 按所涉及范围

| | |
|---|---|
| 局部优化 | 单个基本块内 |
| 循环优化 | 可能涉及多个基本块 |
| 全局优化 | 涉及所有代码 |

# 优化技术简介

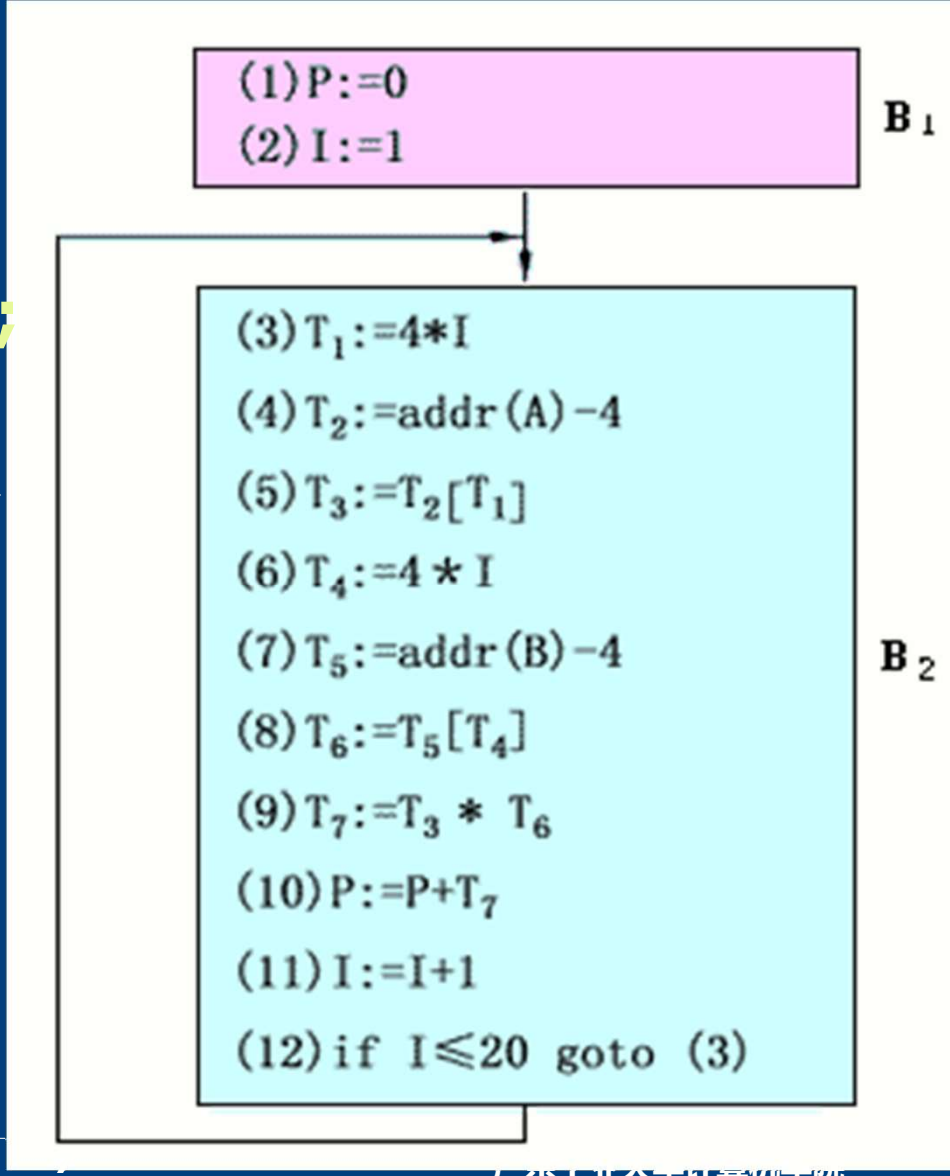
- 常用的优化技术有：
- 1. 删除多余运算
- 2. 循环不变代码外提
- 3. 强度削弱
- 4. 变换循环控制条件
- 5. 合并已知量
- 6. 复写传播与删除无用赋值

# 优化技术应用举例

- 看下面的源程序：

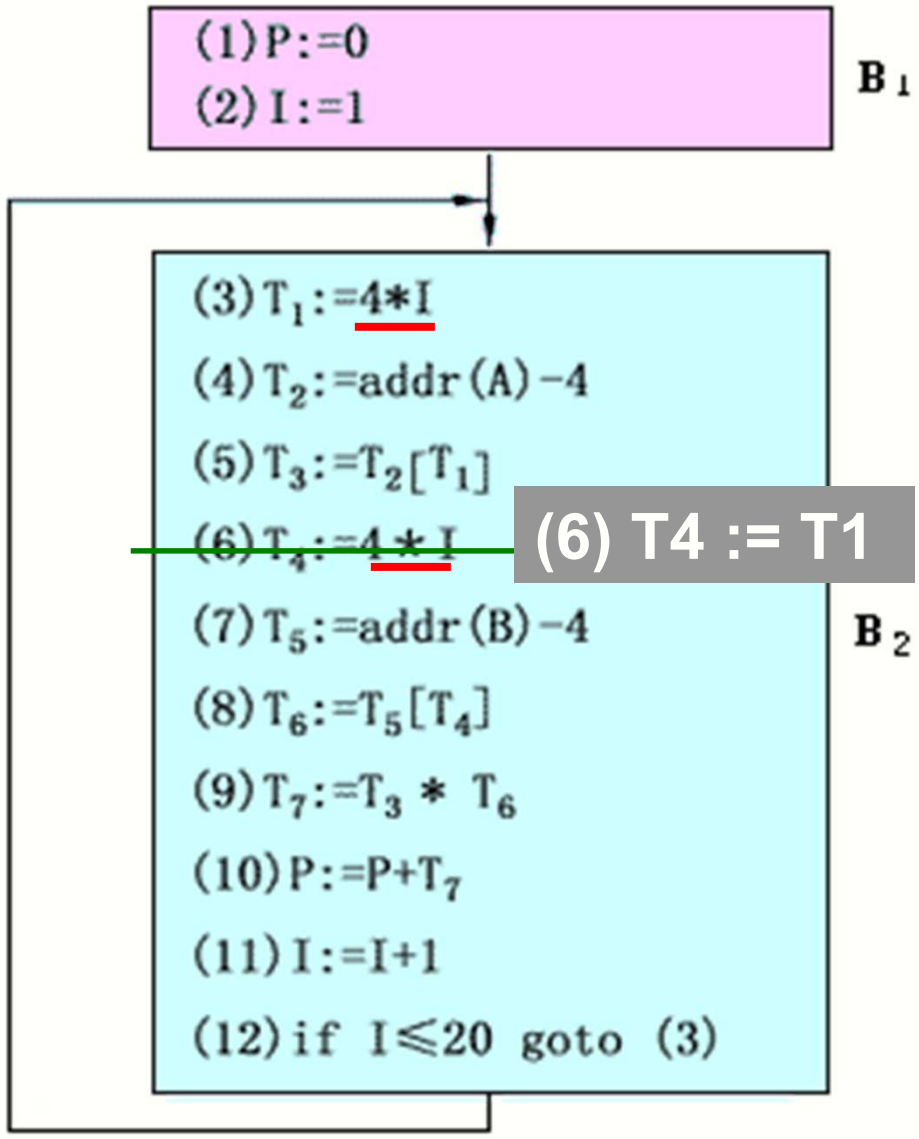
```
P := 0
for I := 1 to 20 do
    P :=P + A[I] * B[I];
```

- 这个程序段由B1和B2两个部分组成，B2是一个循环，假定每个元素占4个字长编址。



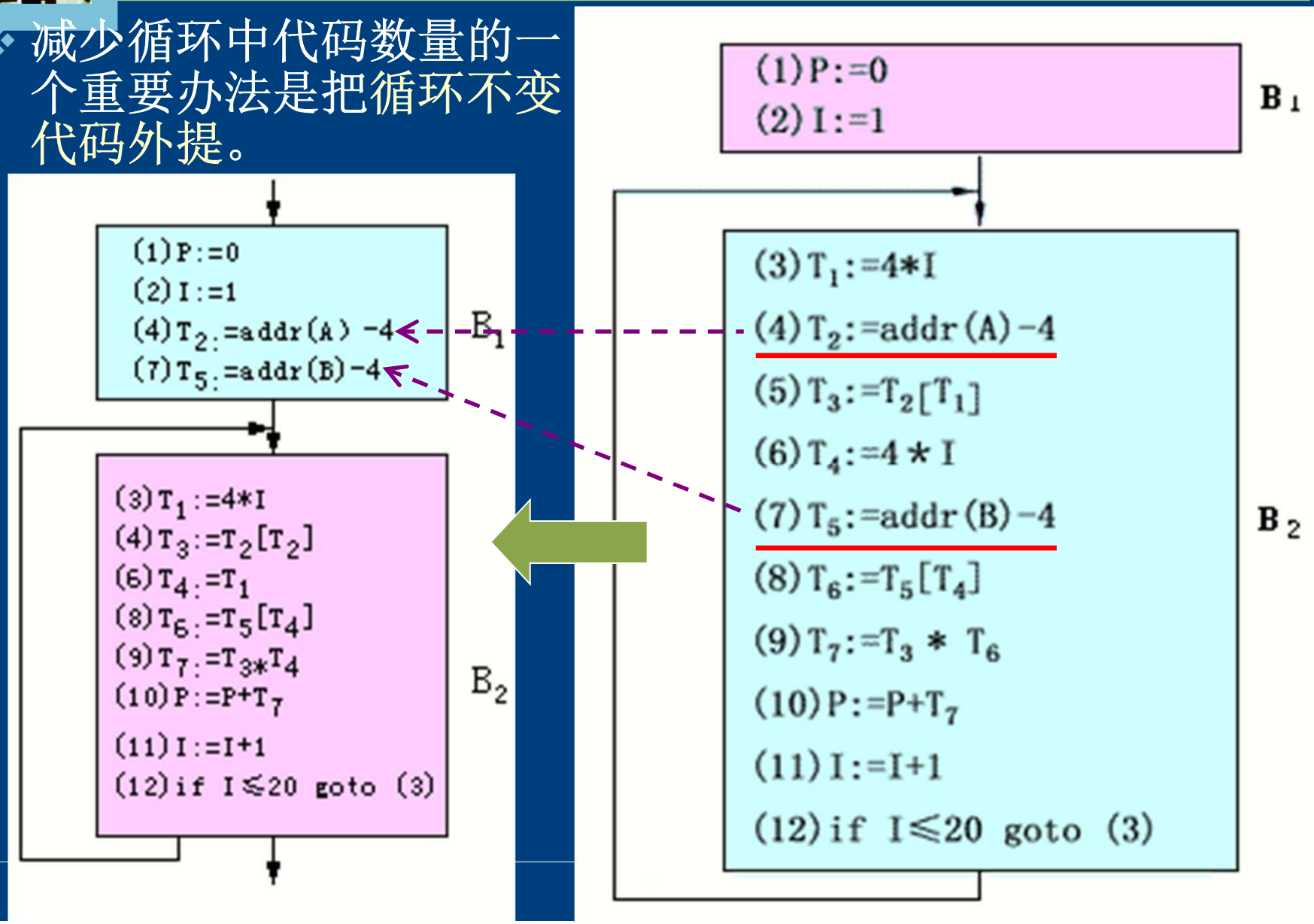
# 1. 删除多余运算

- 优化技术简介——删除多余运算(删除公共子表达式)
- 如图所示，哪个运算多余?
- 我们可以把四元式（6）变换成：**T4 := T1**。
- 这种优化称为删除多余运算或称为删除公共子表达式。



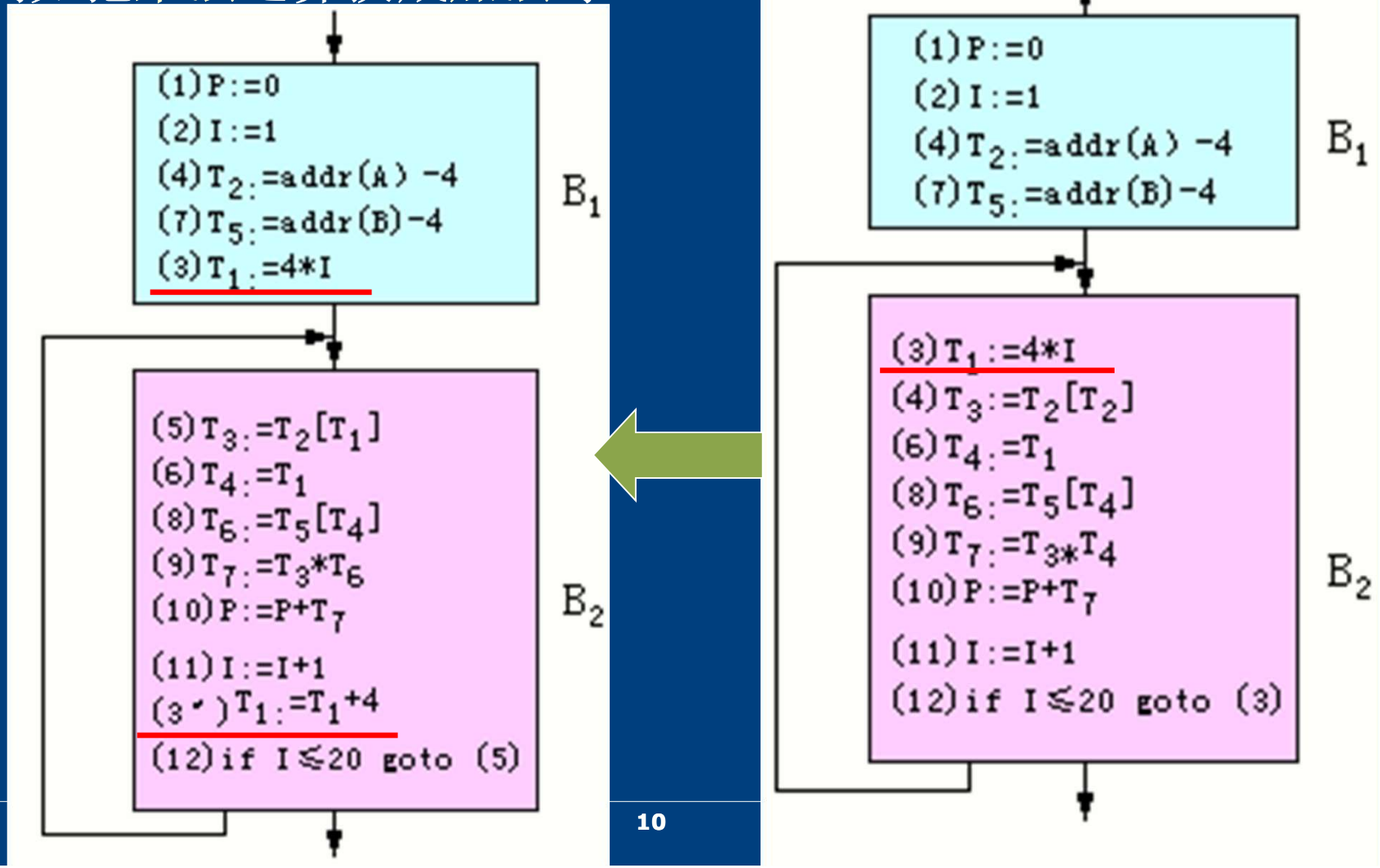
# 2. 代码外提

- 减少循环中代码数量的一个重要办法是把循环不变代码外提。



# 3. 强度削弱

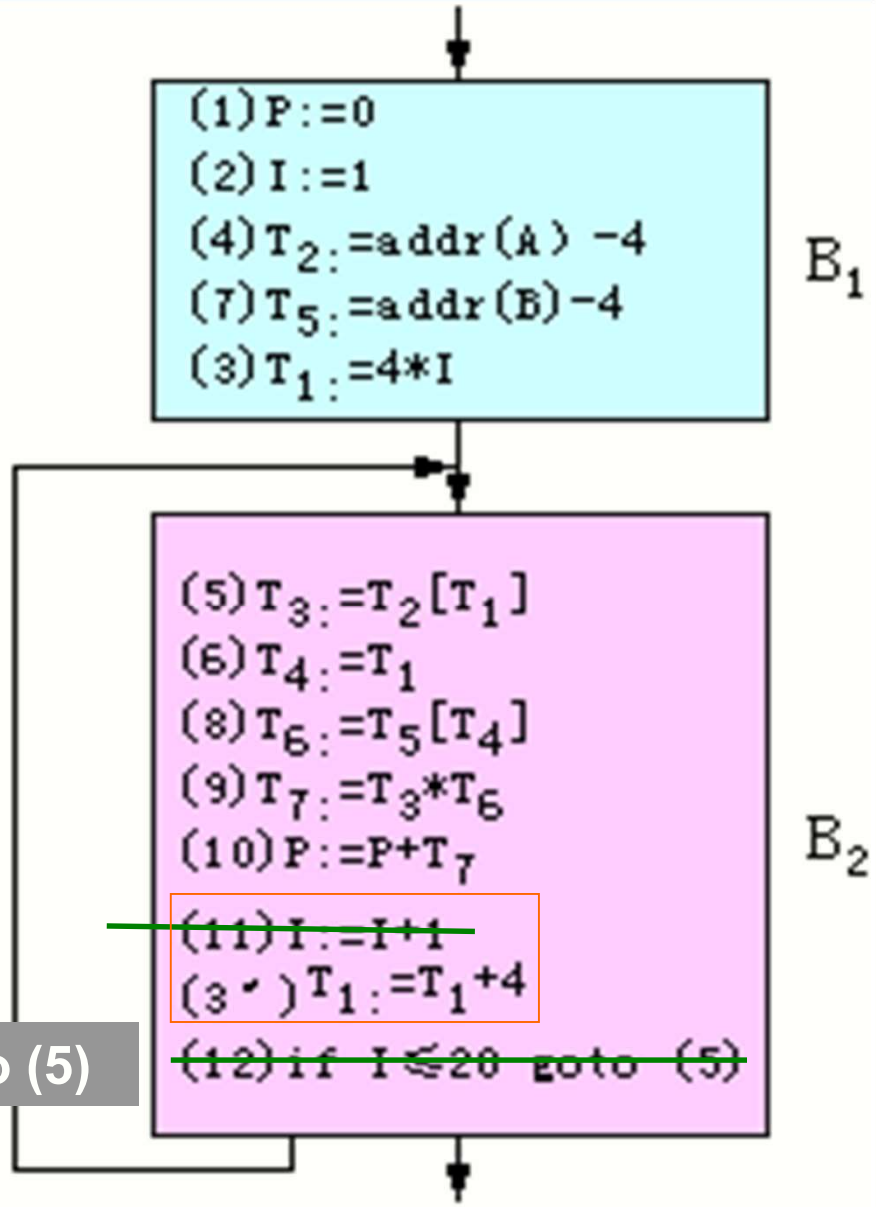
- 削弱强度削弱的思想是把强度大的运算换算成强度小的运算。例如把乘法运算换成加法等。



# 4. 变换循环控制条件

- 在右图的代码中，**I**和**T1**始终保持**T1＝4*I**的线性关系。
- 可以把四元式**(12)**的循环控制条件**I ≤ 20**变换成**T1 ≤ 80**，这样整个程序的运行结果不变。
- 经过这一变换后，循环中**I**的值在循环后不会被引用，四元式（**11**）成为多余运算。

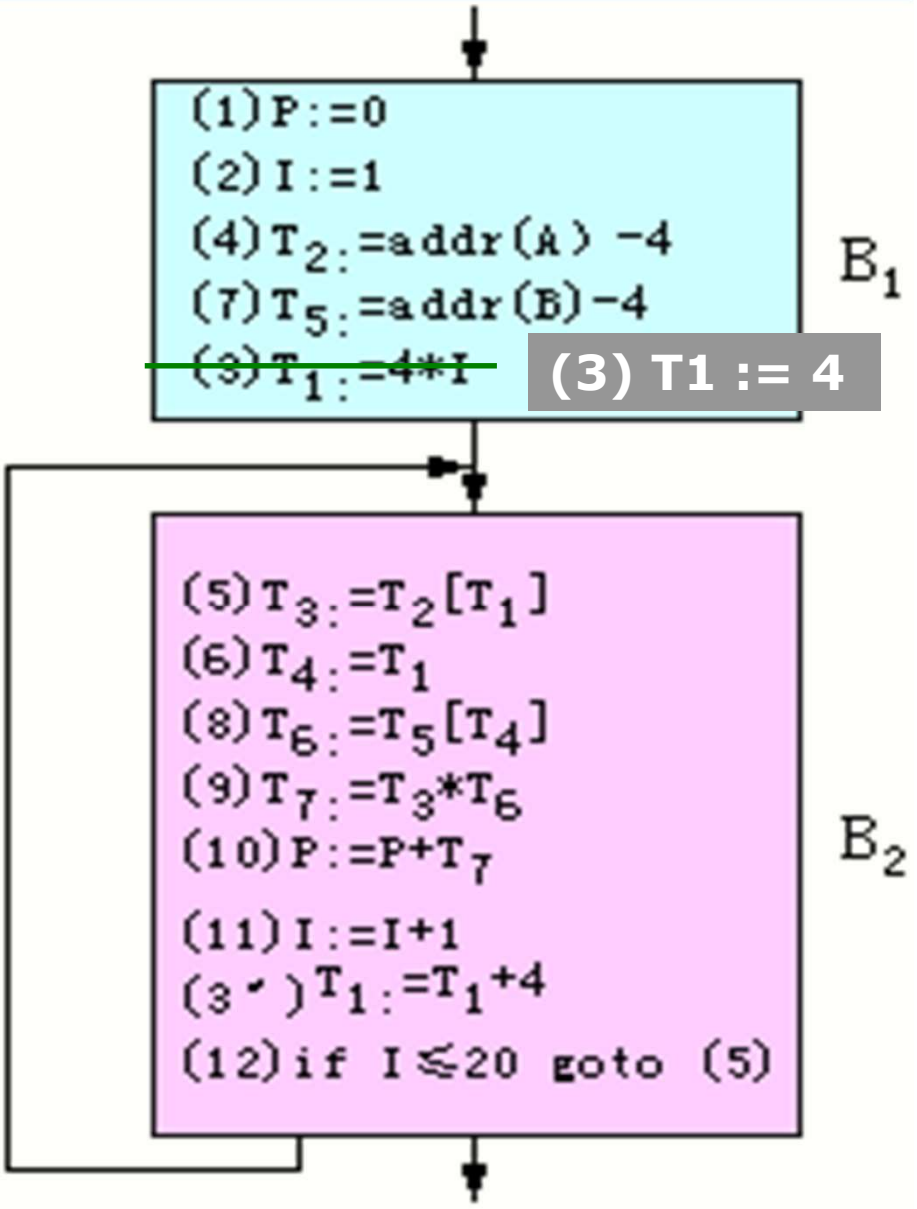
**(12) if T1≤80 goto (5)**



# 5. 合并已知量与复写传播(1)

## (1) 合并已知量

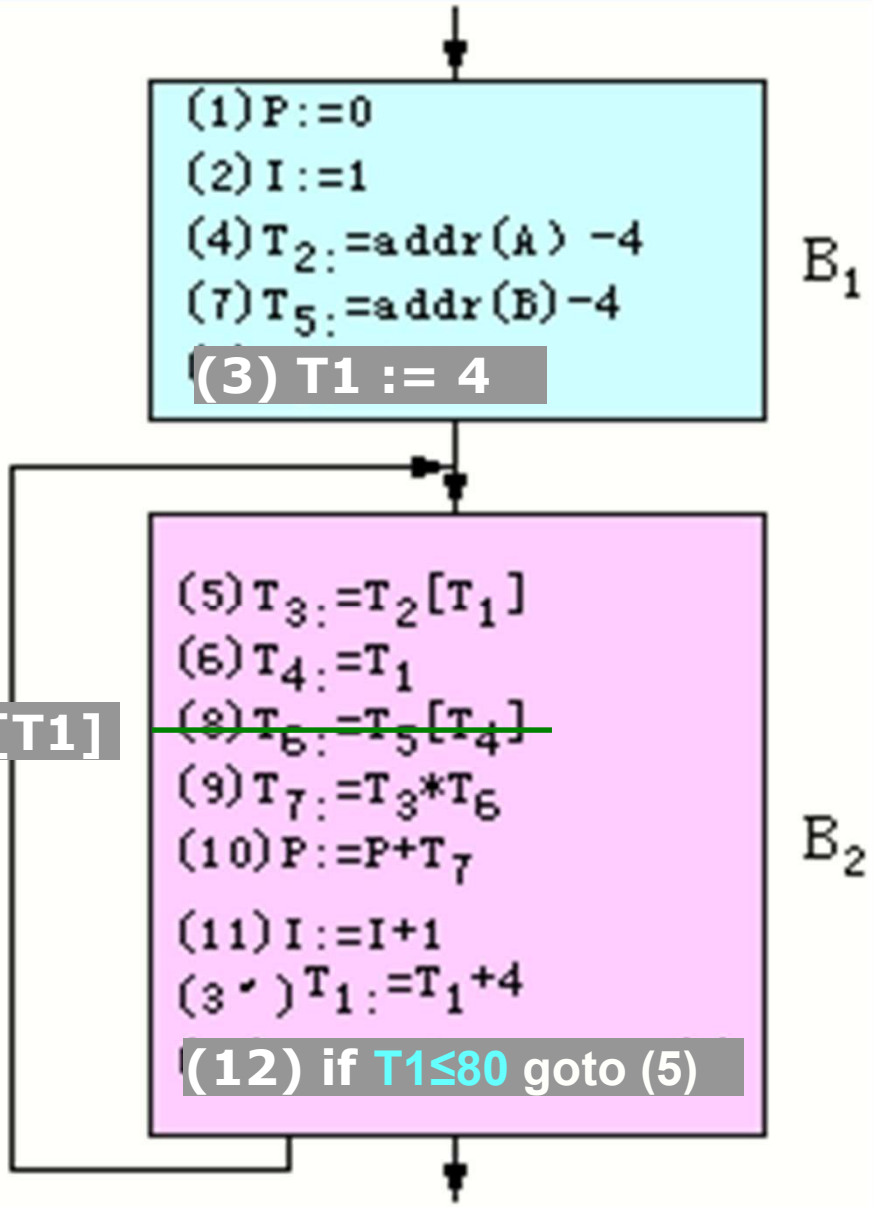
- 在四元式(3)计算4*I时，I必为1。
- 即4*I的两个运算对象都是编码时的已知量，可在编译时计算出它的值是4。
- 所以四元式(3)可变为 T1 := 4。



# 5. 合并已知量与复写传播(2)

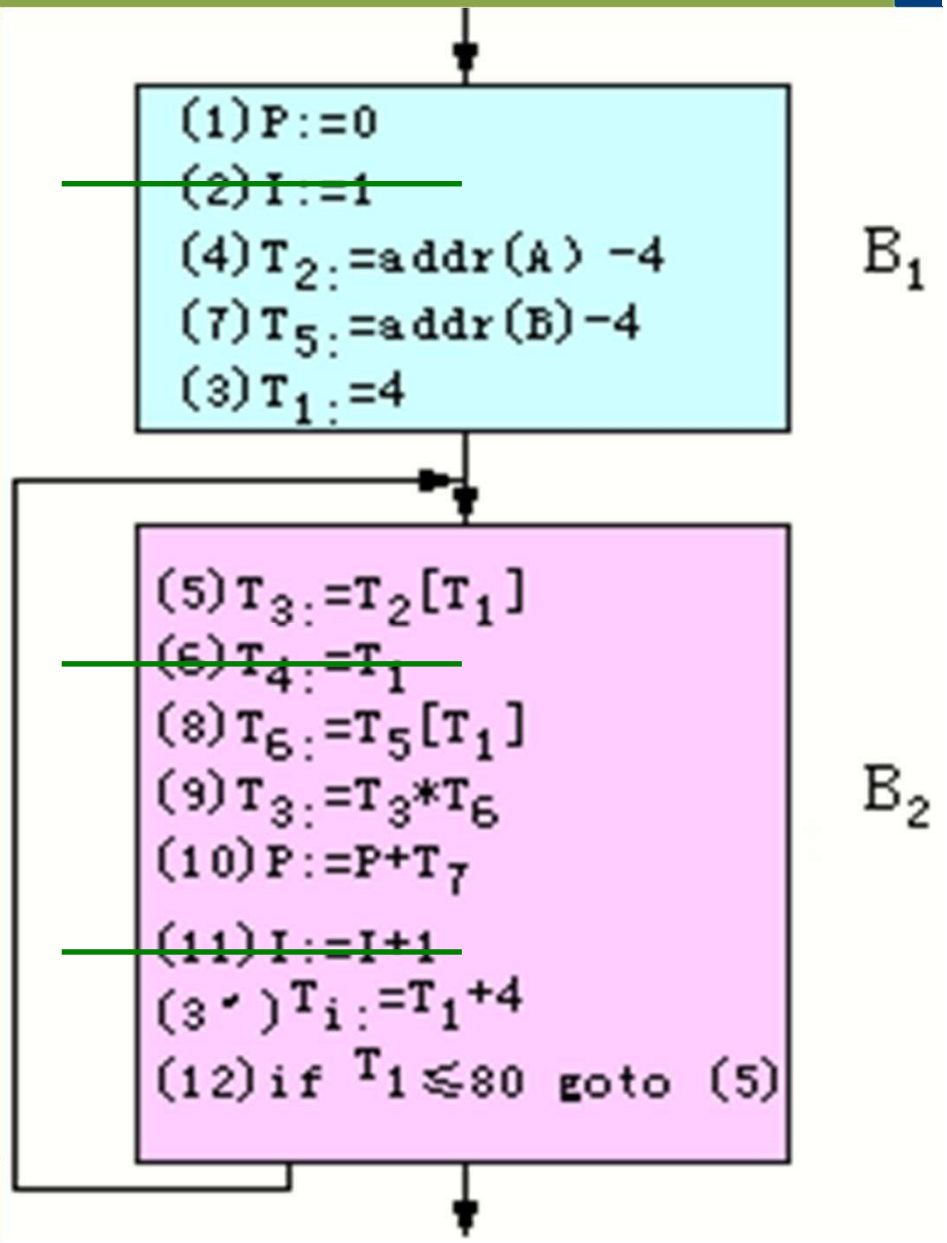
## (2) 复写传播

- 四元式(6)把T1的值复写到T4中。
- (8)要引用T4的值，而从(6)到(8)之间未改变T4和T1的值，则将(8)改为T6 := T5[T1]。
- 注意：复写传播之后运算结果保持不变。



# 6. 删除无用赋值

- 在右图中，**(6)**对**T4**赋值，但**T4**未被引用；
- **(2)**和**(11)**对**I**赋值，但只有**(11)**引用**I**。
- 只要程序中其它地方不需要引用**T4**和**I**，则代码**(6)**，**(2)**和**(11)**对程序的运行结果无任何作用。



# 本课内容

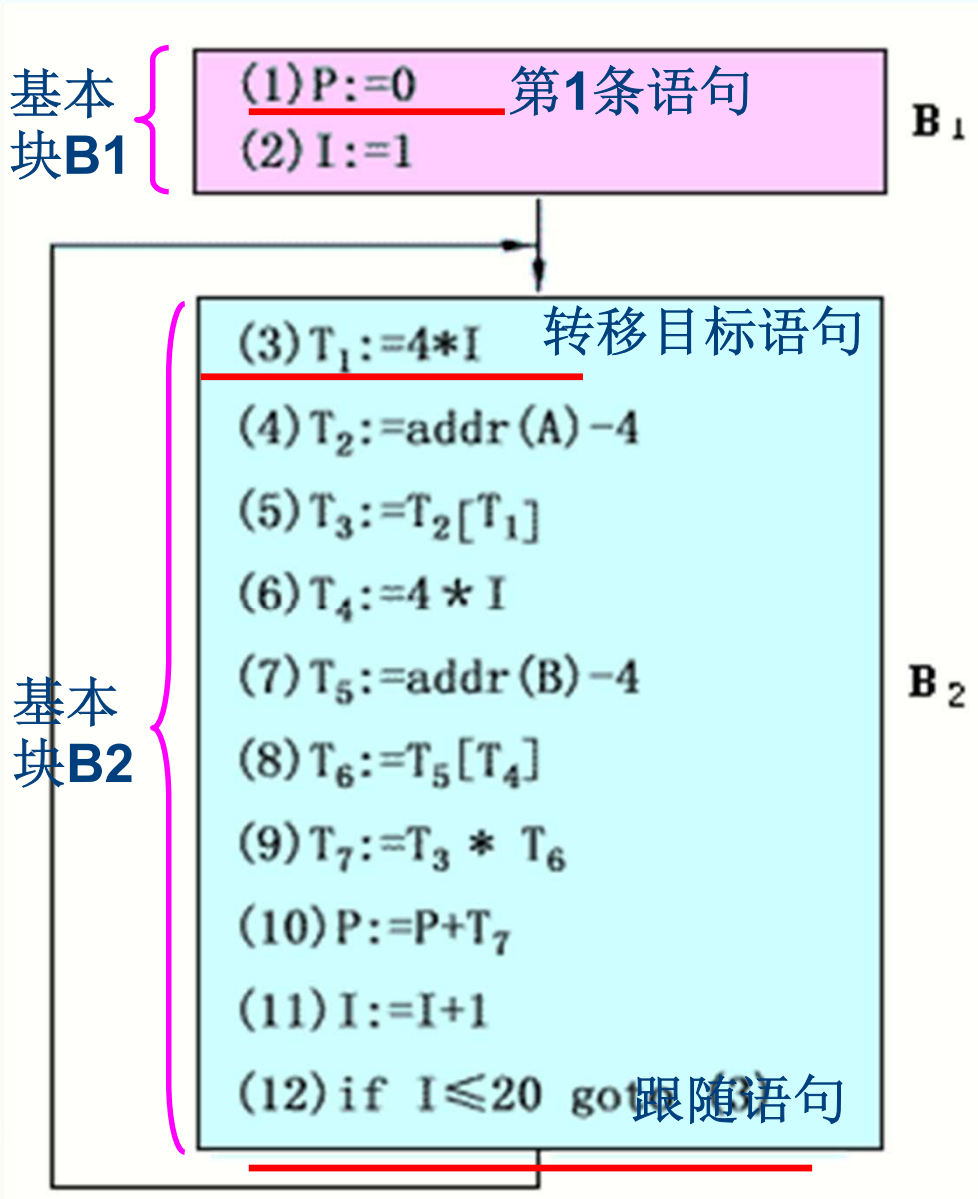
- **11.1** 优化技术简介
- **11.2** 局部优化
- **11.3** 控制流分析和循环优化

# 局部优化

- 局部优化是指基本块内的优化。
- 基本块：指程序中一个顺序执行的语句序列，其中只有一个入口语句和一个出口语句。控制流只能从其入口语句进入，从其出口语句退出，没有中途停止或分支。
- **1.** 基本块的划分
- 需要先定义基本块的入口语句，即以下情况之一的语句：
- ① 程序的第一个语句；
- ② 条件转移语句或无条件转移语句的转移目标语句；
- ③ 紧跟在条件转移语句后面的语句。

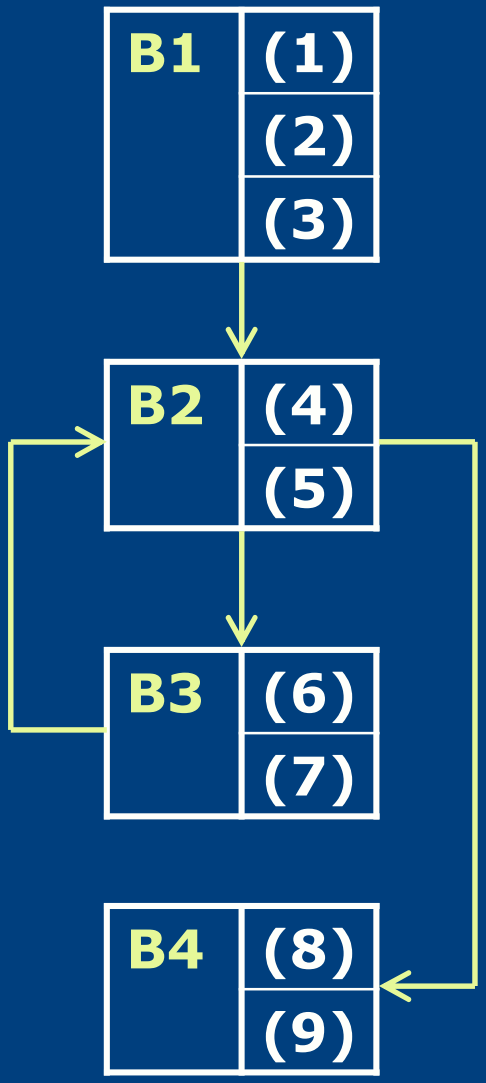
# 1. 基本块的划分步骤

- 划分中间代码的基本步骤：
- ① 求出四元式程序中各个基本块的入口语句。
- ② 对每一入口语句，构造基本块：由该入口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。



# 基本块划分和流图举例1

- (1) read (C)
- (2) A:= 0
- (3) B:= 1
- (4) L1: A:=A + B
- (5) if B>= C goto L2
- (6) B:=B+1
- (7) goto L1
- (8) L2: write (A)
- (9) halt



# 基本块划分和流图举例2

- 将下面的C程序翻译成四元式序列，并且划分基本块

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

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

# 举例(续1)

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

- (1) i := m - 1
- (2) j := n
- (3) t1 := 4 * n
- (4) v := a[ t1 ]

- while( 1 ) {
- while( a[++i] < v );
- while( a[--j] > v );
- ……
- }

- (5) i := i + 1
- (6) t2 := 4 * i;
- (7) t3 := a[ t2 ];
- (8) if t3 < v goto (5)

- (9) j := j - 1
- (10) t4 := 4 * j;
- (11) t5 := a[ t4 ];
- (12) if t5 > v goto (9)

# 举例(续2)

```
while( 1 ) {
        ......
        if( i >= j )   break;
        x = a[ i ];   a[ i ] = a[ j ];   a[ j ] = x;
}
```

(13) if i >= j goto (23)
(14) t6 := 4 * i
(15) x := a[t6]
(16) t7 := 4 * i
(17) t8 := 4 * j
(18) t9 := a[ t8 ]
(19) a[ t7 ] := t9
(20) t10 := 4 * j

(21) a[ t10 ] := x
(22) goto (5)

# 举例(续3)

- while(1) {……}
- x = a[ i ];　　a[ i ] = a[ n ];　　a[ n ] = x;

- (23) t11 := 4 * i
- (24) x := a[t11]
- (25) t12 := 4 * i
- (26) t13 := 4 * n
- (27) t14 := a[ t13 ]
- (28) a[ t12 ] := t14
- (29) t15 := 4 *n
- (30) a[ t15 ] := x

# 举例(续4)

- (1) i := m - 1
- (2) j := n
- (3) t1 := 4 * n;
- (4) v := a[ t1 ]
- (5) i := i + 1
- (6) t2 := 4 * i;
- (7) t3 := a[ t2 ];
- (8) if t3 < v goto (5)
- (9) j := j - 1
- (10) t4 := 4 * j;
- (11) t5 := a[ t4 ];
- (12) if t5 > v goto (9)
- (13) if i >= j goto (23)
- (14) t6 := 4 * i
- (15) x := a[t6]
- (16) t7 := 4 * i
- (17) t8 := 4 * j
- (18) t9 := a[ t8 ]
- (19) a[ t7 ] := t9
- (20) t10 := 4 * j
- (21) a[ t10 ] := x
- (22) goto (5)
- (23) t11 := 4 * i
- (24) x := a[t11]
- (25) t12 := 4 * i
- (26) t13 := 4 * n
- (27) t14 := a[ t13 ]
- (28) a[ t12 ] := t14
- (29) t15 := 4 *n
- (30) a[ t15 ] := x

# 程序流图

- (1) i := m – 1

  ……

  (4) v := a[ t1 ]
- (5) i := i + 1

  ……

  (8) if t3 < v goto (5)
- (9) j := j – 1

  ……

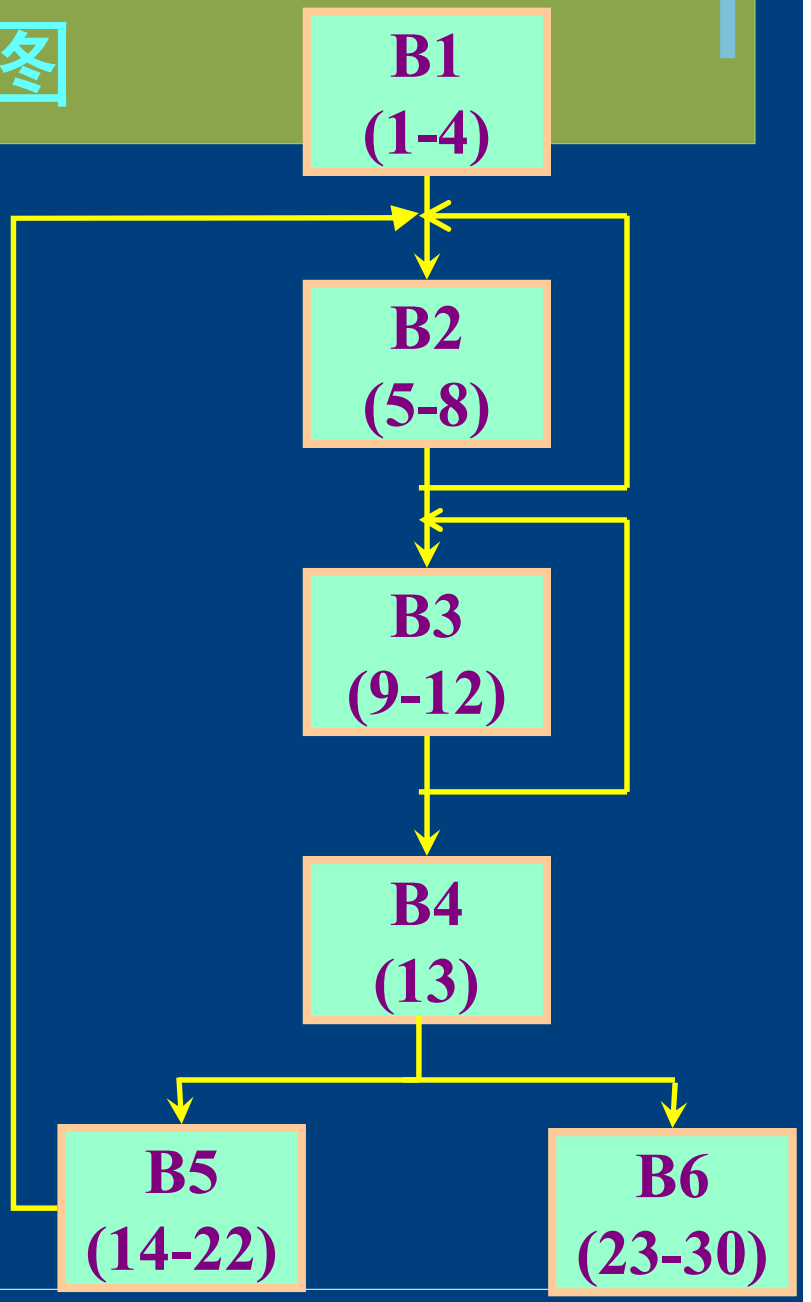
  (12) if t5 > v goto (9)
- (13) if i >= j goto (23)
- (14) t6 := 4 * i

  ……

  (22) goto (5)
- (23) t11 := 4 * i

  ……

  (30) a[ t15 ] := x



# 本课内容

- **11.1** 优化技术简介
- **11.2** 局部优化
- **11.3** 控制流分析和循环优化

# 程序流图与循环

- 我们将使用程序的控制流程图对所讨论的循环给出定义，并介绍怎样从程序的控制流程图中找出程序的循环。

- **1.** 程序流图与循环
- 一个控制流程图(简称为流图)可以表示成一个三元组**G = (N，E，$n_0$)**，其中:

- **N**：图中所有结点集，流图中的有限结点集**N**即是程序的基本块集。

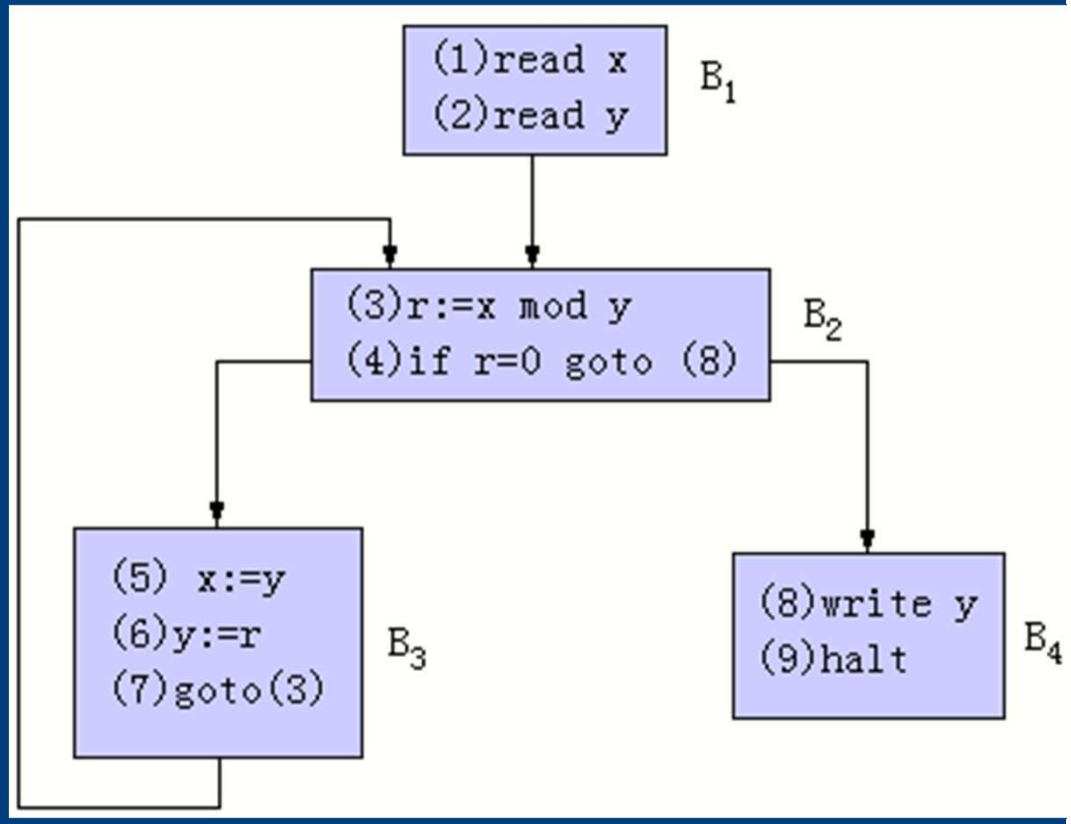
- **E**：图中所有有向边集：表示基本块之间的执行顺序

- **$n_0$**：首结点。流图的首结点就是包含程序第一个语句的基本块。

# 程序流图构造举例

- 将下面这段程序划分为基本块，然后构造有向边，得到程序流图：

```
(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
```



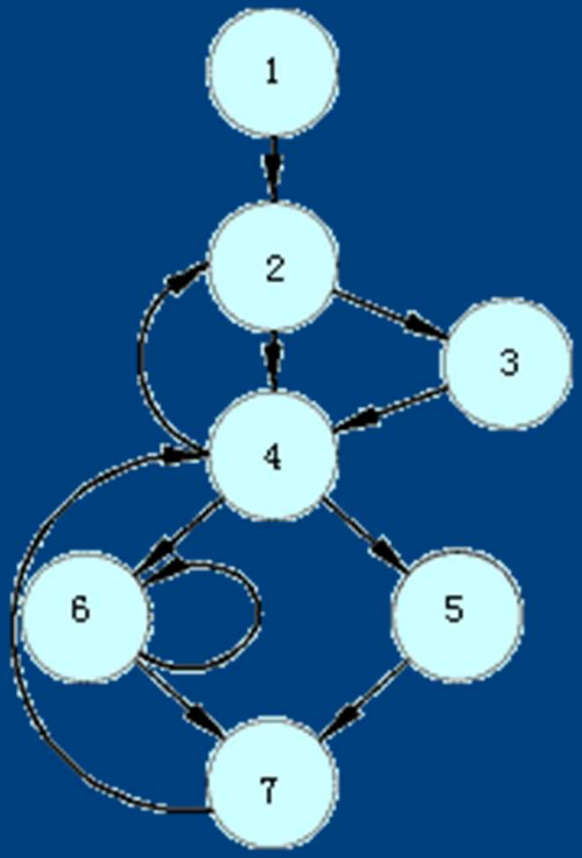
# 2. 循环的定义

- 在程序流图中，具有下列性质的结点序列称为循环：
- ① 各结点是强连通的。也即，其中任意两个结点之间，必有一条通路。
- 特别地，如果序列只包含一个结点，则必有一有向边从该结点引到其自身。
- ②在结点序列中，有且只有一个入口结点。
- 所谓入口结点，是指序列中具有下述性质的结点：
- 从序列外某结点，有一有向边引到它，或者它就是程序流图的首结点。

例如，图中的程序流图，根据定义:

结点序列{6}，{4，5，6，7}以及{2，3，4，5，6，7}都是循环;

而结点序列{2，4}，{2，3，4}，{4，6，7}以及{4，5，7}虽然都是强连通的，但因它们的入口结点不唯一，所以都不是上述意义下的循环。



# 循环的查找——基本概念

- **1.** 必经结点：在程序流图中，对任意两个结点**i**和**j**，如果从流图的首结点出发，到达**j**的任一通路都要经过**i**，则称**i**是**j**的必经结点，记为**i DOM j**。
- 注意：**(1)** 首结点是所有结点的必经结点。
- **(2)** 对于任意结点，必有**a DOM a**。

- **2.** 必经结点集：流图中结点**n**的所有必经结点的集合，称为结点**n**的必经结点集，记为**D(n)**。

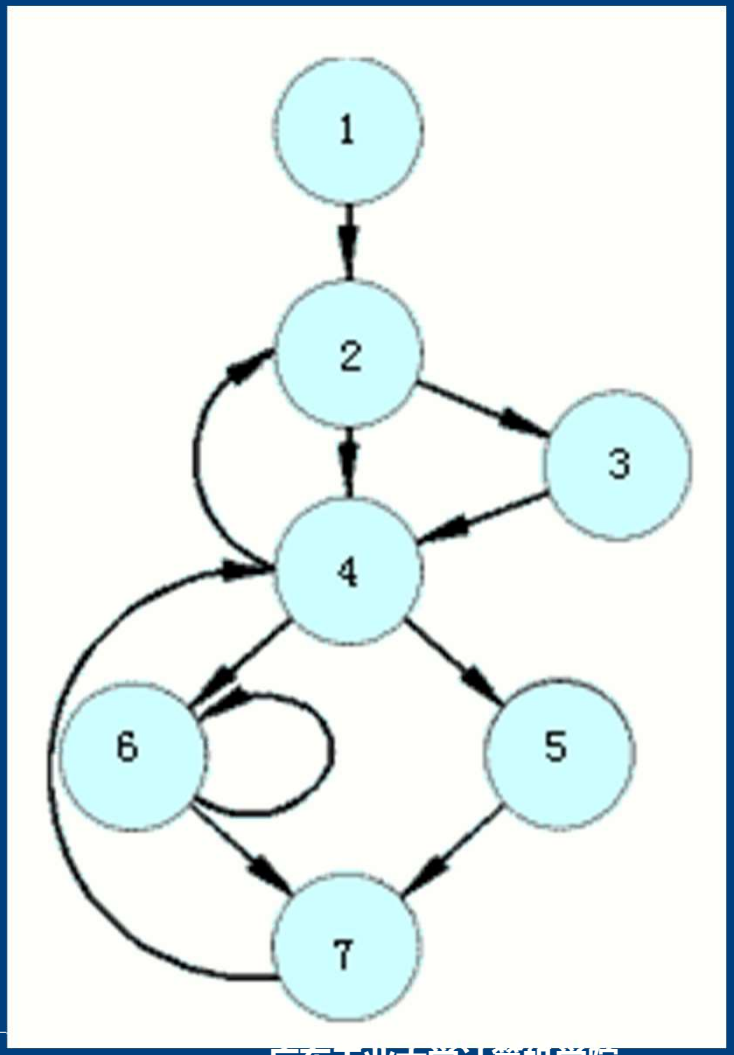
- **3.** 回边：如果存在有向边**a → b**，且**b DOM a**，则**a → b**是一条回边。
- 其实求得必经结点集后，即可求得回边。（书**P259)**

# 循环的查找——概念应用举例

- 如右图所示，可以求得各结点的DOM集：
- D(1) = {1}
- D(2) = {1, 2}
- D(3) = {1, 2, 3}
- D(4) = ?
- D(5) = ?
- D(6) = ?
- D(7) = ?

{1, 2, 4}
{1, 2, 4, 5}
{1, 2, 4, 6}
{1, 2, 4, 7}

- 回边：4 → 2
  7 → 4
  6 → 6
- 循环：{2,3,4,5,6,7}
  {4,5,6,7}
  {6}



# 3. 循环优化——代码外提

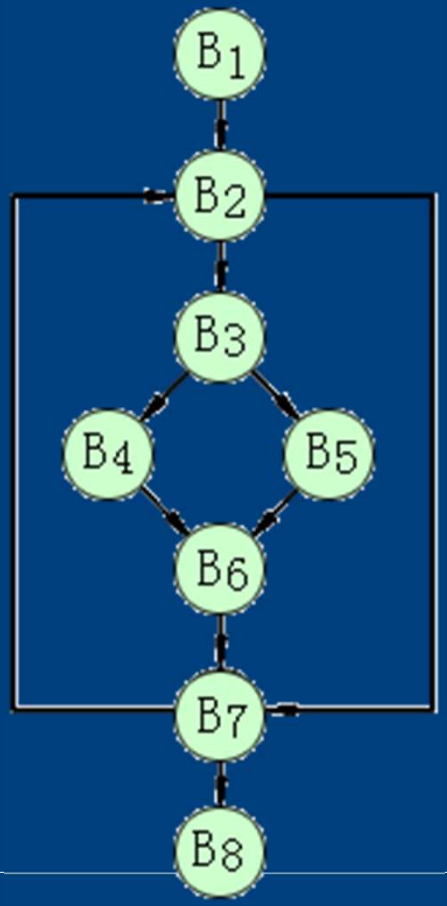
- 在找出了程序流图中的循环之后，我们就可以针对每个循环进行优化工作。循环优化的三种重要技术是：
- **1.** 代码外提；　**2.** 删除归纳变量；　**3.** 强度削弱。

对图中的流图：

(1) 求出流图中各结点n的必经结点集D(n)；

(2) 求出流图中的回边；

(3) 求出流图中的循环。




解答：

（1）

D(1)={1}

D(2)={1,2}

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

D(4)={1,2,3,4}

D(5)={1,2,3,5}

D(6)={1,2,3,6}

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

D(8)={1,2,7,8}

(2)回边 7→ 2

(3)循环 {2，3，4，5，6，7}

# 作业

- 第13周星期四交
- 写上日期
- 及时交作业。

# 主观题 10分

设置

2. 图 10.26 是图 10.25 的 C 代码的部分三地址代码序列。

```
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);
}
```

图 10.25

| | | | |
|---|---|---|---|
| (1) | $i:=m-1$ | (16) | $t_7:=4*i$ |
| (2) | $j:=n$ | (17) | $t_8:=4*j$ |
| (3) | $t_1:=4*n$ | (18) | $t_9:=a[t_8]$ |
| (4) | $v:=a[t_1]$ | (19) | $a[t_7]:=t_9$ |
| (5) | $i:=i+1$ | (20) | $t_{10}:=4*j$ |
| (6) | $t_2:=4*i$ | (21) | $a[t_{10}]:=x$ |
| (7) | $t_3:=a[t_2]$ | (22) | goto (5) |
| (8) | if $t_3<v$ goto (5) | (23) | $t_{11}:=4*i$ |
| (9) | $j:=j-1$ | (24) | $x:=a[t_{11}]$ |
| (10) | $t_4:=4*j$ | (25) | $t_{12}:=4*i$ |
| (11) | $t_5:=a[t_4]$ | (26) | $t_{13}:=4*n$ |
| (12) | if $t_5<v$ goto (9) | (27) | $t_{14}:=a[t_{13}]$ |
| (13) | if $i>=j$ goto (23) | (28) | $a[t_{12}]:=t_{14}$ |
| (14) | $t_6:=4*i$ | (29) | $t_{15}:=4*n$ |
| (15) | $x:=a[t_6]$ | (30) | $a[t_{15}]:=x$ |

图 10.26

(1) 请将图 10.26 的三地址代码序列划分为基本块并给出其流图。

(2) 将每个基本块的公共子表达式删除。

2.0以上版本雨课堂

作答

# 主观题 10分

设置

只需要完成10.29的就可以了。

5. 分别对图 10.28 和图 10.29 的流图：

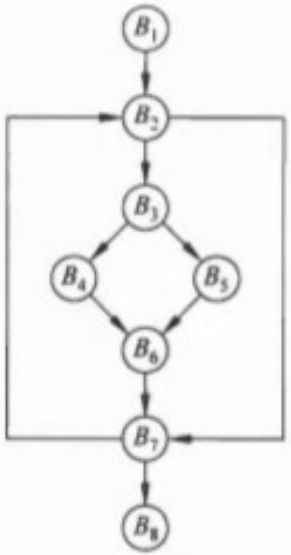



图 10.28

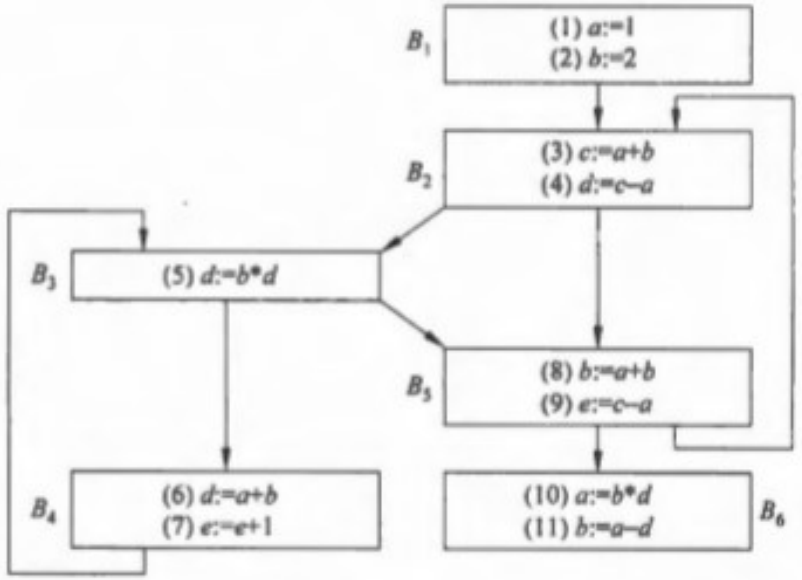



图 10.29

(1) 求出流图中各结点 $n$ 的必经结点集 $D(n)$。

(2) 求出流图中的回边。

(3) 求出流图中的循环。

正常使用主观题需2.0以上版本雨课堂

作答