第 8 章　代码优化（optimization）

二. 循环优化

技术准备 { 控制流分析；数据流分析 }

控制流分析 ⟶ Loop

中间code ⟶ 基本块 ⟶ G ⟶ D(n) ⟶ 回边

数据流分析：中间code + 控制流 ⟹（采集）优化所需信息

一. 数据流分析基础

- **数据流分析**

循环或全局

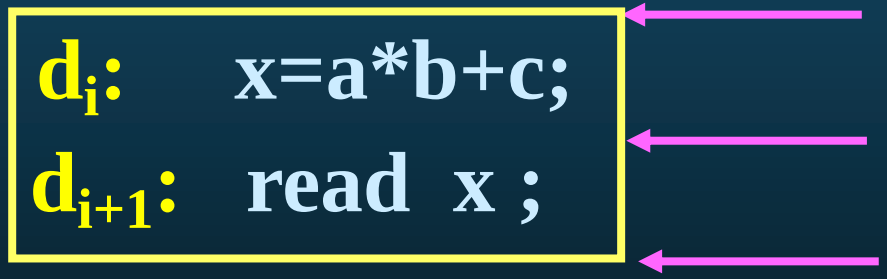
涉及多个基本块范围的优化，编译程序需要收集**整个程序范围内**的有关信息及分布在程序流程图每个基本块的信息，这些信息是程序中**数据流的信息**，这一工作称为数据流分析。

- **点**：指明语句在流图基本块中的位置。

  指一个中间语言语句在其代码序列中的位置。

  如，**入口点**指基本块第一个中间代码前位置；**出口点**指基本块最后一个中间代码后位置；**相邻点**指两个中间代码之间的点。

  例如，



- **定值点：**

是变量x获得值的中间代码的位置d，称为x的定值点。

例如，d_i: x=a*b+c; d_j: read x ;

定值方式
- 赋值语句
- 输入语句
- 函数调用的形参与实参结合

- **引用点：**

  引用变量x的中间代码的位置d，称为x的引用点。

如，d_j：i = i+1

如，d_k：i ++

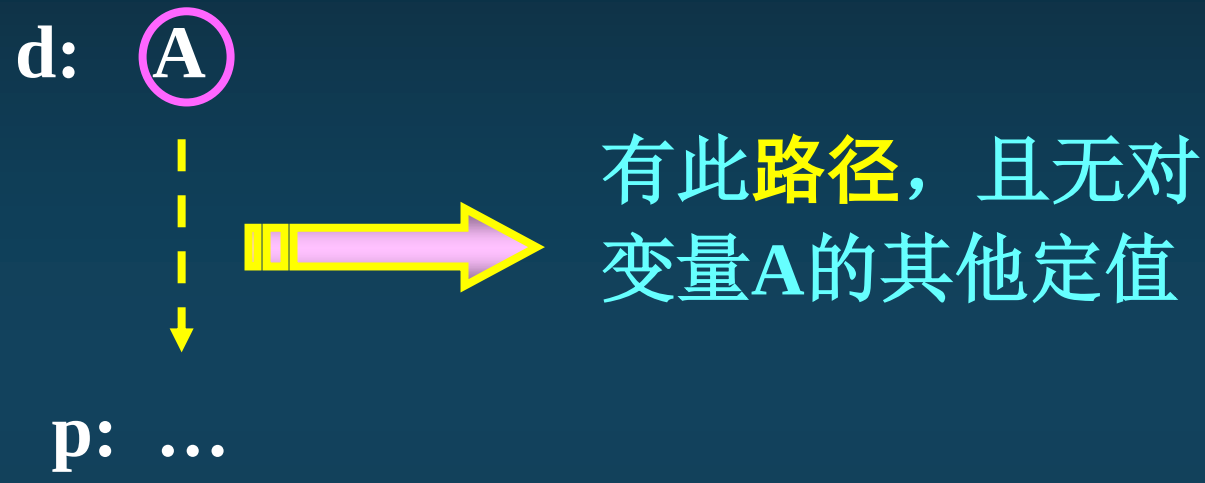
- **到达—定值：**

设有流图G，变量A在G中某点d的定值到达另一点p，是指流图中从点d有一通路到达点p且该通路上没有对变量A的再定值。

约定： <A> — 对变量A的引用；

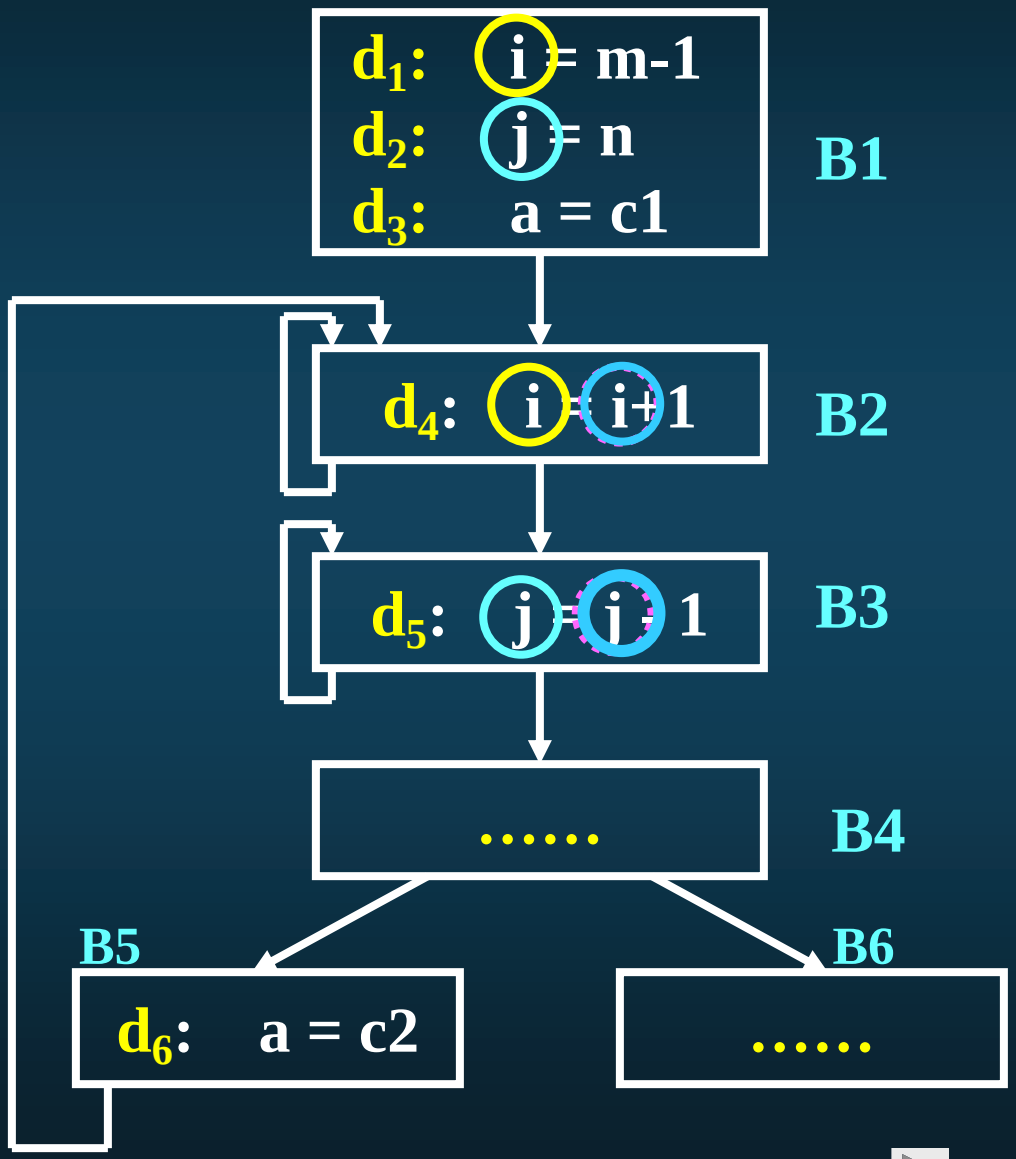
Ⓐ — 对变量A的定值；

- 到达—定值



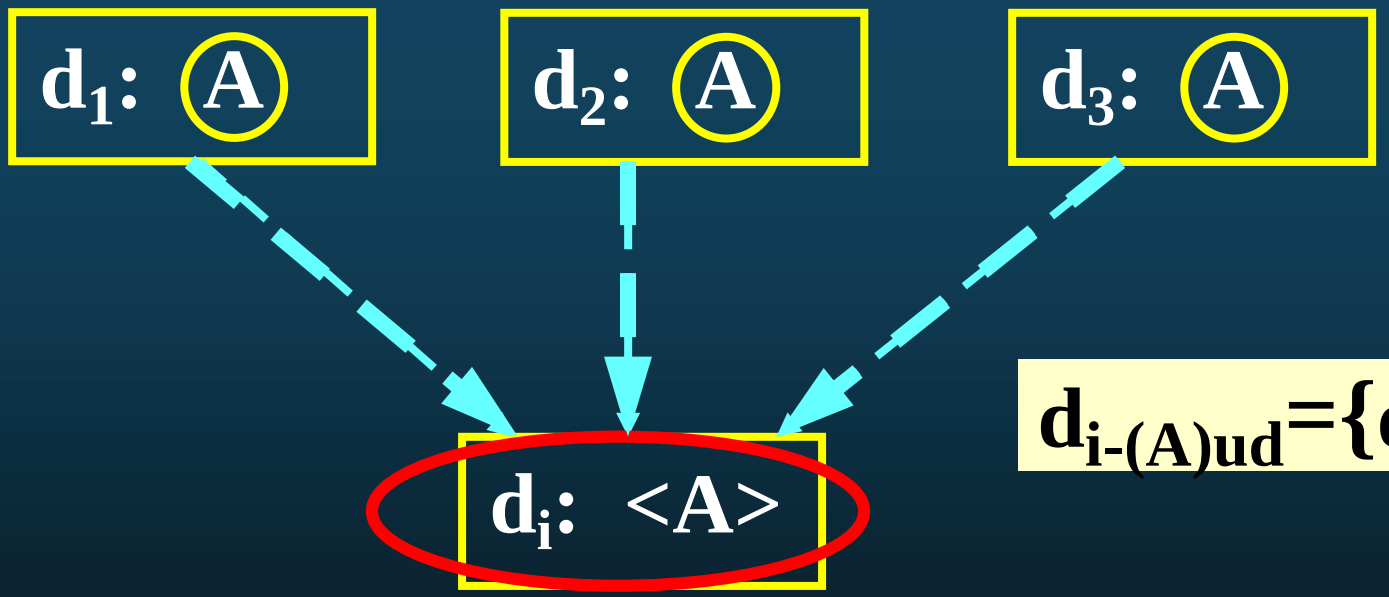
变量A在点d的定值到达点P

例8.6 设有如下流图



- **定义8.6 （ud链)**

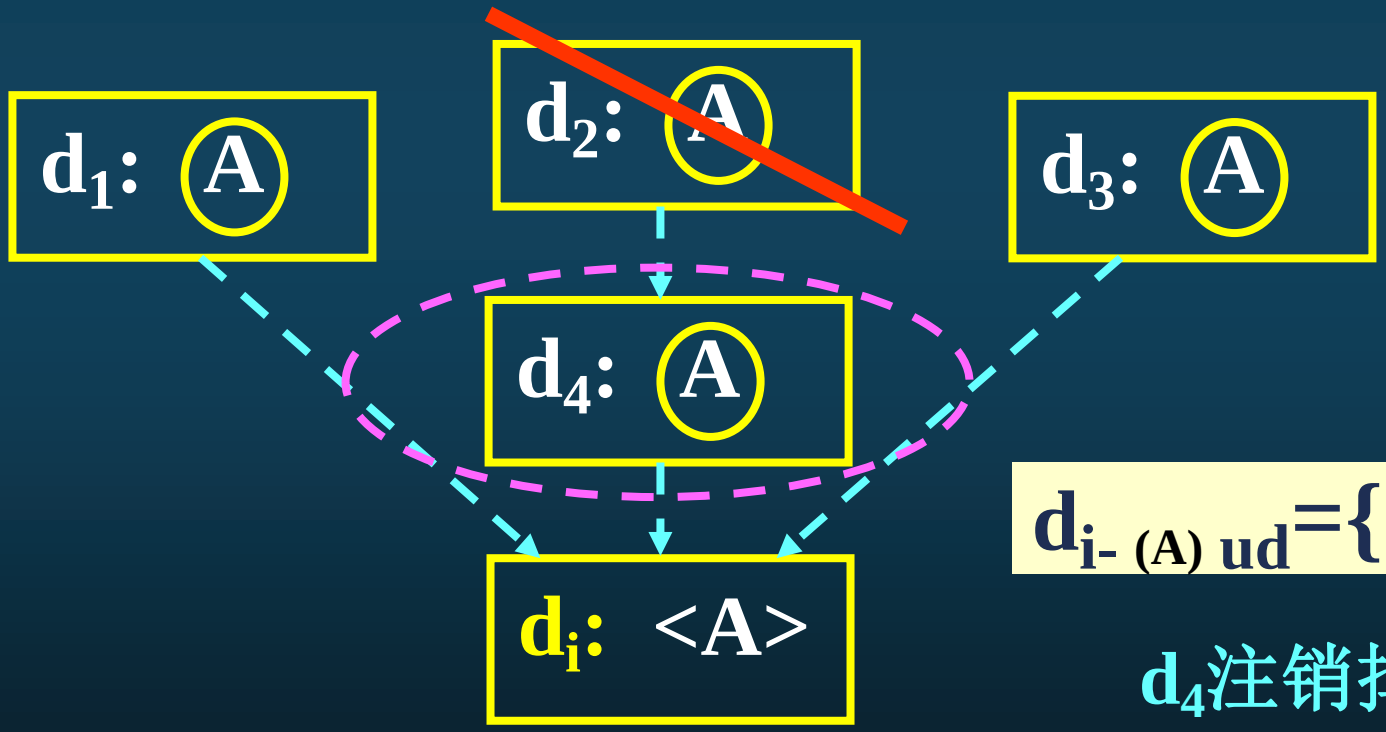
假设在程序中某点P引用了变量A的值，则把G中能到达P的A的定值点的全体，称为A在引用点P的引用—定值链（即ud链）。



$d_{i\text{-}(A)ud}=\{d_1,d_2,d_3\}$

☺ ud链是相对于引用点的定值情况。

即某变量A在点d的引用的ud链，是变量A引用前所有可能到达d点的对A定值的定值表。

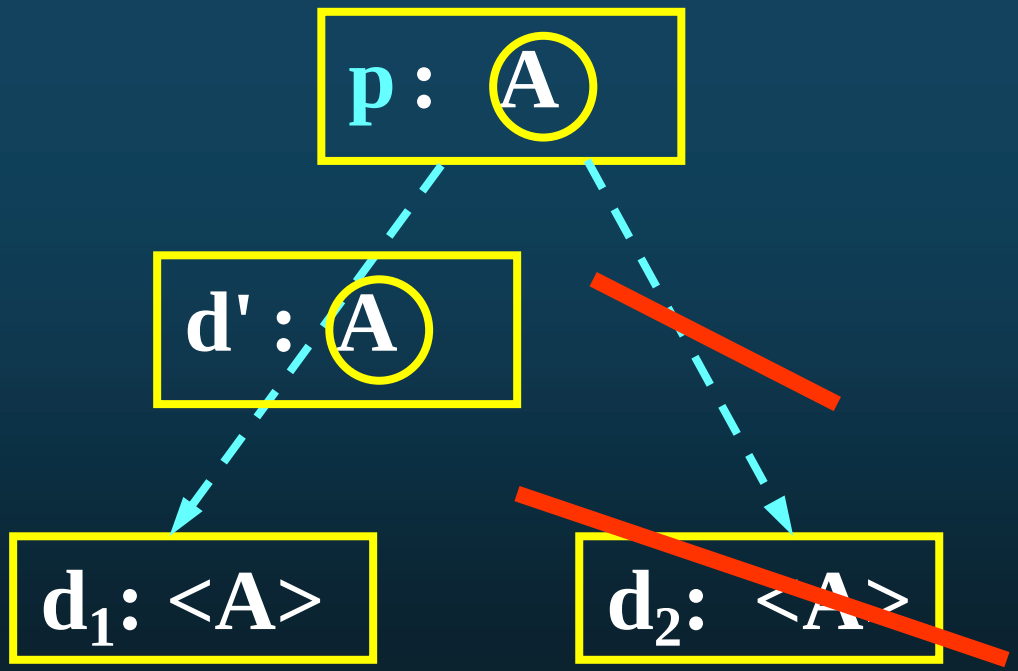


$d_{i-(A)\,ud}=\{d_1,d_4,d_3\}$

d_4注销掉d_2

- **活跃变量：**

在程序中对某变量A和某点P，如果存在一条从P开始的通路，其中引用了A在点P的值，则称A在点P是活跃的，否则称A在点P是死亡的。



A在点p活跃

A在点p, d'活跃

A在点d'活跃，在点p死亡

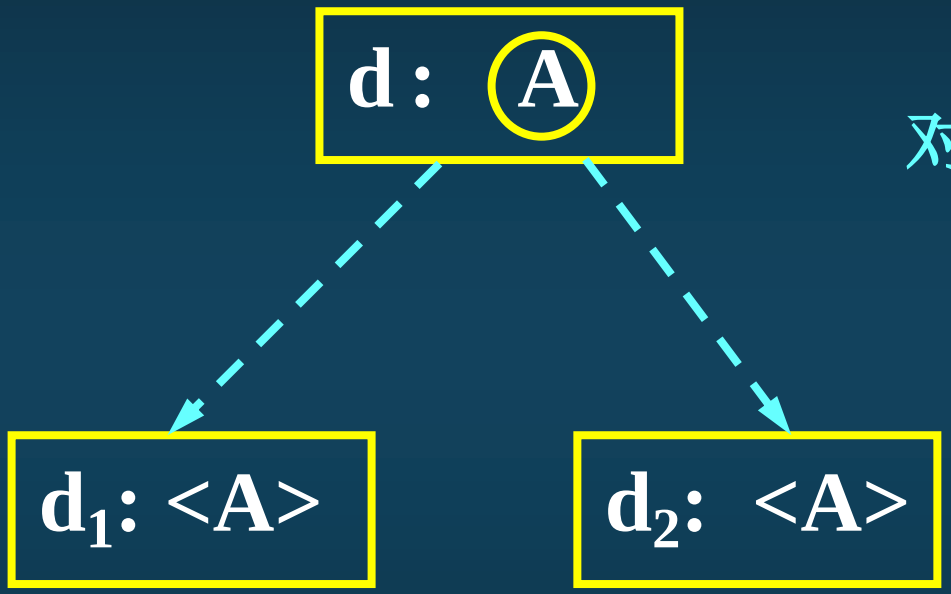
- **定义8.7（du链）**

假设在程序中某点P对一个变量A定值，则把该定值能到达A的引用点的全体，称为A在定值点P的定值—引用链（即du链）。

☺ du链是相对于定值点的引用情况。

即某变量A在点d的定值的du链，是变量A定值后所有可能的引用表。

- du链



对变量A：

$d\text{-}_{du} = \{d_1, d_2\}$

例8.7 设有流图

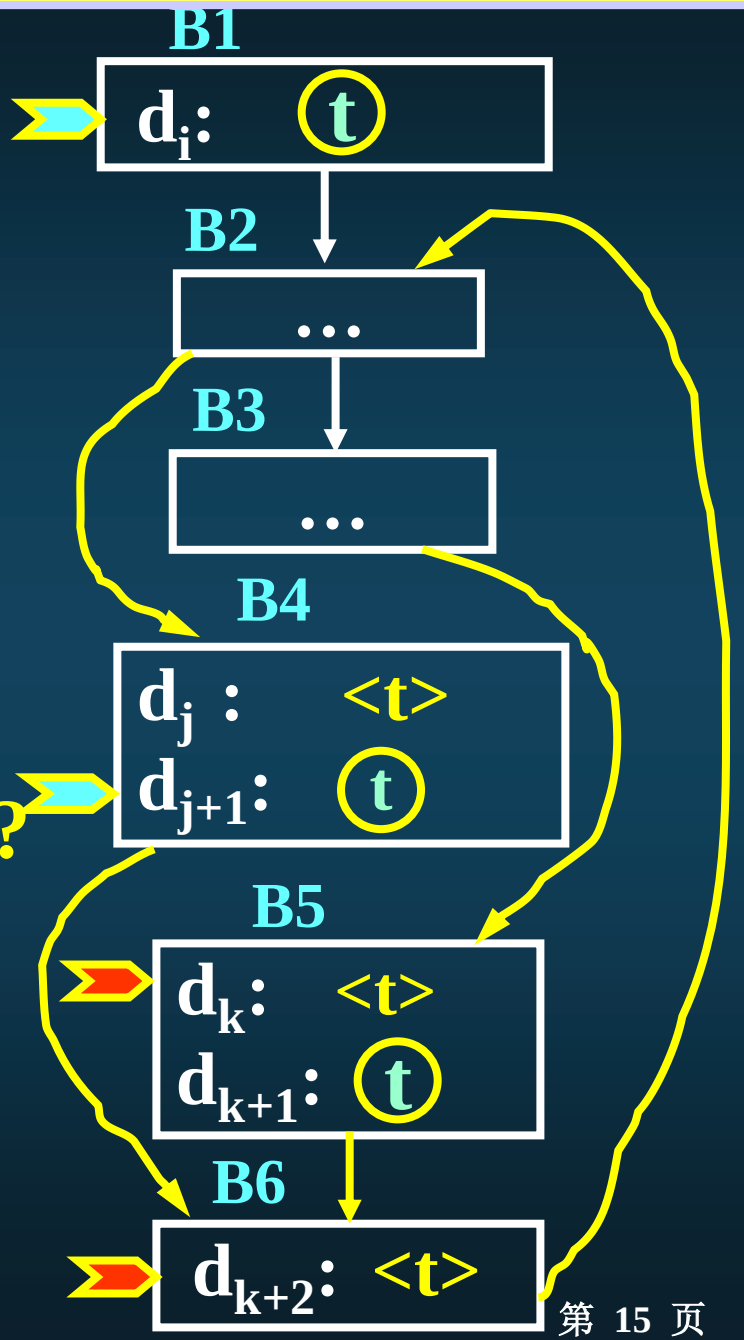
t在点d_{k+2}的ud链={d_{j+1},d_{k+1}}

t在点d_k的ud链={d_i,d_{j+1},d_{k+1}}

t在点d_i的du链={d_j, d_k} d_{k+2} ?

t在点d_{j+1}的du链

={ d_{k+2}, d_j, d_k }



二. 重要数据流方程

编译器把程序的一部分或全部看作一个整体来收集信息，并把收集的信息分配给流图中的各个基本块。

- **到达定值信息**— 完成常量合并，删除无用赋值；
- **活跃变量信息**— 寄存器优化；
- **公共子表达式信息**— 删除冗余运算。

- **典型的数据流方程**

$$out[B] = gen[B] \cup (in[B] - kill[B])$$

其中：

B表示G中某个基本块，也可以为语句；

含义：

当控制流通过一个基本块时，在基本块末尾得到的信息(out)是在该基本块中产生的信息(gen)，或是进入基本块开始点(in)且没有被该基本块注销的信息(kill)；

- **制约建立和求解数据流方程的因素**

1. 产生、注销的概念依赖所需要的信息，考虑数据流方向：

前 ⟶ 后 （每个基本块B_i的有关信息利用其前驱基本块的信息来计算）

如，到达—定值，可用表达式，复写传播。

由in[B] 决定out[B]

后 ⟶ 前 （每个基本块B_i的有关信息利用其后继基本块的信息来计算）

如，活跃变量，非常忙表达式等。

由out[B]决定in[B]

2. 由于数据沿流图的控制路径流动，故数据流分析受程序控制结构影响。

- **到达—定值数据流方程**

在数据流分析中采集程序中量的定值情况（即到达点P的各变量的全部定值点信息）。

$in(B_i)$: 能到达基本块B_i入口点之前的各个变量的所有定值点集。

$out(B_i)$: 能到达基本块B_i出口之后的各变量定值点的集合。

$gen(B_i)$: 在B_i中定值且能到达B_i出口之后的所有定值点集。

kill(B_i): 在基本块B_i外定值，且在B_i中又重新定值的那些变量的定值点的集合。

到达—定值方程

$$out(B) = in(B) - kill(B) \cup gen(B) \quad (8.1)$$

$$in(B) = \cup out(P) \quad P \in P(B) \quad (8.2)$$

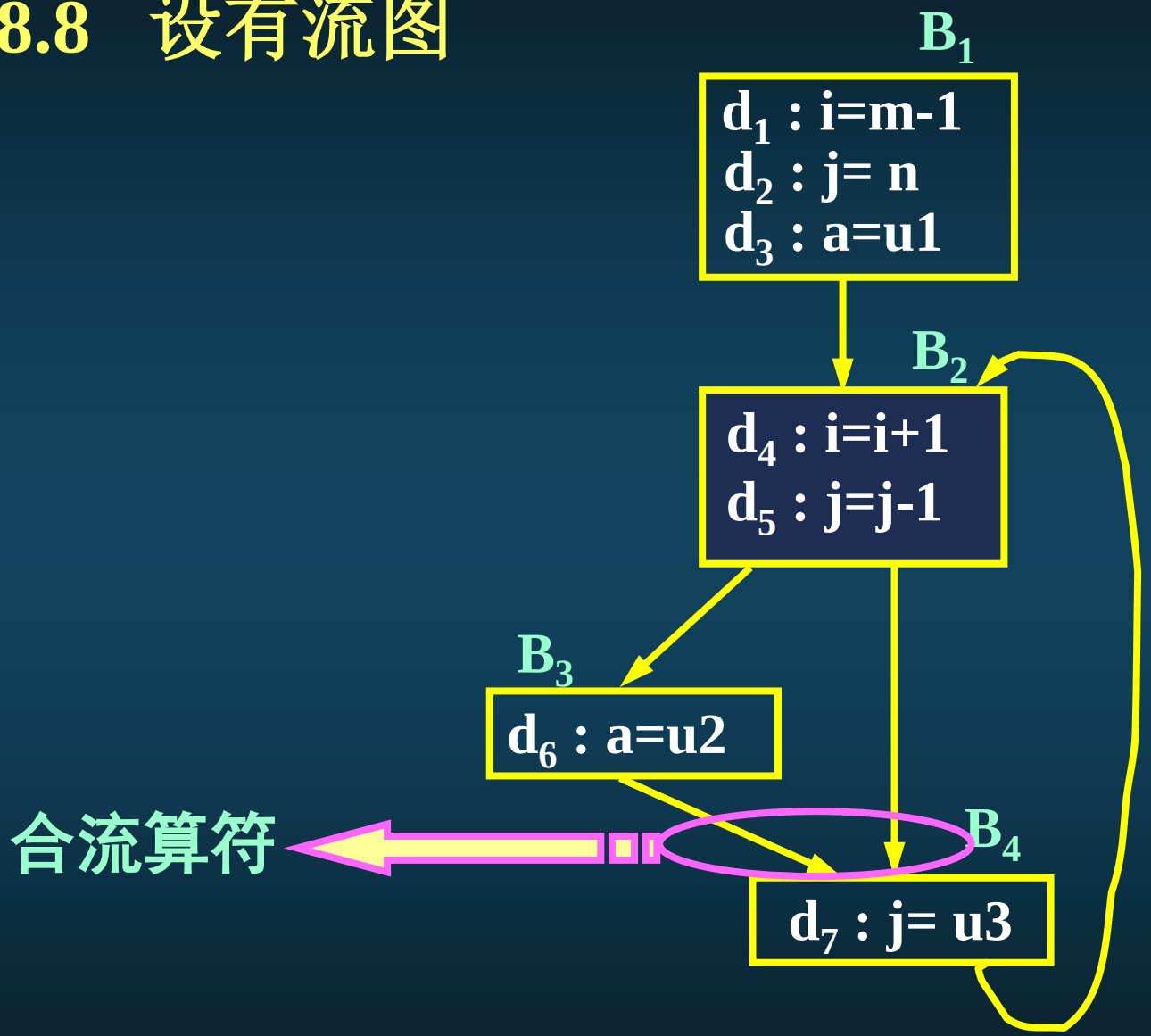
其中：

gen(B)和kill(B)可从其定义出发，直接从给定的流图求出。

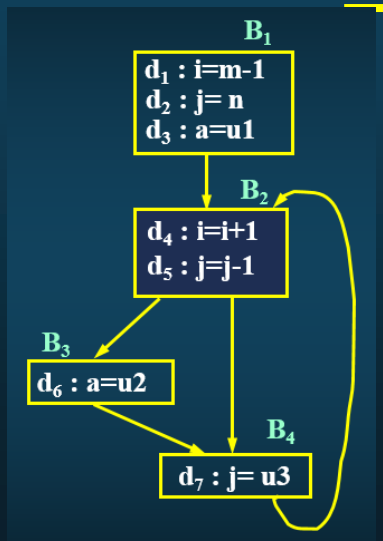
P(B)表示B的所有前驱基本块的集合。

Pred(B)

例8.8 设有流图



	gen(B)	kill(B)
B_1	$\{d_1, d_2, d_3\}$	$\{d_4, d_5, d_6, d_7\}$
B_2	$\{d_4, d_5\}$	$\{d_1, d_2, d_7\}$
B_3	$\{d_6\}$	$\{d_3\}$
B_4	$\{d_7\}$	$\{d_2, d_5\}$



对out(B)，可由以下条件得到：

① 如果某定值点d在in(B)中，而且被d定值的变量在B中未被重新定值，则d也在out(B)中；

② 如果定值点d在gen(B)中,则它一定在out(B)中；

③ 除以上两种情况外，没有其它定值点d∈out(B)。

而对于in(B)，则可知，某定值点d到达基本块B的入口点，当且仅当它到达B的某一前驱基本块的出口点。

对in(B)：

是B的所有前驱基本块的out之和。

- **算法8.2 （到达—定值）**

输入：G中每个基本块B的kill[B]和gen[B]

输出：G中每个基本块B的in[B]和out[B]

方法：

```
{
    for (i=1;i<=N;i++)
     {
        in[Bi]=Φ; out[Bi]=gen[Bi];          /* in[Bi]和out[Bi]的迭代初值 */
     }
        change="真"；       /*change记录相继2次迭代所得的in[Bi]之值不等则为"真"
,
        while (change)          需要继续迭代；若相等，则迭代过程结束，其值为"假"
          {  change= "假"；
             for (i=1;i<=N;i++)
              { NEWIN= ∪ out[P];                                  /*  P∈Pred(Bi)
                if ( NEWIN != in[Bi] )      /* NEWIN记录每次迭代后IN[Bi] 的新值 *
                 {  change= "真"；
                    in[Bi]= NEWIN;
                    out[Bi]=gen[Bi] ∪( in[Bi]-kill[Bi]);
                 }
              }
          }
```

利用到达—定值信息计算ud链

(1) 若在基本块B中，某变量A的引用点u之前有A的定值点d，且d点A的定值能到达点u, 则A在u点的ud链为{d};

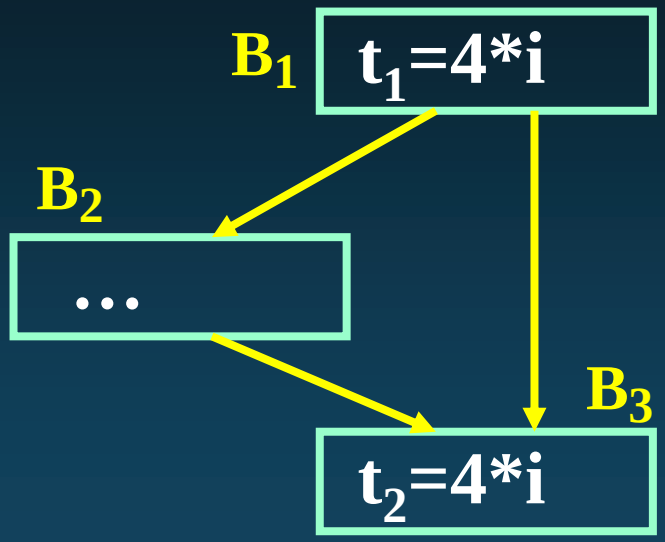
(2) 若在基本块B中，某变量A的引用点u之前无A的定值点，则包含在IN[B]中的全部A的定值点均可到达点u, 所以in[B]中的这些A的定值点组成A在u点的ud链。

- **可用表达式数据流方程**

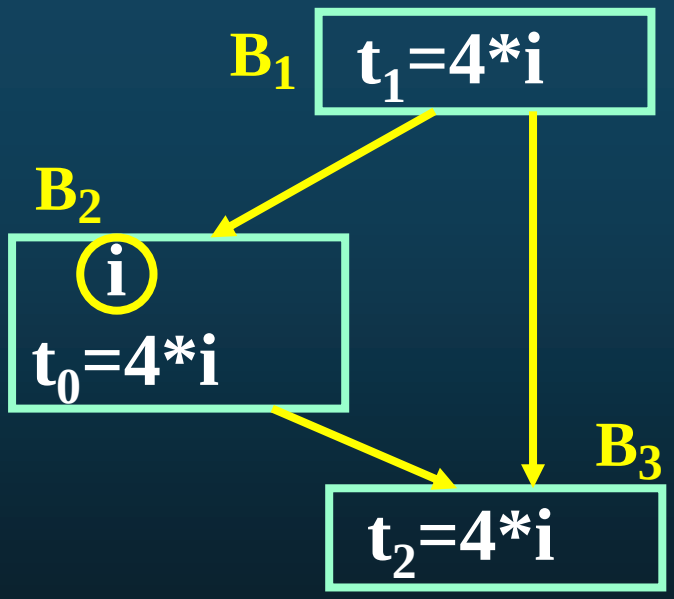
表达式x+y在点P可用：

如果从初始结点到P的每条路径上都计算x+y，并且在最后一个x+y到P之间未对x或y定值，则表达式x+y在点P可用。

若有对x或y的定值，则可用的x+y被注销。



B_2中没有对变量i的定值，则B_1中的4*i 在B_3开始点可用。



B_2中对变量i定值后又计算4*i，则B_1中的4*i 在B_3开始点不一定的可用。

可用表达式数据流方程

$$\begin{cases} out[B] = in[B] - kill[B] \cup gen[B] & (8.3) \\ in(B) = \bigcap_{P是B的前驱} out[P] \quad (B不是开始块) & (8.4) \\ 设in(B_1) = \Phi \quad (B_1是开始块) \end{cases}$$

** 与到达—定值区别：

合流算符是∩；（∵ 一个表达式在块的开始点可用，只有当它在该块的所有前趋块的结束点可用时才行）

活跃变量数据流方程

$$\begin{cases} in_L(B) = out_L(B) - def_L(B) \cup use_L(B) & (8.5) \\ out_L(B) = \bigcup_{S \in Succ(B)} in_L(S) & (8.6) \end{cases}$$

$def_L(B)$：在基本块B中定值，且定值之前未曾在B中引用过的变量的集合。

$use_L(B)$：在基本块B中引用的，但在引用前未曾在B中定值的变量集。

$in_L(B)$: 在基本块B入口点的活跃变量的集合。

$out_L(B)$: 在基本块B的出口点的活跃变量的集合。

循环优化准备

1. 循环查找；(控制流分析)
2. 涉及循环的所有基本量的定值——引用情况信息

数据流是沿着控制流

ud链　　du链

可实施的循环优化：
- 代码外提 (频度削弱)
- 强度削弱
- 删除归纳变量
- 循环展开、合并 …

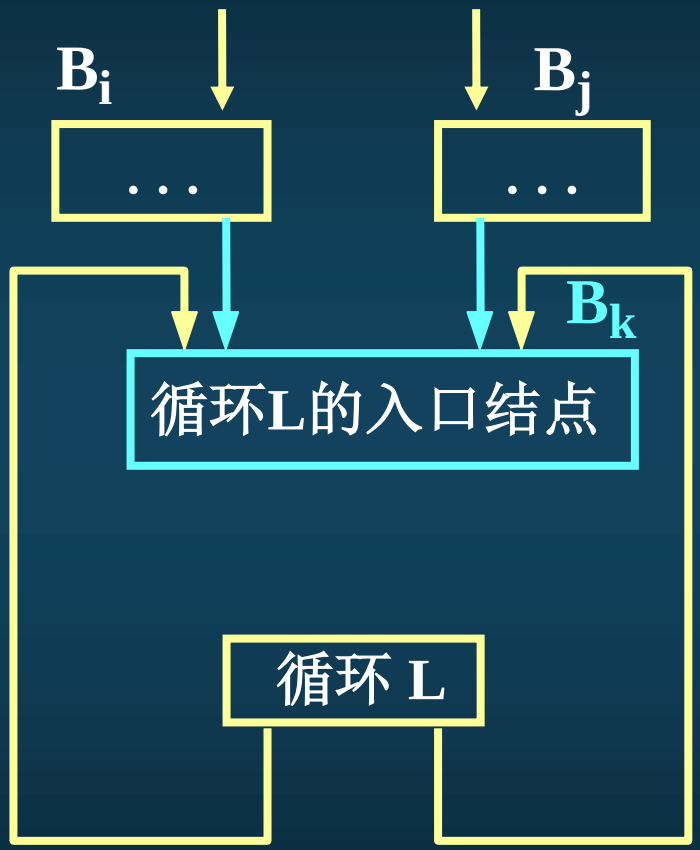
- **循环的前置结点**

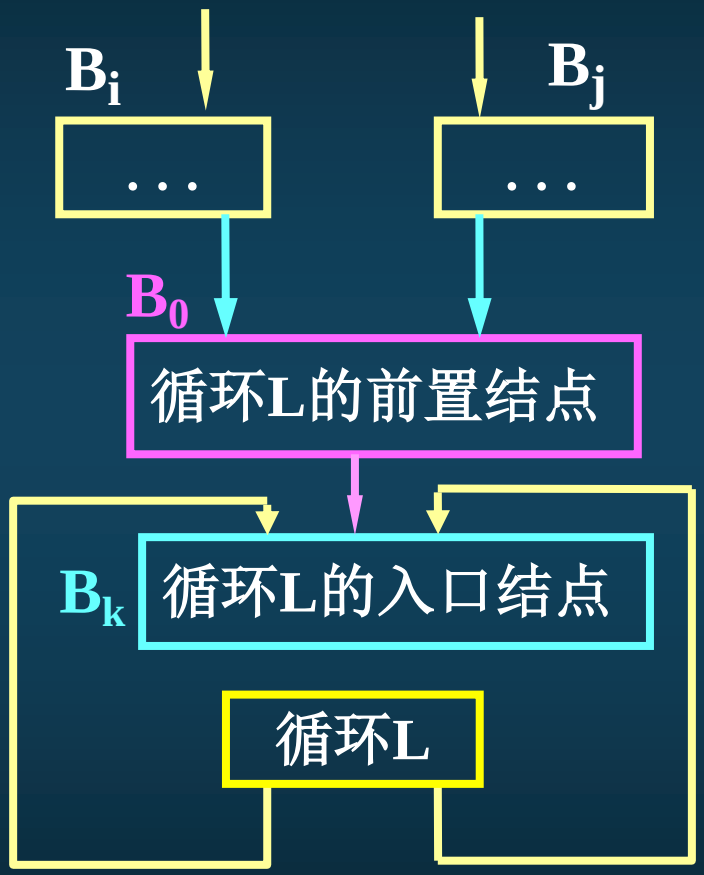
循环的前置结点是在循环的入口结点前建立的一个**新结点**（基本块），它以循环的入口结点为其惟一后继，并将原程序流图中从循环外引至循环入口结点的有向边改引至循环前置结点。

∵ 循环的入口惟一

∴ 前置结点惟一

例8.9 设有流图 (如下面左图)





建立前置结点B_0前的循环L

建立循环L的前置结点B_0后

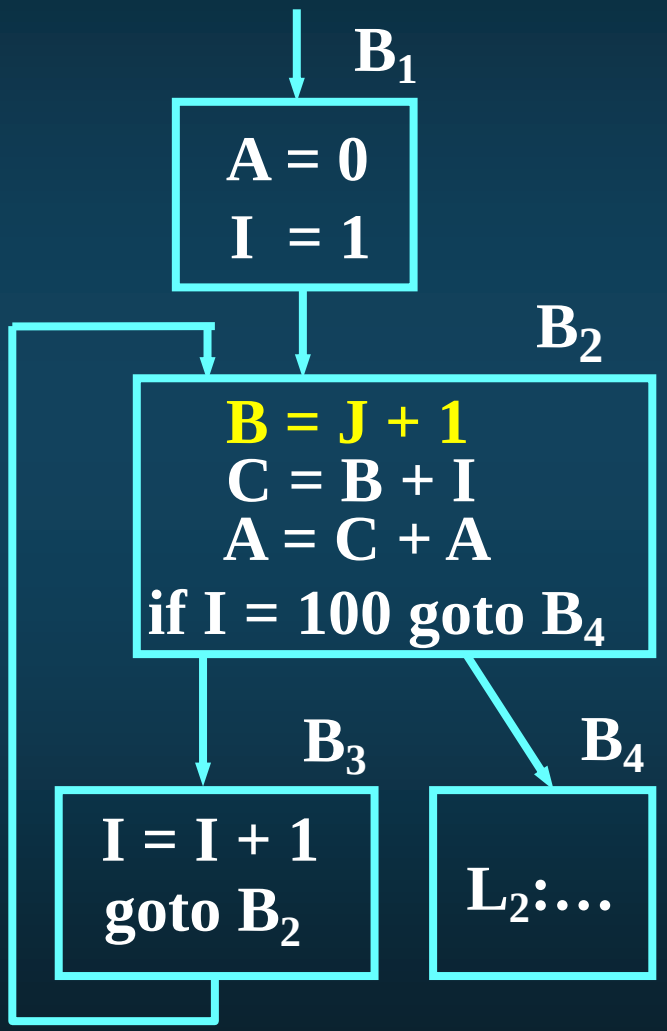
一．代码外提

代码外提就是将循环中的**不变运算**提到循环的前置结点中。这里所指的不变运算，是指与循环执行次数无关的运算或不受循环控制变量影响的那些运算。

例如，设循环L中有形如A=B op C的语句，如果B和C是常数，或者B和C虽然是变量，但到达B和C的定值点皆在循环L外，则在循环中每次计算出的B op C的值始终不变。

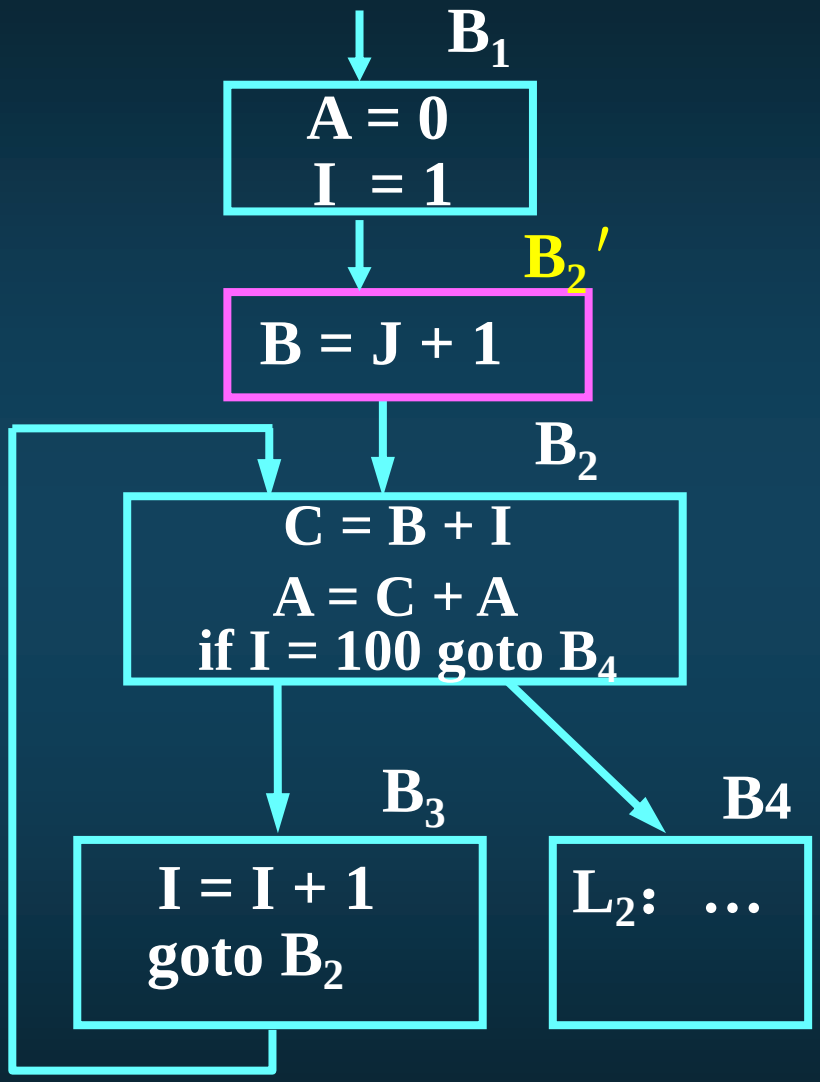
例8.10 给出以下源程序及该程序流图

```
        A = 0;
        I = 1;
L1:     B = J + 1;
        C = B + I;
        A = C + A;
        if (I=100) goto L2;
        I = I + 1;
        goto L1 ;
L2 :    ...
```



$<B_3, B_2>_{loop} = \{B_2, B_3\}$

循环中，B_2中的语句B = J+1，由于循环中没有对J的定值点，所以J的所有引用的定值点都在循环外，它是循环的不变运算，可提到循环的前置结点B_2'中。



- **代码外提算法的设计**

  (1) 查找循环中的不变运算； （X1）

  (2) 实施代码外提； （X2）

- **算法8.3 （ X1: 查找循环中不变运算）**

  设有循环L

  输入： 循环L；L中的所有变量引用点的ud链信息；

  输出： 查找、标识“不变运算”后的循环L；

方法：

(1) 依次查看L中各基本块的每个语句，如果其中的每个运算对象为常数或定值点在L外（据ud链判断），将该语句标记为“不变运算”；

(2) 重复第(3)步，直至没有新的语句被标记为“不变运算”为止；

(3) 依次查看未被标记为“不变运算”的语句，如果其运算对象为常数或定值点在L外，或只有一个到达-定值点且该点上的语句已标记为“不变运算”，则将被查看的语句标为“不变运算”。

例8.11 给出以下循环体代码和数据流信息

$r_{ud}=\{ d_{11}\}$

变量r的定值点在循环外

Loop

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

- **算法8.4　（X2: 代码外提）**

输入：　循环L；ud链信息和必经结点$D(n_i)$信息

输出：　L'；(加前置块，已经外提“不变运算”后的循环L)

方法：

(1) 求出循环L中所有不变运算。(call X1)

(2) 对(1)求出的每一不变运算

S：A=B op C 或 A=op B 或 A=B，

检查是否满足如下条件之一：

①

（i）S所在的结点是L的所有出口结点的必经结点；

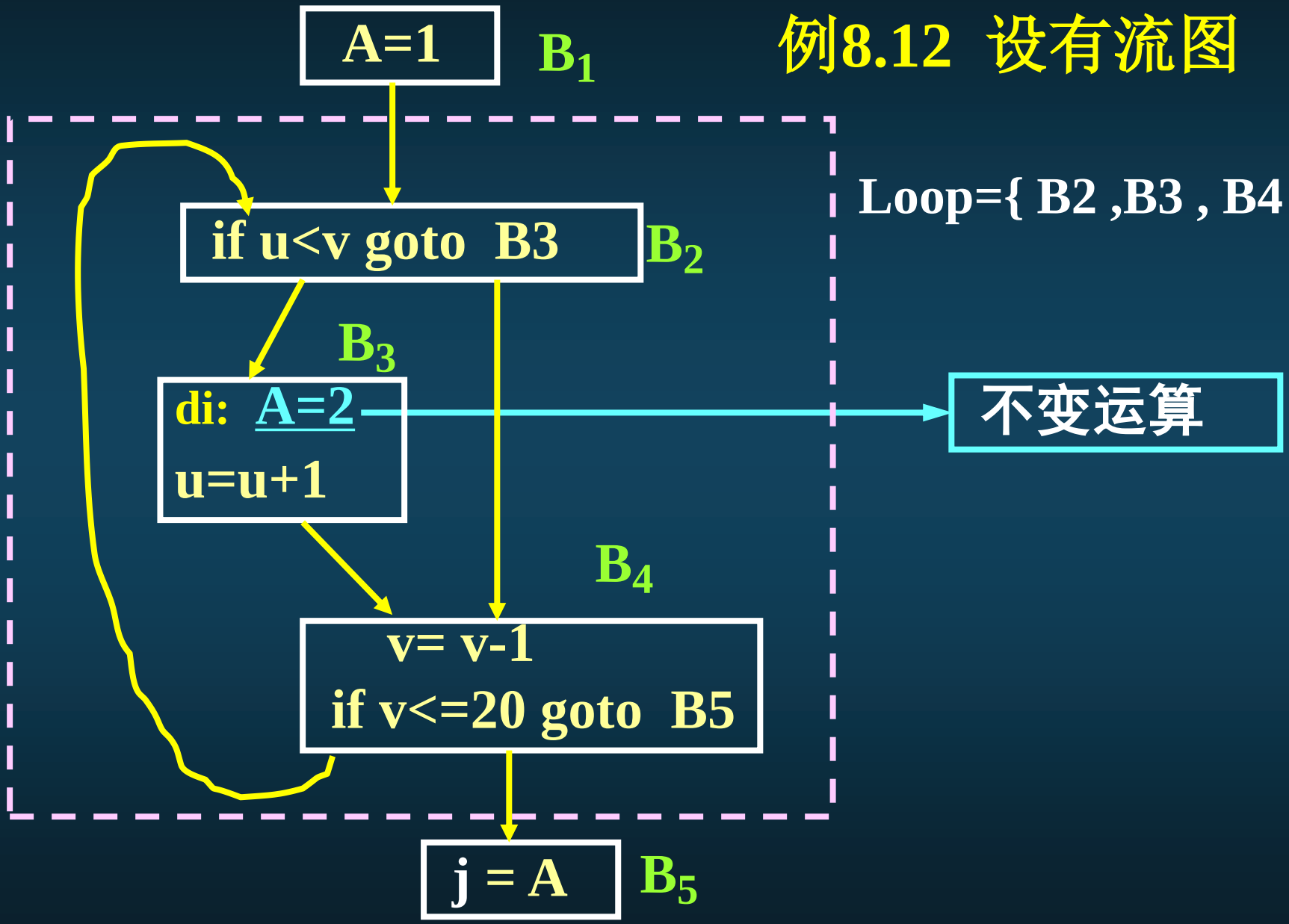
（ii）A在L中其它地方未再定值；

（iii）L中所有A的引用点只有S中A的定值才能到达；

② A在离开L后不再是活跃的，且条件①的(ii)和(iii)成立。所指的A在离开L后不再是活跃的是指，A在L的任何出口结点的后继结点(指不属于L的后继）的入口处不是活跃的。

(3) 按第 (1)步找出的不变运算的顺序，**依次**把符合(2)的条件之一的不变运算S外提到L的前置结点中。但若S中的运算对象（B或C）是在L中定值的，那么，只有当这些定值语句都提到前置结点中后，才可把S也外提。

例8.12 设有流图



Loop={ B2 ,B3 , B4 }

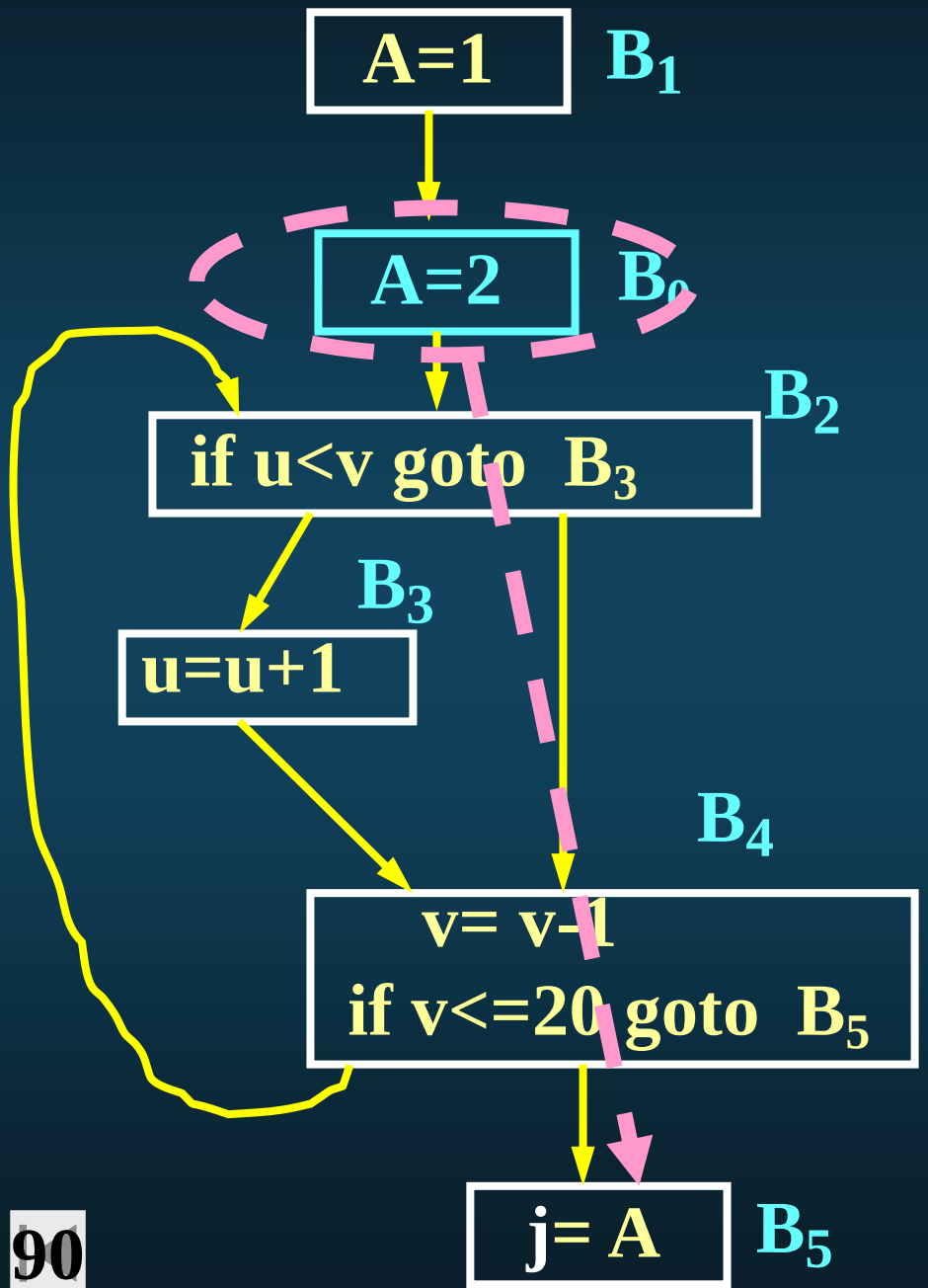
说明外提的限制条件：

** d_i所在的结点是L的所有出口结点的必经结点；（i）

否则，将“不变运算”外提后，会改变程序的计算结果。

外提不变运算后的流图

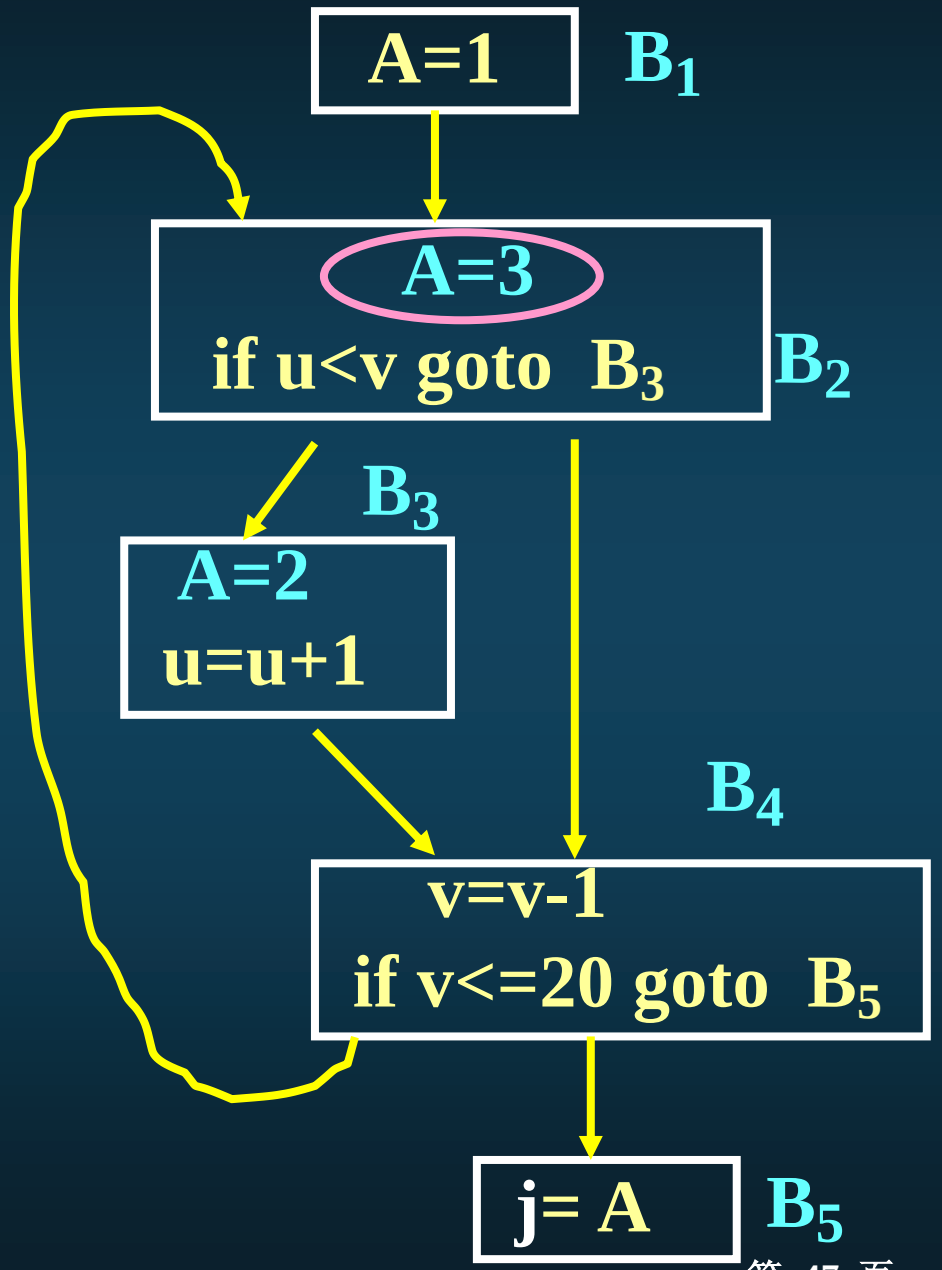
说明(i)



** 在L中对A不止一次定值情况下不允许外提

说明(ii)

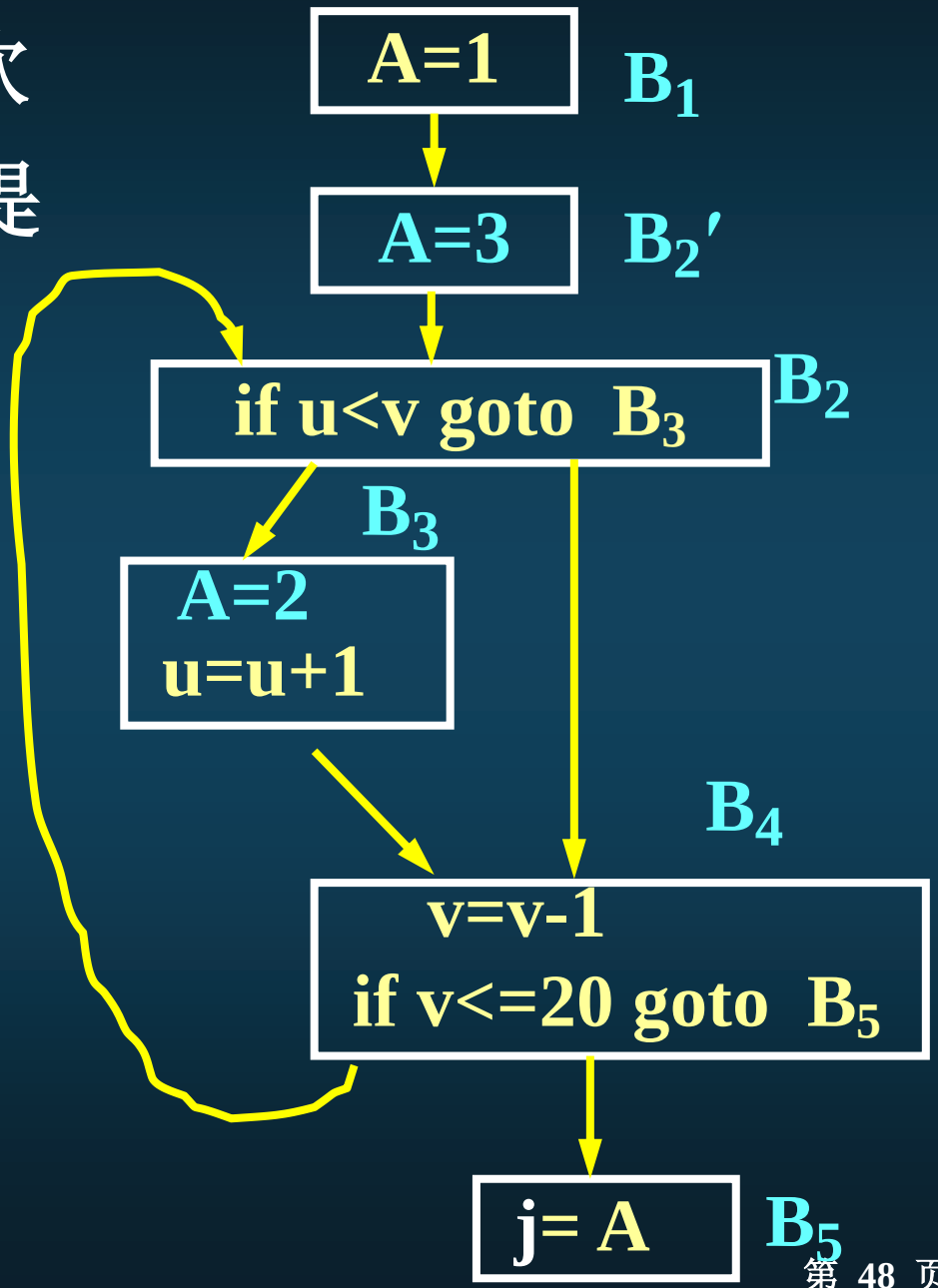
条件(i)不能阻挡将A=3外提



** 在L中对A不止一次定 值情况下不允许外提

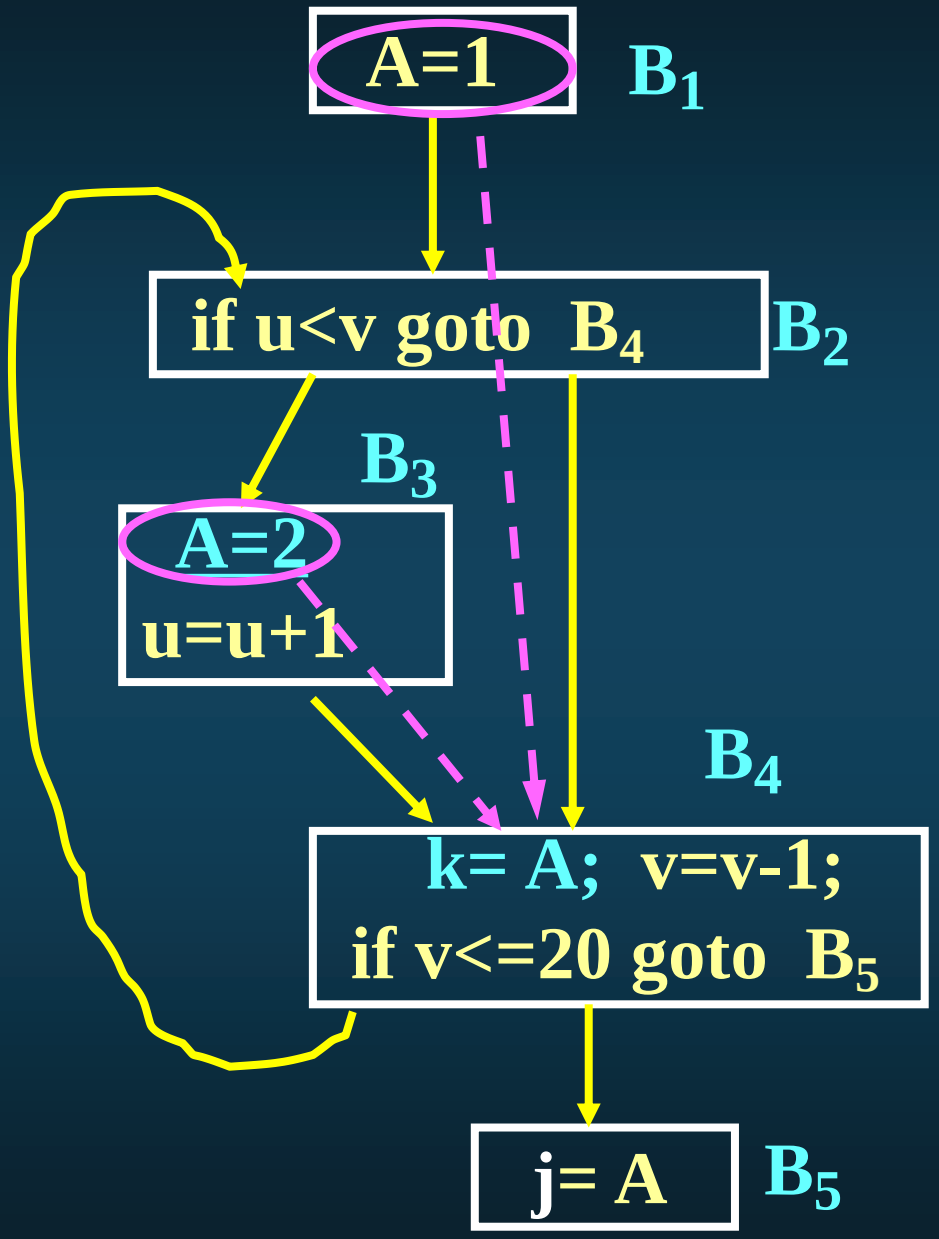
说明(ii)

注意：循环进入B_2、B_3、B_4以后



** L中所有A的引用点只有S中A的定值才能到达；

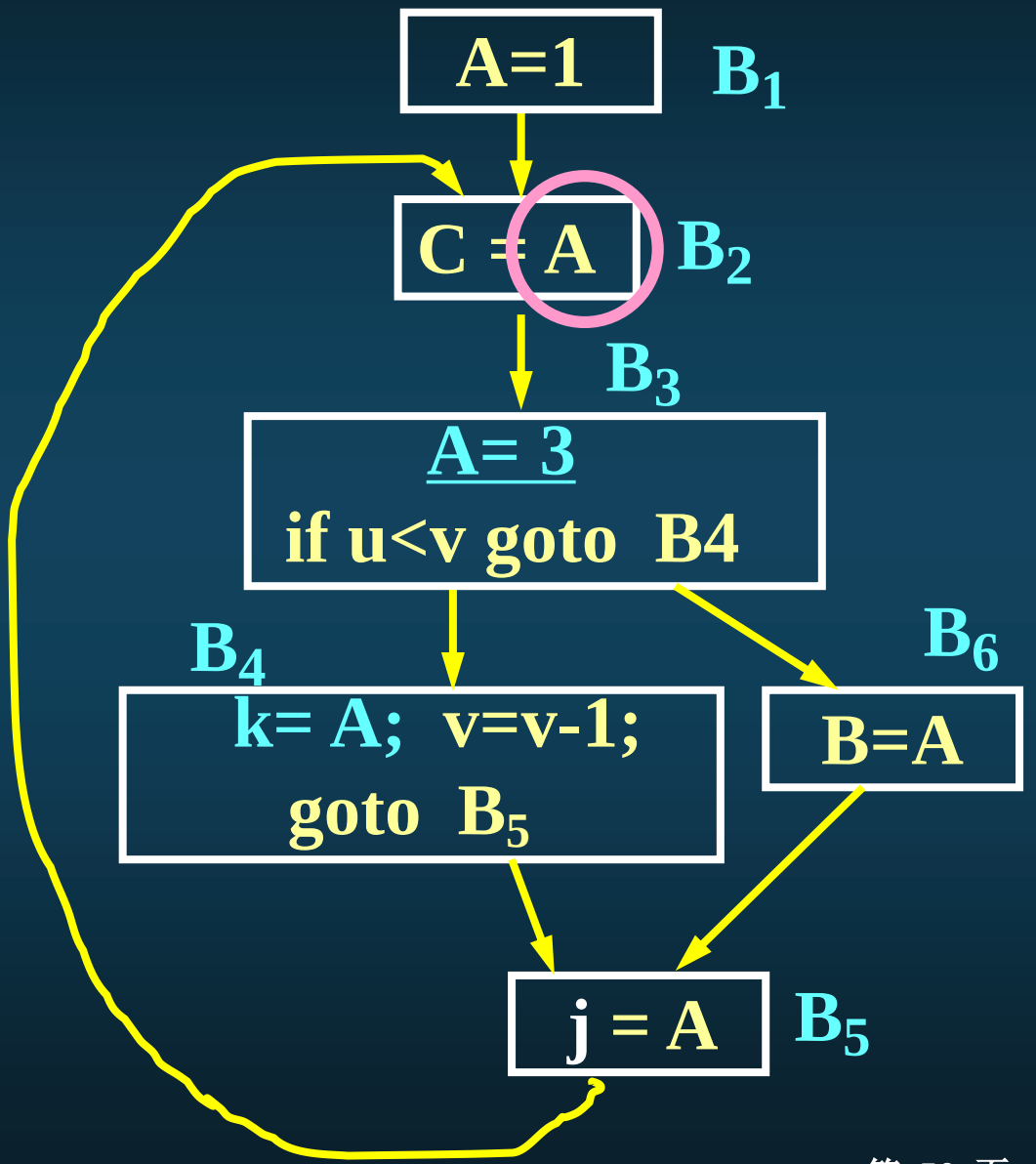
说明(iii)



** L中所有A的引用点只有S中A的定值才能到达；

说明(iii)

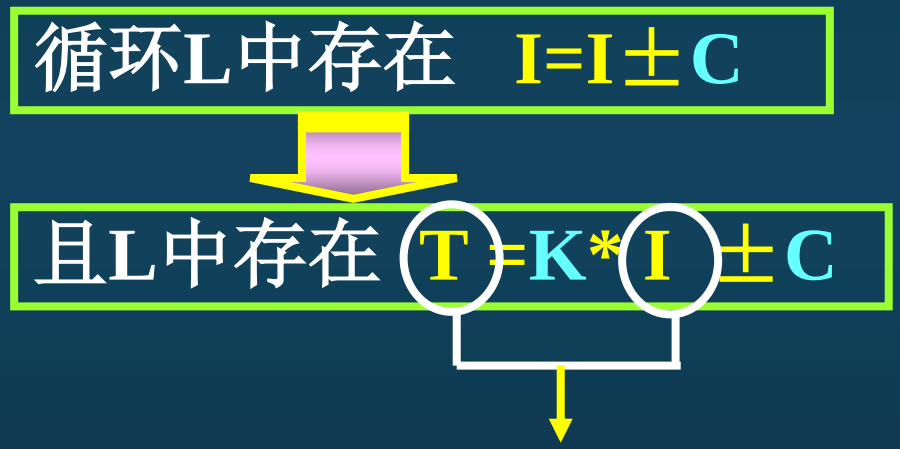
条件(i)(ii)都不能阻挡将A=3外提



二．强度削弱与删除归纳变量

强度削弱是将程序中强度高的运算使用强度低的运算替代，以便使程序运行时间缩短。

▲ 一般情况



- **定义8.9** （基本归纳变量/归纳变量）

如果循环中变量I仅有惟一的 I=I±C 形式的赋值，其中C为循环不变量，则称I为循环中基本归纳变量。

如果I是循环中一基本归纳变量，变量J在循环中的定值总可化为I的同一线性函数的形式：$J= C_1*I±C_2$，其中C_1，C_2是循环不变量，则称J是归纳变量，并称J与I同族。

- 循环优化中强度削弱和删除归纳变量

有次序且相关

(寻找归纳变量) (强度削弱) (删除归纳变量)

- **算法8.5 （W1：查找归纳变量）**

输入： 带有到达—定值信息和循环不变运算信息的循环L

输出： 查找循环L中的一组归纳变量

方法：

step1: 扫描L，找出所有基本归纳变量；（I=I±C）

step2: 寻找L中只有一个定值的K (归纳变量)，其形式为：

K=J*C;　K=C*J;　K=J/C;　K=J ± C;　K=C ± J ;

（其中：C为循环不变量；J为基本归纳变量或归纳变量；）

- **算法8.6 （W2：强度削弱）**

输入： 带有到达—定值信息的L和归纳变量族

输出： 进行强度削弱优化后的L

方法： 依次考察基本归纳变量I，对每个形如

J=C*I ±d的四元式：

step1: 建立新变量S;

step2: 用J=S代替原对J的定值；

step3: 在L中每个I=I+n(n为常量)的四元式后加上

S=S+C*n；

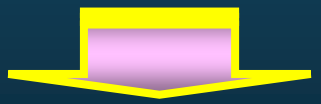
step4: 保证S在L入口的初值为C*I+d；

- **算法8.7　（W3：删除归纳变量）**

输入： 带有到达—定值信息、循环不变运算信息和活跃变量信息的L

输出： 删除归纳变量优化后的L

方法： 考察每个仅用于计算同族中其它归纳变量和条件分支的基本归纳变量I，取I族的一个归纳变量一个归纳变量J，将含I的测试改为用J代替。



据du链信息

替代后的I不再引用时，从L中删去对I定值的语句

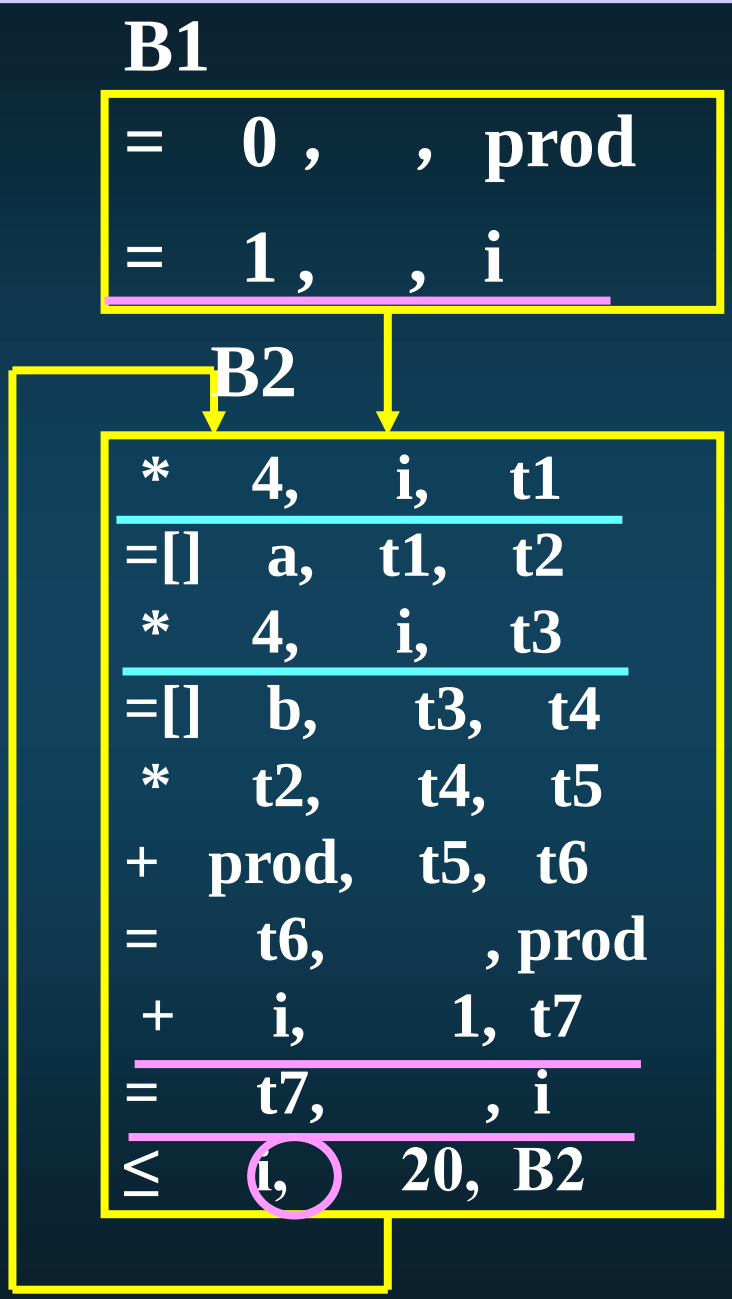
例8.13 P243 — 例7.6

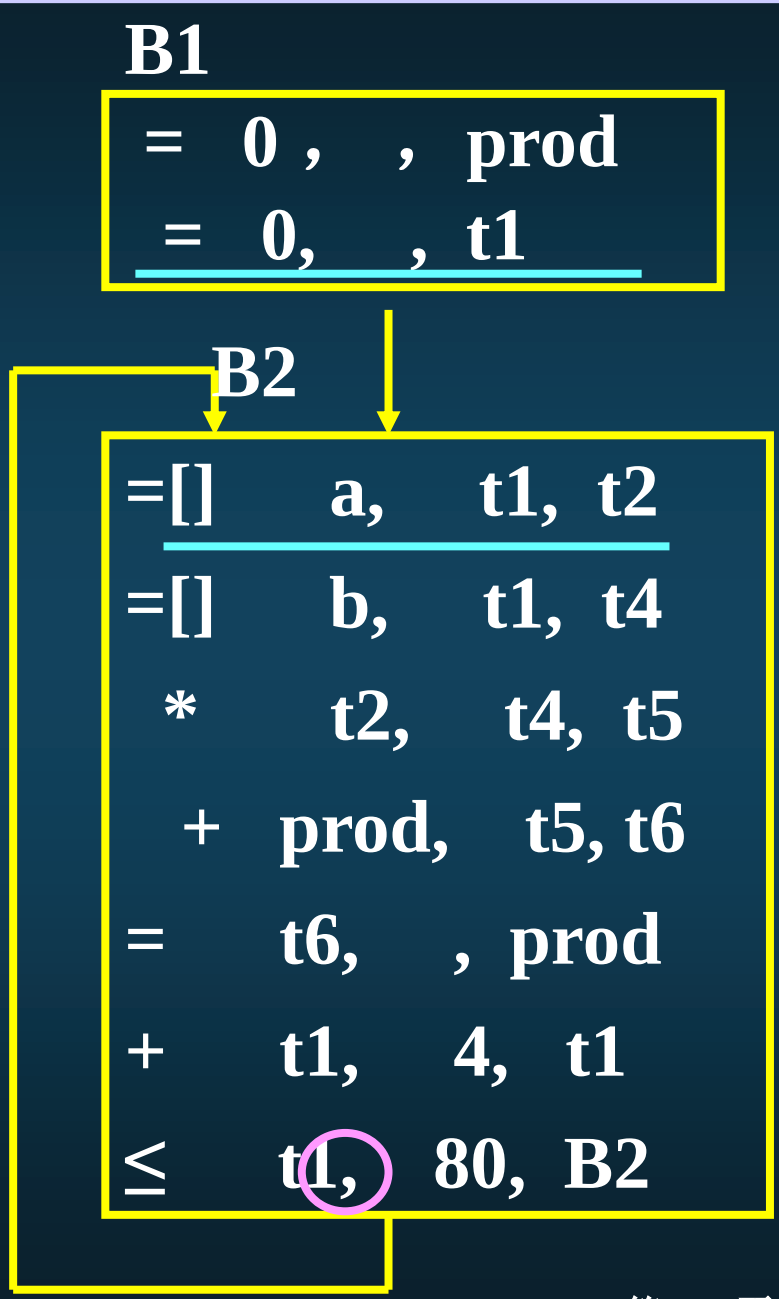
设计算大小为20的两个向量(一维数组表示)a与b的内积公式为

$$prod=a1\times b1+a2\times b2+\ \ldots\ +a20\times b20$$

实现计算的源程序片段如下：

```
prod=0;  i=0;
repeat
    prod=prod+a[i]*b[i];
    i++;
until i>20
```


B1
= 0 , , prod
= 1 , , i
B2
* 4, i, t1
=[] a, t1, t2
* 4, i, t3
=[] b, t3, t4
* t2, t4, t5
+ prod, t5, t6
= t6, , prod
+ i, 1, t7
= t7, , i
≤ i, 20, B2


B1
= 0 , , prod
= 0, , t1
B2
=[] a, t1, t2
=[] b, t1, t4
* t2, t4, t5
+ prod, t5, t6
= t6, , prod
+ t1, 4, t1
≤ t1, 80, B2

实施优化的综合考虑

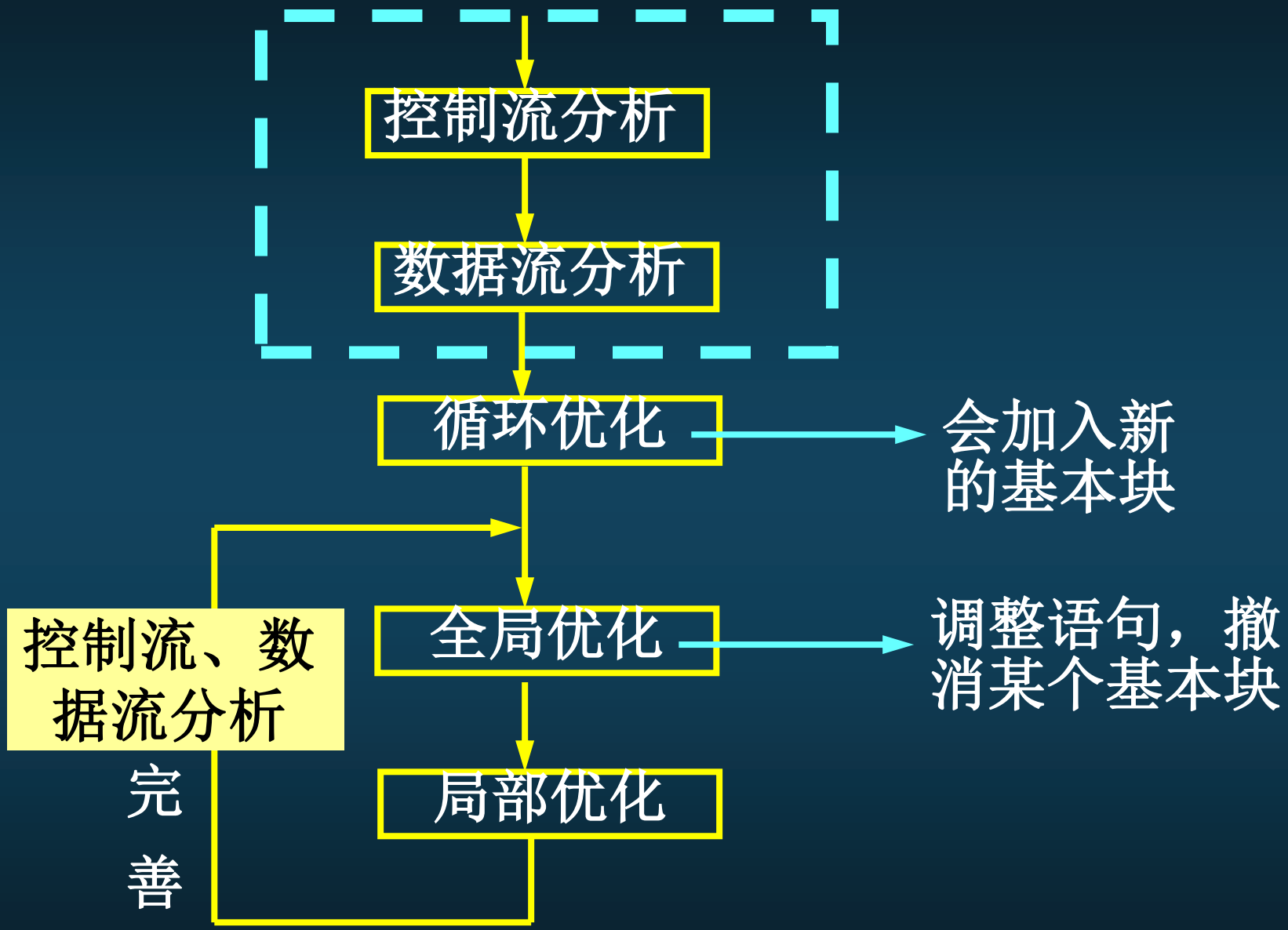
综合应用各类优化技术的共性应考虑的因素：

一. 中间代码的选择

1. 便于生成目标代码；
2. 便于优化；

二. 确定实施各类优化的内容、次序和具体技术

1. 内容：适合实施的具体优化工作；
2. 次序：对提高优化效率，减少优化代价很重要。


控制流分析
数据流分析
循环优化
会加入新的基本块
控制流、数据流分析
全局优化
调整语句，撤消某个基本块
局部优化
完善

三. 平衡提高优化效率、减少优化代价的矛盾

- 优化效率本身的矛盾： 代码执行**时间**的减少；
  存储**空间**占用的减少；
- 优化考虑严密、完善，不顾及完成优化所花费的代价，则会相对抵消整个编译程序的效率、质量甚至影响优化的实际效率；
- **策略：** 针对具体问题抓住主要矛盾，估计主要因素；如，
  * 目标机环境；
  * 循环优化：最内层优化；
  * 数据流分析信息对优化的应用价值；
  * 通用、专用性语言，库函数、包 …