中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle

编 译 原 理

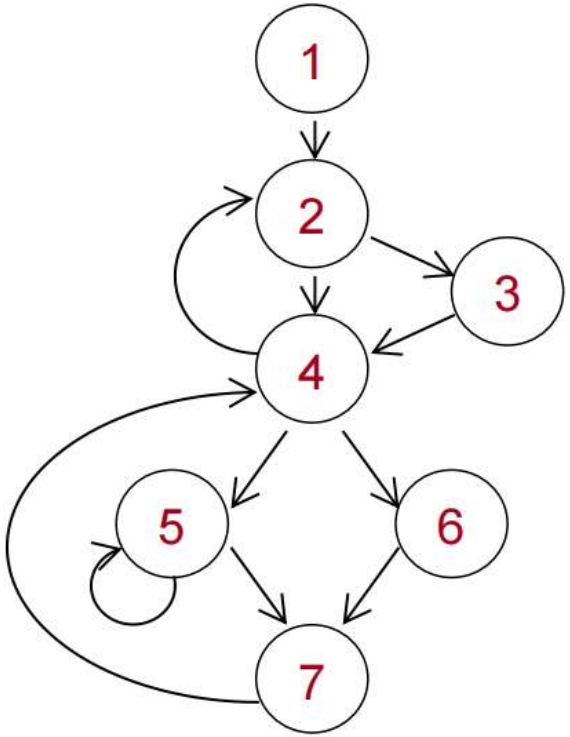
第九章 代码优化（2）

郑馥丹

zhengfd5@mail.sysu.edu.cn

3. 难点&重点：如何在流图中确定循环[Loop]

- 循环[也称自然循环，natural loop]
  - **强连通子图**[strongly connected subgraph]
  - 具有**唯一的入口**（头结点[header]）（**入口：外部结点进入循环的入口**）
  - 循环可以表示为一个结点序列



There are 3 loops:

{5}

{4, 5, 6, 7} 入口为4

{2, 3, 4, 5, 6, 7} 入口为2

They are NOT loops:

{2, 4} Both 2 and 4 are entries

{2, 3, 4} Both 2 and 4 are entries

{4, 5, 7} Both 4 and 7 are entries

{4, 6, 7} Both 4 and 7 are entries

3. 难点&重点：如何在流图中确定循环[Loop]

- 基本思路：根据流图计算所有结点的支配结点集[Dominators]，然后得到流图中的回边[Back Edges]，根据回边就可以确定该流图中的循环。
  - ① 计算支配结点集[Dominators]
  - ② 得到回边[Back Edges]
  - ③ 确定循环[Loop]

3. 难点&重点：如何在流图中确定循环[Loop]

① 计算支配结点集[Dominators]

– 对任意两个结点m和n，若从流图首结点出发，到达n的任一通路都要经过m，则称m是n的支配结点，记为：m **DOM** n

– 流图中结点n的所有支配结点的集合成为结点n的支配结点集，记为D(n)

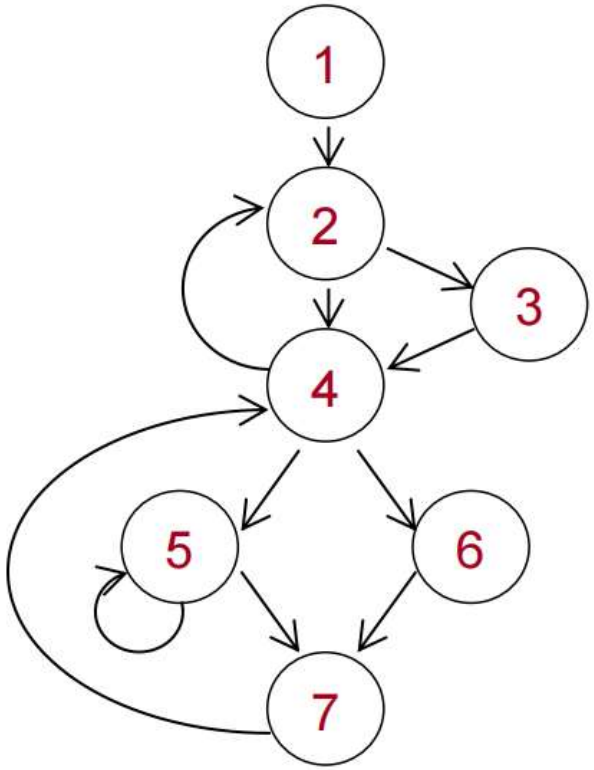
D(n) = {m | m **DOM** n}

– 设n0为流图中的首结点，

流图中任意结点a，必有

✓ a **DOM** a

✓ n0 **DOM** a



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}

3. 难点&重点：如何在流图中确定循环[Loop]

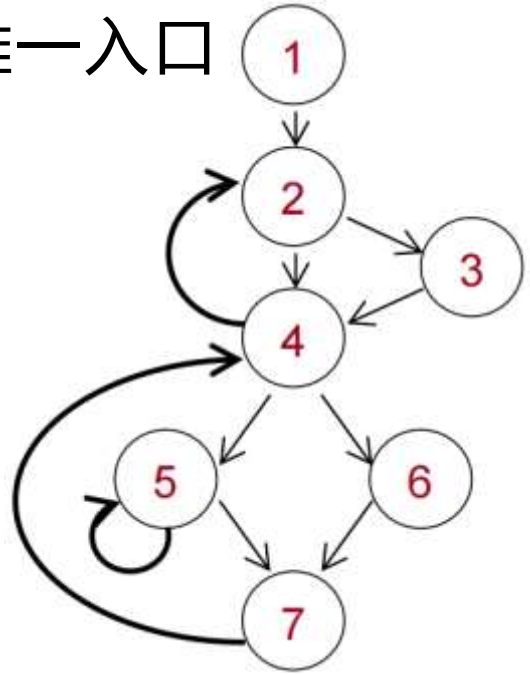
② 得到回边[Back Edges]

– 假设a→b是流图中的一条有向边，如果b DOM a，则称a→b是流图中的一条回边

③ 确定循环[Loop]

– 如果a→b是回边，则结点a，结点b，以及能到达a且不需要经过b的所有结点组成循环，且b是该循环的唯一入口

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}



There are 3 back edges:
5 → 5
7 → 4
4 → 2

There are 3 natural loops:
{5} defined by **5 → 5**
{4, 5, 6, 7} defined by **7 → 4**
{2, 3, 4, 5, 6, 7} defined by **4 → 2**

3. 难点&重点：如何在流图中确定循环[Loop]

③ 确定循环[Loop]

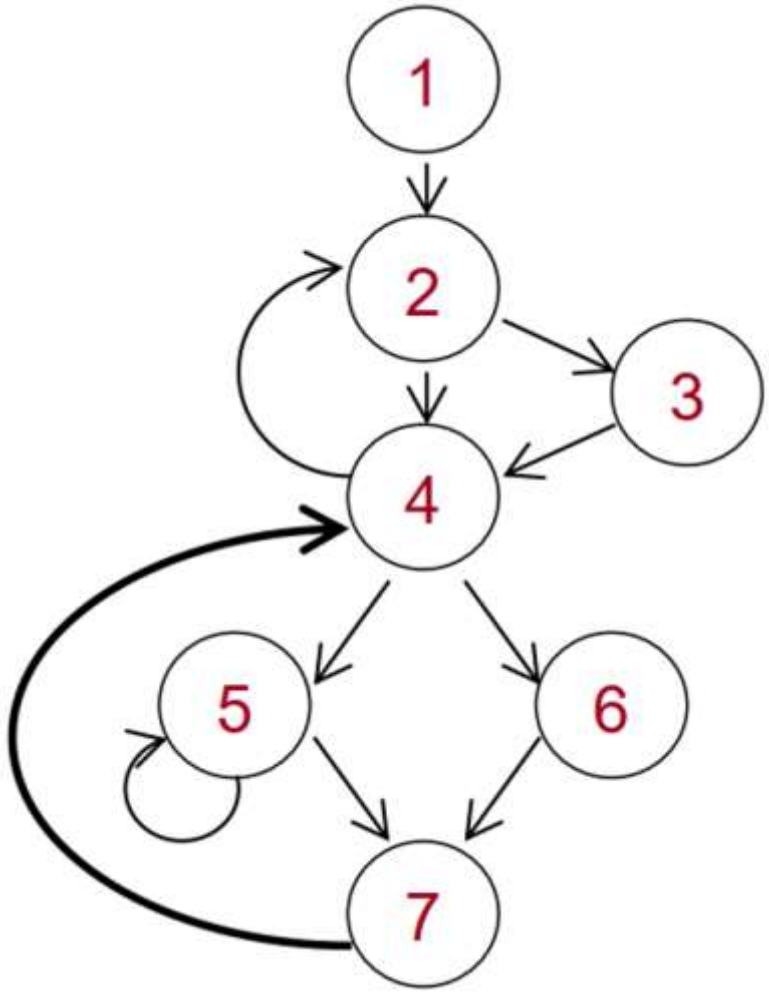
Input: back edge $n \rightarrow d$.

Algorithm:

```
1. void insert(Node m) {
2.    if (m ∉ loop) {  // d will not be pushed
3.      loop = loop ⋃ {m};
4.      stack.push(m)
5.    }
6. }
7. void main() {
8.    Stack stack = new Stack();
9.    loop = {d};  // PRE(d) will not be added
10.   insert(n);
11.   while (stack.notEmpty()) {
12.     m = stack.pop();
13.     foreach (p ∈ PRE(m))  insert(p)
14.   }
15.}
```

3. 难点&重点：如何在流图中确定循环[Loop]

③ 确定循环[Loop]



Given the back edge $7 \rightarrow 4$

1. initialize: loop = {4, 7}, stack = [7].
2. pop 7; insert 5 and 6;

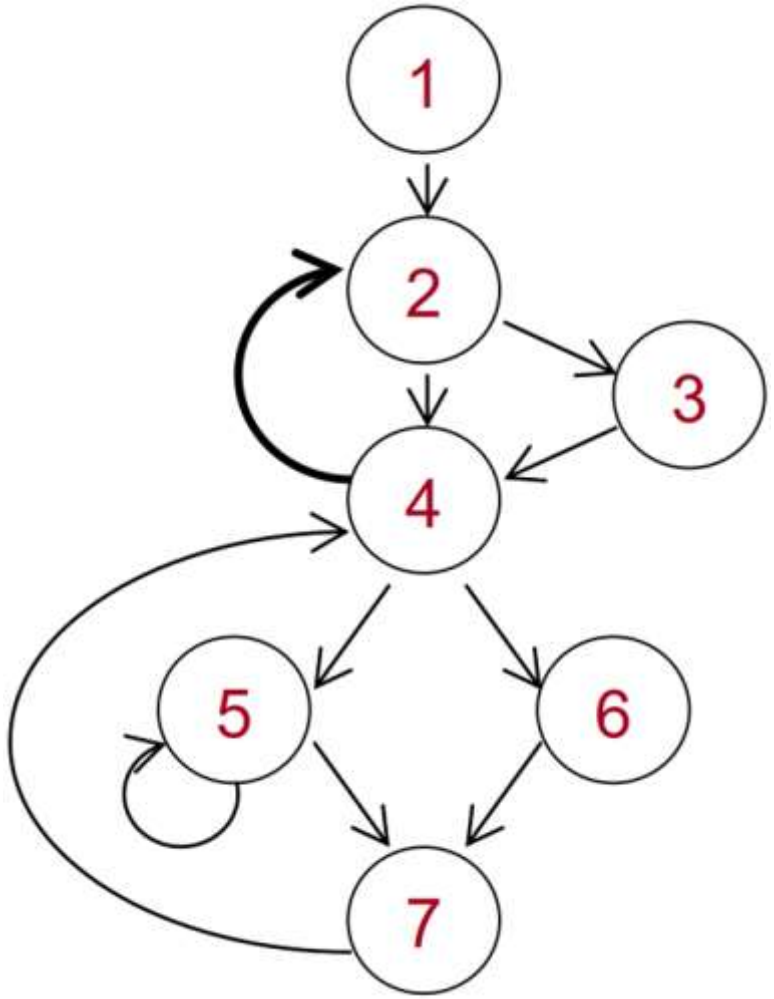
   loop = {4, 7, 5, 6}, stack = [5, 6].
3. pop 6; insert 4 (already in loop);

   loop has no change, stack = [5].
4. pop 5; insert 4 and 5 (both in loop);

   loop has no change, stack = [].
5. result: loop = {4, 7, 5, 6}.

3. 难点&重点：如何在流图中确定循环[Loop]

③ 确定循环[Loop]



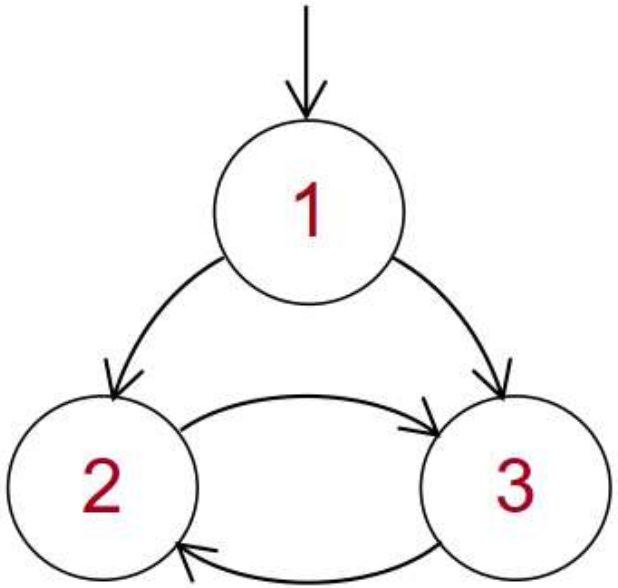
Given the back edge 4 → 2

1. initialize: loop = {2, 4}, stack = [4].
2. pop 4; insert 2, 3 and 7 (2 already in loop);
   loop = {2, 4, 3, 7}, stack = [3, 7].
3. pop 7; insert 5 and 6;
   loop = {2, 4, 3, 7, 5, 6}, stack = [3, 5, 6].
4. pop 6; insert 4 (already in loop);
   loop had no change, stack = [3, 5].
5. pop 5; insert 4 and 5 (both in loop);
   loop has no change, stack = [3].
6. pop 3; insert 2 (already in loop);
   loop has no change, stack = [].
7. result: loop = {2, 4, 3, 7, 5, 6}.

4. 自然循环的性质

- 自然循环并不能涵盖常识中的所有循环
  - 例如，在下面的流图中不存在回边，但按照常识它确实有一个循环：{2, 3}。
  - 只有在可归约流图中，回边才能找到所有循环。

- 可归约流图[Reducible flow graph]
  - 移除所有回边后，子图是无环的。
  - 在可归约流图中，循环的唯一入口是头节点。
  - 由结构化程序生成的流图通常是可归约的。



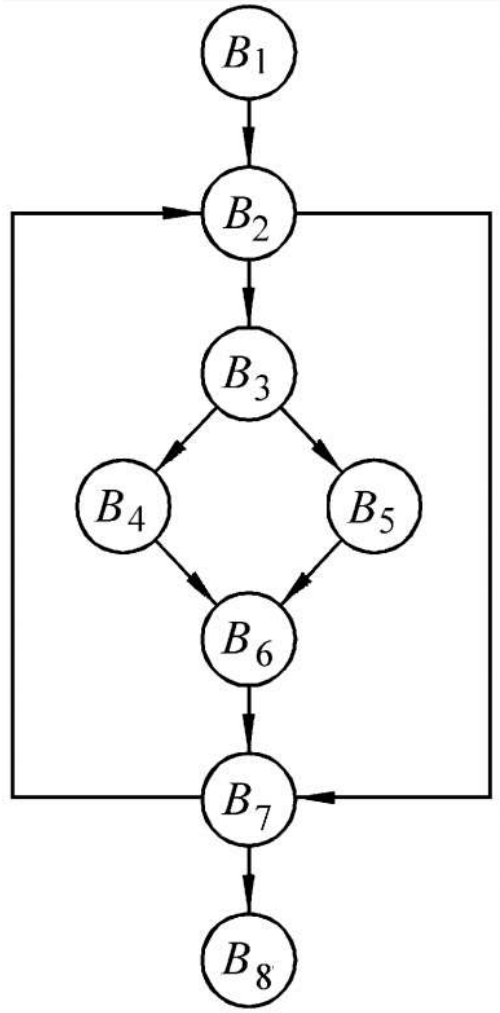
随堂练习（2）

对于右侧流图：

(1) 求出流图中各节点n的支配结点集D(n)；

(2) 求出流图中的回边；

(3) 求出流图中的循环。



(1) 支配结点集：

$D(B_1)=\{B_1\}$
$D(B_2)=\{B_1, B_2\}$
$D(B_3)=\{B_1, B_2, B_3\}$
$D(B_4)=\{B_1, B_2, B_3, B_4\}$
$D(B_5)=\{B_1, B_2, B_3, B_5\}$
$D(B_6)=\{B_1, B_2, B_3, B_6\}$
$D(B_7)=\{B_1, B_2, B_7\}$
$D(B_8)=\{B_1, B_2, B_7, B_8\}$

(2) 回边：(B7→B2)

(3) 循环：对应回边(B7→B2)，循环为：B2,B3,B4,B5,B6,B7

CONTENTS

目 录

1. 概述

- 全局优化作用于**整个过程（函数），**可以进行**跨基本块**的优化
- 常用技术：
  - 全局公共子表达式删除
  - 常量折叠和传播
  - 死代码删除
  - 复写传播
- **关键在于数据流分析**
  - 检测变量、赋值、寄存器分配等

2. 数据流分析[Data-flow Analysis]

- 控制流 vs. 数据流
  - 控制流：将基本块视为black boxes
  - **数据流**：将基本块视为**white boxes**
- 收集关于数据流的信息
  - 变量如何**被赋值（定义）**，左值[L-value]
  - 变量如何**被引用（使用）**，右值[R-value]

2. 数据流分析[Data-flow Analysis]

- 何时需要全局信息?
  - 局部优化
    - ✓ 如果变量从未被使用，则可以删除对该变量的赋值
  - 循环优化：代码外提
    - ✓ 根据变量的定义，确定循环不变运算
    - ✓ 代码外提要求该运算是循环内唯一的定义
    - ✓ 代码外提还要求所定义的变量在循环出口处不是活跃的（not live）
  - 循环优化：归纳变量消除
    - ✓ 如果归纳变量在循环外未被使用，则可以将其消除
  - 代码生成
    - ✓ 出口处的活跃性信息有助于提高寄存器利用率

2. 数据流分析[Data-flow Analysis]

- 需要哪些全局信息?
  - **定义[Definition]**
    - ✓ 语句中对某个左值[L-value]的所有赋值（来源）
  - **使用[Use]**
    - ✓ 语句中对某个右值[R-value]的所有可能使用
  - **活跃性[Liveness]**
    - ✓ 变量在某个语句之后是否会被作为右值[R-value]引用

2. 数据流分析[Data-flow Analysis]

- 两类数据流分析
  - **到达-定值**数据流分析[Reaching Definition Analysis]
  - **活跃变量**数据流分析[Live Variable Analysis]

2. 数据流分析[Data-flow Analysis]

- **到达-定值**数据流分析[Reaching Definition Analysis]
  - 向前流：信息流的方向与控制流是一致的
  - 变量A的**定值[definition]**是对A的赋值或读值到A的语句，该语句的位置称作A的**定值点**
  - 变量A的**定值点**d**到达**某点p，是指流图中从d有一条路径到达p且该通路上没有A的其它定值
    - ✓ 当我们在p之后立即使用变量A时，A的值可能由d决定

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
    - ✓ **OUT[B]为所有该基本块B入口处的定值点集合IN[B]去除当前基本块“杀死”的定值点，再加入当前基本块新产生的定值点**
  - IN[B]=∪OUT[b]　b∈P[B]
    - ✓ **IN[B]为B的所有前驱基本块的出口处OUT集合的并集**
  - 其中:
    - B：基本块；P[B]：B的所有前驱基本块；
    - IN[B]：到达B入口处的各个变量的所有定值点集合；
    - OUT[B]：到达B出口处的各个变量的所有定值点集合；
    - GEN[B]：B中定值并可到达B出口处的所有定值点集合；
    - KILL[B]：B之外的能够到达B的入口处、且其定值的变量在B中又重新定值（即被B所“杀死”）的定值点集合。

2. 数据流分析[Data-flow Analysis]

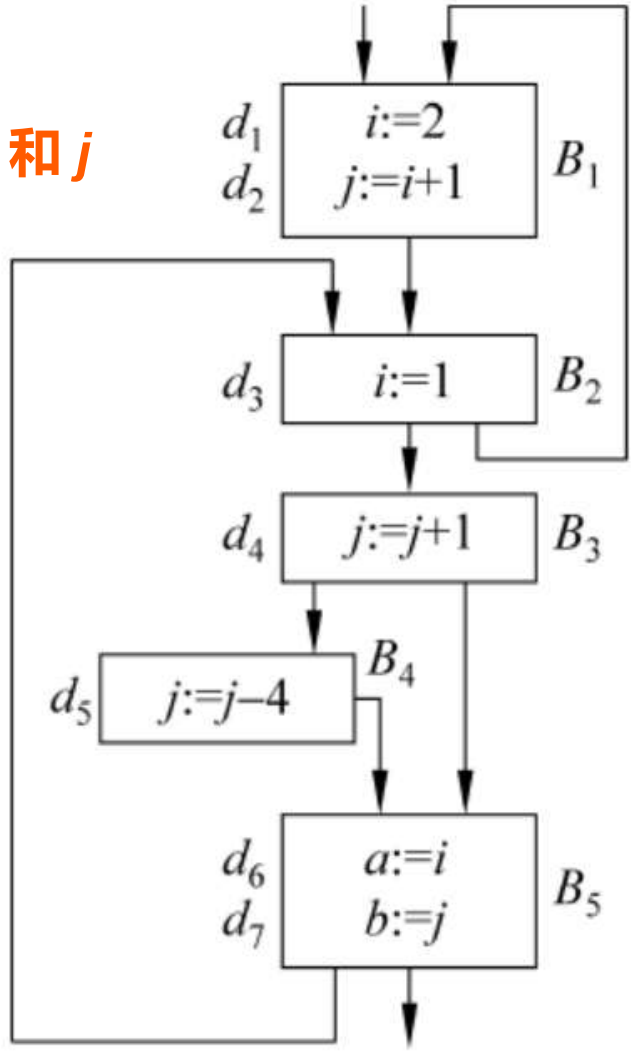
- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*

初始状态

	GEN	KILL	IN	OUT
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	{ }	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	{ }	$\{d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	{ }	$\{d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	{ }	$\{d_5\}$
B_5	{ }	{ }	{ }	{ }

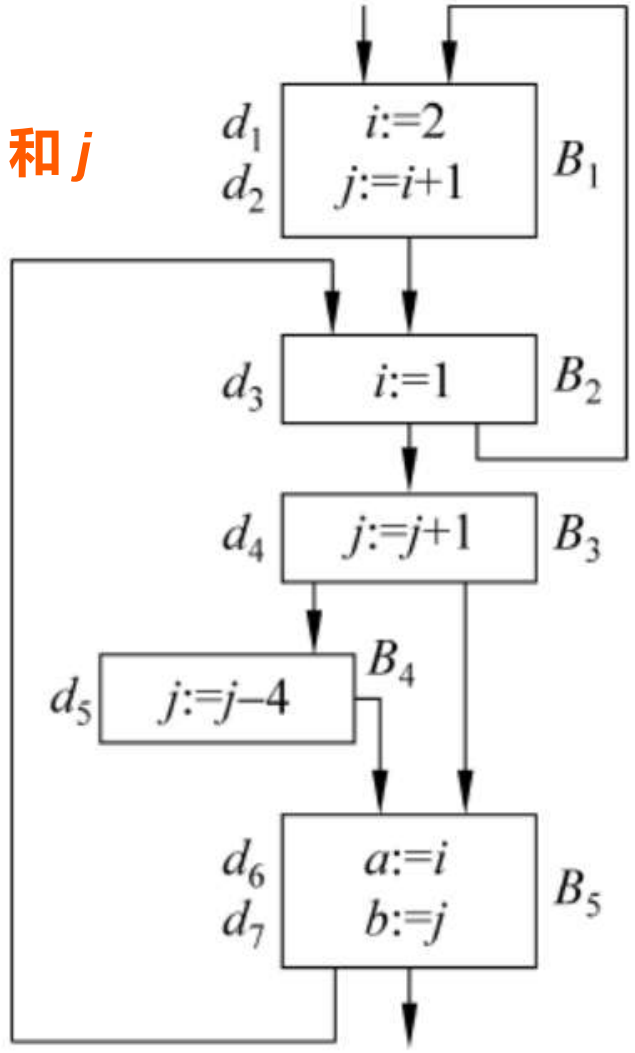
向量表示：1100000



2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



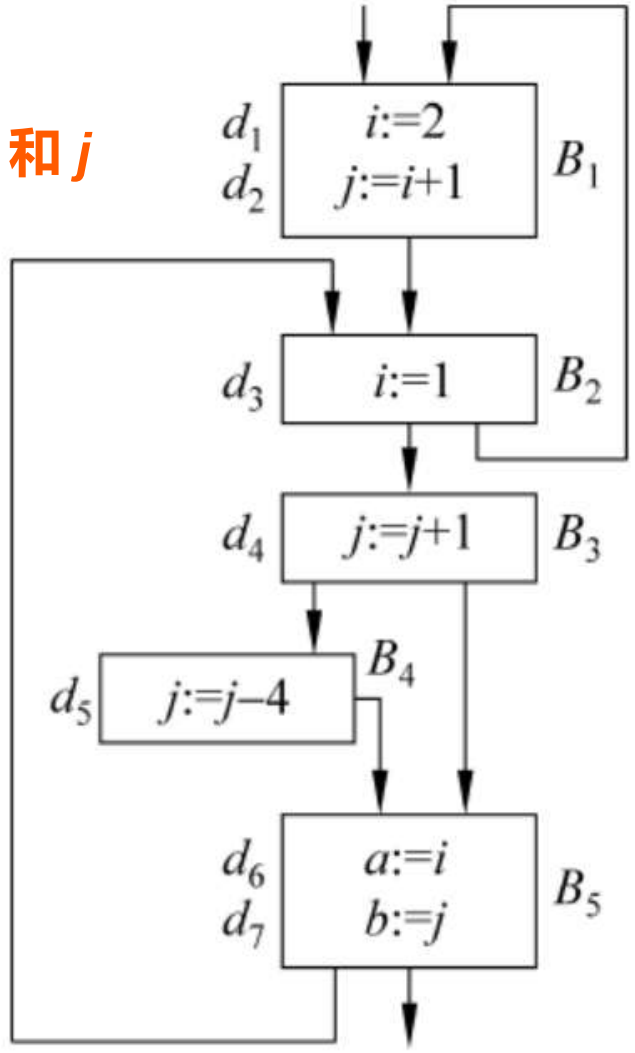
第1次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{\ \}$	$\{d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{\ \}$	$\{d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{\ \}$	$\{d_5\}$
B_5	$\{\ \}$	$\{\ \}$	$\{\ \}$	$\{\ \}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



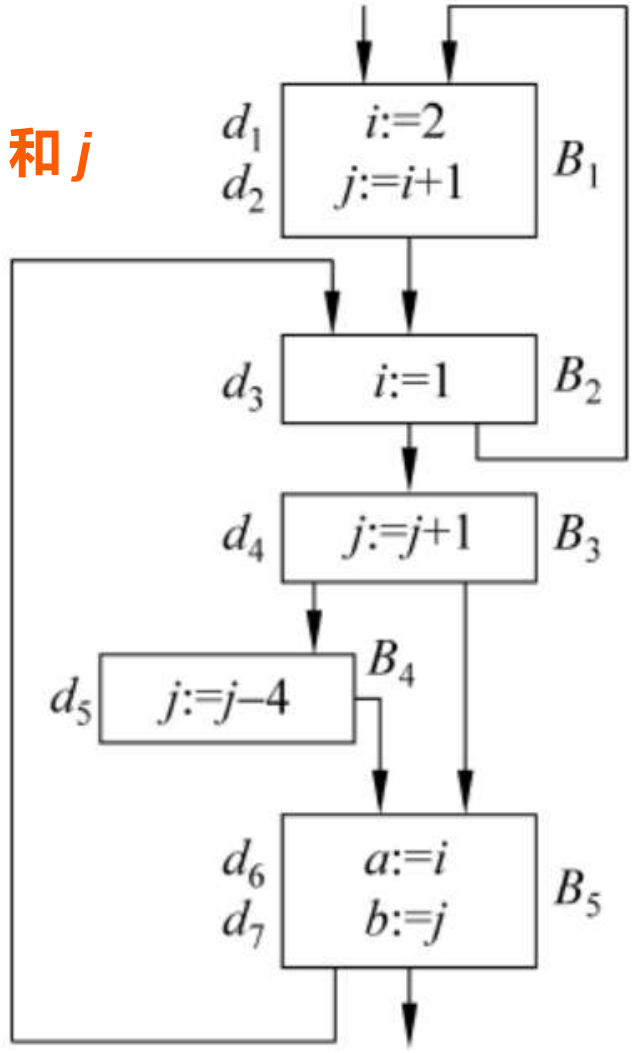
第1次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1,d_2\}$	$\{d_2,d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{\ \}$	$\{d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{\ \}$	$\{d_5\}$
B_5	$\{\ \}$	$\{\ \}$	$\{\ \}$	$\{\ \}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



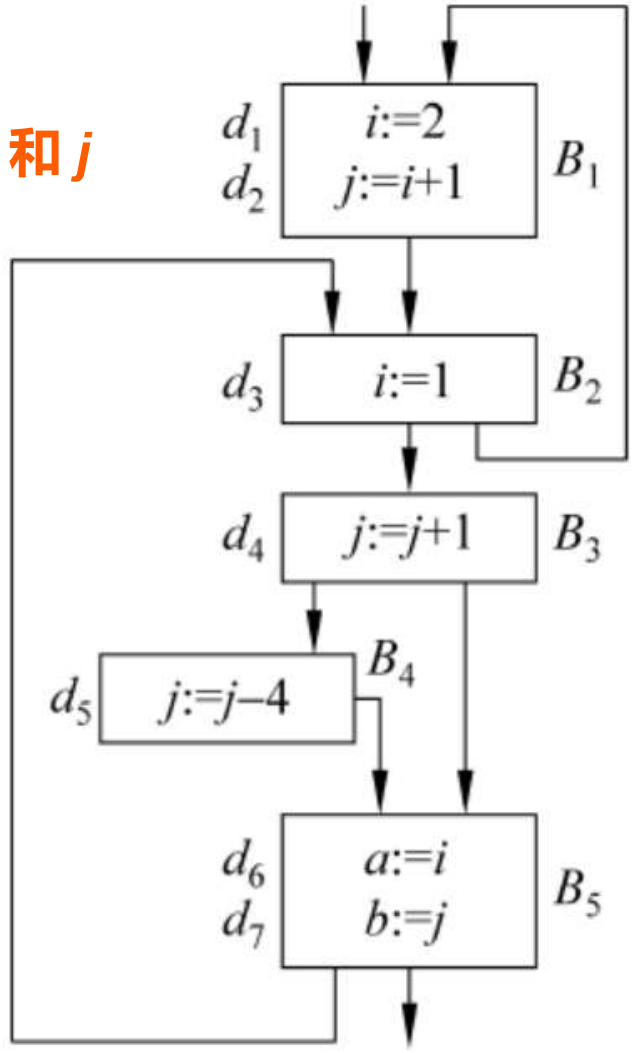
第1次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1,d_2\}$	$\{d_2, d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{\ \}$	$\{d_5\}$
B_5	$\{\ \}$	$\{\ \}$	$\{\ \}$	$\{\ \}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



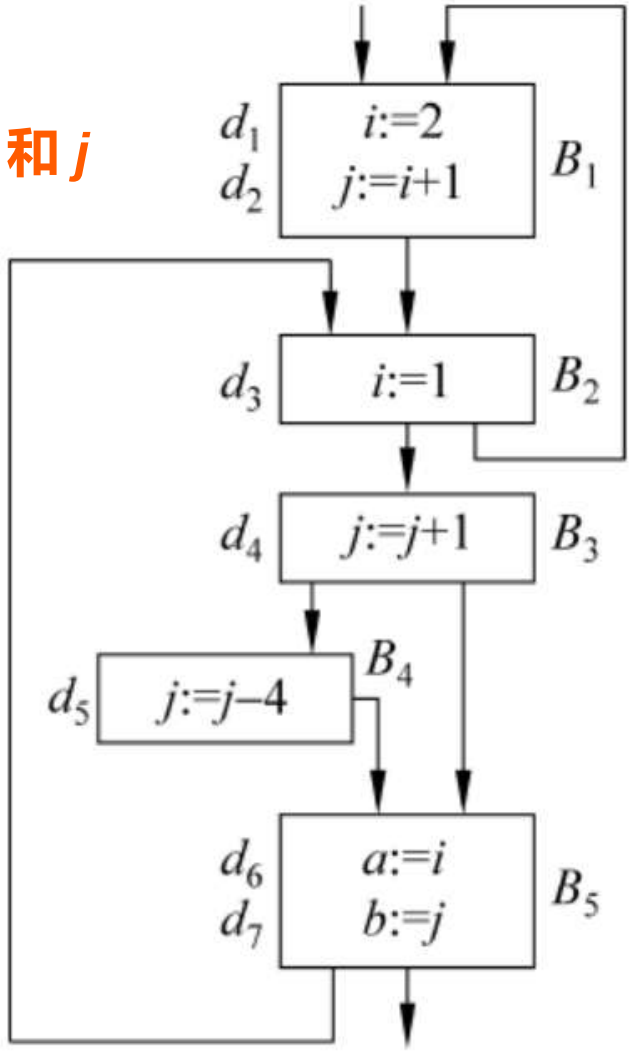
第1次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1,d_2\}$	$\{d_2, d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	{ }	{ }	{ }	{ }

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



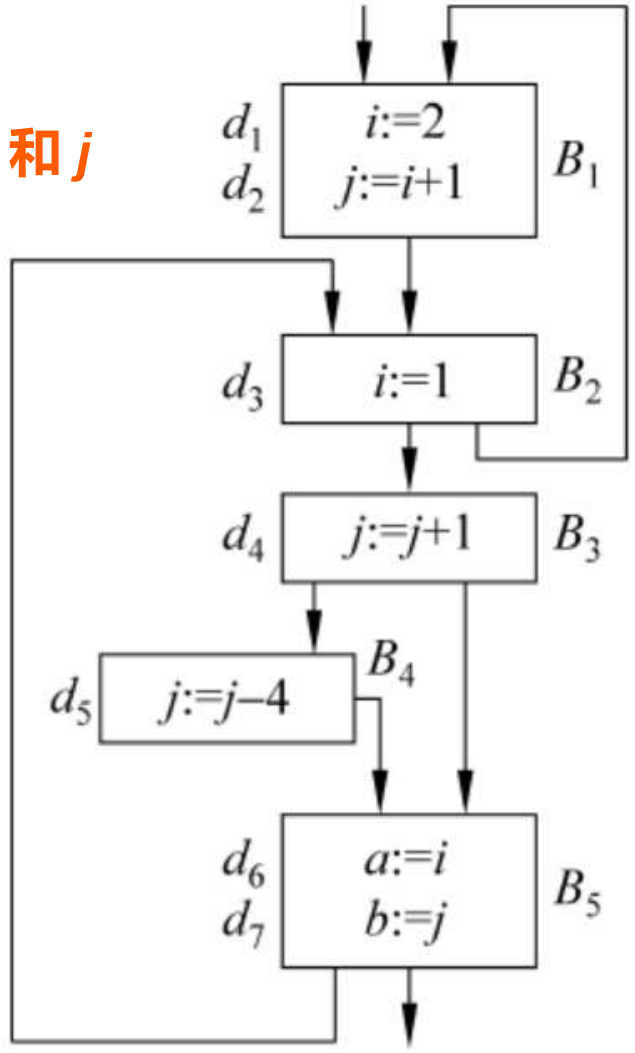
第1次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1,d_2\}$	$\{d_2, d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	{ }	{ }	$\{d_3,d_4,d_5\}$	$\{d_3,d_4,d_5\}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



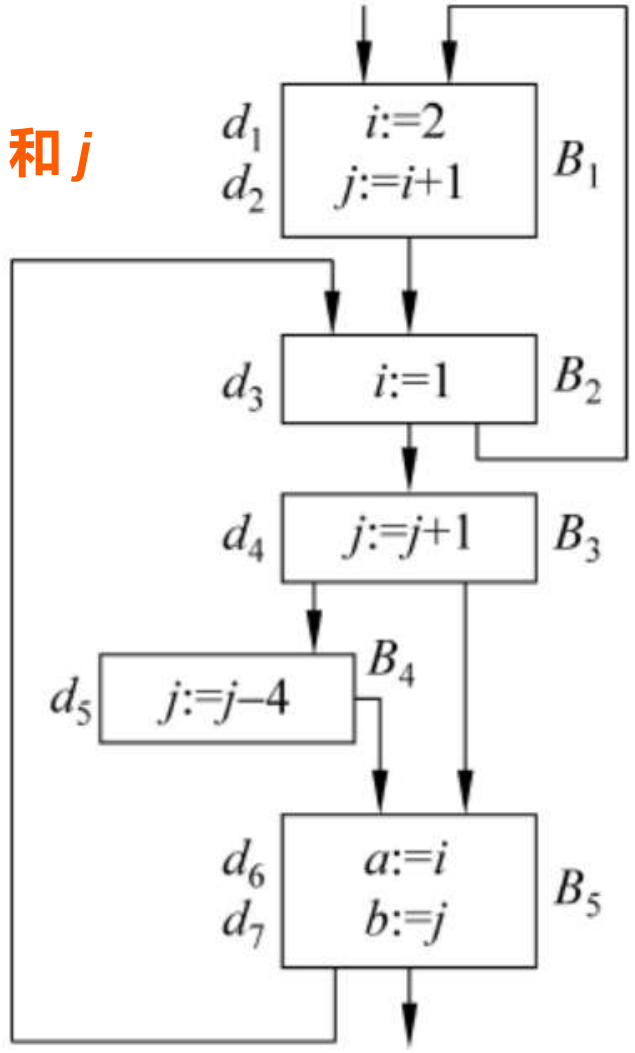
第**2**次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_2, d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1,d_2\}$	$\{d_2, d_3\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	$\{\ \}$	$\{\ \}$	$\{d_3,d_4,d_5\}$	$\{d_3,d_4,d_5\}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



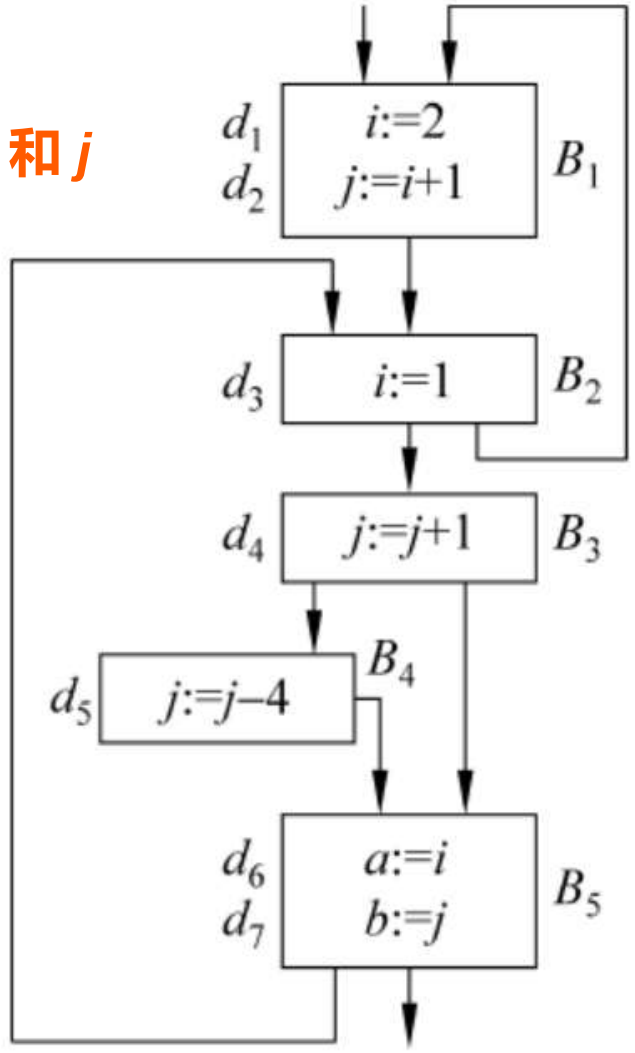
第**2**次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_2, d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1, d_2, d_3,d_4,d_5\}$	$\{d_2, d_3,d_4,d_5\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	{ }	{ }	$\{d_3,d_4,d_5\}$	$\{d_3,d_4,d_5\}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



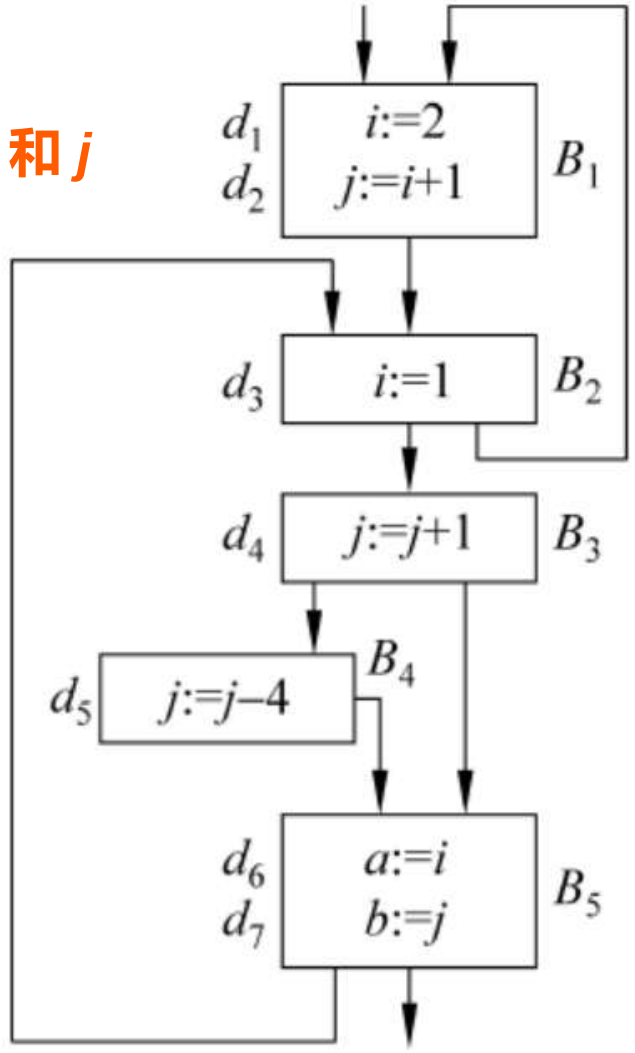
第2次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_2, d_3\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1, d_2, d_3,d_4,d_5\}$	$\{d_2, d_3,d_4,d_5\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3,d_4,d_5\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	{ }	{ }	$\{d_3,d_4,d_5\}$	$\{d_3,d_4,d_5\}$

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - OUT[B]=GEN[B]∪(IN[B]–KILL[B])
  - IN[B]=∪OUT[b]　b∈P[B]

假设仅考虑变量 *i* 和 *j*



第3次迭代

	GEN	*KILL*	*IN*	*OUT*
B_1	$\{d_1,d_2\}$	$\{d_3,d_4,d_5\}$	$\{d_2, d_3,d_4,d_5\}$	$\{d_1,d_2\}$
B_2	$\{d_3\}$	$\{d_1\}$	$\{d_1, d_2, d_3,d_4,d_5\}$	$\{d_2, d_3,d_4,d_5\}$
B_3	$\{d_4\}$	$\{d_2,d_5\}$	$\{d_2, d_3,d_4,d_5\}$	$\{d_3,d_4\}$
B_4	$\{d_5\}$	$\{d_4\}$	$\{d_3,d_4\}$	$\{d_3,d_5\}$
B_5	{ }	{ }	$\{d_3,d_4,d_5\}$	$\{d_3,d_4,d_5\}$

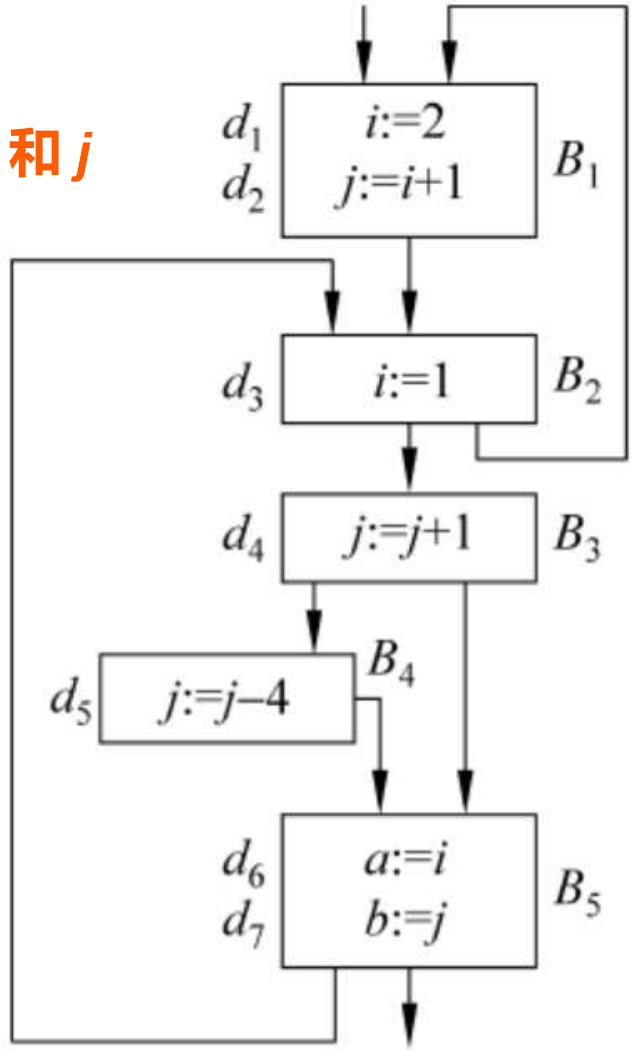
2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - **UD链**[Use-Definition Chaining, 引用-定值链]：变量A在引用语句u的定值链
    - ✓ **能够到达语句u的所有变量A定值的集合**
    - ✓ 可用于查找循环不变量，也可用于全局常量折叠优化
    - ✓ 定义：设在程序中某点u引用了变量A的值，则把能到达u的A的所有定值点的全体，称为A在引用点u的引用-定值链，简称UD链。
    - ✓ 如何计算：
      - 如果在基本块B中，变量 A 的引用点u之前有A的定值点d，并且A在点d的定值到达u，那么A在点u的UD链就是 {d} ；
      - 如果在基本块B中，变量A的引用点u之前没有A的定值点，那么，IN[B]中A的所有定值点均到达u，它们就是A在点u的UD链。

2. 数据流分析[Data-flow Analysis]

- 到达-定值数据流分析[Reaching Definition Analysis]
  - **UD链**

假设仅考虑变量 *i* 和 *j*



i 在引用点 d_2 的 UD 链为 $\{d_1\}$

i 在引用点 d_6 的 UD 链为 $\{d_3\}$

j 在引用点 d_4 的 UD 链为 $\{d_2, d_4, d_5\}$

j 在引用点 d_5 的 UD 链为 $\{d_4\}$

j 在引用点 d_7 的 UD链为 $\{d_4, d_5\}$

2. 数据流分析[Data-flow Analysis]

- **活跃变量**数据流分析[Live Variable Analysis]
  - 向后流：信息流的方向与控制流反向
  - 活跃变量
    - ✓ 对程序中的某变量A和某点p而言，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**
    - ✓ 直观地，对于全局范围的分析来说，**一个变量是活跃的，如果存在一条路径使得该变量被重新定值之前它的当前值还要被引用**

2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
    - **变量在B中定值前有引用，或者在B出口处活跃且未在B中被定值，则它在B入口处是活跃的**
  - LiveOut(B) =∪LiveIn(s) $s \in S[B]$
    - **变量在B出口处活跃，当且仅当它在B的某个后继基本块入口处活跃**
  - 其中：
    - B为一个基本块，S[B]为B的所有后继基本块；
    - LiveUse(B)为B中被定值之前要引用变量的集合；
    - Def(B)为在B中定值的且定值前未曾在B中引用过的变量集合；
    - LiveIn(B)为B入口处为活跃的变量的集合；
    - LiveOut(B)为B出口处为活跃的变量的集合。

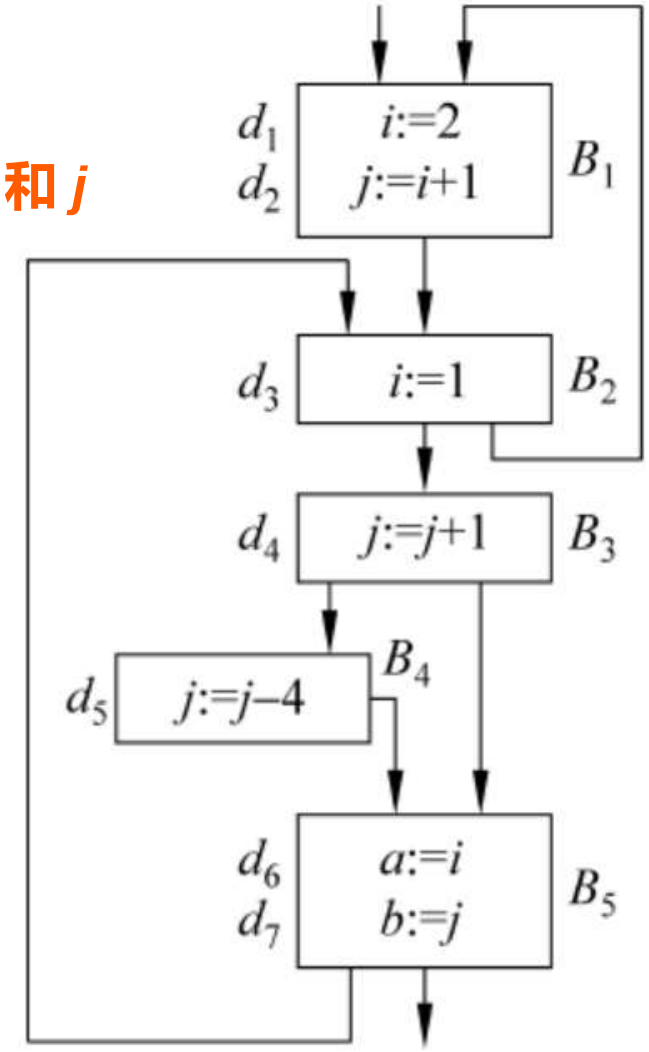
2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
  - LiveOut(B) =∪LiveIn(s)　$s \in S[B]$

初始化

	Def	LiveUse
B_1	{i,j}	Ø
B_2	{i}	Ø
B_3	Ø	{j}
B_4	Ø	{j}
B_5	Ø	{i,j}

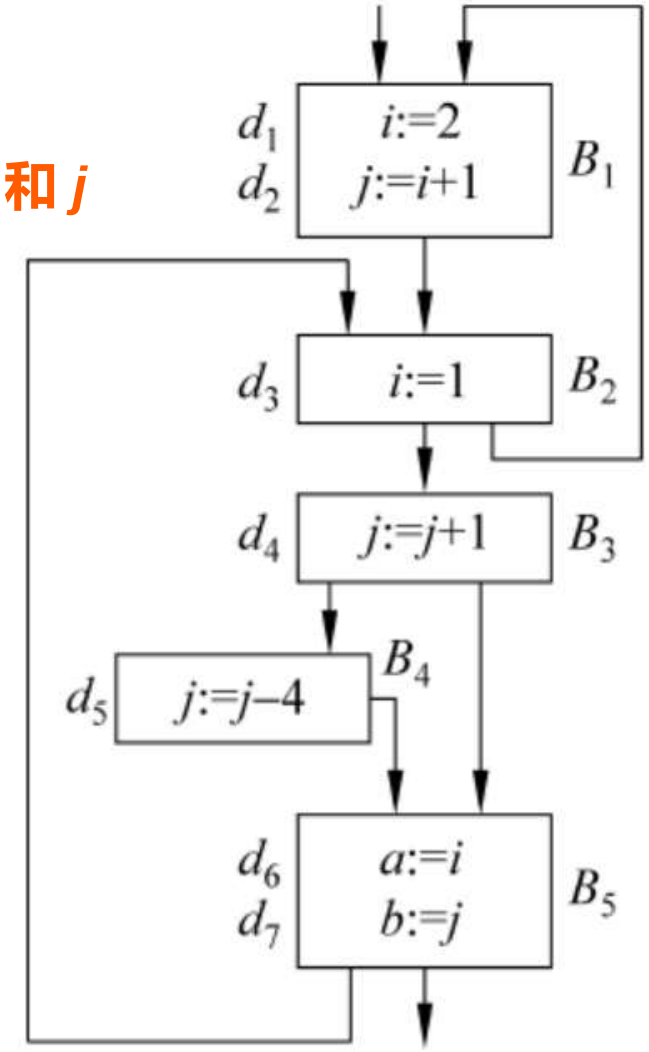
假设仅考虑变量 *i* 和 *j*



2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
  - LiveOut(B) =∪LiveIn(s)　$s \in S[B]$

假设仅考虑变量 *i* 和 *j*



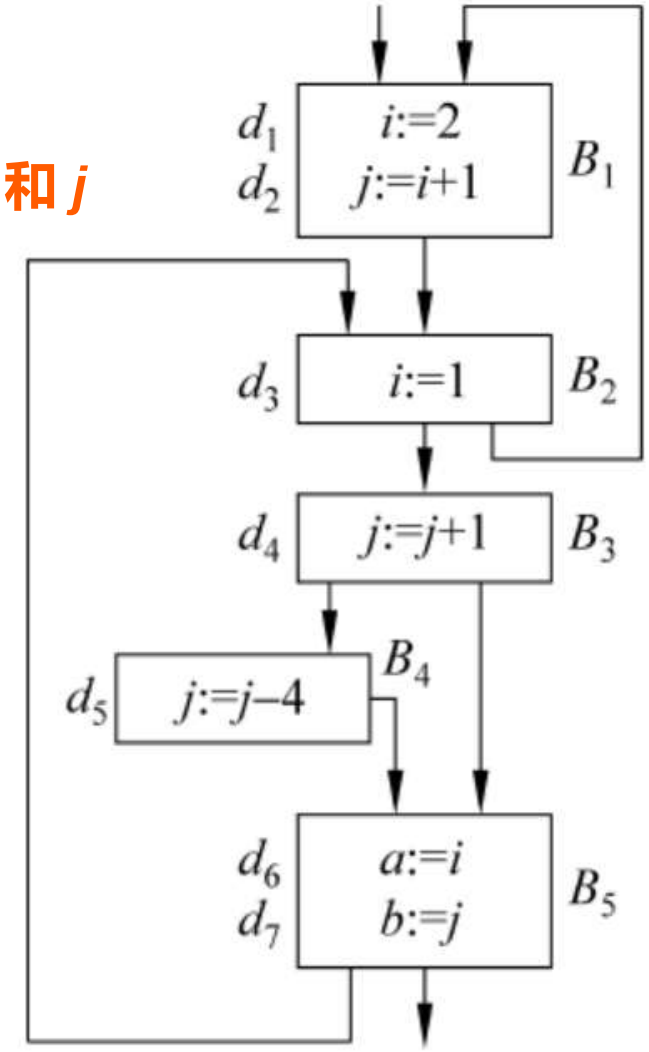
第1次迭代

	Def	LiveUse	LiveOut	LiveIn
B_1	{i,j}	∅	∅	∅
B_2	{i}	∅	∅	∅
B_3	∅	{j}	∅	{j}
B_4	∅	{j}	∅	{j}
B_5	∅	{i,j}	∅	{i,j}

2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
  - LiveOut(B) =∪LiveIn(s)　$s \in S[B]$

假设仅考虑变量 *i* 和 *j*



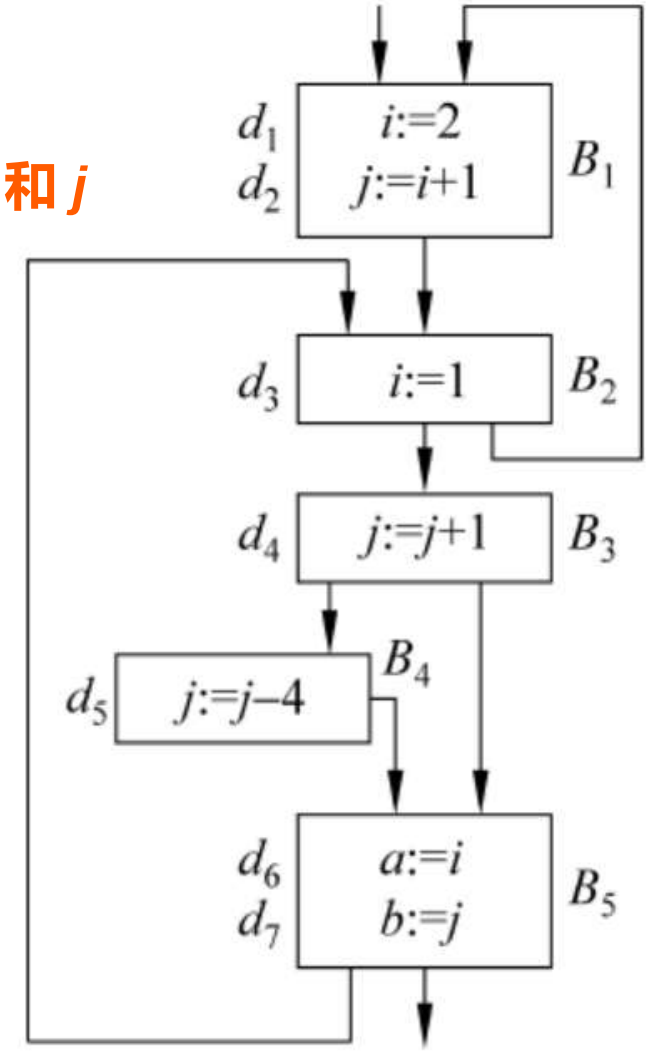
第2次迭代

	Def	LiveUse	LiveOut	LiveIn
B_1	{i,j}	∅	∅	∅
B_2	{i}	∅	{j}	∅
B_3	∅	{j}	{i,j}	{j}
B_4	∅	{j}	{i,j}	{j}
B_5	∅	{i,j}	∅	{i,j}

2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
  - LiveOut(B) =∪LiveIn(s)　$s \in S[B]$

假设仅考虑变量 *i* 和 *j*



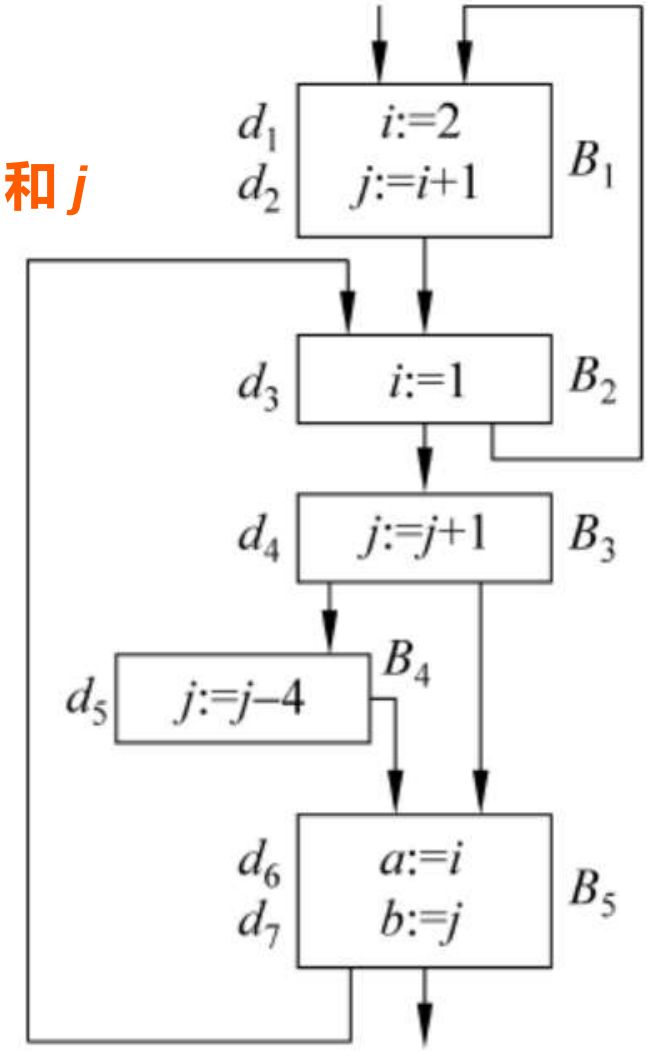
第3次迭代

	Def	LiveUse	LiveOut	LiveIn
B_1	{i,j}	∅	∅	∅
B_2	{i}	∅	{j}	{j}
B_3	∅	{j}	{i,j}	{i,j}
B_4	∅	{j}	{i,j}	{i,j}
B_5	∅	{i,j}	∅	{i,j}

2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - LiveIn(B)= LiveUse(B)∪(LiveOut(B)-Def(B))
  - LiveOut(B) =∪LiveIn(s)　$s \in S[B]$

假设仅考虑变量 *i* 和 *j*



第4次迭代

	Def	LiveUse	LiveOut	LiveIn
B_1	{i,j}	∅	{j}	∅
B_2	{i}	∅	{i,j}	{j}
B_3	∅	{j}	{i,j}	{i,j}
B_4	∅	{j}	{i,j}	{i,j}
B_5	∅	{i,j}	{j}	{i,j}

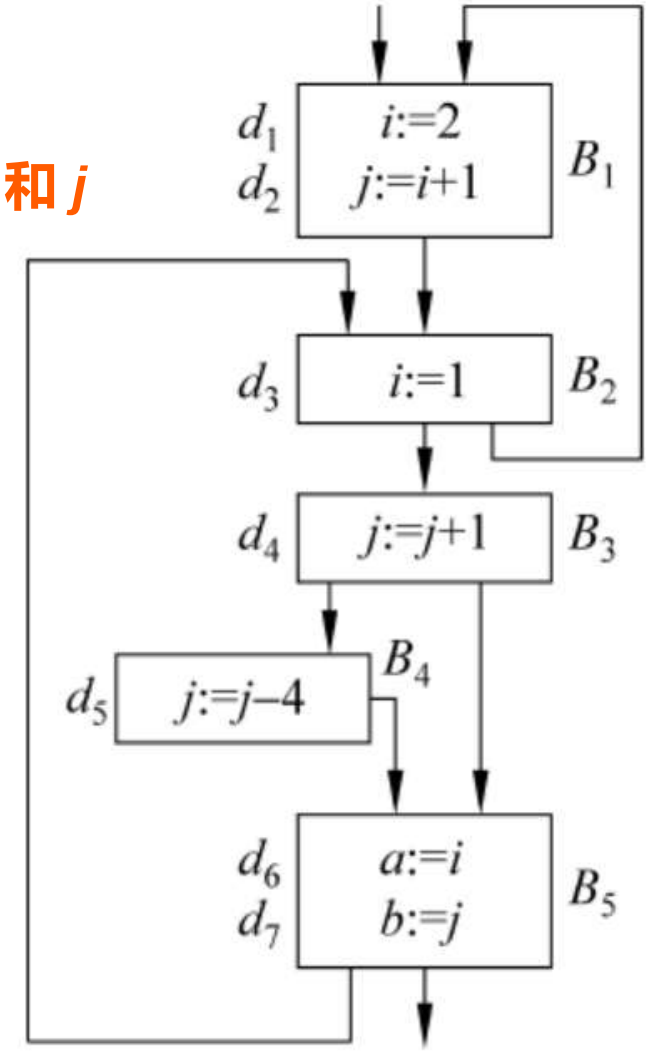
2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - **DU链**[Definition-Use Chaining, 定值-引用链]：变量A在定值语句u的使用链
    - ✓ **能够从u到达的所有变量A使用的集合**
    - ✓ 定义：假设在程序中某点u定义了变量A的值，从u存在一条到达A的某个引用点s的路径，且该路径上不存在A的其他定值点，则把所有此类引用点s的全体称为A在定值点u的定值-引用链，简称DU链。
    - ✓ 计算：DU链的计算可采用类似活跃变量数据流的向后流方法

2. 数据流分析[Data-flow Analysis]

- 活跃变量数据流分析[Live Variable Analysis]
  - **DU链**[Definition-Use Chaining, 定值-引用链]

假设仅考虑变量 *i* 和 *j*



i 在定值点 d1 的 DU 链为 {d2}

j 在定值点d2 的 DU 链为 {d4}

i 在定值点 d3 的 DU 链为 {d6}

j 在定值点 d4 的 DU 链为 {d4, d5, d7}

j 在定值点 d5 的 DU 链为 {d4, d7}

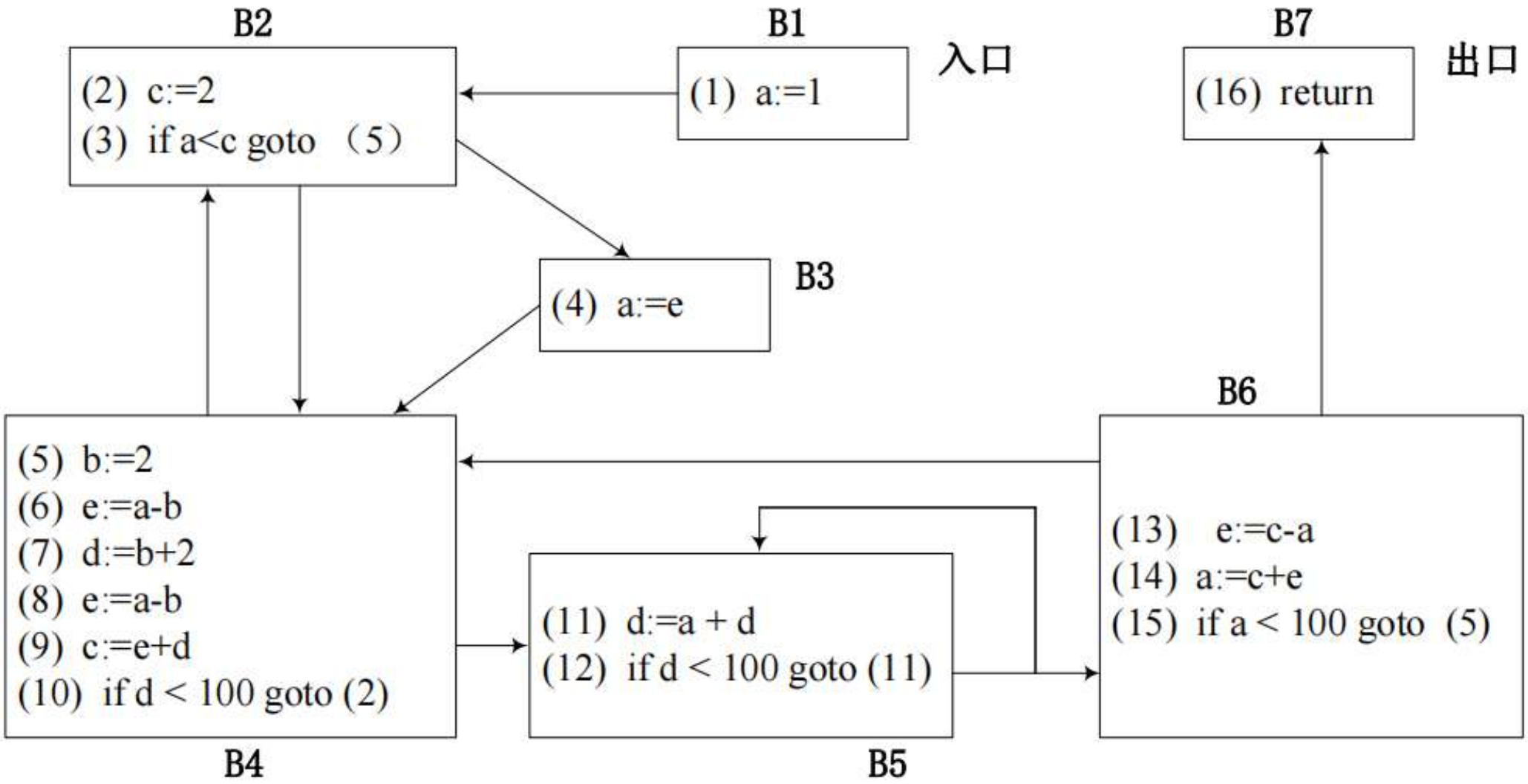
本章小结

- 重点
  - 局部优化：**为基本块构造DAG；根据DAG图优化基本块**
  - 循环优化：**如何在流图中确定循环；基本循环优化技术**
  - 全局优化：**到达-定值数据流分析/UD链；活跃变量数据流分析/DU链**

- 难点
  - 循环优化：如何在流图中确定循环
  - 全局优化：到达-定值数据流分析/UD链；活跃变量数据流分析/DU链

练习

- 下图是包含7个基本块的流图，其中B1为入口，B7为出口



练习

(1) 对该流图进行到达-定值数据流分析，假设B1的IN信息为ø，请将迭代结束时的结果填充在下表中；

(2) 指出该流图范围内，变量a在(11)的UD链；

	GEN	KILL	IN	OUT
B1			∅	
B2				
B3				
B4				
B5				
B6				
B7				

练习

(3) 对该流图进行活跃变量数据流分析，假设B7的LiveOut信息为ø，请将迭代结束时的结果填充在下表中;

(4) 指出该流图范围内，变量c在(2)的DU链。

	LiveUse	DEF	LiveIn	LiveOut
B1				
B2				
B3				
B4				
B5				
B6				
B7				∅