并行编译与优化
Parallel Compiler and Optimization

计算机研究所编译系统室 方建滨

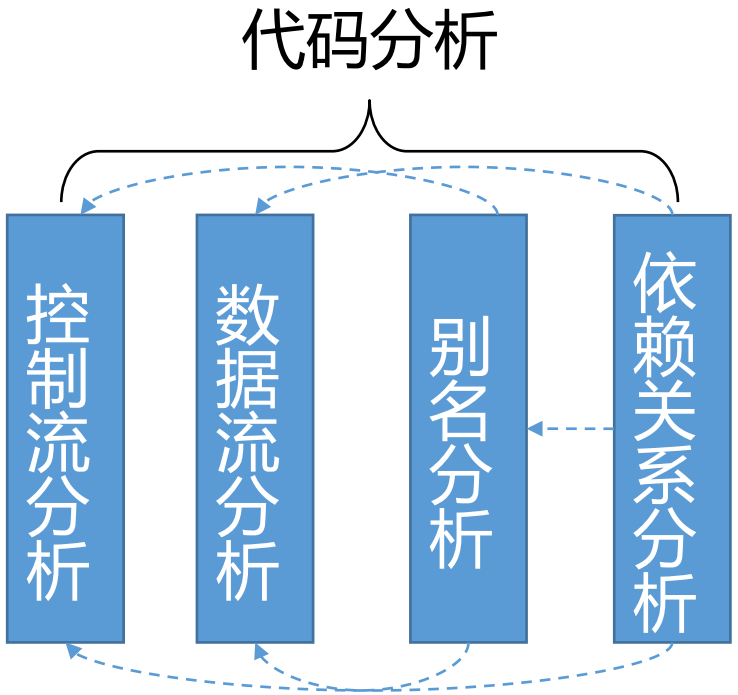
Lecture Eight: Dependence Analysis

第八课：依赖关系分析（一）

2024-04-11

复习

- 编译优化提升性能
  - 标量类优化
  - 指令级并行
  - 向量级并行
  - 核心级并行
- 编译优化 = 分析 + 转换
  - 代码分析是编译优化的基础



导入

```
S1:        a ← b + c
S2:        if (a > 10) goto L1
S3:        d ← b * e
S4:        e ← d + 1
S5:  L1:   e ← d / 2
```

```
S1:        a ← b + c
S3:        d ← b * e
S2:        if (a > 10) goto L1
S4:        e ← d + 1
     L1:   e ← d / 2
```

语句S3可否在语句S2前执行?

学习内容

COMP
CCRG Open MP

- 1. 依赖关系概念及分类
- 2. 控制依赖关系
  - 2.1 控制依赖关系的概念
  - 2.2 后必经关系
  - 2.3 控制依赖关系的严格定义
  - 2.4 控制依赖关系的计算方法
  - 2.5 课堂小结与作业
- 3. 数据依赖关系
  - 3.1 数据依赖关系的概念
  - 3.2 循环中的数据依赖关系
  - 3.3 依赖距离、方向、层次
  - 3.4 数据依赖关系的计算方法
  - 3.5 课堂小结与作业

什么是依赖关系?

- 在程序执行过程中，如果指令(语句)A必须在指令(语句)B之前执行，则称B依赖于A
- 依赖关系是程序执行顺序上的约束

```
S1:        a ← b + c
S2:        if (a > 10) goto L1
S3:        d ← b * e
S4:        e ← d + 1
S5:  L1:e ← d / 2
```

依赖关系分类

- 根据约束发生的来源
  - 如果约束是由程序的控制流引起的，则称之为控制依赖
  - 如果约束是由程序的数据流引起的，则称之为数据依赖
    - 由程序的定值-使用关系导致

分析依赖关系的作用

- 依赖关系分析最初是用在自动向量化中
  - 广泛应用于向量机，如Cray I和银河-I
- 依赖关系分析常用于
  - 循环优化
  - 并行化
  - 向量化
  - 指令调度

银河-I

Cray I

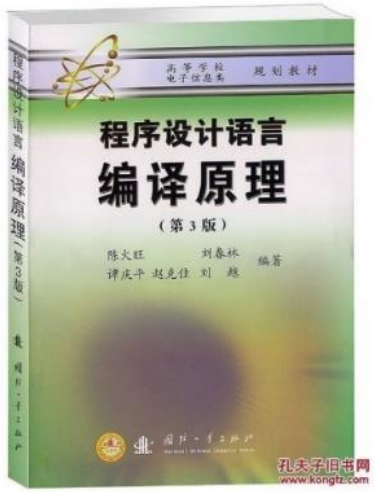
分析依赖关系的作用

YH-1巨型机

- 1983年银河 I 巨型计算机研制成功，采用向量机体系结构
- 运算速度达每秒1亿次，是我国首台亿次计算机
- 陈院士主持研制了我国首个向量FORTRAN77语言编译系统

银河-I

陈火旺院士



编译原理教材

学习内容

COMP
CCRG Open MP

- 1. 依赖关系概念及分类
- 2. 控制依赖关系
  - 2.1 控制依赖关系的概念
  - 2.2 后必经关系
  - 2.3 控制依赖关系的严格定义
  - 2.4 控制依赖关系的计算方法
  - 2.5 课堂小结与作业
- 3. 数据依赖关系
  - 3.1 数据依赖关系的概念
  - 3.2 循环中的数据依赖关系
  - 3.3 依赖距离、方向、层次
  - 3.4 数据依赖关系的计算方法
  - 3.5 课堂小结与作业

什么是控制依赖关系?

- 当语句S2的执行与否依赖于语句S1的输出，称语句S2控制依赖于语句S1
- 语句S2控制依赖于语句S1，记作 S1 δ^c S2

```
S1:        a ← b + c
S2:        if (a > 10) goto L1
S3:        d ← b * e
S4:        e ← d + 1
S5:  L1:e ← d / 2
```

S2 δ^c S3

S2 δ^c S4

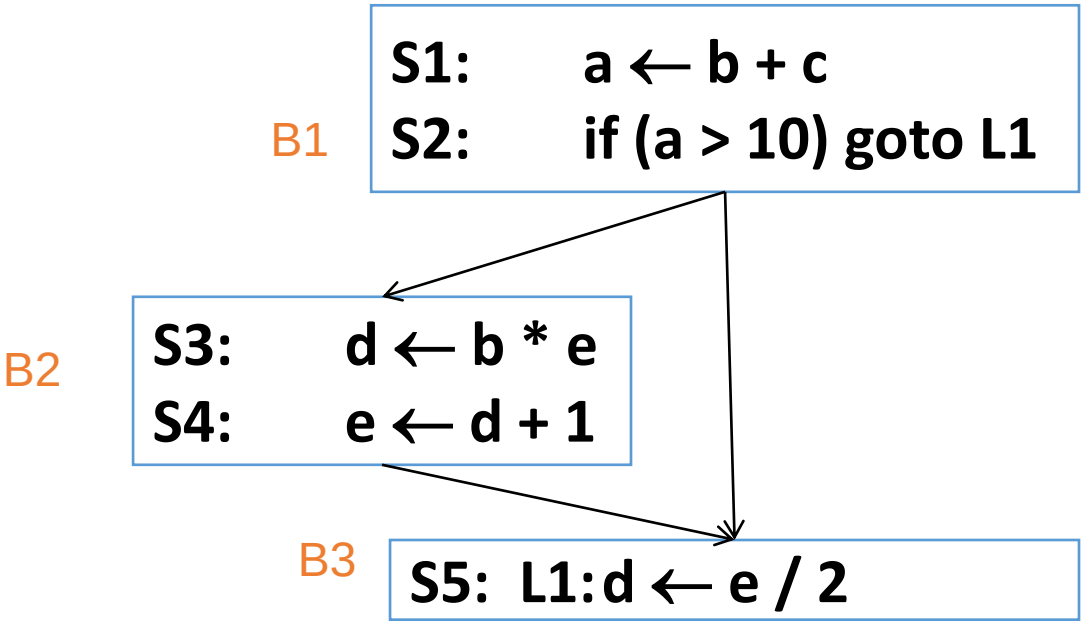
S2 δ^c S5 ??

什么是控制依赖关系?

- 当基本块B2的执行与否依赖于基本块B1的输出，称基本块B2控制依赖于基本块B1
- 记作 B1 δ^c B2

```
S1:       a ← b + c
S2:       if (a > 10) goto L1
S3:       d ← b * e
S4:       e ← d + 1
S5:  L1:  e ← d / 2
```

B1 δ^c B2



B1 δ^c B3 ??

后必经结点的定义

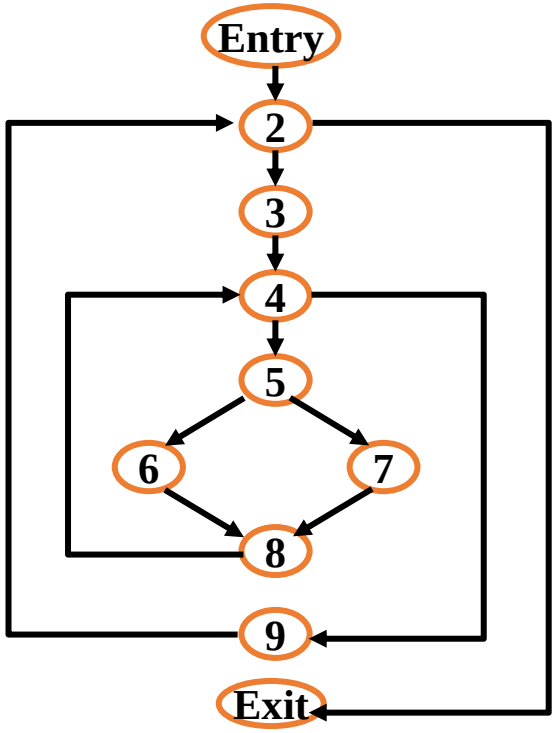
“在一个图（Graph）中，如果从结点B到出口的每一条路径都包含结点A”

我们称：A是B的一个**后必经结点** (Postdominator)

记作：A **pdom** B，或 A∈pdom(B)

CFG中的后必经结点

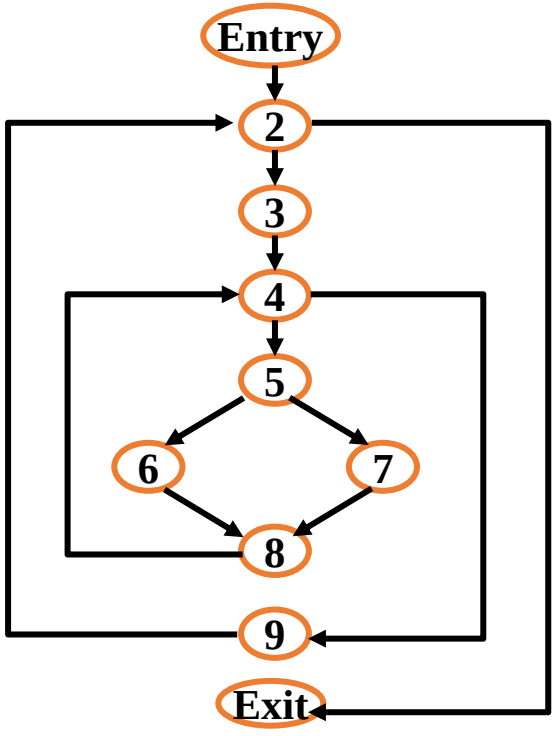
- 控制流图(CFG)是一类特殊的图，每个结点是一个基本块
- 如果从基本块B出发，到Exit的每一条路径都需要经过基本块A，那么A就是B的后必经结点



pdom(Entry) = {Entry, 2, Exit}
pdom(2) = {2, Exit}
pdom(3) = {2, 3, 4, 9,Exit}
pdom(4) = {2, 4, 9, Exit}
pdom(5) = {2, 4, 5, 8, 9, Exit}
pdom(6) = {2, 4, 6, 8, 9, Exit}
pdom(7) = {2, 4, 7, 8, 9, Exit}
pdom(8) = {2, 4, 8, 9, Exit}
pdom(9) = {2, 9, Exit}
pdom(Exit) = {Exit}

严格后必经结点

- 如果结点A是结点B的后必经结点 且 A≠B，则称A是B的严格后必经结点(strict postdominator)
- 一个结点的严格后必经结点集合是将其自身从必经结点集合中排除而得到

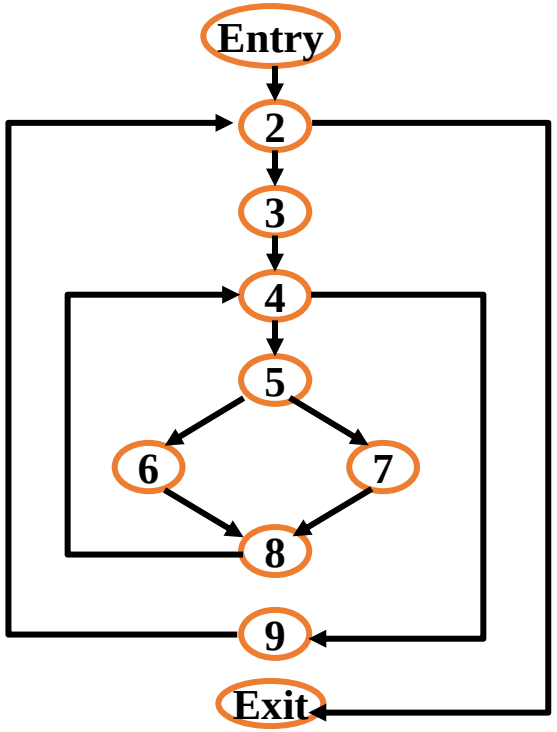


pdom(Entry) = {Entry, 2, Exit}
pdom(2) = {2, Exit}
pdom(3) = {2, 3, 4, 9,Exit}
pdom(4) = {2, 4, 9, Exit}
pdom(5) = {2, 4, 5, 8, 9, Exit}
pdom(6) = {2, 4, 6, 8, 9, Exit}
pdom(7) = {2, 4, 7, 8, 9, Exit}
pdom(8) = {2, 4, 8, 9, Exit}
pdom(9) = {2, 9, Exit}
pdom(Exit) = {Exit}

spdom(Entry) = {2, Exit}
spdom(2) = {Exit}
spdom(3) = {2, 4, 9,Exit}
spdom(4) = {2, 9, Exit}
spdom(5) = {2, 4, 8, 9, Exit}
spdom(6) = {2, 4, 8, 9, Exit}
spdom(7) = {2, 4, 8, 9, Exit}
spdom(8) = {2, 4, 9, Exit}
spdom(9) = {2, Exit}
spdom(Exit) = {}

直接后必经结点

- 如果结点A是结点B的后必经结点，且不存在结点C，满足 A spdom C, and C spdom B ，则称A是B的直接后必经结点 (immediate postdominator)
- 直接后必经结点是“距离最近”的严格后必经结点

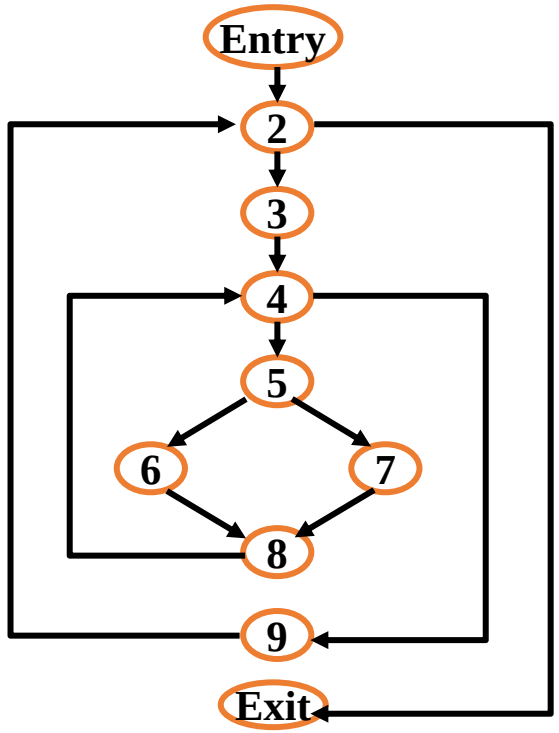


pdom(Entry) = {Entry, 2, Exit}
pdom(2) = {2, Exit}
pdom(3) = {2, 3, 4, 9,Exit}
pdom(4) = {2, 4, 9, Exit}
pdom(5) = {2, 4, 5, 8, 9, Exit}
pdom(6) = {2, 4, 6, 8, 9, Exit}
pdom(7) = {2, 4, 7, 8, 9, Exit}
pdom(8) = {2, 4, 8, 9, Exit}
pdom(9) = {2, 9, Exit}
pdom(Exit) = {Exit}

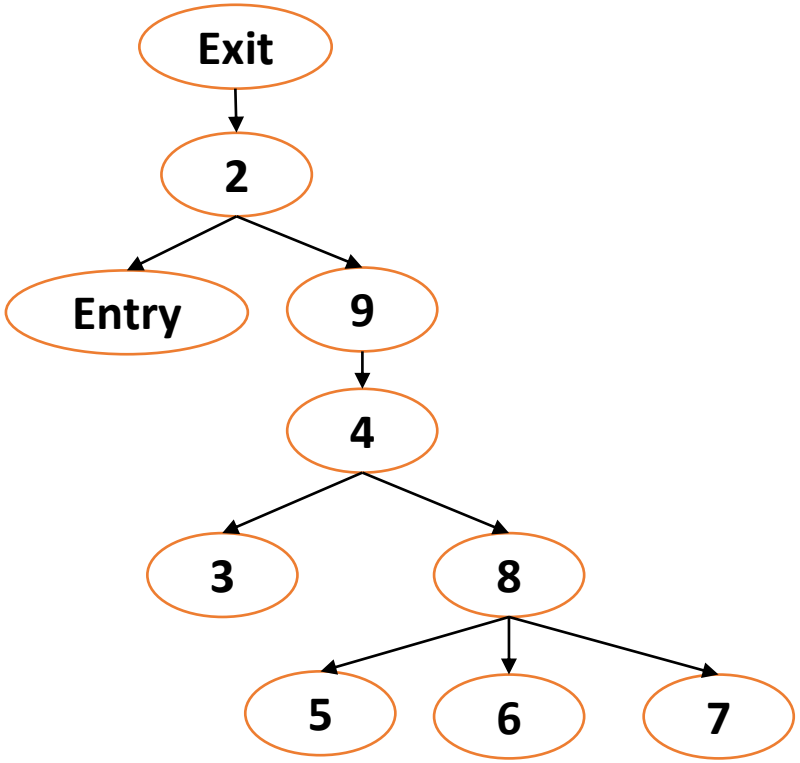
ipdom(Entry) = {2}
ipdom(2) = {Exit}
ipdom(3) = {4}
ipdom(4) = {9}
ipdom(5) = {8}
ipdom(6) = {8}
ipdom(7) = {8}
ipdom(8) = {4}
ipdom(9) = {2}
ipdom(Exit) = {}

后必经结点树

- 直接后必经结点关系信息可以存为树的形式，即**后必经结点树** (dominator tree)
- 由直接后必经关系可以构建后必经结点树

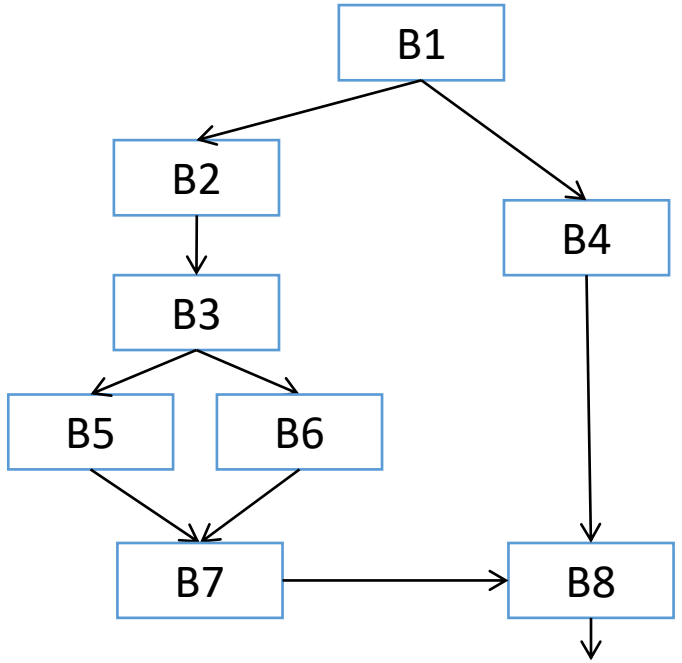


ipdom(Entry) = {2}
ipdom(2) = {Exit}
ipdom(3) = {4}
ipdom(4) = {9}
ipdom(5) = {8}
ipdom(6) = {8}
ipdom(7) = {8}
ipdom(8) = {4}
ipdom(9) = {2}
ipdom(Exit) = {}

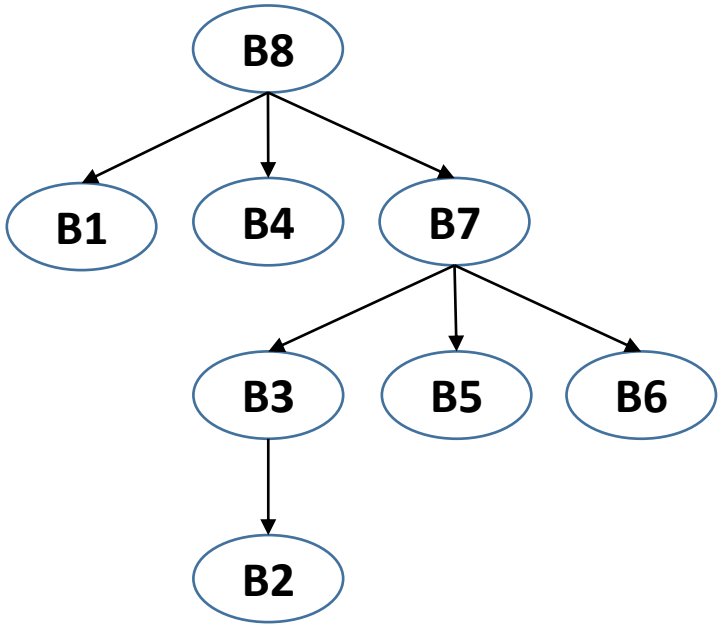


练习

- 请找出下面控制流图中每个结点的直接后必经结点并画出后必经结点树



ipdom(B1) = {B8}
ipdom(B2) = {B3}
ipdom(B3) = {B7}
ipdom(B4) = {B8}
ipdom(B5) = {B7}
ipdom(B6) = {B7}
ipdom(B7) = {B8}
ipdom(B8) = {}

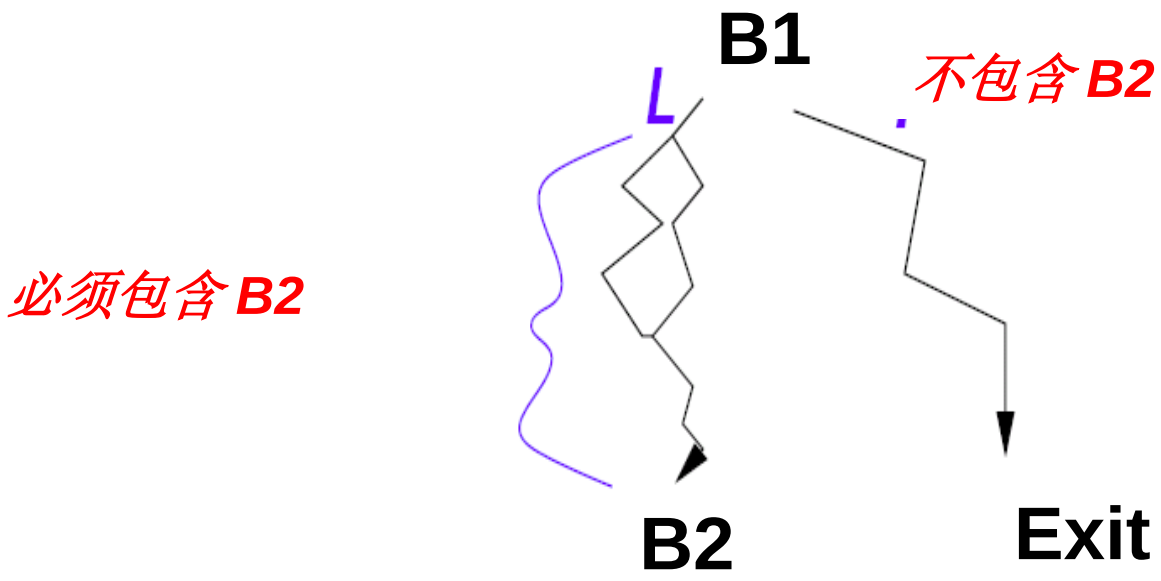


控制依赖的严格定义(1/2)

- 如果基本块B2控制依赖于基本块B1，当且仅当

  *B2是B1的**某些**后继的后必经结点，但*

  *B2不是B1的**所有**后继的后必经结点*

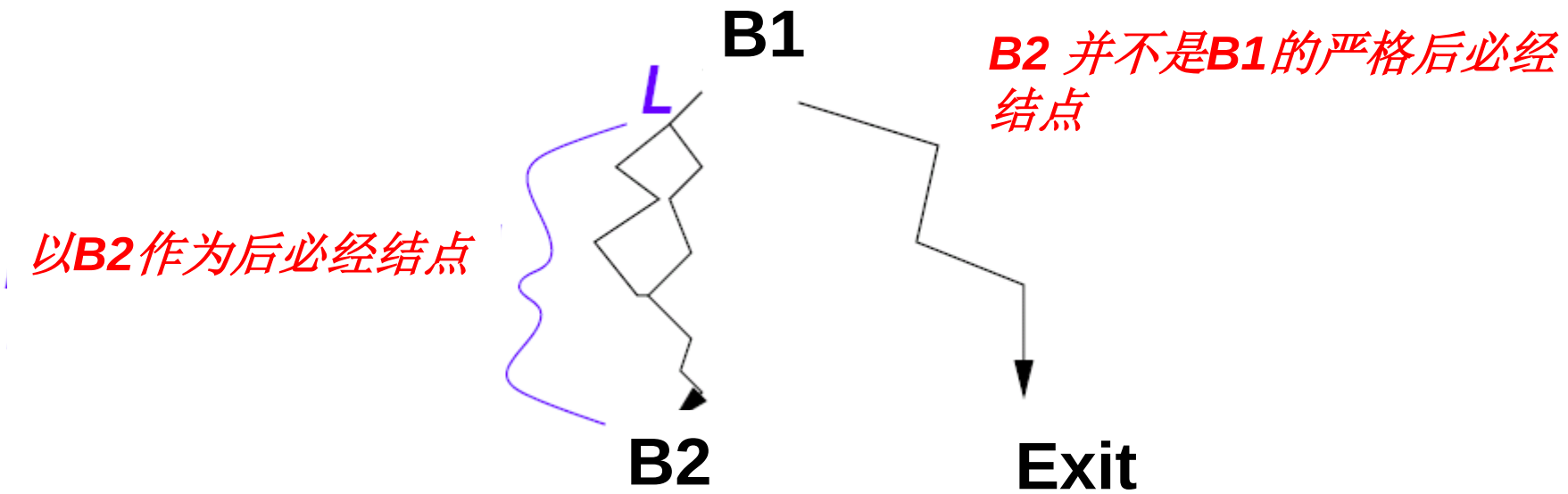


控制依赖的严格定义(2/2)

- **如果基本块B2控制依赖于基本块B1，当且仅当**

  B1到B2有一条非空路径L，并且B2是除了B1外，这条路径上其他基本块的后必经结点

- **开关条件 如果B2控制依赖于B1，那么存在两条路径：一条从B1到出口的路径不包括B2，另外一条则必然经过B2**

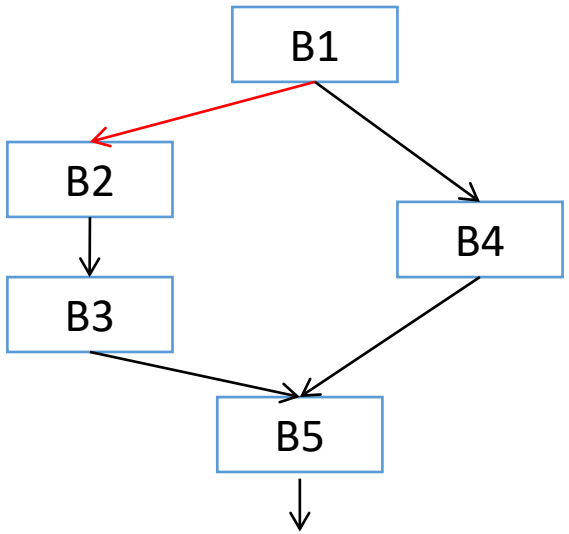


边依赖

- 基本块B3控制依赖于边(B1,B2) 当且仅当

B1到B3有一条起始于(B1, B2)非空路径L，并且B3是除了B1外，这条路径上其他基本块的后必经结点

- 引入边依赖可以让编译器知道从B1经过哪条边到达B3



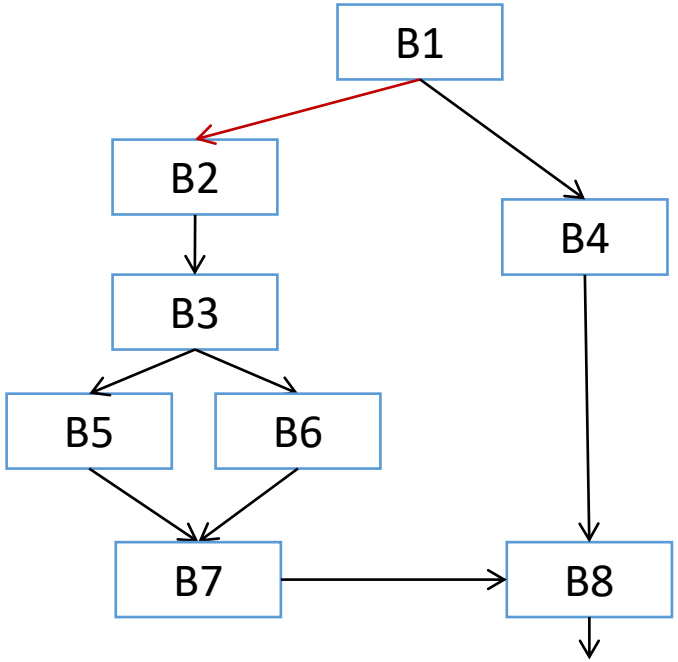
(B1,B2) δ^c B2

(B1,B2) δ^c B3

(B1,B4) δ^c B4

计算控制依赖关系

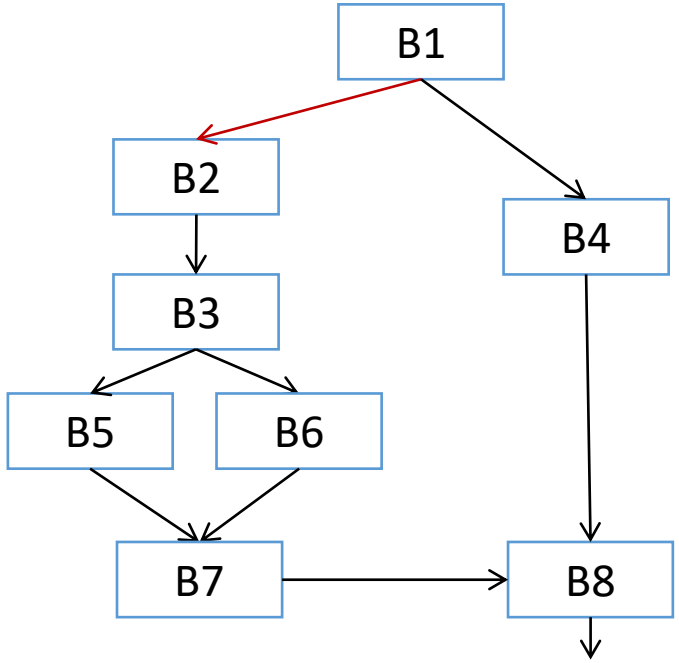
- 对于给定边(B, S)，哪些基本块控制依赖于该边?
  - 是S的后必经结点但不是B的后必经结点的结点

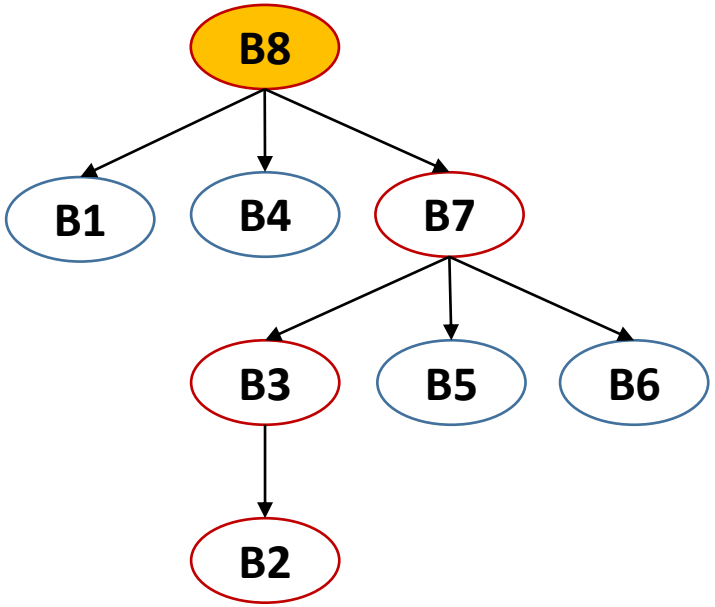


(B1,B2) ⇒ {B2, B3, B7}

计算控制依赖关系

- 对于给定边(B, S)，哪些基本块控制依赖于该边?
  - 在后必经结点树中，以S为起点，迭代地计算ipdom(S), ipdom(ipdom(S)), …直到（但不包括）遇到ipdom(B)为止



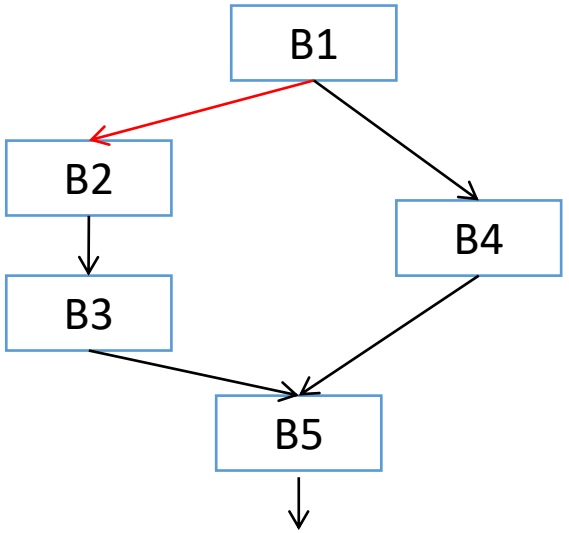


$(B1,B2)\ \delta^{c}\ \{B2, B3, B7\}$

控制依赖关系计算方法

```
find_control_dependent (edge: (B,S))
{
    depends = ∅;
    X = S;  //X is successor of B
    while X ≠ (ipdom(B)) do
    {
       add X to depends
       X = ipdom(X);
    }
    return depends;
}
```

例子：计算控制依赖




```
find_control_dependent (edge: (B,S))
{
    depends = ∅;
    X = S;  //X is successor of B
    while X ≠ (ipdom(B)) do
    {
       add X to depends
       X = ipdom(X);
    }
    return depends;
}
```

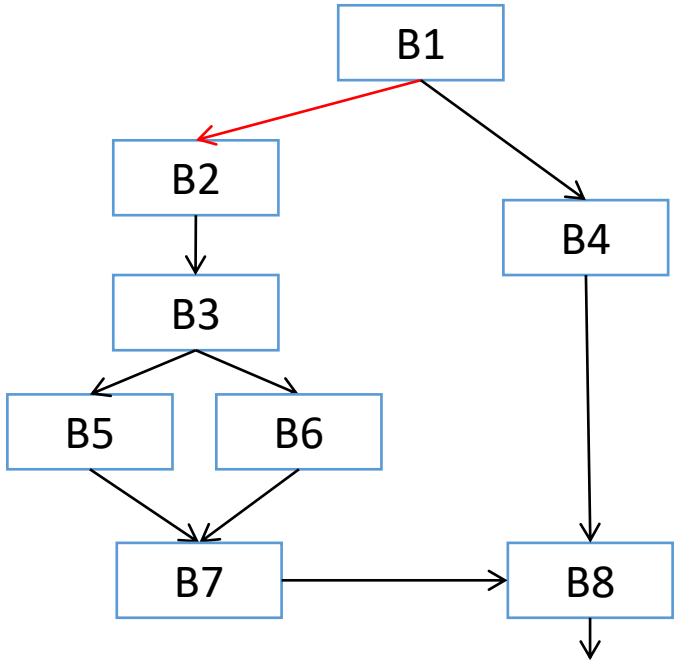
计算边(B1,B2)的控制依赖集合:

X = {B2}
B2 ≠ ipdom(B1), depend(B1,B2)={B2}

X = ipdom(B2) = {B3}
B3 ≠ ipdom(B1), depend(B1,B2)={B2, B3}

X = ipdom(B3) = {B5}
B5 = ipdom(B1), depend(B1,B2)={B2, B3}

练习：计算控制依赖




```
find_control_dependent (edge: (B,S))
{
    depends = ∅;
    X = S;  //X is successor of B
    while X ≠ (ipdom(B)) do
    {
       add X to depends
       X = ipdom(X);
    }
    return depends;
}
```

Calculate the control dependence of (B1,B2):

X = {B2}
B2 ≠ ipdom(B1), depend(B1,B2)={B2}

X = ipdom(B2) = {B3}
B3 ≠ ipdom(B1), depend(B1,B2)={B2, B3}

X = ipdom(B3) = {B7}
B7 ≠ ipdom(B1), depend(B1,B2)={B2, B3, B7}

X = ipdom(B7) = {B8}
B8 = ipdom(B1), depend(B1,B2)={B2, B3, B7}

扩展思考：算法的正确性

- **推论：如果S是B的一个后继结点，则要么S是B的后必经结点，要么pdom(B)是pdom(S)的后必经结点**

  证明：假设S不是B的后必经结点。对于任何一条从S到出口的路径L，L可以扩展为一条从B到出口的路径L’。因此，pdom(B) 必定在这条路径L上，不为S，且在从S到出口的每一条路径上。因此， pdom(B) 必定是S的后必经结点，进一步它肯定也是pdom(S)的后必经结点□

- **如果 (B,S) δ^c Y, 那么pdom(B)是pdom(S)的后必经结点**

- **寻找与B的一个后继S相关的所有控制结点时，一定会追寻到属于B的后必经结点，并且追寻过程到此为止**

回看导入

```
S1:       a ← b + c
S2:       if (a > 10) goto L1
S3:       d ← b * e
S4:       e ← d + 1
S5:  L1:  e ← d / 2
```

```
S1:       a ← b + c
S3:       d ← b * e
S2:       if (a > 10) goto L1
S4:       e ← d + 1
S5:  L1:  e ← d / 2
```

语句S3可否在语句S2前执行?

$S2\ \delta^c\ S3$ or $B(S2)\ \delta^c\ B(S3)$

学习内容

COMP CCRG Open MP

- 1. 依赖关系概念及分类
- 2. 控制依赖关系
  - 2.1 控制依赖关系的概念
  - 2.2 后必经关系
  - 2.3 控制依赖关系的严格定义
  - 2.4 控制依赖关系的计算方法
  - 2.5 课堂小结与作业
- 3. 数据依赖关系
  - 3.1 数据依赖关系的概念
  - 3.2 循环中的数据依赖关系
  - 3.3 依赖距离、方向、层次
  - 3.4 数据依赖关系的计算方法
  - 3.5 课堂小结与作业

语句与实例

- 语句(statement)是文本程序的基本单位，是静态概念
- 实例(instance)是程序运行中语句的一次执行，是动态概念
- 例子：写出下面代码段的语句及语句实例

```
      do I=1,100
S:        A(I) = B(I+2)+1
T:        B(I) = A(I-1)-1
      enddo
```

语句：S, T

语句实例：S(1), T(1)
S(2), T(2)
…
S(100), T(100)

语句与实例

语句的先序关系

- 静态优先：程序文本中语句S静态优先于语句T，记 S<T
- 动态优先：程序运行中语句S在T之前执行，记 S ◁ T

```
S1:   A = 0
S2:   B = A
S3:   A = B + A
S4:   C = A
```

```
      do I=1,100
S:        A(I) = B(I+2)+1
T:        B(I) = A(I-1)-1
      enddo
```

S1 < S2 < S3 < S4
S1 ◁ S2 ◁ S3 ◁ S4

对于基本块，< = ◁

S < T
S(1) ◁ T(1), T(i) ◁ S(i+1)

对于循环，< ≠ ◁

数据依赖关系定义

- **语句T依赖于语句S是指存在S的一个实例S'和T的一个实例T'以及一个存储单元M，满足**
  - **S'和T'都引用M（读或写），且至少一个是写操作；**
  - **当程序串行执行时， S'是在T'之前执行 (S' ◁ T')；**
  - **在同一次执行中，在S'执行结束与T'开始执行前没有对存储单元M的写操作**

1. 访问同一个存储单元M
2. 其中一次访问是“写”
3. 执行顺序

数据依赖关系定义

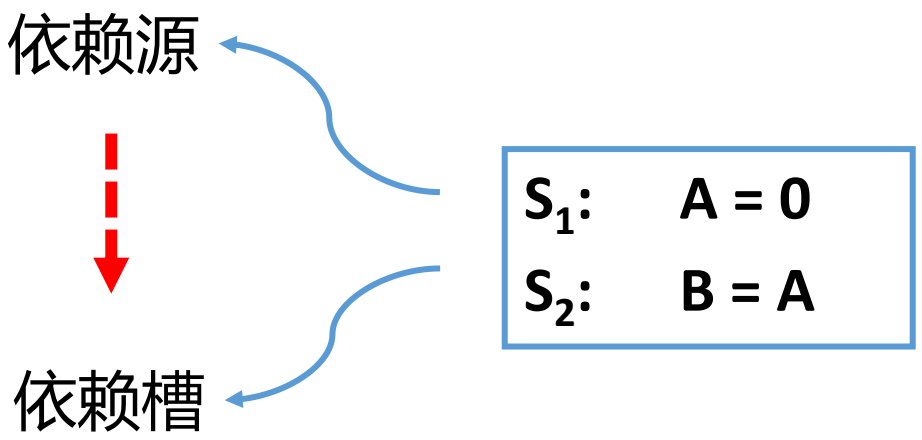
- 讨论：S和T可以是同一语句吗?

```
S: A = B + A
```

S先读A，S再写A ➔ 存在依赖

数据依赖的表示与分类

- 语句T依赖于语句S，记作：S δ T
  - 语句S是依赖关系的源（source）
  - 语句T是依赖关系的槽（sink）

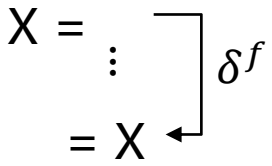


$S_1 \ \delta \ S_2$

数据依赖的表示与分类

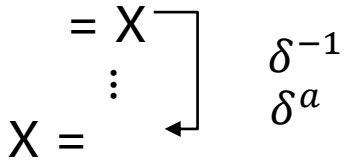
三类数据依赖关系

流依赖（flow-）



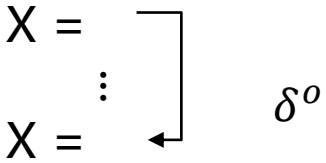
先写后读

反向依赖（anti-）



先读后写

输出依赖（output-）



先写后写

流依赖又称为真依赖

反向依赖和输出依赖可以通过寄存器重命名等优化消除

数据依赖的表示与分类：练习

- 找出下面代码的数据依赖关系

S_1: A = 0
S_2: B = A
S_3: A = B + A
S_4: C = A

S_1 δ^f S_2 for A
S_1 δ^f S_3 for A
S_3 δ^f S_4 for A
S_1 δ^o S_3 for A
S_2 δ^{-1} S_3 for A
S_3 δ^{-1} S_3 for A

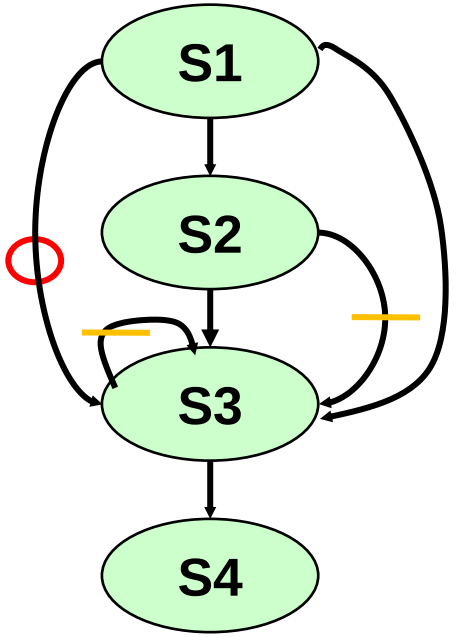
S2 δ^f S3 for B

数据依赖图

- 基本块内数据依赖关系可以用有向无环图（DAG）表示，称为**数据依赖图**
  - 结点表示语句
  - 有向边表示一个数据依赖，区分边的类型
    - 边起点是依赖源语句
    - 边终点是依赖槽语句

S_1: A = 0
S_2: B = A
S_3: A = B + A
S_4: C = A

$S_1\ \delta^f\ S_2$
$S_1\ \delta^f\ S_3$
$S_1\ \delta^0\ S_3$
$S_2\ \delta^{-1}\ S_3$
$S_3\ \delta^{-1}\ S_3$
$S_3\ \delta^f\ S_4$
$S_2\ \delta^f\ S_3$

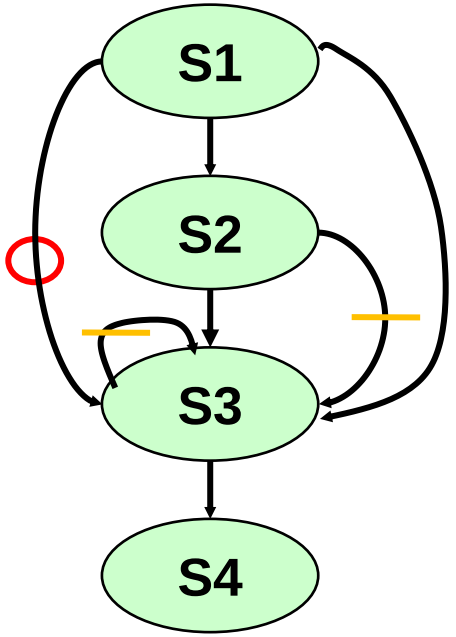


数据依赖图

- 基本块内数据依赖关系可以用有向无环图（DAG）表示，称为**数据依赖图**
  - 结点表示语句
  - 有向边表示一个数据依赖，区分边的类型
    - 边起点是依赖源语句
    - 边终点是依赖槽语句

S_1: A = 0
S_2: B = A
S_3: A = B + A
S_4: C = A

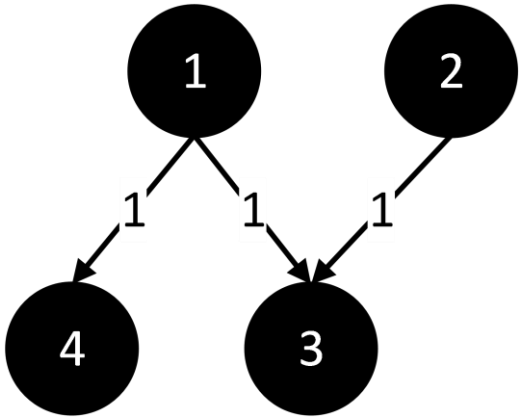
$S_1\ \delta^f\ S_2$
$S_1\ \delta^f\ S_3$
$S_1\ \delta^0\ S_3$
$S_2\ \delta^{-1}\ S_3$
$S_3\ \delta^{-1}\ S_3$
$S_3\ \delta^f\ S_4$
$S_2\ \delta^f\ S_3$



数据依赖图

- The edge from S1 to S2 is labeled with an integer that is the required latency between them

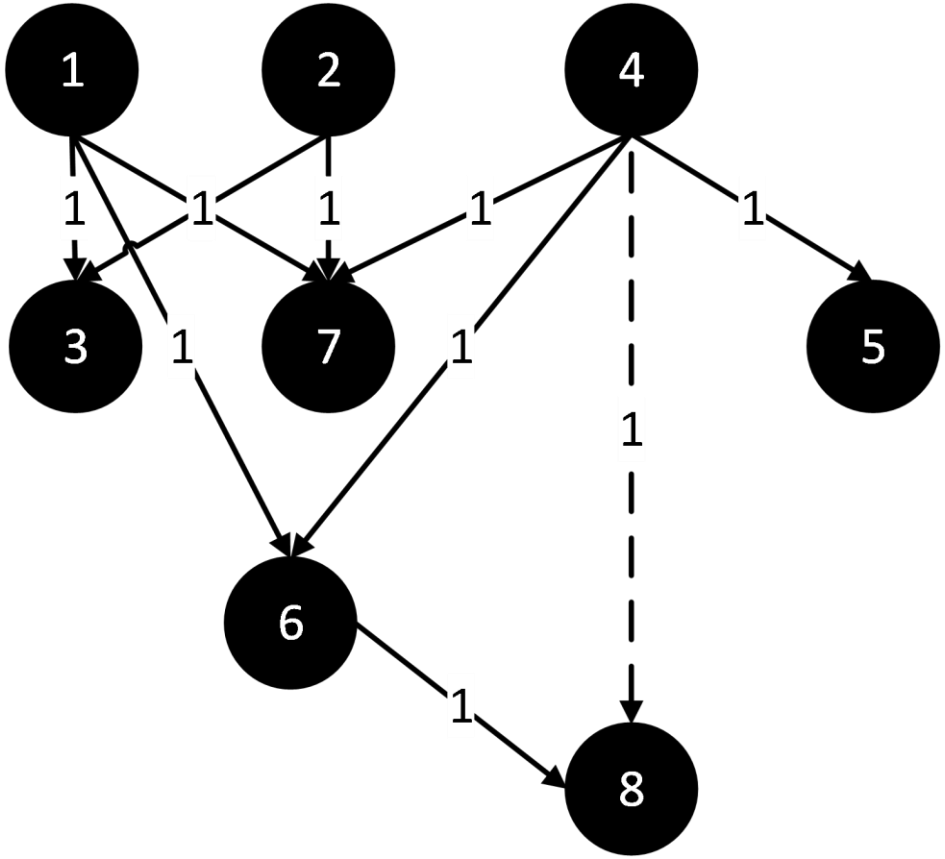
1. r2 ← [r1](4)
2. r3 ← [r1+4](4)
3. r4 ← r2 + r3
4. r5 ← r2 - 1



数据依赖图

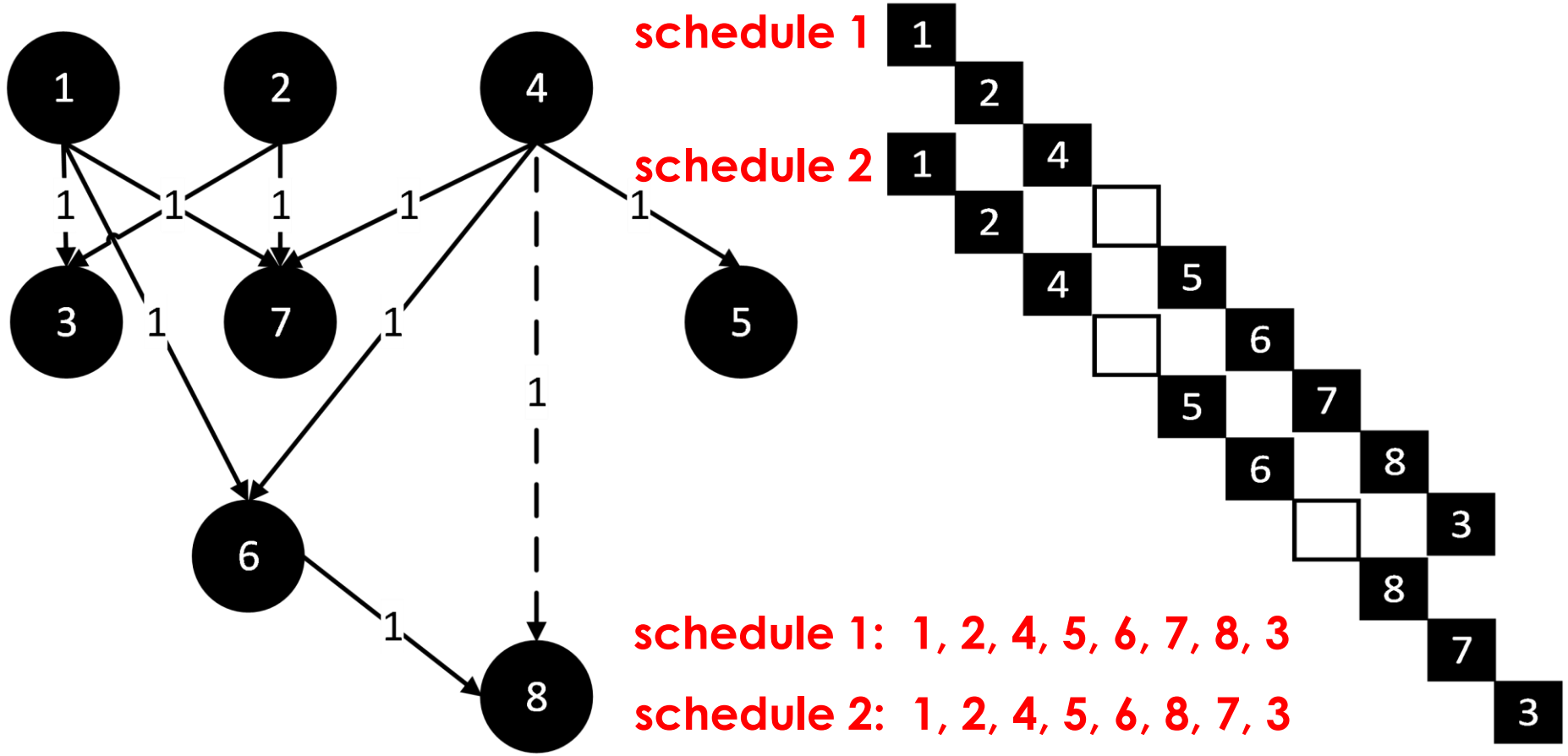
- The edge from S1 to S2 is labeled with an integer that is the required latency between them

1. r3 ← [15](4)
2. r4 ← [r15+4](4)
3. r2 ← r3 – r4
4. r5 ← [r12](4)
5. r12 ← r12 + 4
6. r6 ← r3 * r5
7. [r15+4](4) ← r3
8. r5 ← r6 + 2



数据依赖图

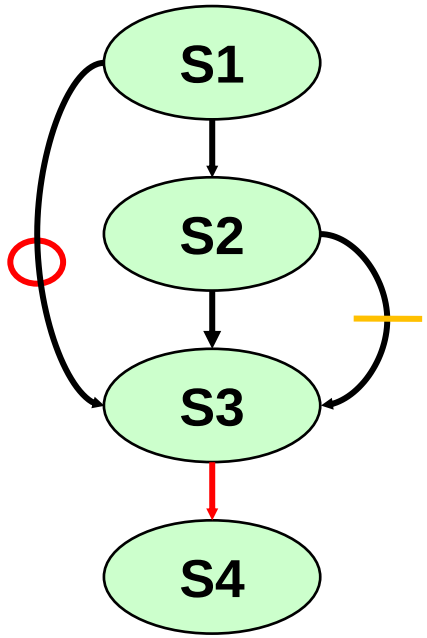
- Dependence DAGs can be used to find an efficient instruction schedule



数据依赖图：练习

- 画出下面代码段的数据依赖图

```
S1:    A = 0
S2:    B = A
S3:    A = B + A
S4:    C = A
```



回顾导入：语句S4流依赖于语句S3，不能提前到S3前执行

课堂小结

- 依赖关系概念及分类
- 控制依赖关系概念与表示
- 后必经关系
  - 严格后必经关系、直接后必经关系、后必经结点树
- 控制依赖关系的严格定义
  - 基于后必经关系定义的控制依赖关系
- 控制依赖关系的计算方法
  - 掌握算法的推演过程

作业

- 对于给定控制流图，计算边的控制依赖集合
  - 写出具体计算步骤
  - 在头歌系统提交

参考资料

COMP CCRG Open MP

- 《高级编译器设计与实现》（鲸书）第9章
- Building an Optimizing Compiler, ch3.6
- Optimizing Compilers for Modern Architectures, ch2-ch3
- 参考论文
  - Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck: An Efficient Method of Computing Static Single Assignment Form. POPL 1989: 25-35
  - Gina Goff, Ken Kennedy, Chau-Wen Tseng: Practical Dependence Testing. PLDI 1991: 15-29