

2023年秋季学期 《编译原理和技术》



# 编译原理和技术 学期复习课

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心  
计算机科学与技术学院

2023年12月13日



# 覆盖内容



- **Lecture note涉及的所有部分**

- 重点考察期中后的内容
- 上半学期的内容也会做少量考核
- 今天涉及的只是一个梳理，不是划重点

- **部分实验内容考核**

- 估计1-2道题

- **请重点参考作业、课后习题、实验、实验报告等**



# 考试形式



- **考试形式：闭卷**
- **不考填空题、选择题、名词解释题等**
- **时间地点等候通知**
  - 1.7后，考试周内



## • 词法分析

- 理解并会使用正规式（即正则表达式）
- 掌握NFA和DFA，以及之间的转换
- 学会为正规式写NFA和DFA
- DFA的化简



# 例题1



- 针对正规集 $L = \{\text{含奇数个1的0、1串}\}$ 
  - 给出描述正规集 $L$ 的正规式(即正则表达式) $R$



# 例题1



- 针对正规集  $L = \{\text{含奇数个1的0、1串}\}$ 
  - 给出描述正规集  $L$  的正规式(即正则表达式)  $R$
- 解答过程:
  - 可以先写出几个合法的串
    - 1, 111, 11111, 可以在相邻的两个1之间或第一个1之前或最后一个1之后插入任意多的0
    - 排除0以外, 合法的1串都可以看做两部分 “1+偶数个1的串”
    - $0^*1(10^*1|0)^*$



## 例题2



- **构造识别正规式  $10|(0|11)0^*1$  的极小化DFA M。**
  - 这里一定要注意优先级，不少同学把|的优先级弄错了。



## 例题2



- 构造识别正规式  $10 | (0 | 11)0^*1$  的极小化DFA M。
- 解答过程
  - 如果不熟练，可以先画出NFA
  - 然后通过子集构造法将NFA转成DFA
  - 最后通过化简得到极小化的DFA





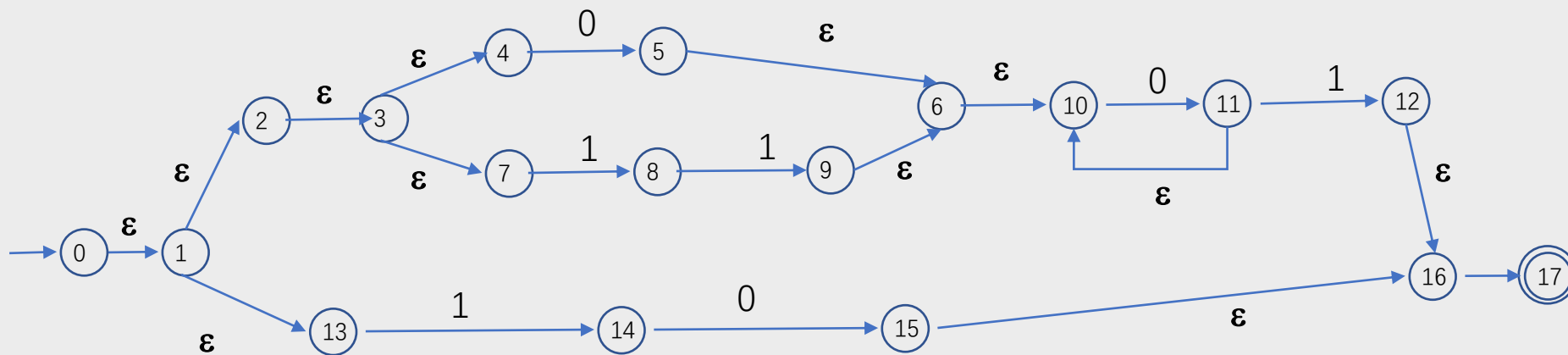
## 例题2



- 构造识别正规式  $10|(0|11)0^*1$  的极小化DFA M。

- 解答过程

- 如果不熟练，可以先画出NFA





- 子集构造法(subset construction)

- ❖  $\epsilon$ -闭包 ( $\epsilon$ -closure) : 状态 $s$ 的 $\epsilon$ -闭包是 $s$ 经 $\epsilon$ 转换所能到达的状态集合
- ❖ NFA的初始状态的 $\epsilon$ -闭包对应于DFA的初始状态
- ❖ 针对每个DFA状态 – NFA状态子集 $A$ , 求输入每个 $a_i$ 后能到达的NFA状态的 $\epsilon$ -闭包并集 ( $\epsilon$ -closure(move( $A, a_i$ ))), 该集合对应于DFA中的一个已有状态, 或者是一个要新加的DFA状态



- 子集构造法(subset construction)

- ❖  $A = \varepsilon\text{-closure}(\text{状态}0) = \{0, 1, 2, 3, 4, 7, 13\}$
- ❖ 接下来考虑转换函数 $move$ , 以及输入字母表 $\{0, 1\}$
- ❖  $move(A, \text{输入}0) = \{5\}$
- ❖  $B = \varepsilon\text{-closure}(move(A, \text{输入}0)) = \{5, 6, 10\}$
- ❖  $move(A, \text{输入}1) = \{8, 14\}$
- ❖  $C = \varepsilon\text{-closure}(move(A, \text{输入}1)) = \{8, 14\}$
- ❖  $move(B, \text{输入}0) = \{11\}$
- ❖ .....



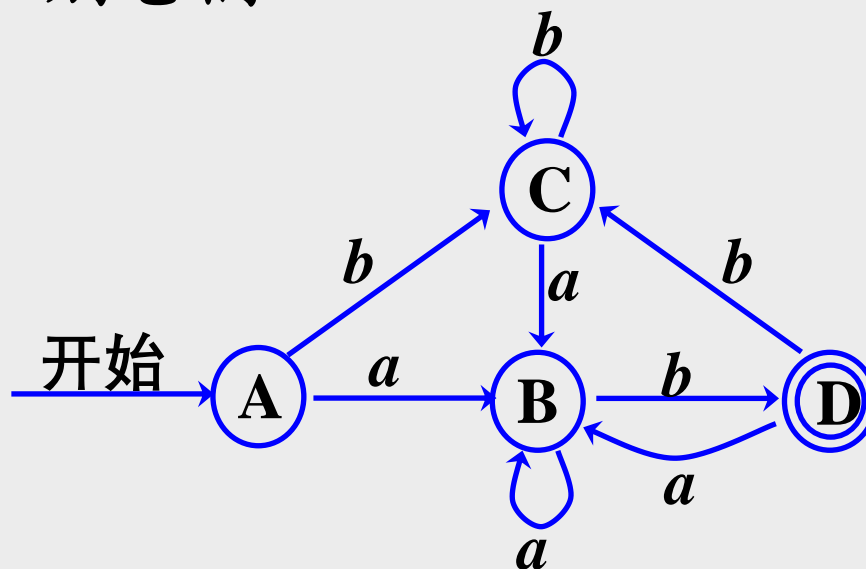
- **A和B是可区别的状态**

- ❖ 从A出发，读过单字符b构成的串，到达非接受状态C，而从B出发，读过串b，到达接受状态D

- **A和C是不可区别的状态**

- ❖ 无任何串可用来像上面这样区别它们

可区别的状态要  
分开对待



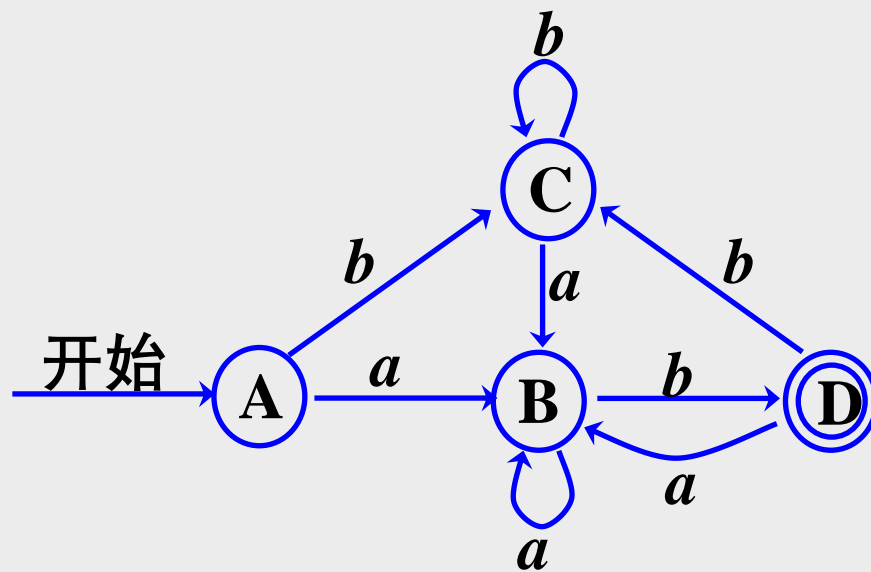


## 1. 按是否是接受状态来区分

$\{A, B, C\}, \{D\}$

$\text{move}(\{A, B, C\}, a) = \{B\}$

$\text{move}(\{A, B, C\}, b) = \{C, D\}$





## 1. 按是否是接受状态来区分

$\{A, B, C\}, \{D\}$

$\text{move}(\{A, B, C\}, a) = \{B\}$

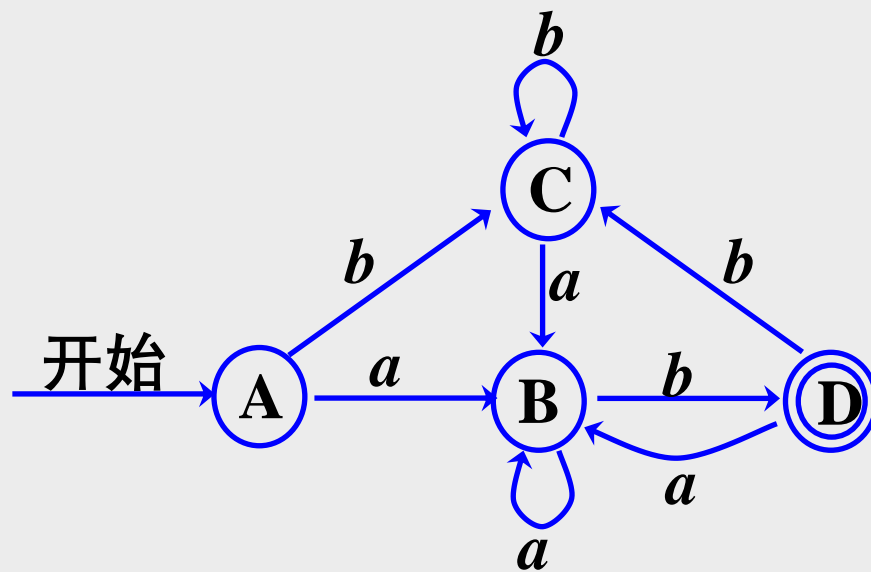
$\text{move}(\{A, B, C\}, b) = \{C, D\}$

## 2. 继续分解

$\{A, C\}, \{B\}, \{D\}$

$\text{move}(\{A, C\}, a) = \{B\}$

$\text{move}(\{A, C\}, b) = \{C\}$



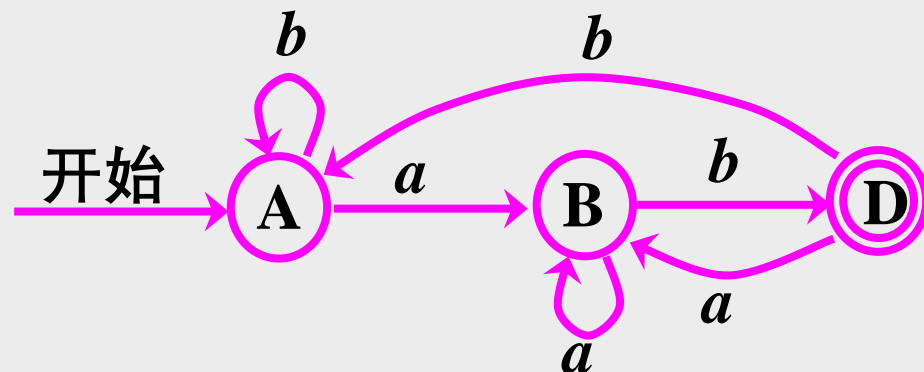


## 1. 按是否是接受状态来区分

$\{A, B, C\}, \{D\}$

$\text{move}(\{A, B, C\}, a) = \{B\}$

$\text{move}(\{A, B, C\}, b) = \{C, D\}$

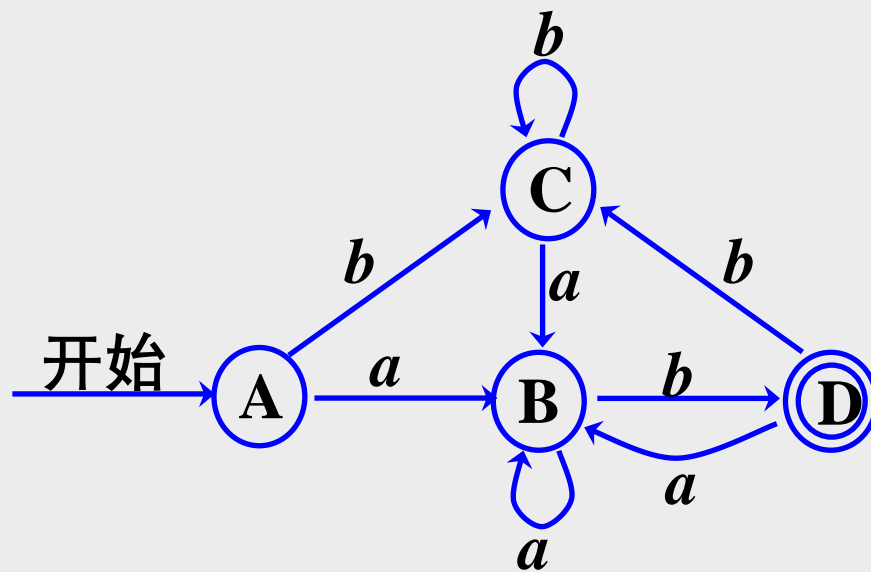


## 2. 继续分解

$\{A, C\}, \{B\}, \{D\}$

$\text{move}(\{A, C\}, a) = \{B\}$

$\text{move}(\{A, C\}, b) = \{C\}$

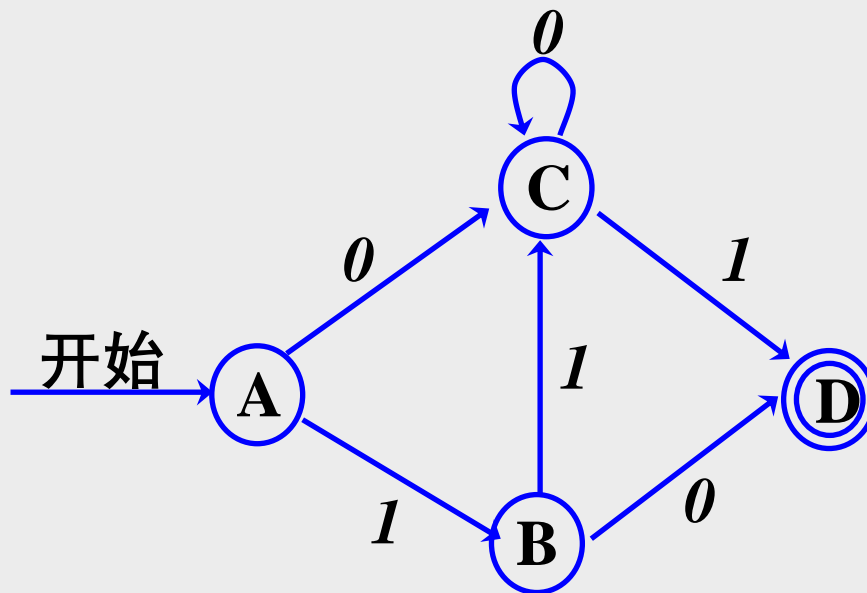




## 例题2



### • 最终结果







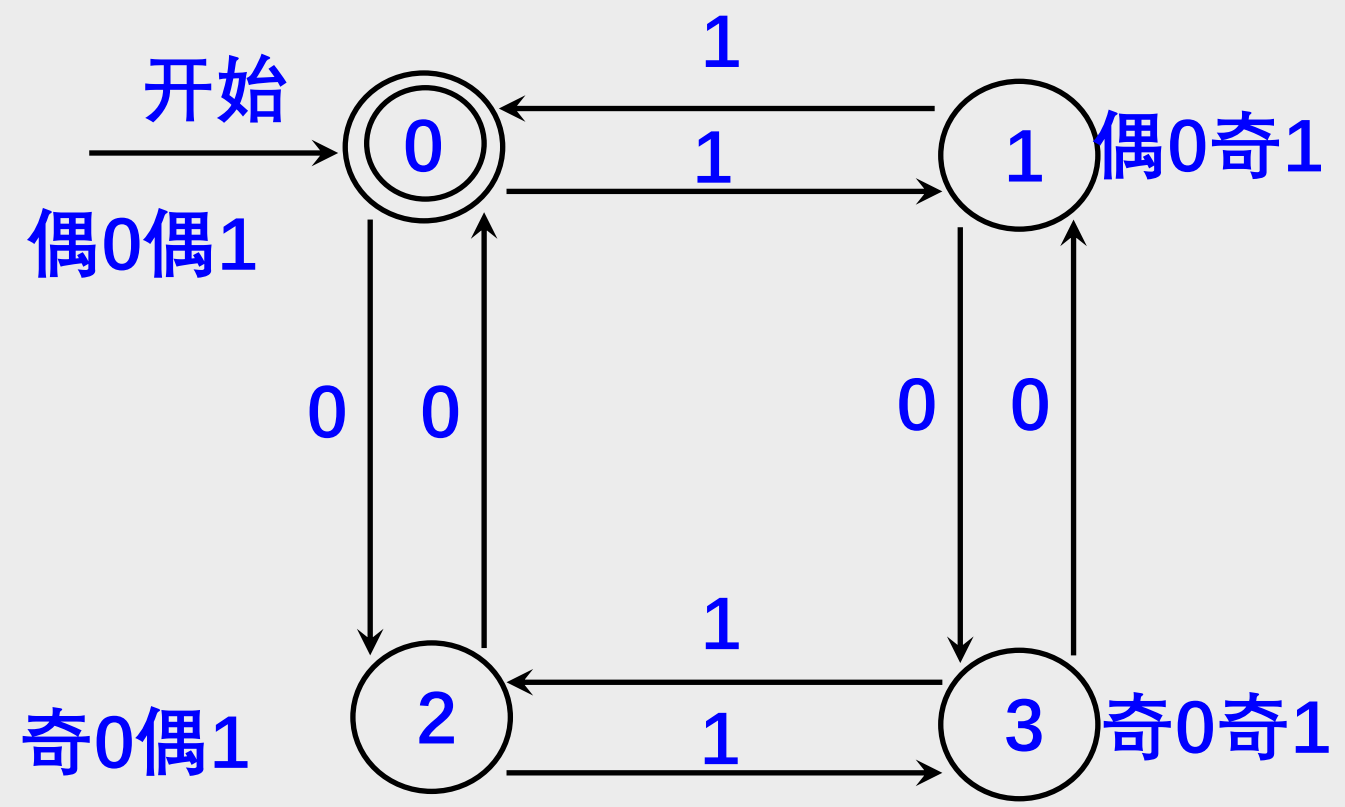
## 例题3



- **针对正规集 $L=\{0\text{和}1\text{的个数均为偶数的}0\text{-}1\text{串}\}$ :**
  - (1)先直接给出识别 $L$ 的极小化DFA  $M$ ;
  - (2)再给出描述 $L$ 的正规式 $R$ 。



# 例题3





## 例题3



- 正则表达式为:
- $(00|11|((01|10)(00|11)^*(01|10)))^*$



## • 语法分析

- 掌握文法的定义和书写格式
- 掌握FIRST/FOLLOW集合计算、二义性分析
- 掌握SLR和LR分析表的构造、移进-归约冲突的分析



## 例题4



- (1)给出产生可被5整除的二进制串集(含空串)的上下文无关文法 $G_0$ ;
- (2)并针对 $G_0$ , 给出FIRST与FOLLOW集合, LL(1)分析表;
- (3)为 $G_0$ 设计相应的递归下降分析程序。
- 解答过程:
  - 首先给出DFA, 然后按照下一页ppt的规则来写文法
  - FIRST和FOLLOW集合计算
  - 递归下降分析程序主要考虑消除左递归和match, 以及错误处理



# 正则表达式与CFG的区别



- 都能表示语言
- 能用正则表达式表示的语言都能用CFG表示

- 正则表达式

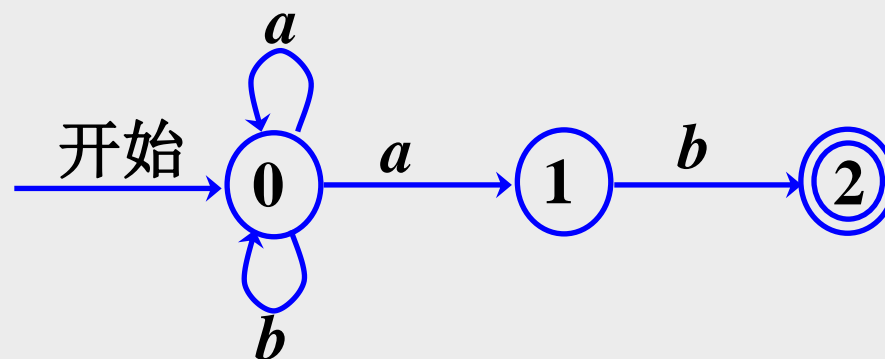
$(a|b)^*ab$

- CFG文法

$A_0 \rightarrow a A_0 \mid b A_0 \mid a A_1$

$A_1 \rightarrow b A_2$

$A_2 \rightarrow \varepsilon$

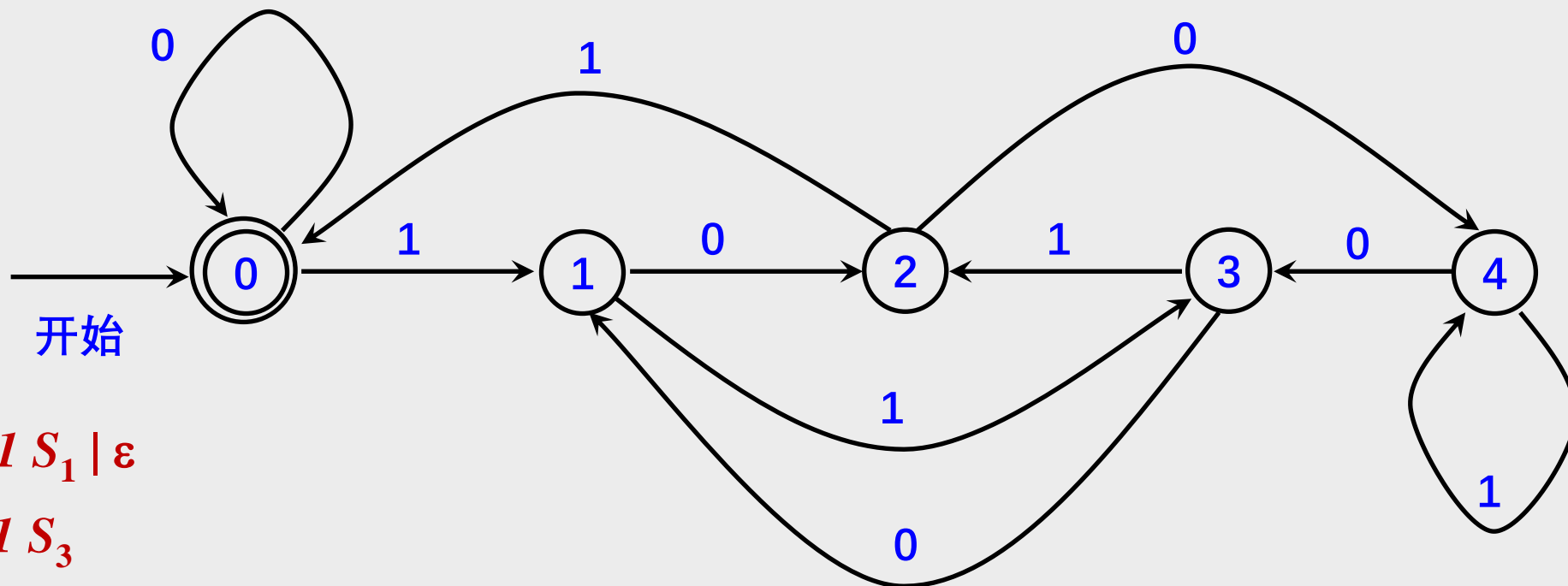




## 例题4



- (1) 给出产生可被5整除的二进制串集(含空串)的上下文无关文法G0;



$$S_0 \rightarrow 0 S_0 \mid 1 S_1 \mid \varepsilon$$

$$S_1 \rightarrow 0 S_2 \mid 1 S_3$$

$$S_2 \rightarrow 0 S_4 \mid 1 S_0$$

$$S_3 \rightarrow 0 S_1 \mid 1 S_2$$

$$S_4 \rightarrow 0 S_3 \mid 1 S_4$$



# LL(1)文法: FIRST(X)



- 计算FIRST(X),  $X \in V_T \cup V_N$

- $X \in V_T$ ,  $\text{FIRST}(X) = \{X\}$

- $X \in V_N$  且  $X \rightarrow \epsilon$

- 则将  $\epsilon$  加入到 FIRST(X)

- $X \in V_N$  且  $X \rightarrow Y_1 Y_2 \dots Y_k$

- 如果  $a \in \text{FIRST}(Y_i)$  且  $\epsilon$  在  $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_{i-1})$  中, 则将  $a$  加入到 FIRST(X)

- 如果  $\epsilon$  在  $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_k)$  中, 则将  $\epsilon$  加入到 FIRST(X)





# LL(1)文法: FOLLOW(A)



- 计算FOLLOW(A),  $A \in V_N$ 
  - \$加入到FOLLOW(A), 当A是开始符号
  - 如果 $A \rightarrow \alpha B\beta$ , 则 $\text{FIRST}(\beta) - \{\epsilon\}$ 加入到FOLLOW(B)
  - 如果 $A \rightarrow \alpha B$  或  $A \rightarrow \alpha B\beta$  且  $\epsilon \in \text{FIRST}(\beta)$ , 则FOLLOW(A)加入到FOLLOW(B)



## 例题4



- (1)给出产生可被5整除的二进制串集(含空串)的上下文无关文法G0;

$$\text{First}(S_0) = \{0, 1, \epsilon\}$$

$$\text{First}(S_1) = \text{First}(S_2) = \text{First}(S_3) = \text{First}(S_4) = \{0, 1\}$$

$$\text{Follow}(S_0) = \dots = \text{Follow}(S_4) = \{\$ \}$$

$$S_0 \rightarrow 0 S_0 \mid 1 S_1 \mid \epsilon$$

$$S_1 \rightarrow 0 S_2 \mid 1 S_3$$

$$S_2 \rightarrow 0 S_4 \mid 1 S_0$$

$$S_3 \rightarrow 0 S_1 \mid 1 S_2$$

$$S_4 \rightarrow 0 S_3 \mid 1 S_4$$



# 预测分析表M的构造



- 对文法的每个产生式  $A \rightarrow \alpha$  , 执行(1)和(2)
  - (1) 对  $\text{FIRST}(\alpha)$  的每个终结符  $a$  , 把  $A \rightarrow \alpha$  加入  $M[A, a]$
  - (2) 如果  $\varepsilon$  在  $\text{FIRST}(\alpha)$  中, 对  $\text{FOLLOW}(A)$  的每个终结符  $b$  (包括  $\$$ ) , 把  $A \rightarrow \alpha$  加入  $M[A, b]$

**$M$ 中其它没有定义的条目都是error**



# 例题4



- (1)给出产生可被5整除的二进制串集(含空串)的上下文无关文法G0;

$$\text{First}(S_0) = \{0, 1, \epsilon\}$$

$$\text{First}(S_1) = \text{First}(S_2) = \text{First}(S_3) = \text{First}(S_4) = \{0, 1\}$$

$$\text{Follow}(S_0) = \dots = \text{Follow}(S_4) = \{\$ \}$$

$$S_0 \rightarrow 0 S_0 \mid 1 S_1 \mid \epsilon$$

$$S_1 \rightarrow 0 S_2 \mid 1 S_3$$

$$S_2 \rightarrow 0 S_4 \mid 1 S_0$$

$$S_3 \rightarrow 0 S_1 \mid 1 S_2$$

$$S_4 \rightarrow 0 S_3 \mid 1 S_4$$

非终结符	输入符号		
	<u>0</u>	<u>1</u>	<u>\$</u>
S0	$S_0 \rightarrow 0 S_0$	$S_0 \rightarrow 1 S_1$	$S_0 \rightarrow \epsilon$
S1	$S_1 \rightarrow 0 S_2$	$S_1 \rightarrow 1 S_3$	
S2	$S_2 \rightarrow 0 S_4$	$S_2 \rightarrow 1 S_0$	
S3	$S_3 \rightarrow 0 S_1$	$S_3 \rightarrow 1 S_2$	
S4	$S_4 \rightarrow 0 S_3$	$S_4 \rightarrow 1 S_4$	



# 递归下降分析法



```
void type( ) {  
    if ( (lookahead == integer) || (lookahead == char) ||  
        (lookahead == num) )  
        simple( );  
    else if ( lookahead == '↑' ) { match('↑'); match(id); }  
    else if (lookahead == array) {  
        match(array); match( '[' ); simple( );  
        match( ']' ); match(of ); type( );  
    }  
    else error( );  
}
```

type → simple  
          ↑ id  
          |  
          array [simple] of type

例题4结果略



# 例题5



文法 G1 和 G2 中有一个是二义性文法，另一个是非二义性文法。

文法 G1,  $s$  为开始符号。

$S \rightarrow a B S \mid b A S$

$S \rightarrow \varepsilon$

$A \rightarrow a \mid b A \underline{A}$

$B \rightarrow b \mid a B \underline{B}$

文法 G2,  $s$  为开始符号。

$S \rightarrow a B \mid b A$

$S \rightarrow \varepsilon$

$A \rightarrow a S \mid b A \underline{A}$

$B \rightarrow b S \mid a B \underline{B}$

- (1) 针对其中的二义性文法，用串 aababb 证明其二义性；
- (2) 针对其中的非二义性文法，给出读过活前缀 aBaaBB 经过的所有 LR(0) 项目集簇。



## 例题5



- **输入串为aababb**

- 推导1:

$S \Rightarrow a B \Rightarrow a a B B \Rightarrow a a b S B \Rightarrow a a b B \Rightarrow a a b a B B \Rightarrow a a b a b S B \Rightarrow a a b a b B \Rightarrow a a b a b b$

- 推导2:

$S \Rightarrow a B \Rightarrow a a B B \Rightarrow a a b S B \Rightarrow a a b a B B \Rightarrow a a b a b S B \Rightarrow a a b a b B \Rightarrow a a b a b b$

- **存在两个最左推导，因此G2二义**



## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot \text{id}$

求项目集的闭包closure(I)

闭包函数closure(I)

1、I的每个项目均加入closure(I)

2、如果 $A \rightarrow \alpha \cdot B \beta$ 在 closure(I)中，  
且 $B \rightarrow \gamma$ 是产生式，那么如果项目 $B \rightarrow \cdot \gamma$ 还不在于closure(I)中的话，  
那么把它加入。





# LR分析法总结



		SLR	LR(1)
初始状态		$[S' \rightarrow \cdot S]$	$[S' \rightarrow \cdot S, \$]$
项目集		LR(0) CLOSURE(I)	LR(1), CLOSURE(I) 搜索符考虑 <b>FISRT</b> ( $\beta a$ )
动作	移进	$[A \rightarrow \alpha a \beta] \in I_i$ $GOTO(I_i, a) = I_j$ <b>ACTION</b> $[i, a] = sj$	$[A \rightarrow \alpha a \beta, b] \in I_i$ $GOTO(I_i, a) = I_j$ <b>ACTION</b> $[i, a] = sj$
	归约	$[A \rightarrow \alpha] \in I_i, A \neq S'$ $a \in FOLLOW(A)$ <b>ACTION</b> $[i, a] = rj$	$[A \rightarrow \alpha, a] \in I_i$ $A \neq S'$ <b>ACTION</b> $[i, a] = rj$
	接受	$[S' \rightarrow S \cdot] \in I_i$ <b>ACTION</b> $[i, \$] = acc$	$[S' \rightarrow S \cdot, \$] \in I_i$ <b>ACTION</b> $[i, \$] = acc$
	出错	空白条目	空白条目
GOTO		$GOTO(I_i, A) = I_j$ <b>GOTO</b> $[i, A] = j$	$GOTO(I_i, A) = I_j$ <b>GOTO</b> $[i, A] = j$
状态量		少(几百)	多(几千)

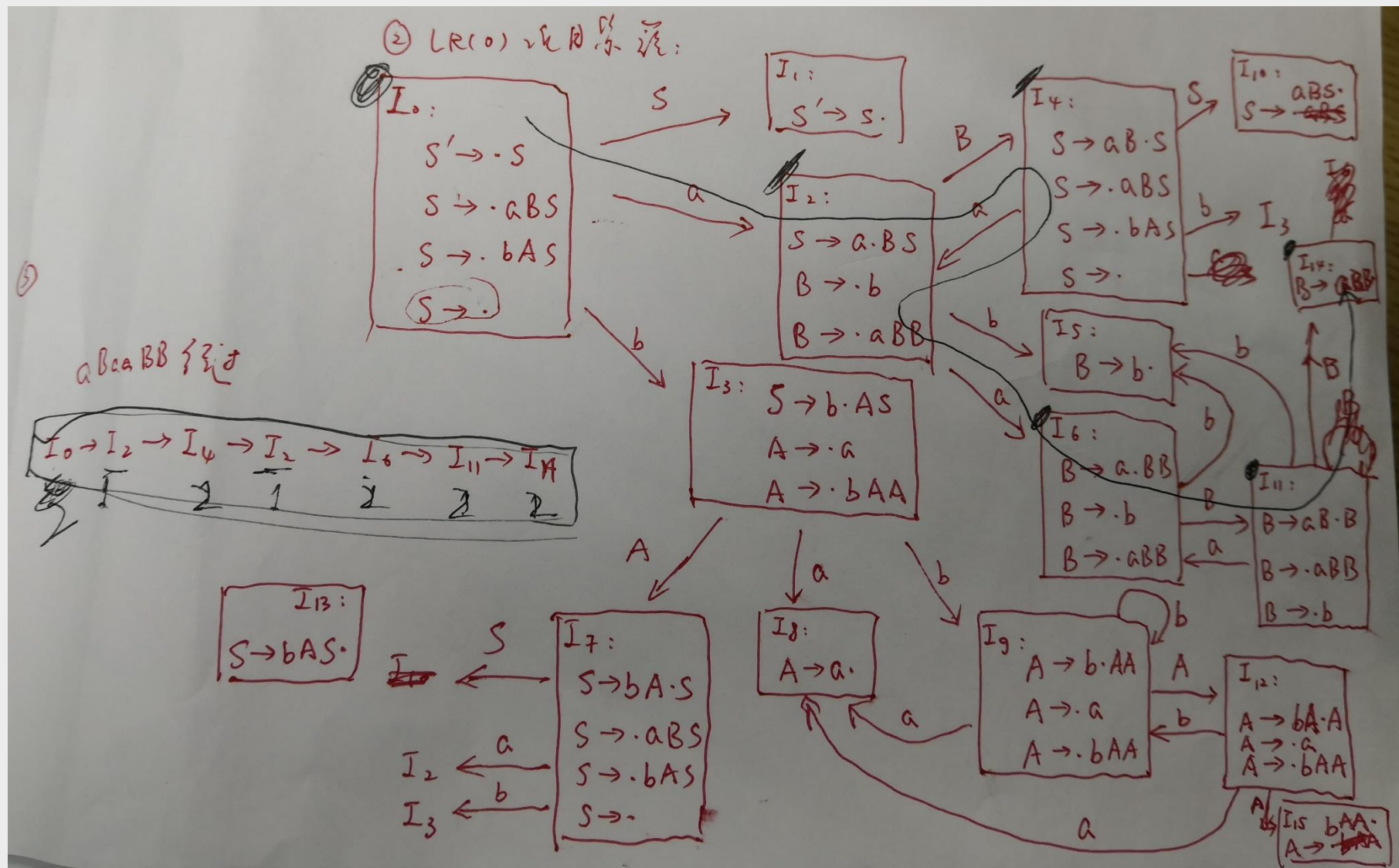


## 例题5



① 拓广文法:

$$\begin{aligned} S' &\rightarrow S, & S &\rightarrow aBS, & S &\rightarrow bAS, & S &\rightarrow \epsilon \\ A &\rightarrow a, & A &\rightarrow bAA, & B &\rightarrow b, & B &\rightarrow aBB \end{aligned}$$





# 例题6



- 针对右图中的文法G3,
  - (1)给出LR(1)项目集簇
  - (2)G3是否为SLR(1)文法?
  - (3)G3是否为LALR(1)文法?
  - (4)G3是否为LR(1)文法?

文法G3, s为开始符号。

$S \rightarrow A$

$A \rightarrow b B$

$B \rightarrow c C$

$B \rightarrow c C e$

$C \rightarrow d A$

$A \rightarrow a$







# 例题6



## • 针对右图中的文法G3,

- (1)给出LR(1)项目集簇
- (2)G3是否为SLR(1)文法?
  - 不是, 在LR(0)中考虑I7

文法G3, s为开始符号。

$S \rightarrow A$

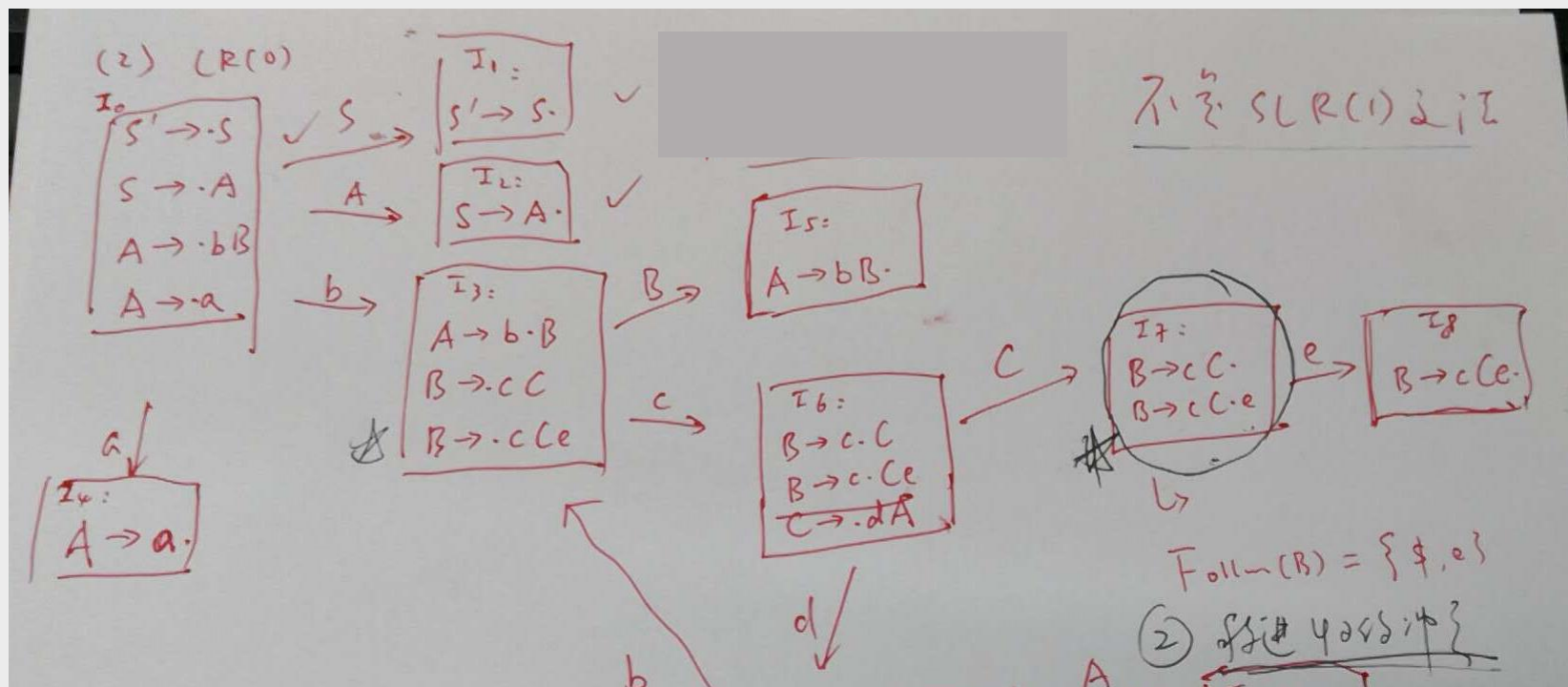
$A \rightarrow b B$

$B \rightarrow c C$

$B \rightarrow c C e$

$C \rightarrow d A$

$A \rightarrow a$





# 例题6



## • 针对右图中的文法G3,

- (1)给出LR(1)项目集簇
- (2)G3是否为SLR(1)文法?
- (4)G3是否为LR(1)文法?
  - 不是, 在LR(1)状态转换图中考虑I15
  - 有移进-归约冲突

文法G3, s为开始符号。

$S \rightarrow A$

$A \rightarrow b B$

$B \rightarrow c C$

$B \rightarrow c C e$

$C \rightarrow d A$

$A \rightarrow a$



## • 语法制导翻译

- 掌握语法制导翻译方案
- 掌握简单的综合属性和继承属性计算
- 掌握继承属性的自下而上计算模拟
  - 栈上的计算，与分析一起





## 例题7



### • 4.12(b) 文法如下:

$$S \rightarrow (L) \mid a$$

$$L \rightarrow L, S \mid S$$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2, 5, 8, 10, 14。

(4)写出自下而上分析的栈操作代码



## • 语义规则和产生式相联系的两种方式

### • 语法制导定义

- 将文法符号和某些属性相关联，并通过语义规则来描述如何计算属性的值，没有描述这些规则的计算时机

### • 语法制导的翻译方案

- 在产生式的右部的适当位置，插入相应的语义动作，按照分析的进程，执行遇到的语义动作，从而明确了语法分析过程中属性的计算时机。



## 例题7



• **a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。**

- 继承属性 in: 该文法符号推出的字符序列的前面已经有多少字符
- 综合属性 total: 该文法符号推出的字符序列所包含的字符总数

$$S' \rightarrow \{ S.in = 0; \} S$$
$$S \rightarrow \{ L.in = S.in + 1; \} (L) \{ S.total = L.total + 2; \}$$
$$S \rightarrow a \{ S.total = 1; \text{print}(S.in + 1); \}$$
$$L \rightarrow \{ L1.in = L.in; \} L1, \{ S.in = L1.in + L1.total + 1; \} S$$
$$\{ L.total = L1.total + S.total + 1; \}$$
$$L \rightarrow \{ S.in = L.in; \} S \{ L.total = S.total; \}$$



## 例题7



• **a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。**

- 继承属性 in: 该文法符号推出的字符序列的前面已经有多少字符
- 综合属性 total: 该文法符号推出的字符序列所包含的字符总数

$S' \rightarrow M S$

$M \rightarrow \varepsilon \{M.s = 0\}$

$S \rightarrow (NL) \{ S.total = L.total + 2; \}$

$N \rightarrow \varepsilon \{ N.i = S.in + 1, N.s = N.i \}$

$S \rightarrow a \{ S.total = 1; \text{print}(S.in + 1); \}$

$L \rightarrow R L1, P S \{ L.total = L1.total + S.total + 1; \}$

$R \rightarrow \varepsilon \{ R.i = L.in; R.s = R.i \}$

$P \rightarrow \varepsilon \{ P.i = L1.in + L1.total + 1, P.s = P.i \}$

$L \rightarrow T S \{ L.total = S.total; \}$

$T \rightarrow \varepsilon \{ T.i = L.in, T.s = T.i \}$



## • 引入标记非终极符M,N,R,P

产生式	语义规则	栈操作代码
$S' \rightarrow M S$		
$M \rightarrow \varepsilon$	$\{M.s = 0\}$	$val[top+1] = 0$
$S \rightarrow (NL)$	$\{ S.total = L.total + 2; \}$	$val[top-3] = val[top-1] + 2$
$N \rightarrow \varepsilon$	$\{ N.i = S.in + 1, N.s = N.i \}$	$val[top+1] = val[top-1] + 1$
$S \rightarrow a$	$\{ S.total = 1; print (S.in + 1); \}$	$val[top] = 1; print(val[top-1]+1)$
$L \rightarrow R L1, P S$	$\{L.total = L1.total + S.total + 1; \}$	$val[top-4] = val[top] + val[top-3] + 1$
$R \rightarrow \varepsilon$	$\{R.i = L.in; R.s = R.i\}$	$val[top+1] = val[top]$
$P \rightarrow \varepsilon$	$\{ P.i = L1.in + L1.total + 1, P.s = P.i \}$	$val[top+1] = val[top-2] + val[top-1] + 1$
$L \rightarrow T S$	$\{ L.total = S.total; \}$	$val[top-1] = val[top]$
$T \rightarrow \varepsilon$	$\{ T.i = L.in, T.s = T.i \}$	$val[top+1] = val[top]$



# 重要知识点复习



- **类型检查**

- 掌握类型表达式书写
  - 指针、数组、结构体、函数等



## 例题8



- 5.5 假如有下列C的声明:

```
typedef struct{
```

```
    int a, b;
```

```
} CELL, *PCELL;
```

```
CELL foo[100];
```

```
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。



# 例题8



- 5.5 假如有下列C的声明:

```
typedef struct{
```

```
    int a, b;
```

```
} CELL, *PCELL;
```

```
CELL foo[100];
```

```
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

CELL foo[100];

array(Range ?, TypeOfElement ?)

array(0..99, TypeOfElement ?)

array(0..99, CELL)

array(0..99, record((int a) × (int b)))

array(0..99, record((a × integer) × (b × integer)))





# 例题8



- 5.5 假如有下列C的声明:

```
typedef struct{
```

```
    int a, b;
```

```
} CELL, *PCELL;
```

```
CELL foo[100];
```

```
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

PCELL bar(x, y) int x; CELL y; {}

TypeOfParameters? -> TypeOfReturnValue?

(int × CELL) -> PCELL

(integer × record((a × integer) × (b × integer))) -> PCELL

(integer × record((a × integer) × (b × integer))) ->  
pointer(record((a × integer) × (b × integer)))

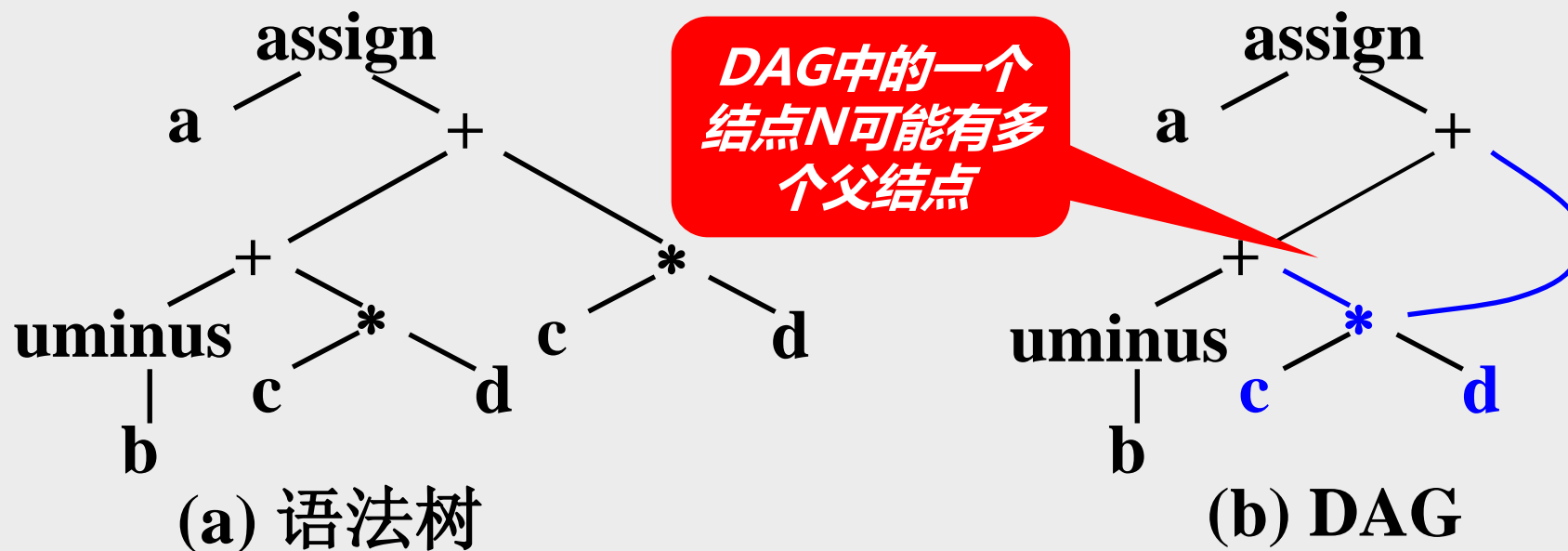


## • 中间代码生成

- 掌握三地址码的格式
- 掌握基本块、流图、循环
  - 给定三地址码，如何划分基本块、画出流图、找出循环、计算回边等



- 语法树是一种图形化的中间表示
- 有向无环图(Directed Acyclic Graph, DAG)也是一种中间表示



$a = (-b + c*d) + c*d$  的图形表示



## • 三地址代码 (Three-Address Code, TAC)

一般形式:  $x = y \text{ op } z$

- 最多一个算符
- 最多三个计算分量
- 每一个分量代表一个地址, 因此三地址

## • 例 表达式 $x + y * z$ 翻译成的三地址语句序列

$$t_1 = y * z$$

$$t_2 = x + t_1$$



- 三地址代码是语法树或DAG的一种线性表示

- 例  $a = (-b + c * d) + c * d$

语法树的代码

$$t_1 = -b$$

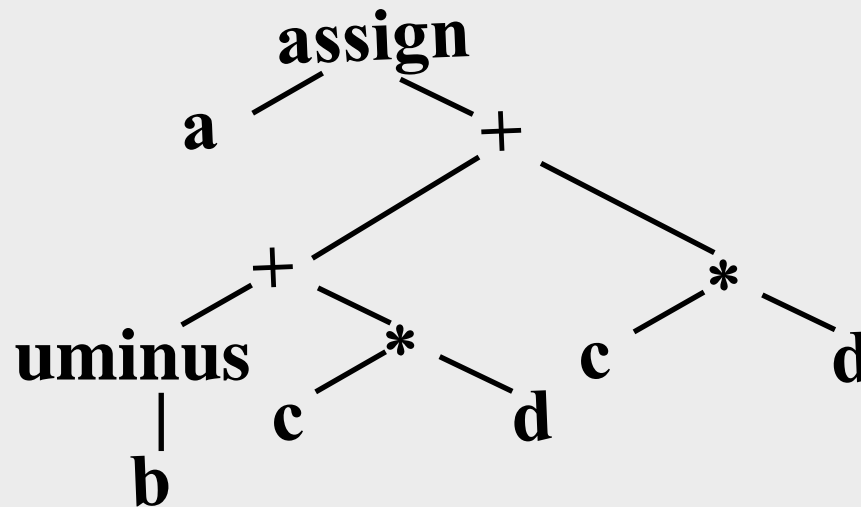
$$t_2 = c * d$$

$$t_3 = t_1 + t_2$$

$$t_4 = c * d$$

$$t_5 = t_3 + t_4$$

$$a = t_5$$





## • 三地址代码是语法树或DAG的一种线性表示

• 例  $a = (-b + c * d) + c * d$

语法树的代码

DAG的代码

$$t_1 = -b$$

$$t_1 = -b$$

$$t_2 = c * d$$

$$t_2 = c * d$$

$$t_3 = t_1 + t_2$$

$$t_3 = t_1 + t_2$$

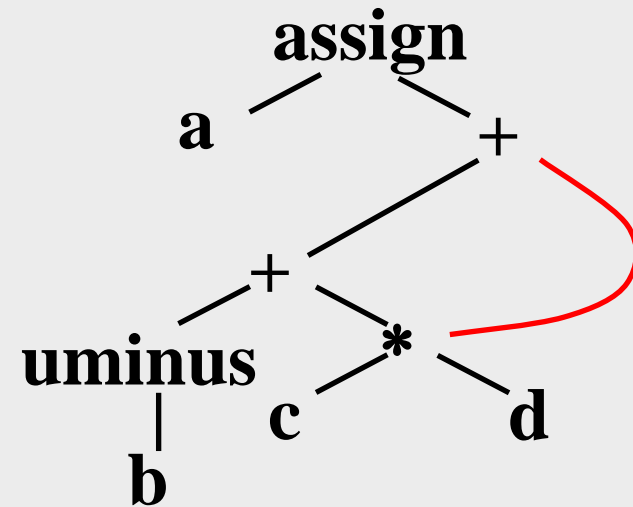
$$t_4 = c * d$$

$$t_4 = t_3 + t_2$$

$$t_5 = t_3 + t_4$$

$$a = t_4$$

$$a = t_5$$





## • 常用的三地址语句

- 运算/赋值语句  $x = y \text{ op } z, \quad x = \text{op } y, \quad x = y$
- 无条件转移 **goto L**
- 条件转移1 **if x goto L, if False x goto L**
- 条件转移2 **if x relop y goto L**



## • 常用的三地址语句

### • 过程调用

- **param  $x_1$**  //设置参数
- **param  $x_2$**
- ...
- **param  $x_n$**
- **call  $p, n$**  //调用子过程p, n为参数个数

### • 过程返回 **return $y$**

### • 索引赋值 **$x = y[i]$ 和 $x[i] = y$**

- 注意:  $i$ 表示距离 $y$ 处 $i$ 个内存单元

### • 地址和指针赋值 **$x = \&y$ , $x = *y$ 和 $*x = y$**





# 三地址代码翻译：举例



## • 考虑语句，令数组a的每个元素占8存储单元

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

```
L:   $t_1 = i + 1$   
     $i = t_1$   
     $t_2 = i * 8$   
     $t_3 = a[t_2]$   
    if  $t_3 < v$  goto L
```

符号标号

```
100:  $t_1 = i + 1$   
101:  $i = t_1$   
102:  $t_2 = i * 8$   
103:  $t_3 = a[t_2]$   
104: if  $t_3 < v$  goto 100
```

位置标号



- 一种便于某些代码优化的中间表示
- 和三地址代码的主要区别
  - 所有赋值指令都是对不同名字的变量的赋值

三地址代码

$$p = a + b$$

$$q = p - c$$

$$p = q * d$$

$$p = e - p$$

$$q = p + q$$

静态单赋值形式

$$p_1 = a + b$$

$$q_1 = p_1 - c$$

$$p_2 = q_1 * d$$

$$p_3 = e - p_2$$

$$q_2 = p_3 + q_1$$

SSA由Barry K. Rosen、Mark N. Wegman和  
F. Kenneth Zadeck于1988年提出



- 一种便于某些代码优化的中间表示
- 和三地址代码的主要区别
  - 所有赋值指令都是对不同名字的变量的赋值
  - 同一个变量在不同控制流路径上都被定值

if (flag)  $x = -1$ ; else  $x = 1$ ;

$y = x * a$ ;

改成

if (flag)  $x_1 = -1$ ; else  $x_2 = 1$ ;

$x_3 = \phi(x_1, x_2)$ ;      //由flag的值决定用 $x_1$ 还是 $x_2$

$y = x_3 * a$ ;



## 快速排序程序片段如下

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

**while (1) {**

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

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

**if ( $i \geq j$ ) break;**

**$x = a[i]; a[i] = a[j]; a[j] = x;$**

**}**

**$x = a[i]; a[i] = a[n]; a[n] = x;$**

(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  $i \geq j$  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)  $a[t15] := x$



# 基本块(Basic block)



- **连续的三地址指令序列，控制流从它的开始进入，并从它的末尾离开，中间没有停止或分支的可能性（末尾除外）**

- **流图(flow graph)**

用有向边表示基本块之间的控制流信息，基本块作为结点



# 基本块划分算法



- 输入：三地址指令序列
- 输出：基本块列表
- 算法：
  - 首先确定基本块的第一个指令，即首指令(leader)
    - 指令序列的第一条三地址指令是一个首指令
    - 任意转移指令的目标指令是一个首指令
    - 紧跟一个转移指令的指令是一个首指令
  - 然后，每个首指令对应的基本块包括了从它自己开始，直到下一个首指令(不含)或指令序列结尾之间的所有指令



# 举例



```
(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
```



# 举例——首指令



(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  $i \geq j$  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)  $a[t15] := x$





# 举例——基本块



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

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

---

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

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

---

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

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

---

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

(13) if  $i \geq j$  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)

---

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

(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)  $a[t15] := x$



# 流图 (Flow graph)



- 流图的结点是一些基本块
- 从基本块 $B$ 到基本块 $C$ 之间有一条边，当且仅当 $C$ 的第一个指令可能紧跟在 $B$ 的最后一条指令之后执行
  - $B$ 是 $C$ 的前驱 (predecessor)
  - $C$ 是 $B$ 的后继 (successor)



# 流图 (Flow graph)



- 流图的结点是一些基本块
- 从基本块 $B$ 到基本块 $C$ 之间有一条边，当且仅当 $C$ 的第一个指令可能紧跟在 $B$ 的最后一条指令之后执行，**判定方法如下**：
  - 有一个从 $B$ 的结尾跳转到 $C$ 的开头的跳转指令
  - 参考原来三地址指令序列中的顺序， $C$ 紧跟在 $B$ 之后，且 $B$ 的结尾没有无条件跳转指令



# 举例——流图



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

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

---

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

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

---

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

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

---

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

(13) if  $i \geq j$  goto (23)  
(14)  $t6 := 4 * i$   
(15)  $x := a[t6]$

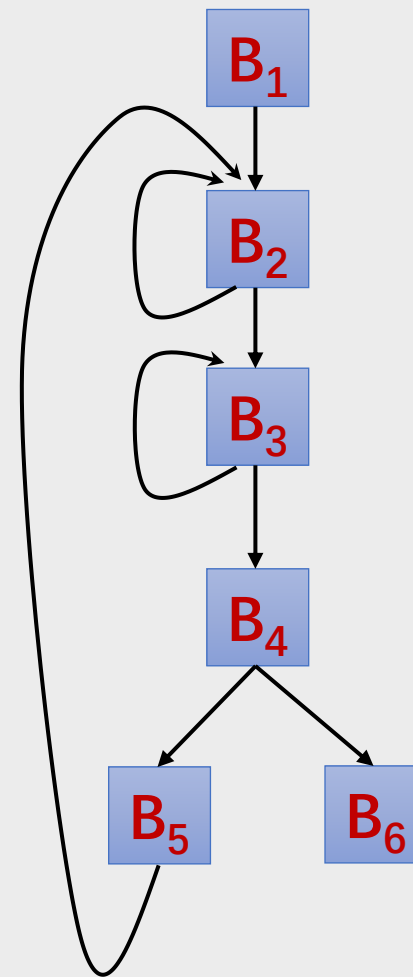
**B<sub>5</sub>**

(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)

---

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

(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)  $a[t15] := x$



## 例题10: $A[i, j] := B[i, j] * k$



- **数组A**:  $A[1..10, 1..20]$  of integer;

**数组B**:  $B[1..10, 1..20]$  of integer;

$w : 4$  (integer)

为高级语言程序写三地址码

- **TAC如下:**

(1)  $t_1 := i * 20$

(2)  $t_1 := t_1 + j$

(3)  $t_2 := A - 84 \ // \ 84 == ( (1*20)+1 ) * 4$

(4)  $t_3 := t_1 * 4 \ // \text{以上} A[i, j] \text{的 (左值) 翻译}$



## 例题10: $A[i, j] := B[i, j] * k$



TAC如下 (续) :

(5)  $t_4 := i * 20$

(6)  $t_4 := t_4 + j$

(7)  $t_5 := B - 84$

(8)  $t_6 := t_4 * 4$

(9)  $t_7 := t_5[t_6]$

//以上计算 $B[i, j]$ 的  
右值

TAC如下 (续) :

(10)  $t_8 := t_7 * k$

//以上整个右值表达

//式计算完毕

(11)  $t_2[t_3] := t_8$

// 完成数组元素的赋值



## 例题11



翻译以下语句序列：

要掌握标号回填技术

**if (  $a < b$  or  $c < d$  and  $e < f$  ) then**

**while (  $a > c$  ) do  $c := c + 1$**

**else  $d := d + 1$ ;**

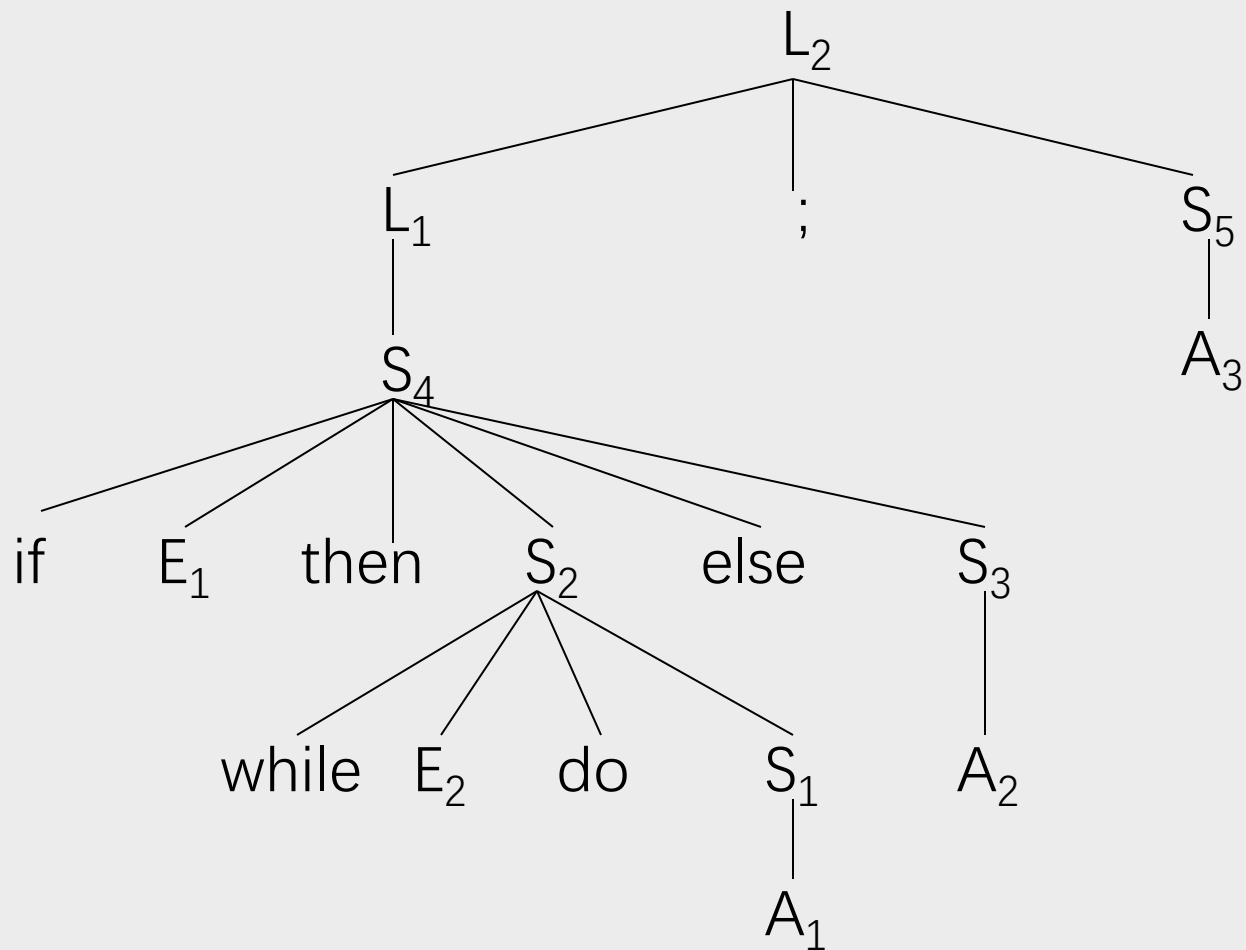
**$e := e + d$ ;**



# 例题11



## • 分析树







# 例题11



**一、翻译  $E_1$ : (  $a < b$  or  $c < d$  and  $e < f$  )**

**(100) if  $a < b$  goto 106**

**(101) goto 102                   //用102回填(101)**

**(102) if  $c < d$  goto 104   //用104回填(102)**

**(103) goto 111**

**(104) if  $e < f$  goto 106**

**(105) goto 111**

**truelist: { 100, 104 } falselist: { 103, 105 }**



# 例题11



**二、翻译  $S_2$ : while  $E_2$  do  $S_1$**

**(106) if  $a > c$  goto 108 //用108回填(106)**

**(107) goto 112**

**(108)  $c := c + 1$  //  $S_1 \rightarrow A_1$   $S_1.nextlist = \{\}$**

**(109) goto 106 // 转至循环入口(106)**

**$S_2.nextlist: \{ 107 \}$  //转至循环外部**

**(110) goto 112 // 由 $N \rightarrow \epsilon$ 生成**

**(111)  $d := d + 1$  //  $S_3 \rightarrow A_2$   $S_3.nextlist = \{\}$**



# 例题11



## 三、分析完 $S_4$

- 用106回填(100)和(104); 用111回填(103)和(105)
- $S_4.nextlist: \{ 107, 110 \}$

## 四、分析完 $L_1$

- $L_1.nextlist: \{ 107, 110 \}$

## 五、分析 $S_5$

(112)  $e := e + d$  //  $S_5 \rightarrow A_3$   $S_5.nextlist = \{\}$



# 例题11



## 六、分析完 $L_2$

- 用112回填(107)和(110)
- $L_2.nextlist: \{\}$



## 例题12:



### • 9.15 a. 计算支配关系

$$D(1) = \{1\}$$

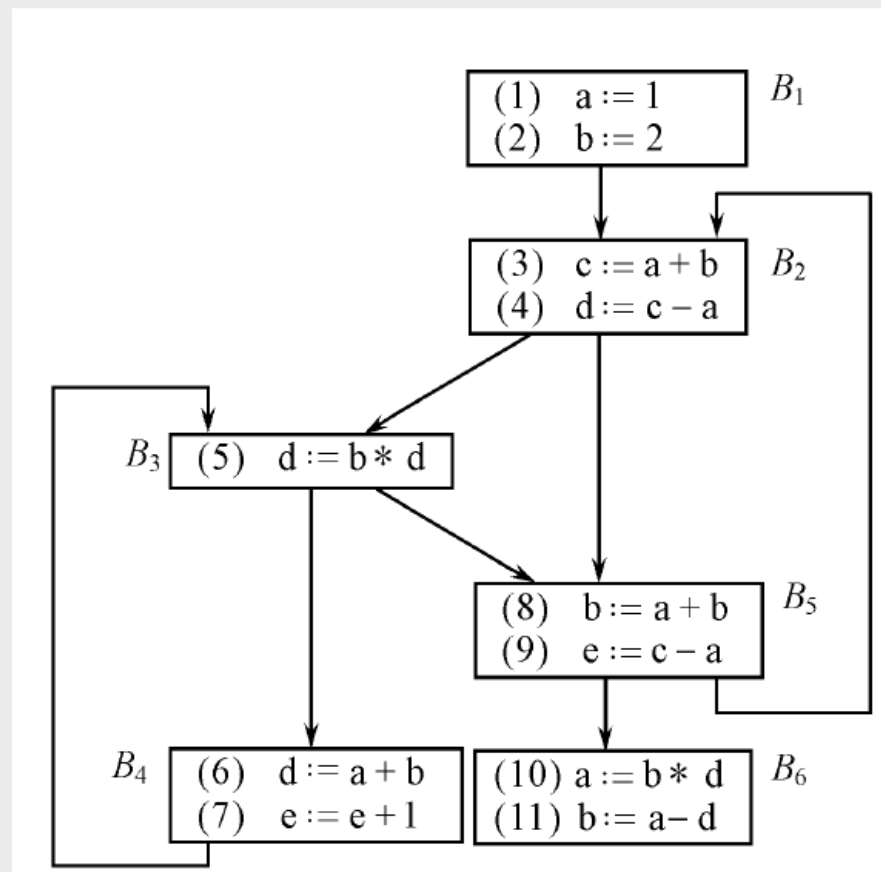
$$D(2) = \{1, 2\}$$

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

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

$$D(5) = \{1, 2, 5\}$$

$$D(6) = \{1, 2, 5, 6\}$$

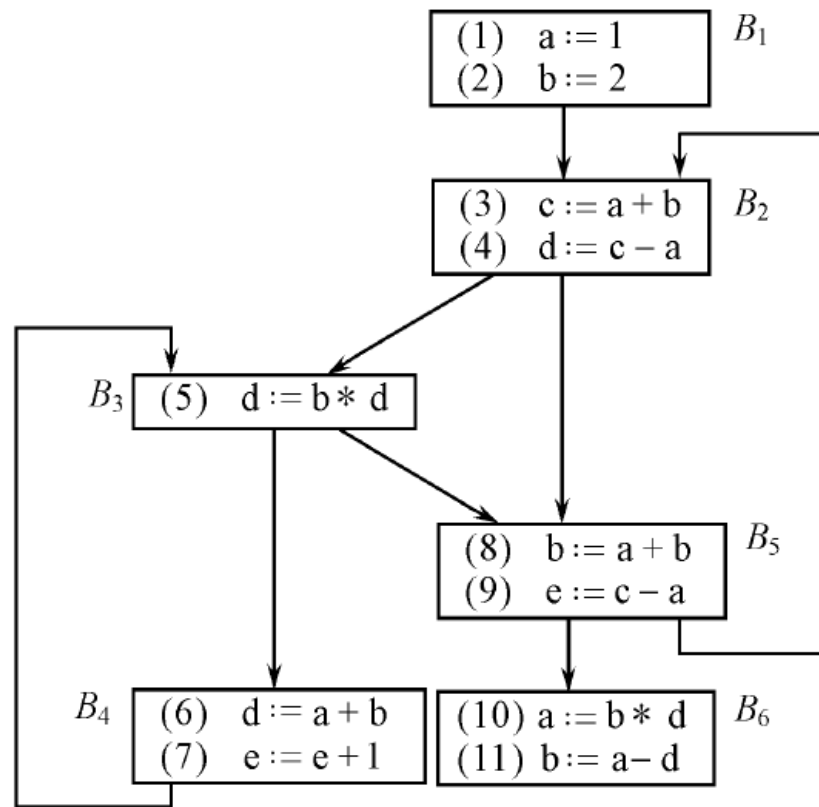




## 例题12:



- 9.15 b.找出一种深度优先排序
- $\{1,2,5,6,3,4\}$
- Or
- $\{1,2,3,4,5,6\}$

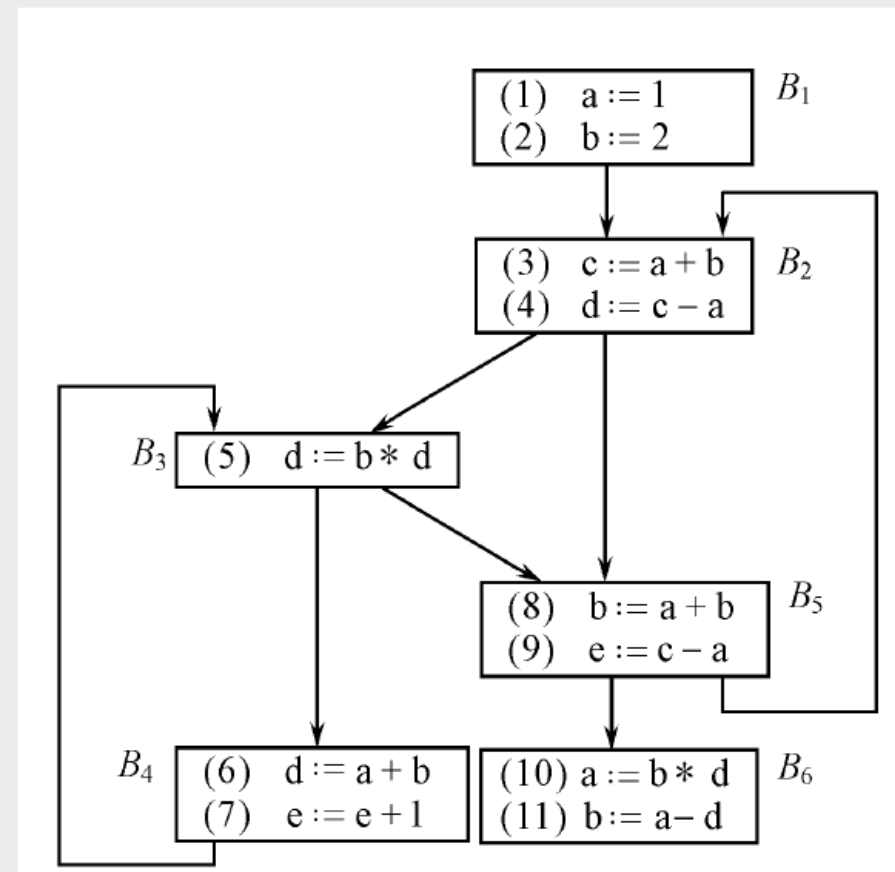




## 例题12:



- 9.15 c.对 (b) 的结果, 标明前进边, 后撤边和交叉边
- 前进边:  $1 \rightarrow 2$ ;  $2 \rightarrow 5$ ;  $2 \rightarrow 3$ ;  $5 \rightarrow 6$ ;  $3 \rightarrow 4$
- 后撤边:  $4 \rightarrow 3$ ;  $5 \rightarrow 2$
- 交叉边:  $3 \rightarrow 5$

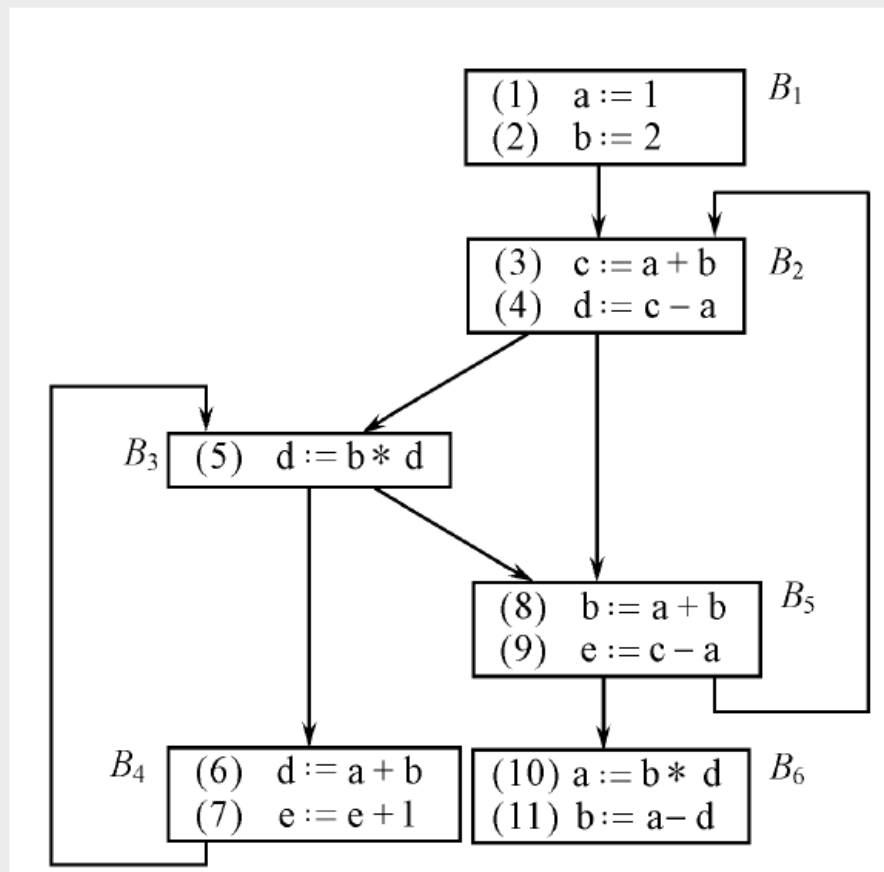




## 例题12:



- 9.15 d.该图是否可归约
- 后撤边:  $4 \rightarrow 3$ ;  $5 \rightarrow 2$
- 判断他们是不是回边
- 显然是
- 所以可以归约



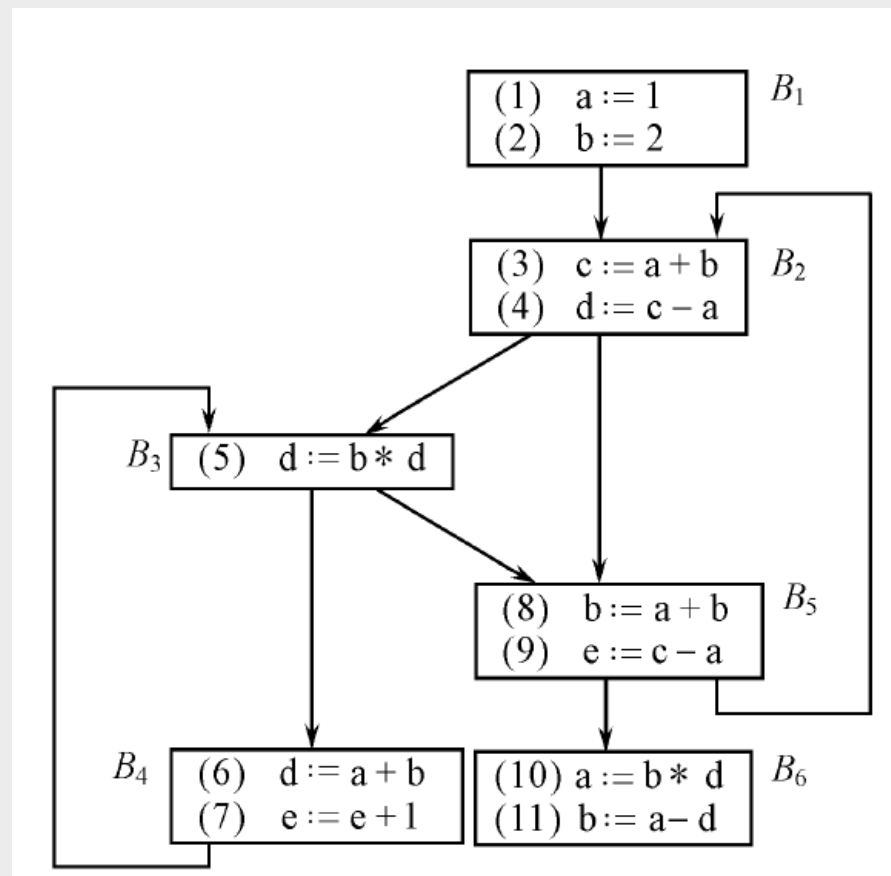




## 例题12:



- 9.15 f.找出该图的自然循环
- 针对回边:
- 4->3: {3,4}
- 5->2: {2,3,4,5}





## • 独立于机器的优化

- 掌握数据流分析的基本概念
- 掌握一些数据流分析的方法
  - 如计算到达定值等
- 掌握基本块内部优化的思想

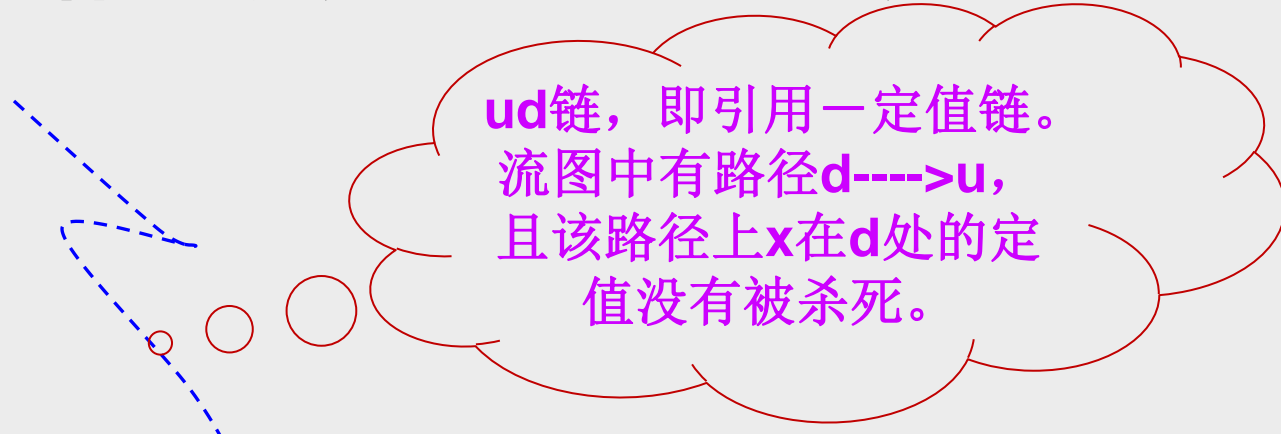


- **到达一个程序点的所有定值(gen)**
- **定值的注销(kill)**
  - 在一条执行路径上，对x的赋值注销先前对x的所有赋值
- **别名给到达一定值的计算带来困难，因此，本章其余部分仅考虑变量无别名的情况**



## • 定值与引用

**d :**     $x := y + z$  // 语句d 是变量x的一个定值点



ud链，即引用一定值链。  
流图中有路径 $d \rightsquigarrow u$ ，  
且该路径上x在d处的定  
值没有被杀死。

**u :**     $w := x + v$  // 语句u 是变量x的一个引用点

## • 变量x在d点的定值到达u点



# 到达一定值分析的用途



- **循环不变计算的检测**

- 如果循环中含有赋值 $x=y+z$ ，而 $y$ 和 $z$ 所有可能的定值都在循环外，那么 $y+z$ 就是循环不变计算

- **常量合并**

- 如果对变量 $x$ 的某次使用只有一个定值到达，且该定值把一个常量赋给 $x$ ，则可以用该常量替换 $x$

- **错误检测**

- 判定变量 $x$ 在 $p$ 点上是否未经定值就被引用



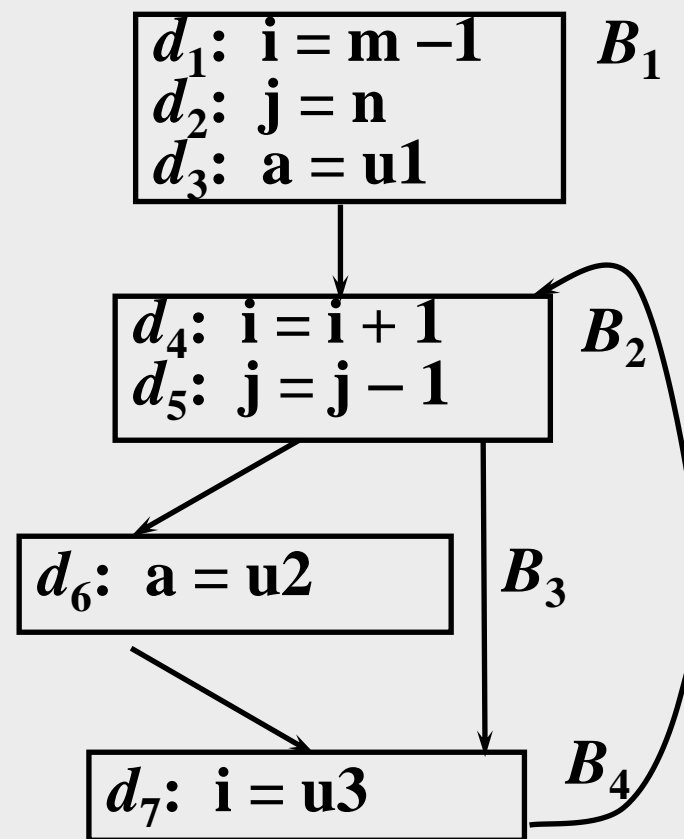
- *gen*和*kill*分别表示一个基本块生成和注销的定值

$gen [B_1] = \{d_1, d_2, d_3\}$   
 $kill [B_1] = \{d_4, d_5, d_6, d_7\}$

$gen [B_2] = \{d_4, d_5\}$   
 $kill [B_2] = \{d_1, d_2, d_7\}$

$gen [B_3] = \{d_6\}$   
 $kill [B_3] = \{d_3\}$

$gen [B_4] = \{d_7\}$   
 $kill [B_4] = \{d_1, d_4\}$





- 基本块的 $gen$ 和 $kill$ 是怎样计算的

- 对三地址指令  $d: u = v + w$ ，它的状态传递函数是

$$f_d(x) = gen_d \cup (x - kill_d)$$

- 若：  $f_1(x) = gen_1 \cup (x - kill_1)$ ,  $f_2(x) = gen_2 \cup (x - kill_2)$

$$\text{则： } f_2(f_1(x)) = gen_2 \cup (gen_1 \cup (x - kill_1) - kill_2)$$

$$= (gen_2 \cup (gen_1 - kill_2)) \cup (x - (kill_1 \cup kill_2))$$

- 若基本块 $B$ 有 $n$ 条三地址指令

$$f_B(x) = gen_B \cup (x - kill_B)$$

$$kill_B = kill_1 \cup kill_2 \cup \dots \cup kill_n$$

$$gen_B = gen_n \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \dots \cup (gen_1 - kill_2 - kill_3 - \dots - kill_n)$$



## • 到达一定值的数据流等式

- $gen_B$ :  $B$ 中能到达 $B$ 的结束点的定值语句
- $kill_B$ : 整个程序中决不会到达 $B$ 结束点的定值
- $IN[B]$ : 能到达 $B$ 的开始点的定值集合
- $OUT[B]$ : 能到达 $B$ 的结束点的定值集合

## 两组等式 (根据 $gen$ 和 $kill$ 定义 $IN$ 和 $OUT$ )

- $IN[B] = \bigcup_{P \text{ 是 } B \text{ 的前驱}} OUT[P]$
  - $OUT[B] = gen_B \cup (IN[B] - kill_B)$
  - $OUT[ENTRY] = \emptyset$
- ## • 到达一定值方程组的迭代求解, 最终到达不动点





## // 正向数据流分析

引入两个虚拟块：ENTRY、EXIT

- (1)  $\text{OUT}[\text{ENTRY}] = \emptyset;$
- (2) for (除了ENTRY以外的每个块B)  $\text{OUT}[B] = \emptyset;$
- (3) while (任何一个OUT出现变化){
- (4)     for (除了ENTRY以外的每个块B) {
- (5)          $\text{IN}[B] = \bigcup_{P \text{ 是 } B \text{ 的前驱}} \text{OUT}[P];$
- (6)          $\text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B);$
- (7)     }}

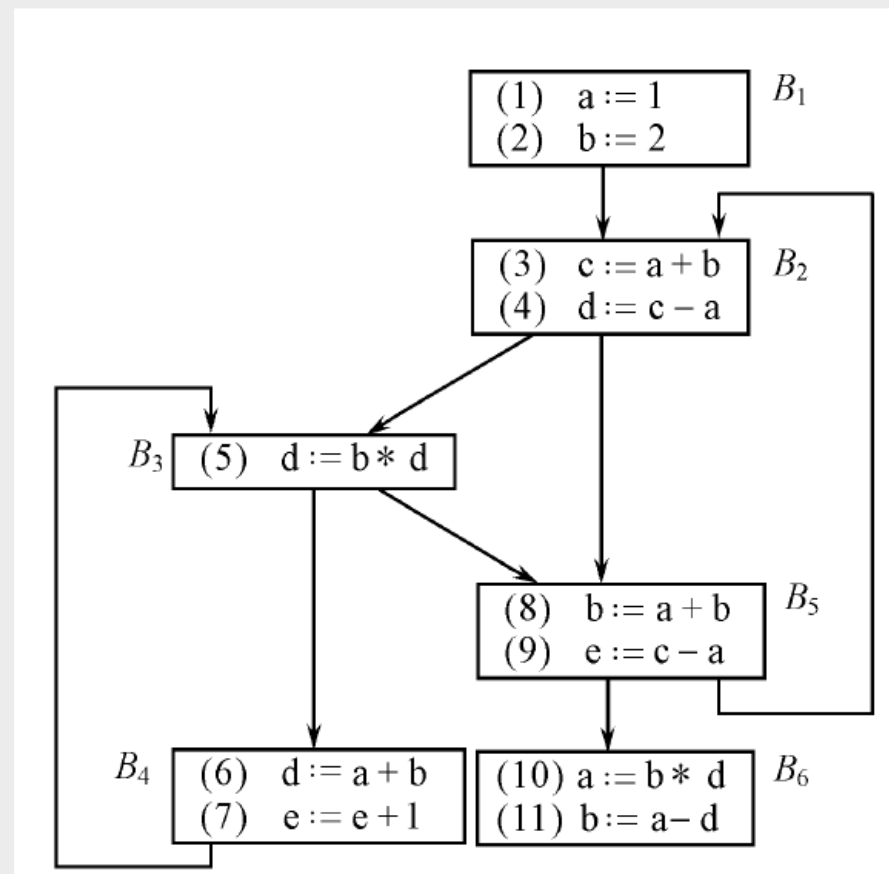
向量求解：集合并操作使用逻辑或，集合相减使用后者求补再逻辑与



# 例题13



- 9.3 a.为到达-定值分析, 计算每个块的gen,kill,IN和OUT集合
- $GEN[B1] = \{d1,d2\}$
- $KILL[B1] = \{d8,d10,d11\}$
- $GEN[B2] = \{d3,d4\}$
- $KILL[B2] = \{d5,d6\}$
- $GEN[B3] = \{d5\}$
- $KILL[B3] = \{d4,d6\}$
- $GEN[B4] = \{d6,d7\}$
- $KILL[B4] = \{d4,d5,d9\}$
- $GEN[B5] = \{d8, d9\}$
- $KILL[B5] = \{d2,d11,d7\}$
- $GEN[B6] = \{d10,d11\}$
- $KILL[B6] = \{d1,d2,d8\}$





# 例题13



- 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	初始	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\})$ $= \{d1,d2\}$		
B2	$\emptyset$				
B3	$\emptyset$				
B4	$\emptyset$				
B5	$\emptyset$				
B6	$\emptyset$				



# 例题13



- 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	初始	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\})$ $= \{d1,d2\}$		
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$		
B3	$\emptyset$				
B4	$\emptyset$				
B5	$\emptyset$				
B6	$\emptyset$				



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\})$ $= \{d1,d2\}$		
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\})$ $= \{d1,d2,d3,d4\}$		
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$		
B4	$\emptyset$				
B5	$\emptyset$				
B6	$\emptyset$				



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$		
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$		
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$		
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$		
B5	$\emptyset$				
B6	$\emptyset$				



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\})$ $= \{d1,d2\}$		
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\})$ $= \{d1,d2,d3,d4\}$		
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$		
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$		
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\}$ $= \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$		
B6	$\emptyset$				



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$		
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$		
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$		
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$		
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$		
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$		





# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$		
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$		
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$		
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$		



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d6,d7\} = \{d1,d2,d3,d4,d6,d7,d8,d9\}$	$\{d5\} + (\{d1,d2,d3,d4,d6,d7,d8,d9\} - \{d4,d6\}) = \{d1,d2,d3,d5,d7,d8,d9\}$
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$		
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$		
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$		



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d6,d7\} = \{d1,d2,d3,d4,d6,d7,d8,d9\}$	$\{d5\} + (\{d1,d2,d3,d4,d6,d7,d8,d9\} - \{d4,d6\}) = \{d1,d2,d3,d5,d7,d8,d9\}$
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$	$\{d1,d2,d3,d5,d7,d8,d9\}$	$\{d6,d7\} + \{d1,d2,d3,d5,d7,d8,d9\} - \{d4,d5,d9\} = \{d1,d2,d3,d6,d7,d8\}$
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$		
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$		



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d6,d7\} = \{d1,d2,d3,d4,d6,d7,d8,d9\}$	$\{d5\} + (\{d1,d2,d3,d4,d6,d7,d8,d9\} - \{d4,d6\}) = \{d1,d2,d3,d5,d7,d8,d9\}$
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$	$\{d1,d2,d3,d5,d7,d8,d9\}$	$\{d6,d7\} + \{d1,d2,d3,d5,d7,d8,d9\} - \{d4,d5,d9\} = \{d1,d2,d3,d6,d7,d8\}$
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d5,d7,d8,d9\} = \{d1,d2,d3,d4,d5,d6,d7,d8,d9\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5,d6,d7,d8,d9\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d6,d8,d9\}$
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$		



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d3,d5\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d6,d7\} = \{d1,d2,d3,d4,d6,d7,d8,d9\}$	$\{d5\} + (\{d1,d2,d3,d4,d6,d7,d8,d9\} - \{d4,d6\}) = \{d1,d2,d3,d5,d7,d8,d9\}$
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$	$\{d1,d2,d3,d5,d7,d8,d9\}$	$\{d6,d7\} + \{d1,d2,d3,d5,d7,d8,d9\} - \{d4,d5,d9\} = \{d1,d2,d3,d6,d7,d8\}$
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d5,d7,d8,d9\} = \{d1,d2,d3,d4,d5,d6,d7,d8,d9\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5,d6,d7,d8,d9\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d6,d8,d9\}$
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$	$\{d1,d3,d4,d5,d6,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d6,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d6,d9,d10,d11\}$



# 例题13



## • 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	$\emptyset$	$\emptyset$	$\{d1,d2\} \cup (\emptyset - \{d8,d10,d11\}) = \{d1,d2\}$	$\emptyset$	$\{d1,d2\}$
B2	$\emptyset$	$\{d1,d2\}$	$\{d3,d4\} + (\{d1,d2\} - \{d5,d6\}) = \{d1,d2,d3,d4\}$	$\{d1,d2\} \cup \{d1,d3,d4,d5,d8,d9\} = \{d1,d2,d3,d4,d5,d8,d9\}$	$\{d3,d4\} + (\{d1,d2,d3,d4,d5,d8,d9\} - \{d5,d6\}) = \{d1,d2,d3,d4,d6,d8,d9\}$
B3	$\emptyset$	$\{d1,d2,d3,d4\}$	$\{d5\} + (\{d1,d2,d3,d4\} - \{d4,d6\}) = \{d1,d2,d5\}$	$\{d1,d2,d3,d4,d6,d8,d9\}$	$\{d5\} + (\{d1,d2,d3,d4,d6,d7,d8,d9\} - \{d4,d6\}) = \{d1,d2,d3,d5,d7,d8,d9\}$
B4	$\emptyset$	$\{d1,d2,d3,d5\}$	$\{d6,d7\} + (\{d1,d2,d3,d5\} - \{d4,d5,d9\}) = \{d1,d2,d3,d6,d7\}$	$\{d1,d2,d3,d5,d7,d8,d9\}$	$\{d6,d7\} + \{d1,d2,d3,d5,d7,d8,d9\} - \{d4,d5,d9\} = \{d1,d2,d3,d6,d7,d8\}$
B5	$\emptyset$	$\{d1,d2,d3,d4\} \cup \{d1,d2,d3,d5\} = \{d1,d2,d3,d4,d5\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d8,d9\}$	$\{d1,d2,d3,d4,d6,d8,d9\} \cup \{d1,d2,d3,d5,d7,d8,d9\} = \{d1,d2,d3,d4,d5,d6,d7,d8,d9\}$	$\{d8,d9\} + (\{d1,d2,d3,d4,d5,d6,d7,d8,d9\} - \{d2,d11,d7\}) = \{d1,d3,d4,d5,d6,d8,d9\}$
B6	$\emptyset$	$\{d1,d3,d4,d5,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d9,d10,d11\}$	$\{d1,d3,d4,d5,d6,d8,d9\}$	$\{d10,d11\} + (\{d1,d3,d4,d5,d6,d8,d9\} - \{d1,d2,d8\}) = \{d3,d4,d5,d6,d9,d10,d11\}$

继续迭代直到out没有变化，此处略去，  
由于时间关系，有可能计算有误，做一个  
免责声明。



# 重要知识点复习



## • 代码生成

- 寄存器分配及汇编代码生成
- 指令调度和代价计算



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

R1	R2	R3	a	b	c	d	t	u	v
			a	b	c	d			





# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R1, a**

**LD R2, b**

**SUB R2, R1, R2**

R1	R2	R3	a	b	c	d	t	u	v
			a	b	c	d			



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R1, a**

**LD R2, b**

**SUB R2, R1, R2**

R1	R2	R3	a	b	c	d	t	u	v
a	b		a, R1	b, R2	c	d			



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R1, a**

**LD R2, b**

**SUB R2, R1, R2**

R1	R2	R3	a	b	c	d	t	u	v
a	t		a, R1	b	c	d	R2		



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R3, c**

**SUB R1, R1, R3**

R1	R2	R3	a	b	c	d	t	u	v
a	t		a, R1	b	c	d	R2		



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R3, c**

**SUB R1, R1, R3**

R1	R2	R3	a	b	c	d	t	u	v
a	t	c	a, R1	b	c, R3	d	R2		



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R3, c**

**SUB R1, R1, R3**

R1	R2	R3	a	b	c	d	t	u	v
u	t	c	a	b	c, R3	d	R2	R1	



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**ADD R3, R2, R1**

R1	R2	R3	a	b	c	d	t	u	v
u	t	c	a	b	c, R3	d	R2	R1	



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**ADD R3, R2, R1**

R1	R2	R3	a	b	c	d	t	u	v
u	t	v	a	b	c	d	R2	R1	R3





# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R2, d**

R1	R2	R3	a	b	c	d	t	u	v
u	t	v	a	b	c	d	R2	R1	R3



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**LD R2, d**

R1	R2	R3	a	b	c	d	t	u	v
u	a, d	v	R2	b	c	d, R2		R1	R3



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**ADD R1, R3, R1**

R1	R2	R3	a	b	c	d	t	u	v
u	a, d	v	R2	b	c	d, R2		R1	R3



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

**exit**

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**ST a, R2**

**ST d, R1**

R1	R2	R3	a	b	c	d	t	u	v
d	a	v	R2	b	c	R1			R3



# 寄存器分配选择——举例



基本块三地址代码如下：

$t = a - b$

$u = a - c$

$v = t + u$

$a = d$

$d = v + u$

**exit**

$t$ 、 $u$ 、 $v$ 为临时变量

$a$ 、 $b$ 、 $c$ 、 $d$ 在出口处活跃

**ST a, R2**

**ST d, R1**

R1	R2	R3	a	b	c	d	t	u	v
d	a	v	a, R2	b	c	d, R1			R3



# 重要知识点复习



## • 运行时

- 栈上内存分配
- 活动记录管理



# 全局栈式存储分配



有C程序如下:

```
void g() { int a ; a = 10 ; }
```

```
void h() { int a ; a = 100; g(); }
```

```
main()
```

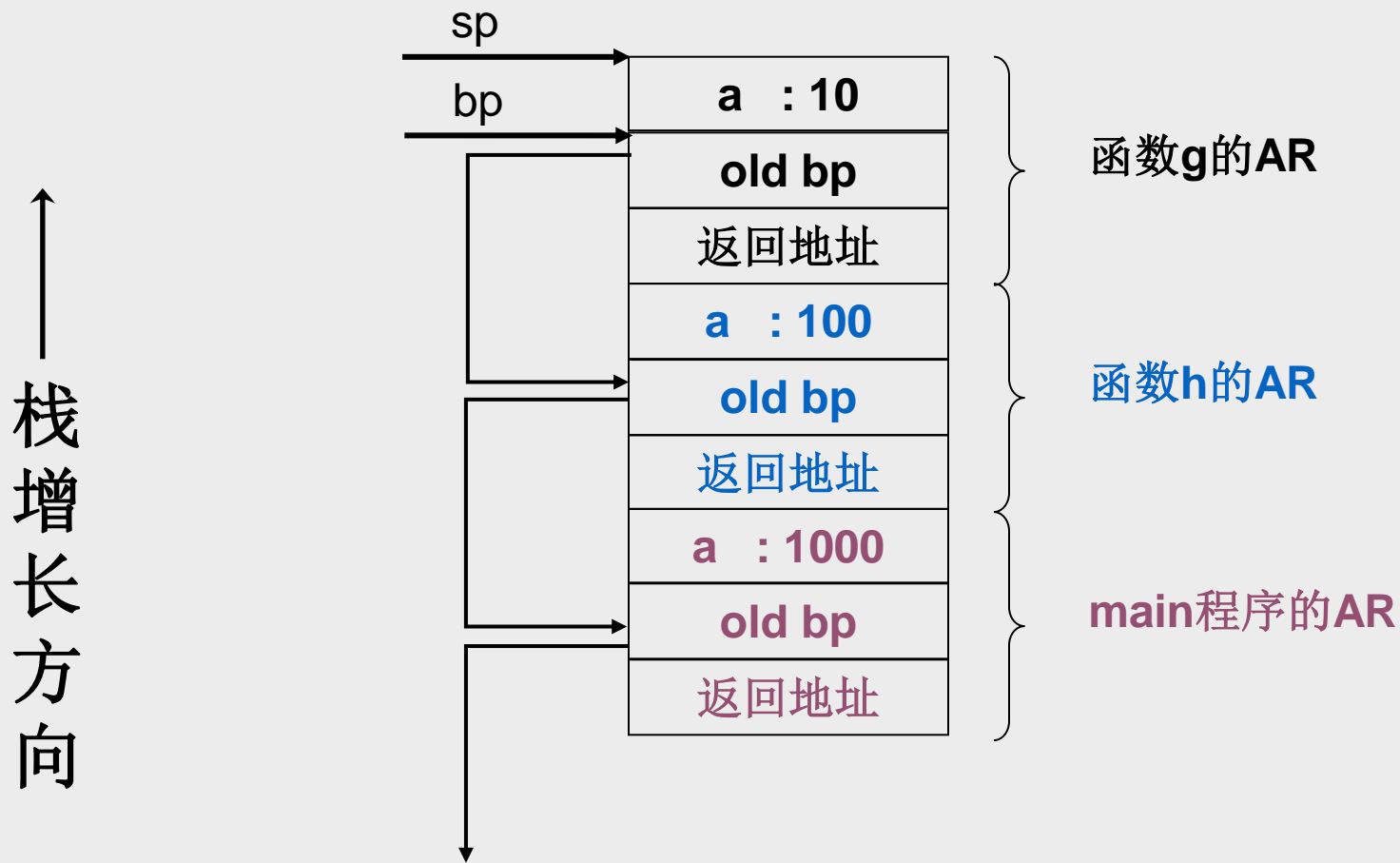
```
{ int a = 1000; h(); }
```



# 全局栈式存储分配



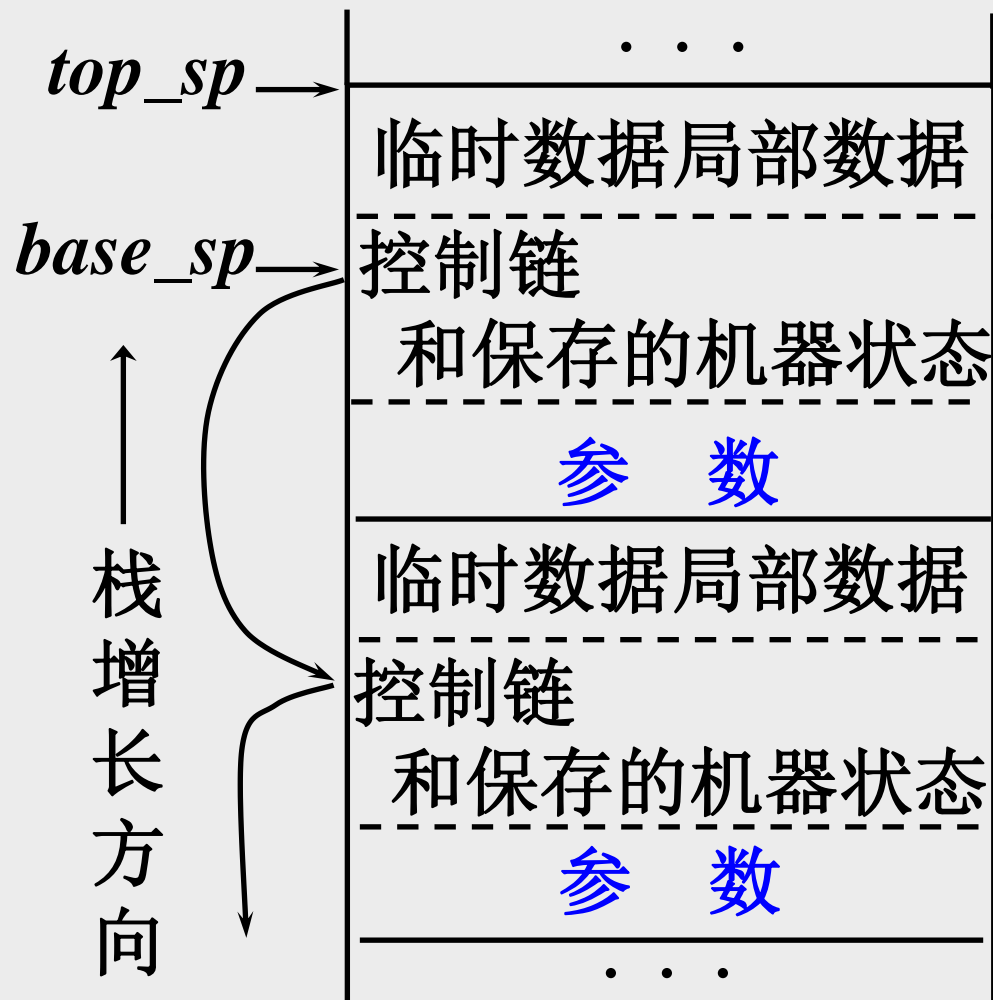
- 过程g被调用时，活动记录栈的（大致）内容







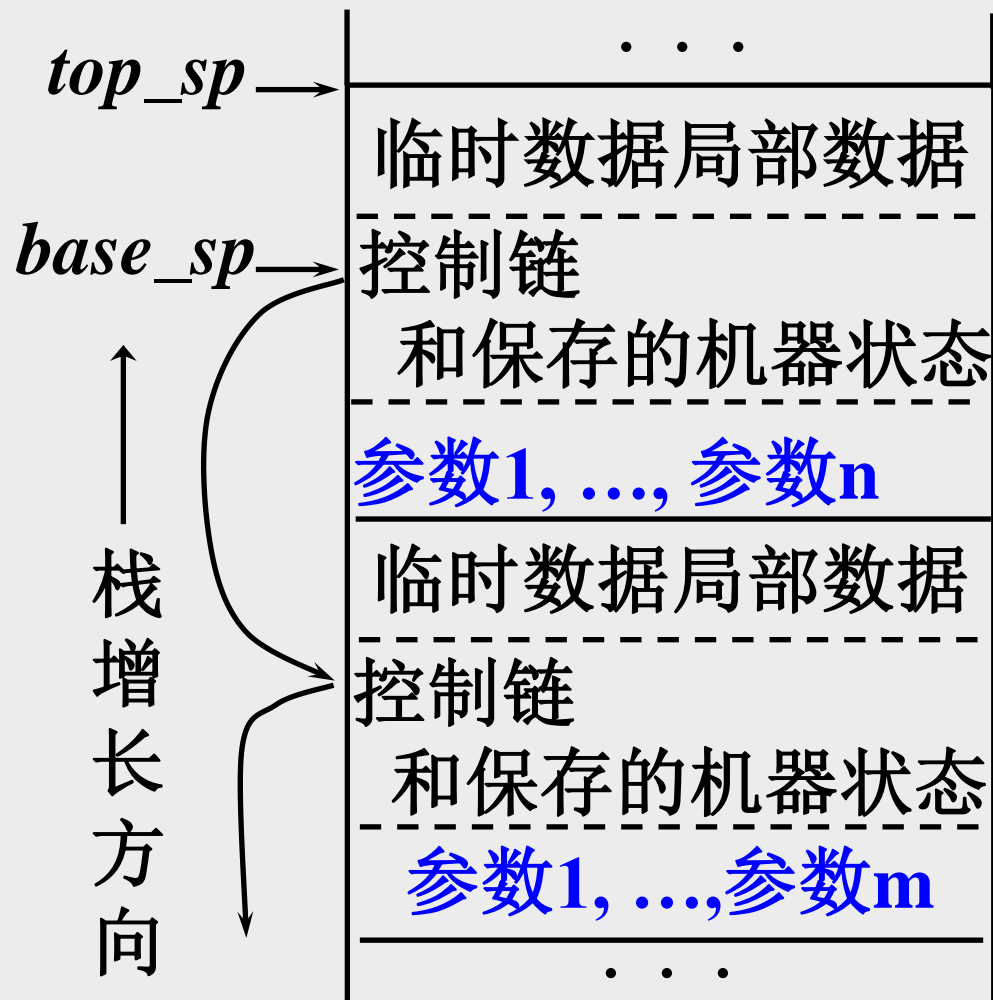
## • 过程的参数个数可变的情况



(1) 函数返回值改成  
用寄存器传递



## • 过程的参数个数可变的情况

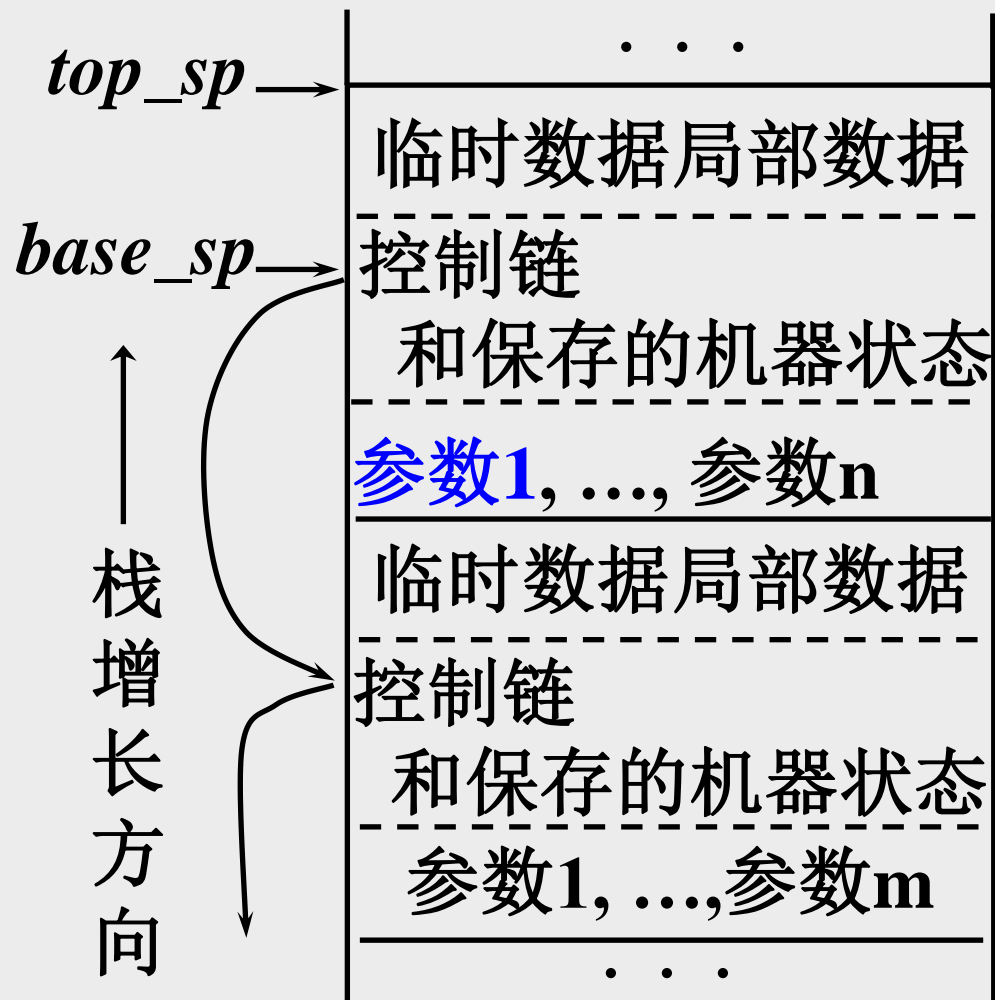


(2) 编译器产生将实参表达式逆序计算并将结果进栈的代码

自上而下依次是参数1, ..., 参数n



## • 过程的参数个数可变的情况

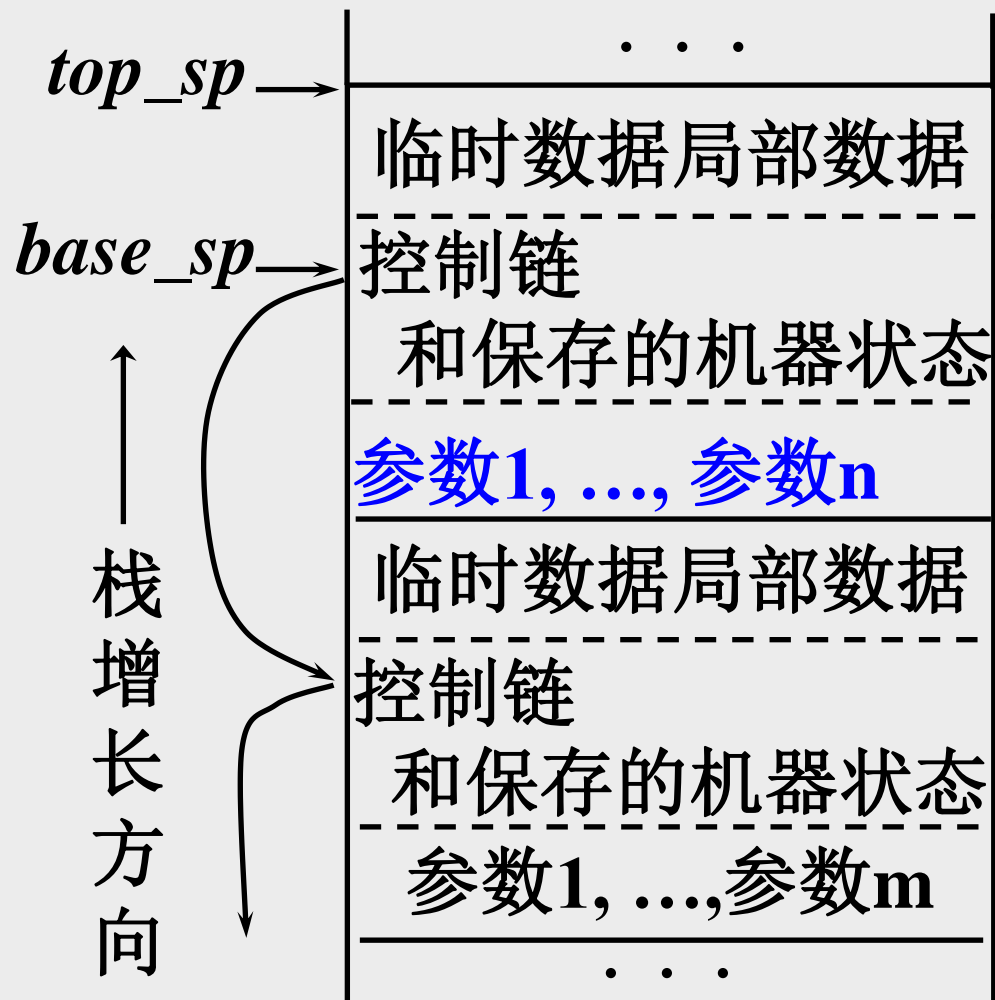


(3) 被调用函数能准确地知道第一个参数的位置

But why?



## • 过程的参数个数可变的情况



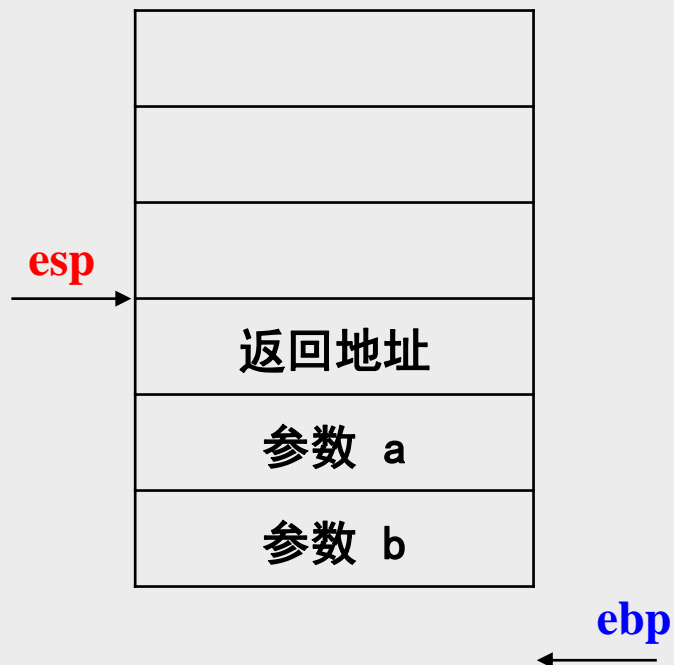
(4) 被调用函数根据第一个参数到栈中取第二、第三个参数等等



# 有参函数的活动记录



```
void func( int a , int b )  
{  
    int c , d;  
    c = a;  
    d = b;  
}
```



```
.file "ar.c"  
.text  
.globl func  
.type func,@function  
func:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $8, %esp  
    movl 8(%ebp), %eax  
    movl %eax, -4(%ebp)  
    movl 12(%ebp), %eax  
    movl %eax, -8(%ebp)  
    leave  
    ret
```



# 有参函数的活动记录



```
void func( int a , int b )  
{  
    int c , d;  
    c = a;  
    d = b;  
}
```



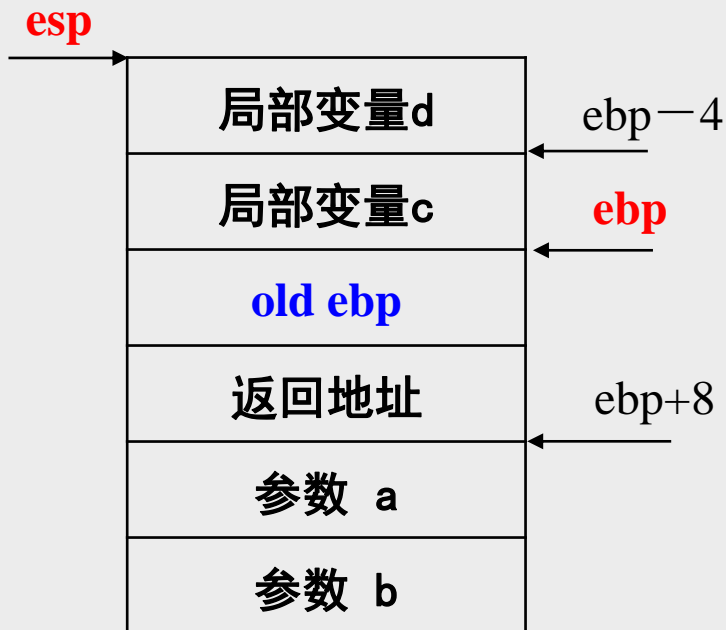
```
.file "ar.c"  
.text  
.globl func  
.type func,@function  
func:  
    pushl %ebp //老基地址压栈  
    movl %esp, %ebp //基地址指针=栈顶指针  
    subl $8, %esp  
    movl 8(%ebp), %eax  
    movl %eax, -4(%ebp)  
    movl 12(%ebp), %eax  
    movl %eax, -8(%ebp)  
    leave  
    ret
```



# 有参函数的活动记录



```
void func( int a , int b )  
{  
    int c , d;  
    c = a;  
    d = b;  
}
```



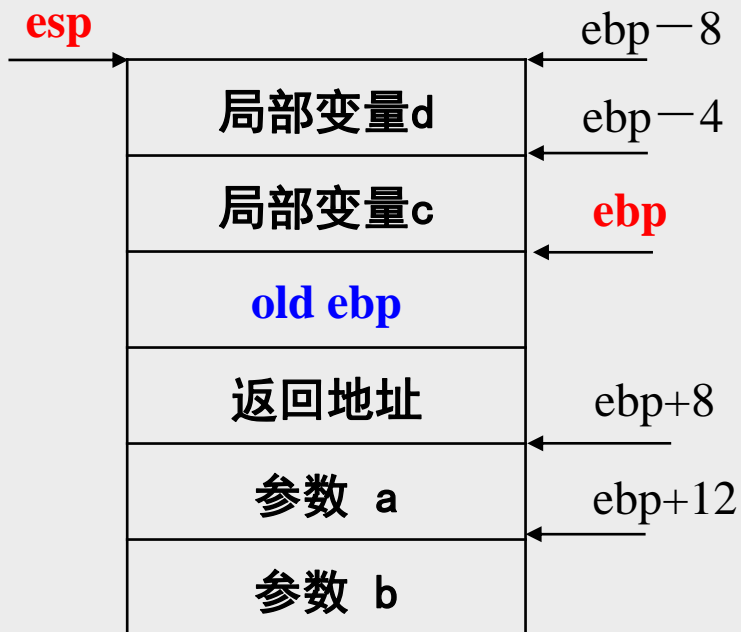
```
.file "ar.c"  
.text  
.globl func  
.type func,@function  
func:  
    pushl %ebp //老基地址压栈  
    movl %esp, %ebp //基地址指针=栈顶指针  
    subl $8, %esp //分配c,d局部变量空间  
    movl 8(%ebp), %eax //将a值放进寄存器  
    movl %eax, -4(%ebp) //将a值赋给c  
    movl 12(%ebp), %eax  
    movl %eax, -8(%ebp)  
    leave  
    ret
```



# 有参函数的活动记录



```
void func( int a , int b )  
{  
    int c , d;  
    c = a;  
    d = b;  
}
```



```
.file "ar.c"  
.text  
.globl func  
.type func,@function  
func:  
    pushl %ebp //老基地址压栈  
    movl %esp, %ebp //基地址指针=栈顶指针  
    subl $8, %esp //分配c,d局部变量空间  
    movl 8(%ebp), %eax //将a值放进寄存器  
    movl %eax, -4(%ebp) //将a值赋给c  
    movl 12(%ebp), %eax //将b值放进寄存器  
    movl %eax, -8(%ebp) //将b值赋给d  
    leave  
    ret
```

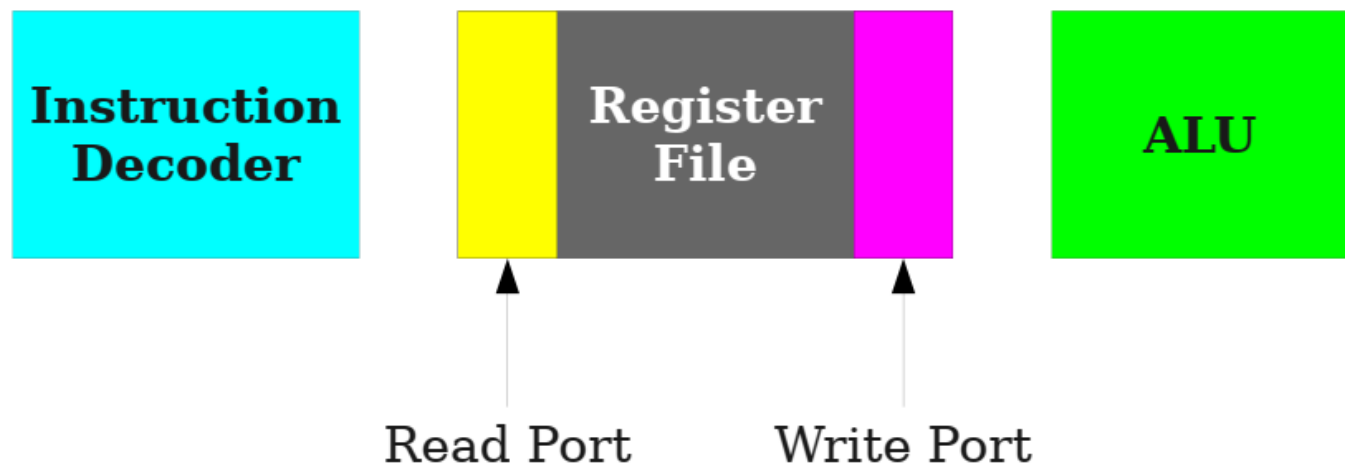




## • 指令调度

- 流水线并行的例子
- 指令调度与数据依赖分析
- 数据依赖指导下的指令调度

# 复杂的流水线并行



**add \$t2, \$t0, \$t1      # \$t2 = \$t0 + \$t1**

**add \$t4, \$t3, \$t2      # \$t4 = \$t3 + \$t2**

**add \$t7, \$t5, \$t6      # \$t7 = \$t5 + \$t6**

**add \$t0, \$t0, \$t7      # \$t0 = \$t0 + \$t7**

[illegible]

## Instruction Decoder

Reg  
F

## Read Port

## Write Port

数据未准备好

**add \$t2, \$t0, \$t1      # \$t2 = \$t0 + \$t1**

add \$t4, \$t3, \$t2      # \$t4 = \$t3 + \$t2

**add \$t7, \$t5, \$t6      # \$t7 = \$t5 + \$t6**

add \$t0, \$t0, \$t7      # \$t0 = \$t0 + \$t7

[illegible]



ID	RR	ALU	RW



ID	RR	ALU	RW

[illegible]



**+ \$t1**

$+ \$t2$

add \$t4, \$t3, \$t1

**add \$t7, \$t5, \$t6      # \$t7 = \$t5 + \$t6**

**add \$t0, \$t0, \$t7      # \$t0 = \$t0 + \$t7**

[illegible]



[illegible]

[illegible]

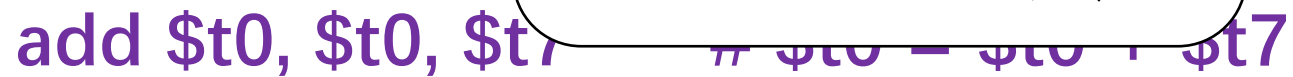
[illegible]

[illegible]

[illegible]



ID	RR	ALU	RW



# 节省两个 时钟周期

[illegible]

# 分析数据依赖关系



$$t_0 = t_1 + t_2$$

$$t_1 = t_0 + t_1$$

$$t_3 = t_2 + t_4$$

$$t_0 = t_1 + t_2$$

$$t_5 = t_3 + t_4$$

$$t_6 = t_2 + t_7$$



# 分析数据依赖关系



$$t_0 = t_1 + t_2$$

$$t_1 = t_0 + t_1$$

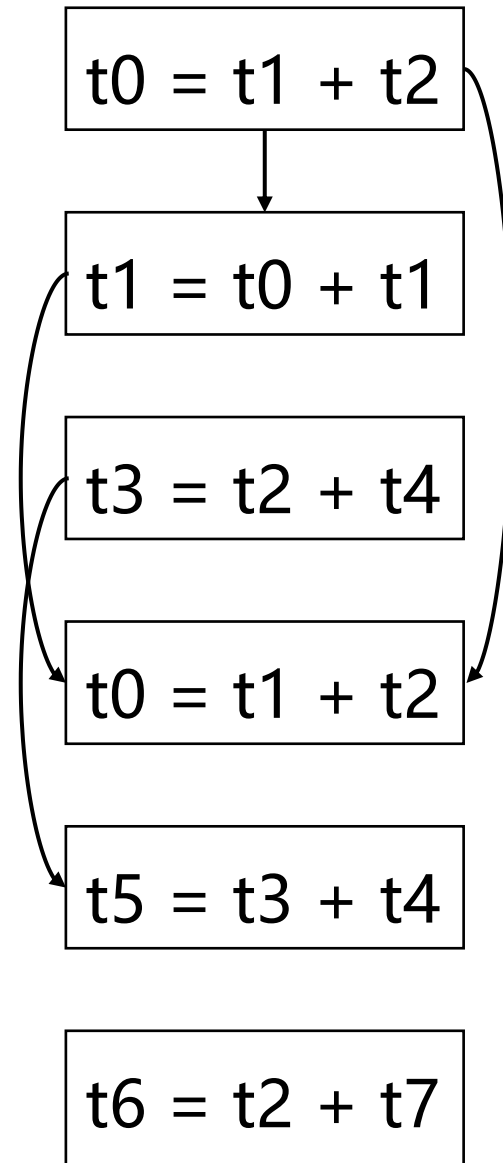
$$t_3 = t_2 + t_4$$

$$t_0 = t_1 + t_2$$

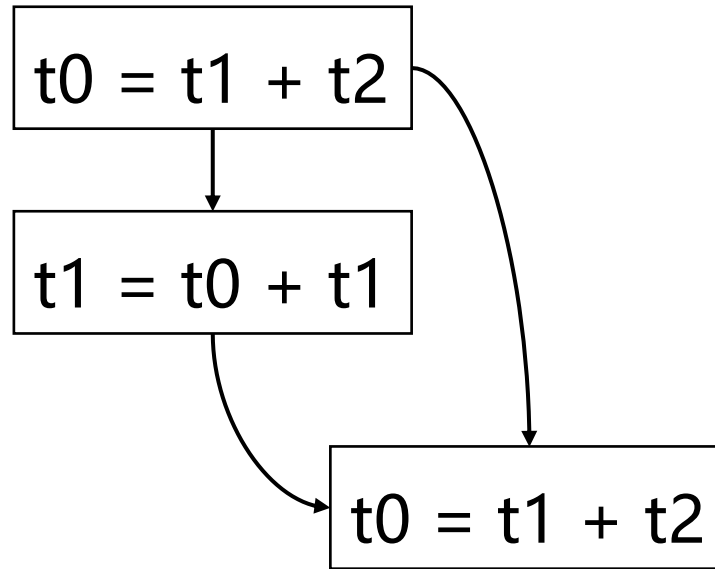
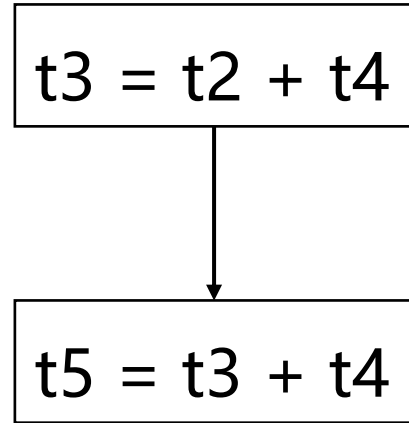
$$t_5 = t_3 + t_4$$

$$t_6 = t_2 + t_7$$

# 分析数据依赖关系



# 分析数据依赖关系



$t6 = t2 + t7$

# 数据依赖指导下的指令调度



$$t3 = t2 + t4$$



$$t5 = t3 + t4$$

$$t0 = t1 + t2$$



$$t1 = t0 + t1$$



$$t0 = t1 + t2$$

$$t6 = t2 + t7$$

$t3 = t2 + t4$
$t5 = t3 + t4$
$t0 = t1 + t2$
$t1 = t0 + t1$
$t0 = t1 + t2$
$t6 = t2 + t7$

$t0 = t1 + t2$
$t3 = t2 + t4$
$t6 = t2 + t7$
$t1 = t0 + t1$
$t5 = t3 + t4$
$t0 = t1 + t2$

# 升级版的指令调度



0	$t3 = t2 + t4$
---	----------------

+3

0	$t5 = t3 + t4$
---	----------------

0	$t6 = t2 + t7$
---	----------------

$t0 = t1 + t2$	0
----------------	---

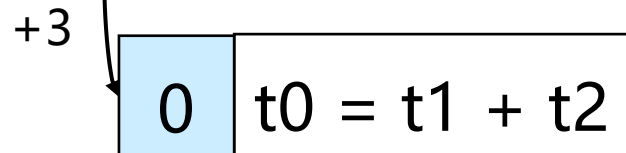
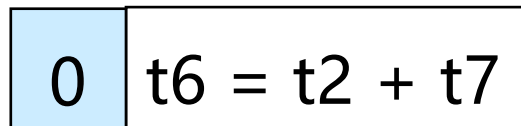
+3

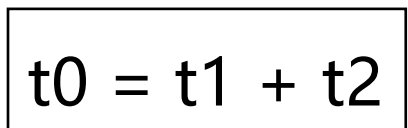
$t1 = t0 + t1$	0
----------------	---

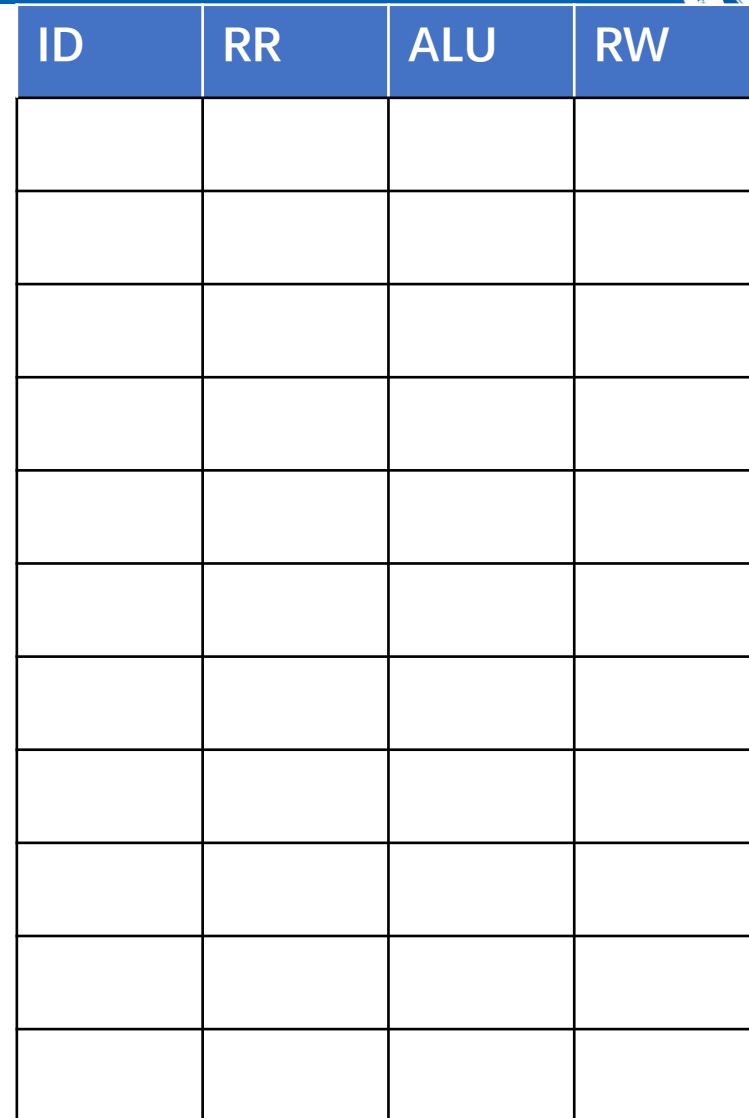
+3

0	$t0 = t1 + t2$
---	----------------

+3

[illegible]

[illegible]





# 升级版的指令调度



0

$t3 = t2 + t4$

+3

0

$t5 = t3 + t4$

0

$t6 = t2 + t7$

$t0 = t1 + t2$

$t1 = t0 + t1$

3

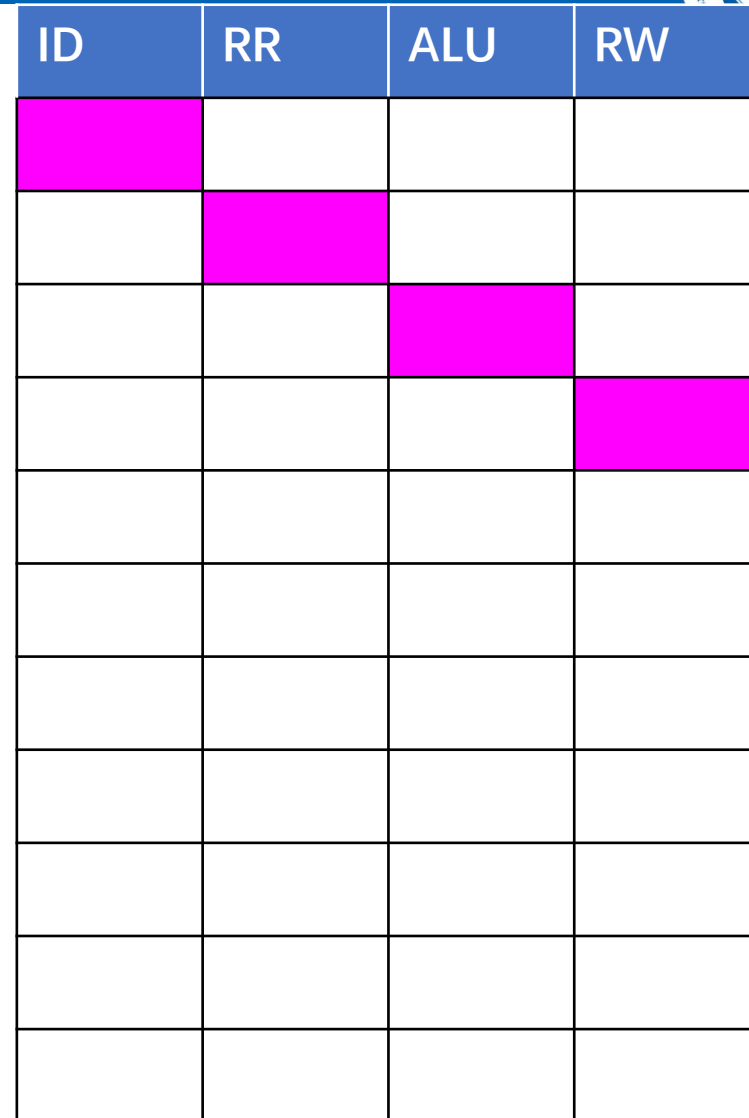
+3

3

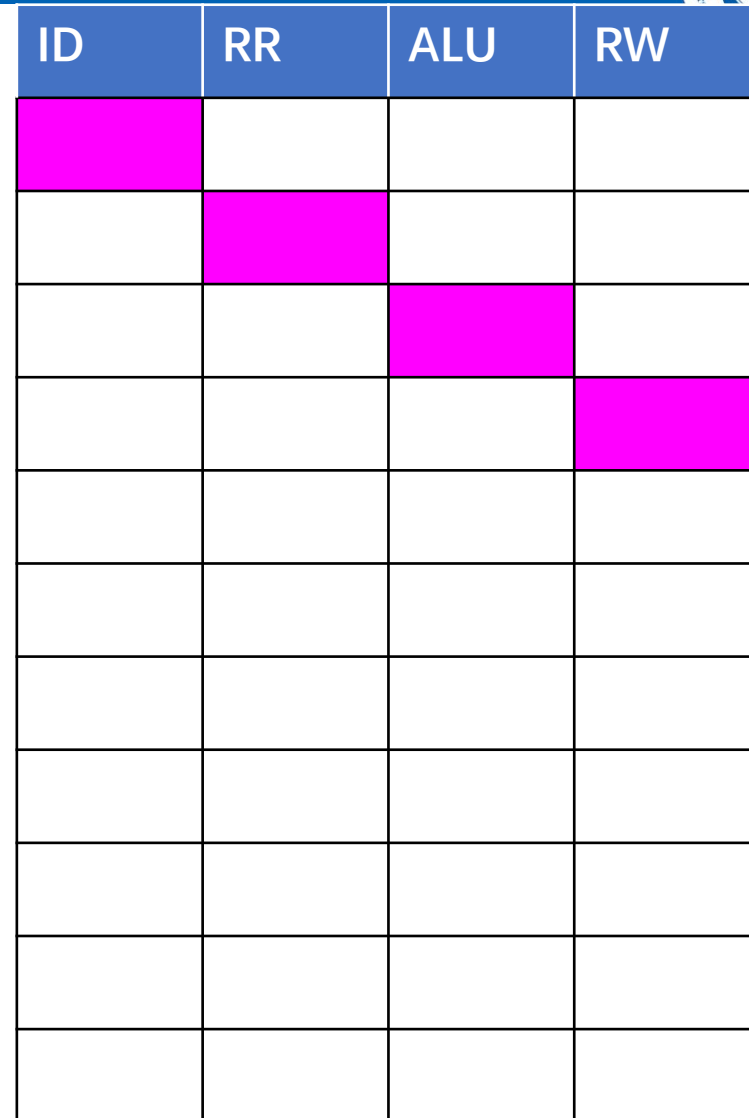
$t0 = t1 + t2$

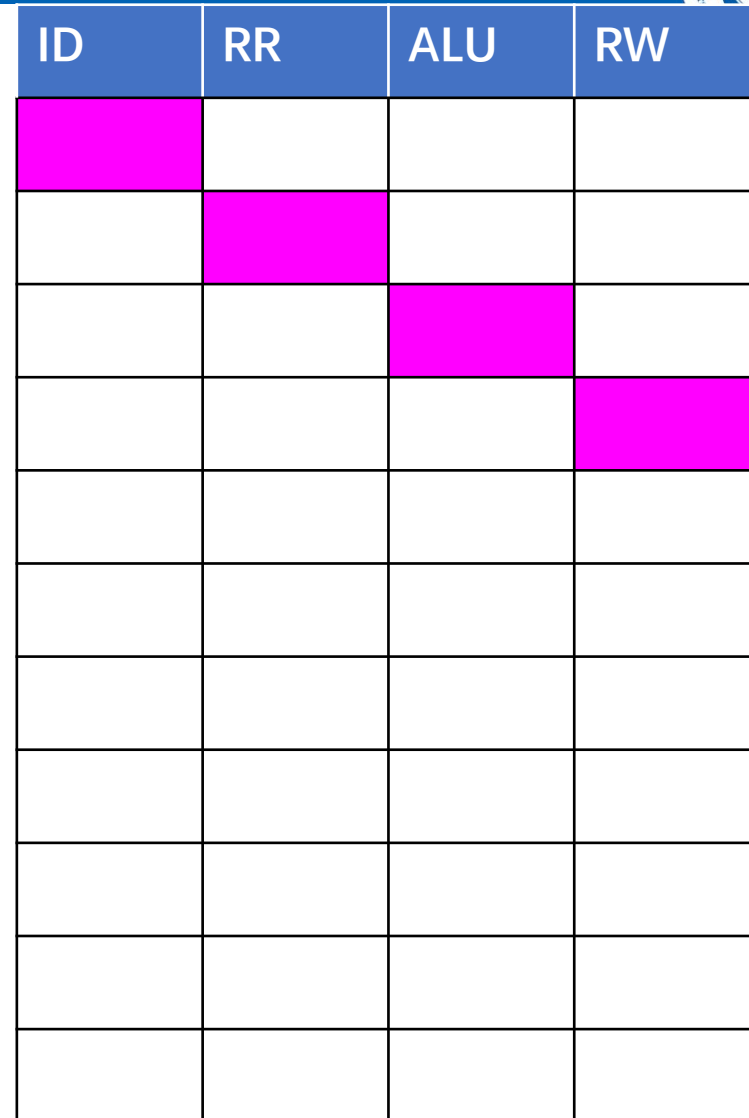
ID	RR	ALU	RW

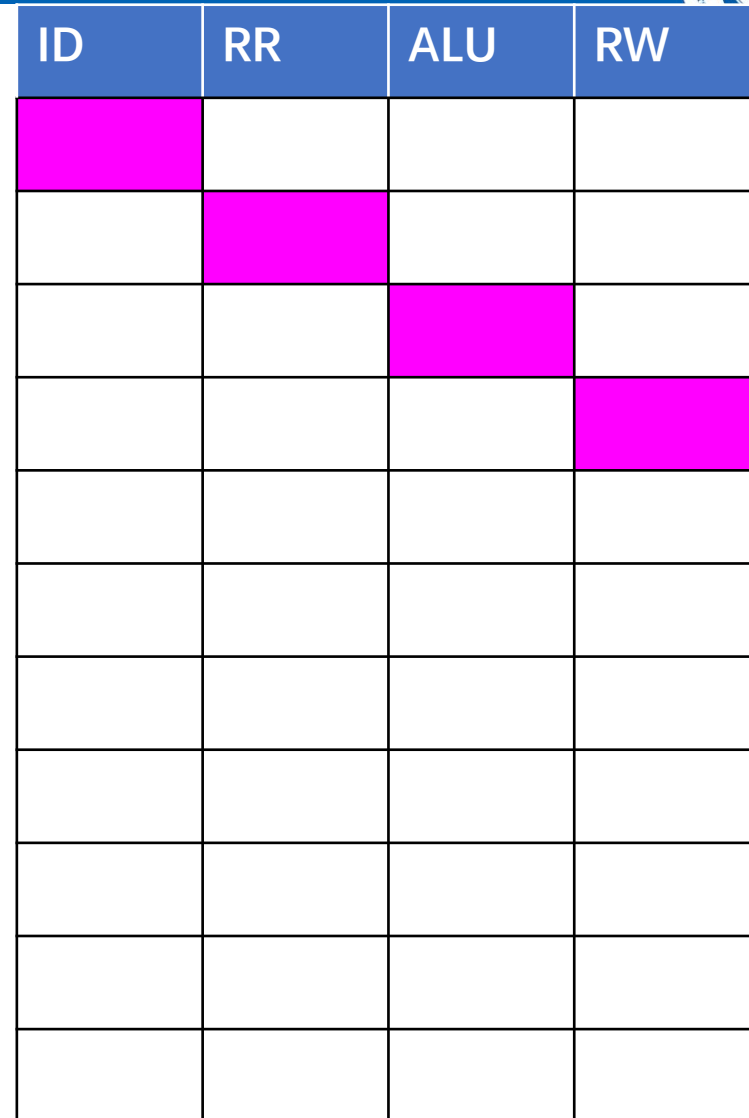
$t0 = t1 + t2$

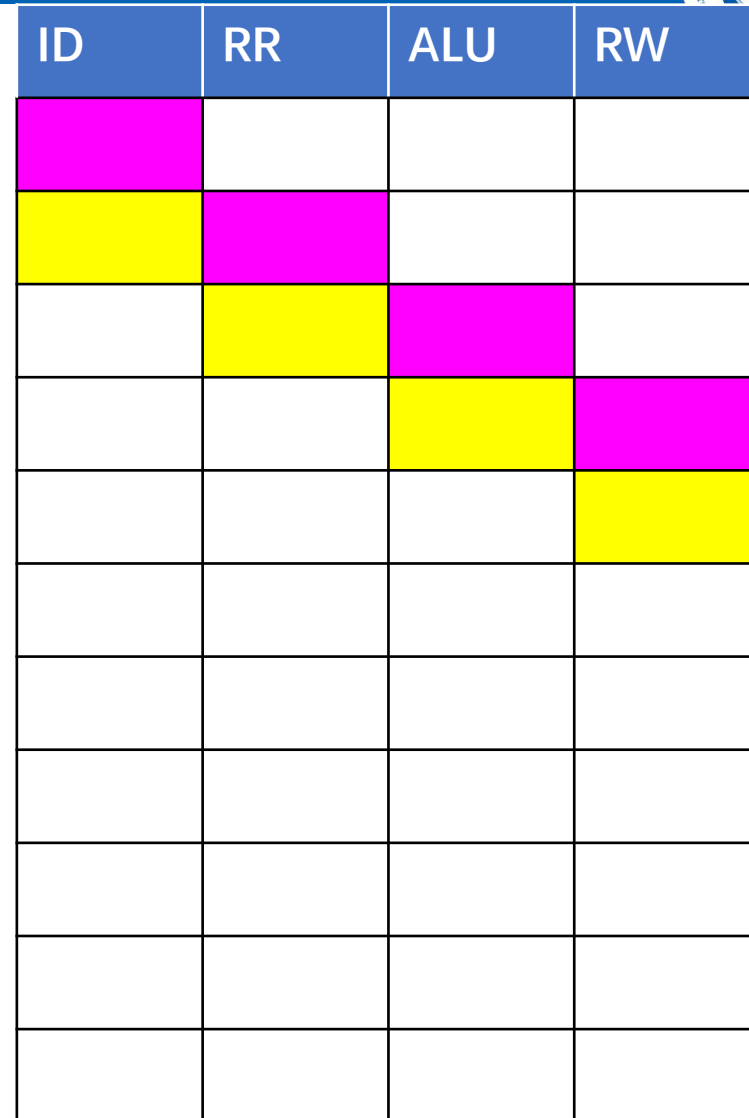


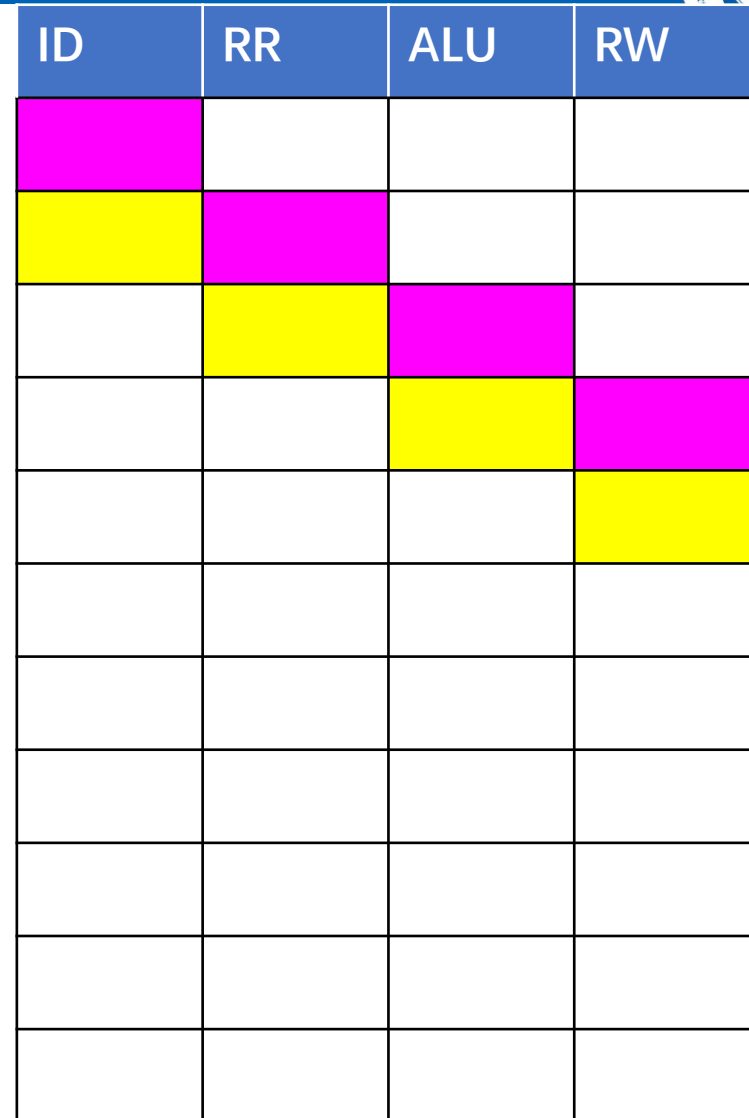
$$t_0 = t_1 + t_2$$

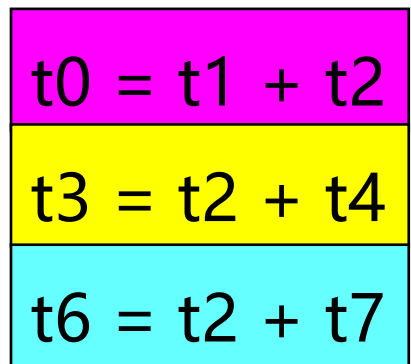

$$t_3 = t_2 + t_4$$


$$t_3 = t_2 + t_4$$

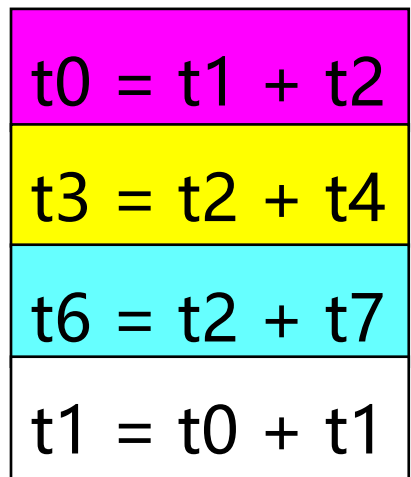


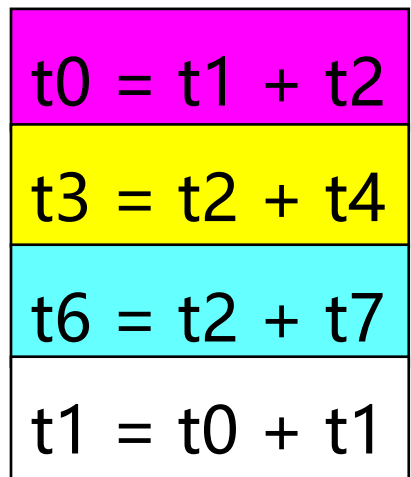


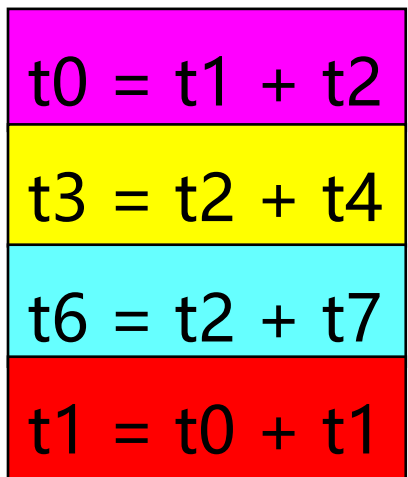
[illegible]

[illegible]



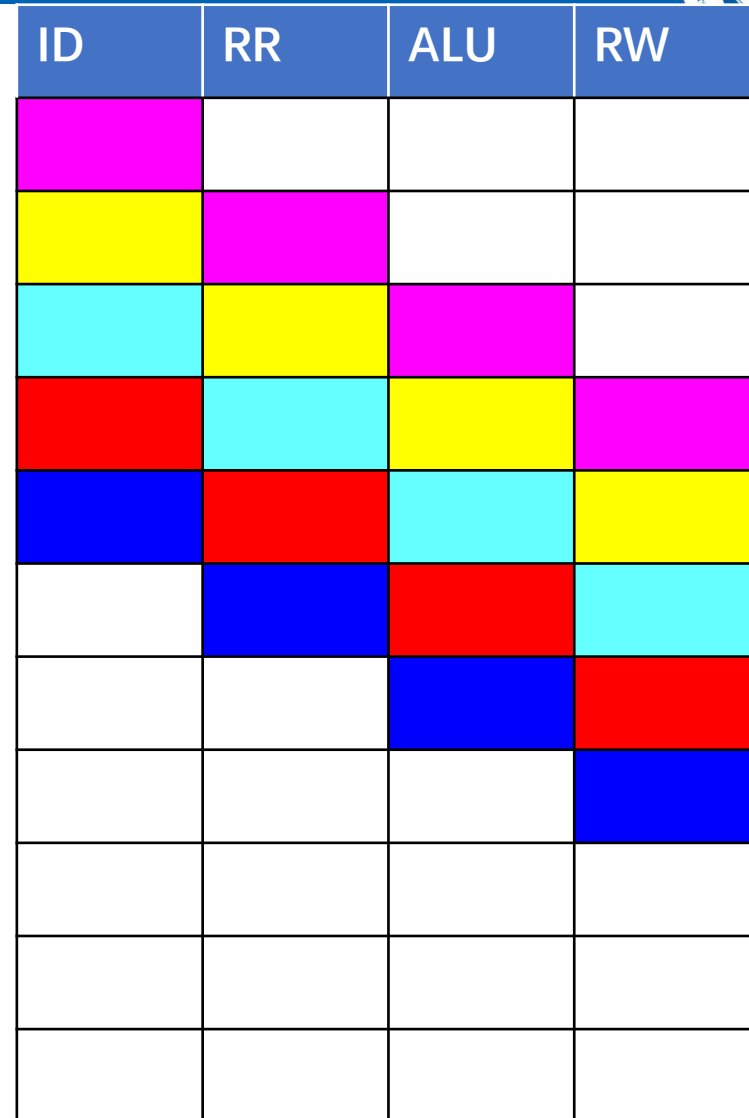
[illegible]

[illegible]

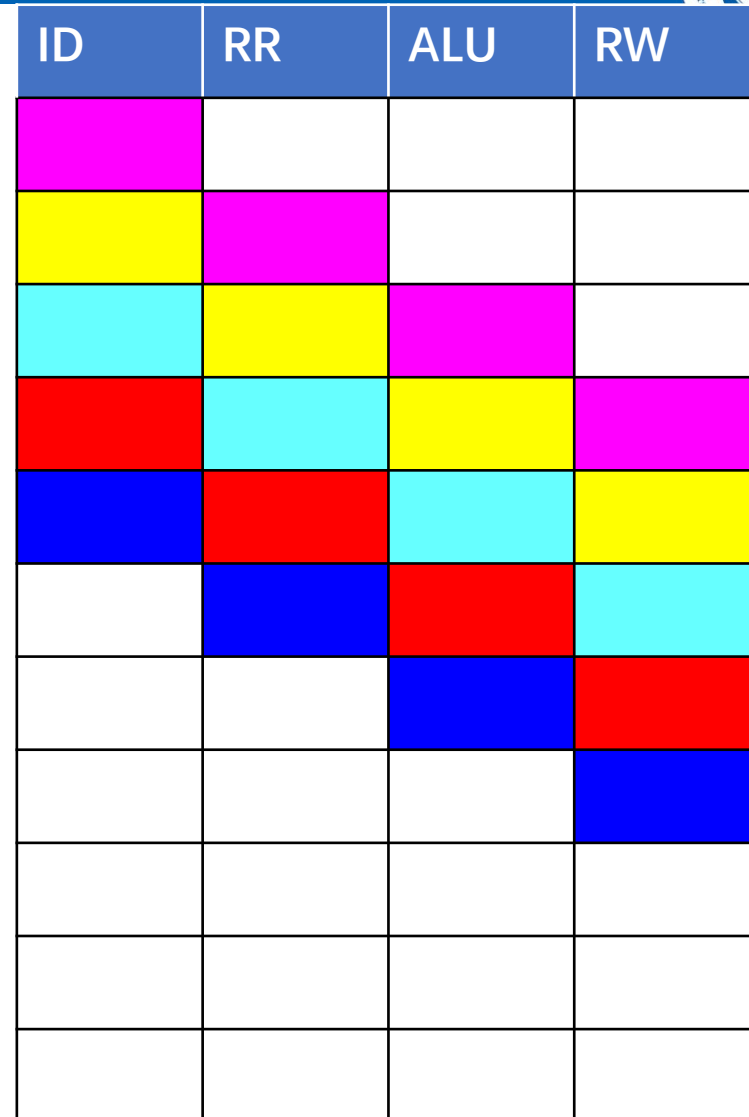
[illegible]



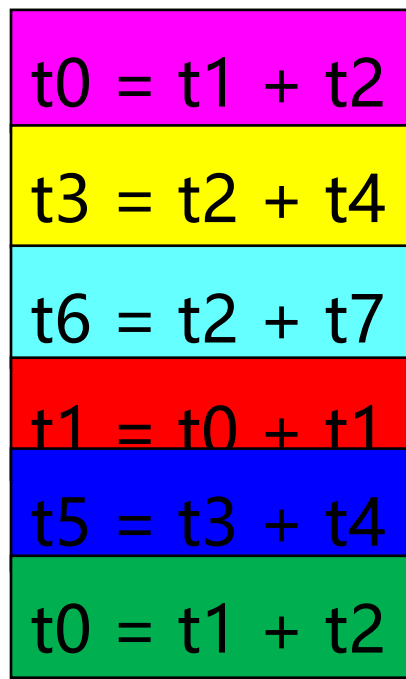
$t_0 = t_1 + t_2$
$t_3 = t_2 + t_4$
$t_6 = t_2 + t_7$
$t_1 = t_0 + t_1$
$t_5 = t_3 + t_4$



$t_0 = t_1 + t_2$
$t_3 = t_2 + t_4$
$t_6 = t_2 + t_7$
$t_1 = t_0 + t_1$
$t_5 = t_3 + t_4$



$t_0 = t_1 + t_2$
$t_3 = t_2 + t_4$
$t_6 = t_2 + t_7$
$t_1 = t_0 + t_1$
$t_5 = t_3 + t_4$
$t_0 = t_1 + t_2$

[illegible]

ID	RR	ALU	RW

ID	RR	ALU	RW



## ■参考Lab1-Lab4的实验内容

- 理解整个框架和全链条实验的逻辑
- 会阅读LightIR和LLVM IR代码
- 熟悉后端代码生成

# 致谢



- ❑ 感谢所有选课同学的信任和支持！
- ❑ 感谢教学实验室徐伟老师及各位助教的辛勤努力！
- ❑ 感谢学校学院教务相关老师们的支持！

□ **亦余心之所善兮，虽九死其犹未悔**

■ 信仰的力量

□ **操千曲而后晓声，观千剑而后识器**

■ 时间的力量

□ **牢骚太盛防肠断，风物长宜放眼量**

■ 格局的力量



# 一起努力 打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2023年12月13日