

Lecture 9

静态单赋值

徐辉

xuh@fudan.edu.cn

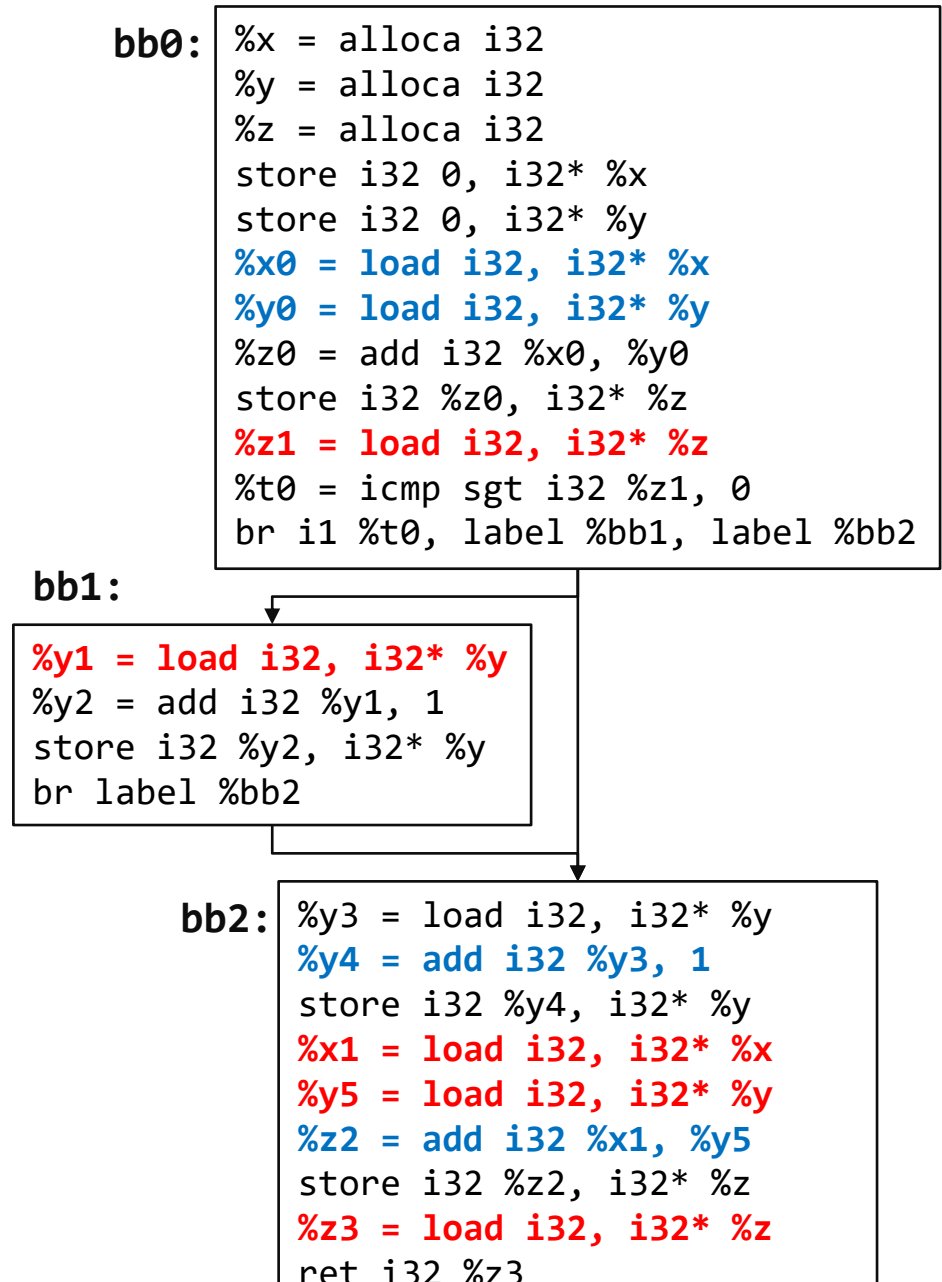
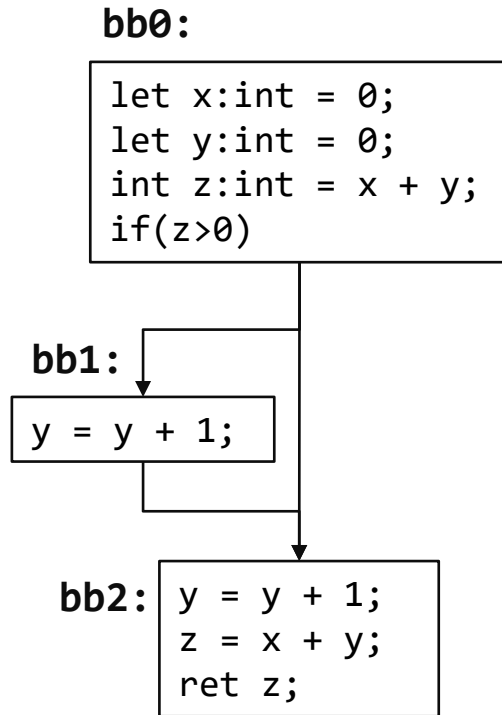


大纲

- 一、优化冗余Load指令
- 二、优化冗余Store指令
- 三、纯寄存器表示
- 四、Phi指令优化

一、优化Load

线性IR中的Load冗余



优化思路: 可用寄存器分析

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%z1 = load i32, i32* %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y1 = load i32, i32* %y
%y2 = add i32 %y1, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%x1 = load i32, i32* %x
%y5 = load i32, i32* %y
%z2 = add i32 %x1, %y5
store i32 %z2, i32* %z
%z3 = load i32, i32* %z
ret i32 %z3
```

- 正向遍历控制流图
- Transfer函数定义:
 - $\%t = \text{load } i32, i32* \%x$
 - $S_x = S_x \cup \{t\}$
 - $\%t = \text{bop } \%t1, \%t2$
 - $S_x = S_x \cup \{t\}, \text{ s.t. } t \in *x$
 - $\text{store } i32 \%t, i32* \%x$
 - $S_x = \{t\}$
- 遇到合并节点

$$IN(n) = \bigcap_{n' \in \text{predecessor}(n)} OUT(n')$$

分析过程

bb0:

```

%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%z1 = load i32, i32* %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
    
```

bb1:

```

%y1 = load i32, i32* %y
%y2 = add i32 %y1, 1
store i32 %y2, i32* %y
br label %bb2
    
```

bb2:

```

%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%x1 = load i32, i32* %x
%y5 = load i32, i32* %y
%z2 = add i32 %x1, %y5
store i32 %z2, i32* %z
%z3 = load i32, i32* %z
ret i32 %z3
    
```

S_x

S_y

S_z

{x0}

{}

{}

{x0}

{y0}

{}

{x0}

{y0}

{z0}

{x0}

{y0}

{z0}

{x0}

{y0}

{z0, z1}

{x0}

{y0, y1}

{z0, z1}

{x0}

{y0, y1, y2}

{z0, z1}

{x0}

{y2}

{z0, z1}

{x0}n{x0}

{y0}n{y2}

{z0, z1}n{z0, z1}

{x0}

{y3}

{z0, z1}

{x0}

{y3, y4}

{z0, z1}

{x0}

{y4}

{z0, z1}

{x0, x1}

{y4}

{z0, z1}

{x0, x1}

{y4, y5}

{z0, z1}

{x0, x1}

{y4, y5}

{z0, z1, z2}

{x0, x1}

{y4, y5}

{z2}

{x0, x1}

{y4, y5}

{z2, z3}

分析结果

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%z1 = load i32, i32* %z
%t0 = icmp sgt i32 %z1, 0
br i1 %t0, label %bb1, label %bb2
```

S_x

S_y

S_z

{x0}

{}

{}

{x0}

{y0}

{}

{x0}

{y0}

{z0}

{x0}

{y0}

{z0}

{x0}

{y0}

{z0, z1}

bb1:

```
%y1 = load i32, i32* %y
%y2 = add i32 %y1, 1
store i32 %y2, i32* %y
br label %bb2
```

{x0}

{y0, y1}

{z0, z1}

{x0}

{y0, y1, y2}

{z0, z1}

{x0}

{y2}

{z0, z1}

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%x1 = load i32, i32* %x
%y5 = load i32, i32* %y
%z2 = add i32 %x1, %y5
store i32 %z2, i32* %z
%z3 = load i32, i32* %z
ret i32 %z3
```

{x0}

{}

{z0, z1}

{x0}

{y3}

{z0, z1}

{x0}

{y3, y4}

{z0, z1}

{x0}

{y4}

{z0, z1}

{x0, x1}

{y4}

{z0, z1}

{x0, x1}

{y4, y5}

{z0, z1}

{x0, x1}

{y4, y5}

{z0, z1, z2}

{x0, x1}

{y4, y5}

{z2}

{x0, x1}

{y4, y5}

{z2, z3}

优化结果

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

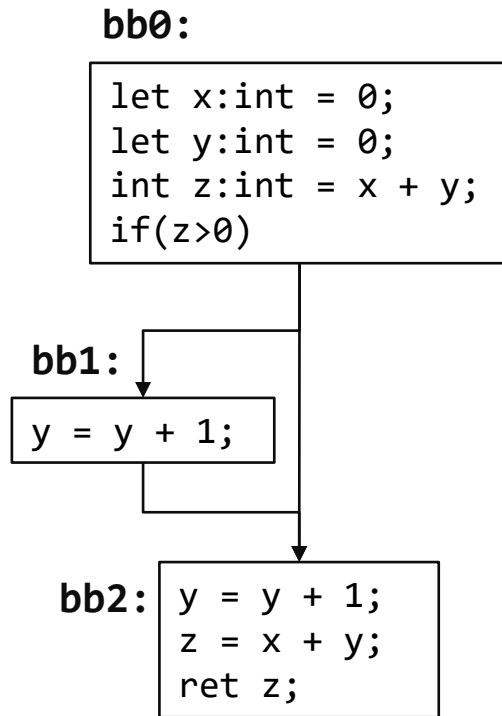
```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```


伪代码

```
For (each instruction n):  
    IN[n] = {<v:  $\emptyset$ >: v is a program variable}  
    OUT[n] = {<v:  $\emptyset$ >}  
Repeat:  
    For(each instruction n):  
        For(each n's predecessor p)  
            IN[n] = IN[n]  $\cup$  OUT[p]  
            OUT[n] = TRANSFER(n)  
Until IN[n] and OUT[n] stops changing for all n
```

二、优化Store

线性IR中的Store冗余



bb0:

```
%x = alloca i32  
%y = alloca i32  
%z = alloca i32  
store i32 0, i32* %x  
store i32 0, i32* %y  
%x0 = load i32, i32* %x  
%y0 = load i32, i32* %y  
%z0 = add i32 %x0, %y0  
store i32 %z0, i32* %z  
%t0 = icmp sgt i32 %z0, 0  
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1  
store i32 %y2, i32* %y  
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y  
%y4 = add i32 %y3, 1  
store i32 %y4, i32* %y  
%z2 = add i32 %x0, %y4  
store i32 %z2, i32* %z  
ret i32 %z2
```

优化思路: 可用Store语句分析

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

- 逆向遍历控制流图
- Transfer函数定义:
 - store i32 %t, i32* %x
 - $S = S \cup \{x\}$
 - %t = load i32, i32* %x
 - $S = S \setminus \{x\}$
 - %t = alloc, i32* %x
 - $S = S \setminus \{x\}$
- 遇到合并节点

$$OUT(n) = \bigcap_{n' \in \text{successor}(n)} IN(n')$$

分析过程

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

S

{}

{z}

{z}

{z}

{z}

{z}

{z}

{z}

{z}

{y,z}n{z}

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

{y,z}

{y,z}

{z}

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

{z}

{y,z}

{y,z}

{z}

{z}

{}

{}

分析结果

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
store i32 %z0, i32* %z
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

{}

{z}

{z}

{z}

{z}

{z}

{z}

{z}

{z}

{z}

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

{y,z}

{y,z}

{z}

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

{z}

{y,z}

{y,z}

{z}

{z}

{}

{}

优化结果

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

三、纯寄存器表示

消除内存存取

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

分析方法：数值流分析

bb0:

```
%x = alloca i32
%y = alloca i32
%z = alloca i32
store i32 0, i32* %x
store i32 0, i32* %y
%x0 = load i32, i32* %x
%y0 = load i32, i32* %y
%z0 = add i32 %x0, %y0
%t0 = icmp sgt i32 %z0, 0
br i1 %t0, label %bb1, label %bb2
```

bb1:

```
%y2 = add i32 %y0, 1
store i32 %y2, i32* %y
br label %bb2
```

bb2:

```
%y3 = load i32, i32* %y
%y4 = add i32 %y3, 1
store i32 %y4, i32* %y
%z2 = add i32 %x0, %y4
store i32 %z2, i32* %z
ret i32 %z2
```

- 正向遍历控制流图
- Transfer函数定义：
 - $\text{store i32 } \%t, \text{ i32* } \%x$
 - $S_x = \{t\}$
- 遇到合并节点

$$IN(n) = \bigcup_{n' \in \text{predecessor}(n)} OUT(n')$$

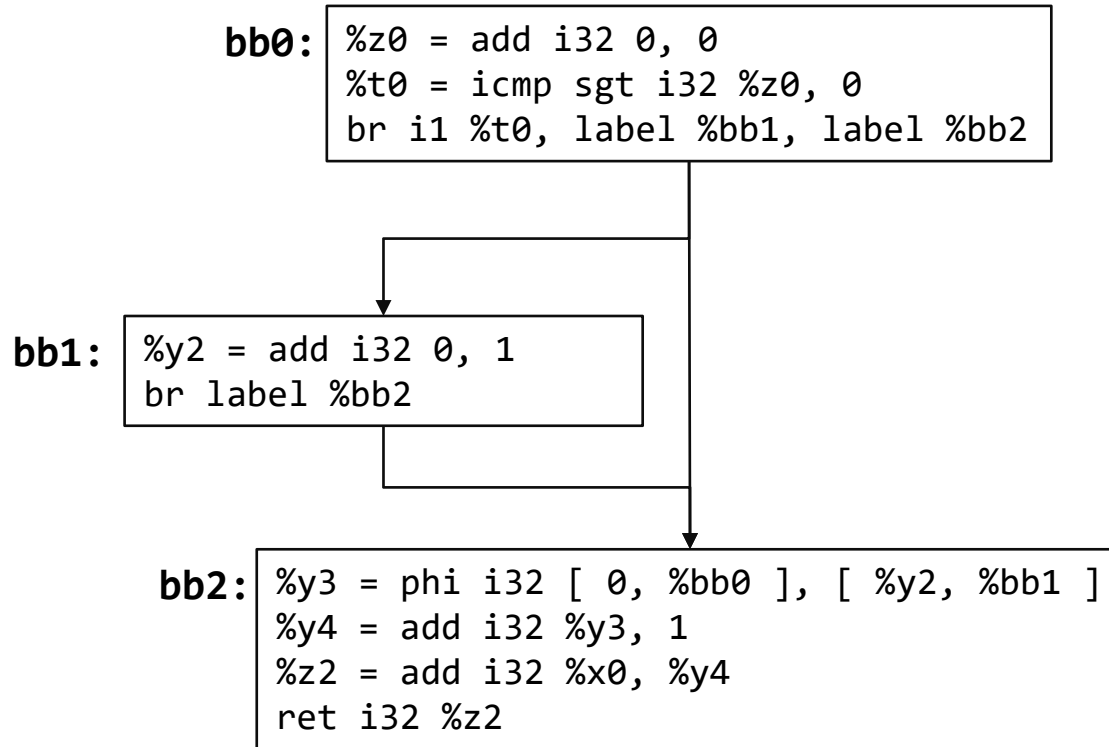
分析过程

bb0:	S_x	S_y	S_z
%x = alloca i32	{}	{}	{}
%y = alloca i32	{}	{}	{}
%z = alloca i32	{}	{}	{}
store i32 0, i32* %x	{0}	{0}	{}
store i32 0, i32* %y	{0}	{0}	{}
%x0 = load i32, i32* %x	{0}	{0}	{}
%y0 = load i32, i32* %y	{0}	{0}	{}
%z0 = add i32 %x0, %y0	{0}	{0}	{}
%t0 = icmp sgt i32 %z0, 0	{0}	{0}	{}
br i1 %t0, label %bb1, label %bb2	{0}	{0}	{}
	{0}	{0}	{}
bb1:			
%y2 = add i32 %y0, 1	{0}	{0}	
store i32 %y2, i32* %y	{0}	{y2}	{}
br label %bb2	{0}	{y2}	{}
	{0}	{y2}	{}
bb2:			
%y3 = load i32, i32* %y	{0} ∪ {0}	{0} ∪ {y2}	
%y4 = add i32 %y3, 1	{0}	{0} ∪ {y2}	{}
store i32 %y4, i32* %y	{0}	{y4}	{}
%z2 = add i32 %x0, %y4	{0}	{y4}	{}
store i32 %z2, i32* %z	{0}	{y4}	{}
ret i32 %z2	{}	{}	{z}

分析结果

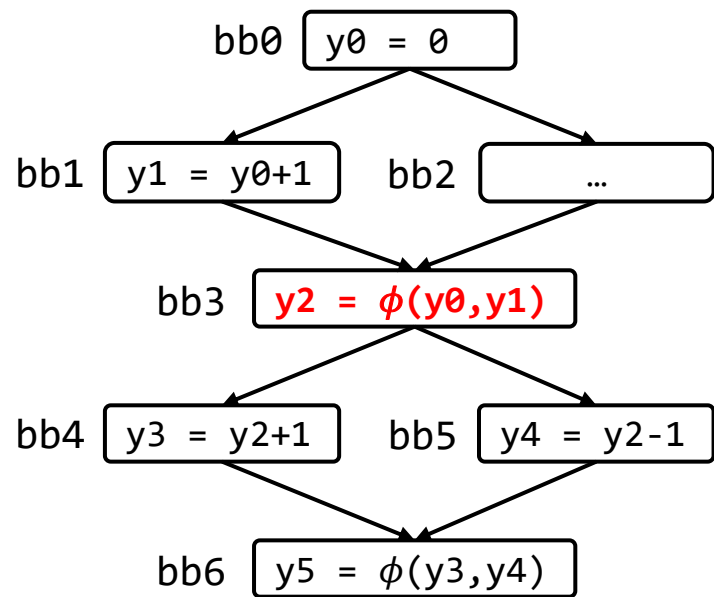
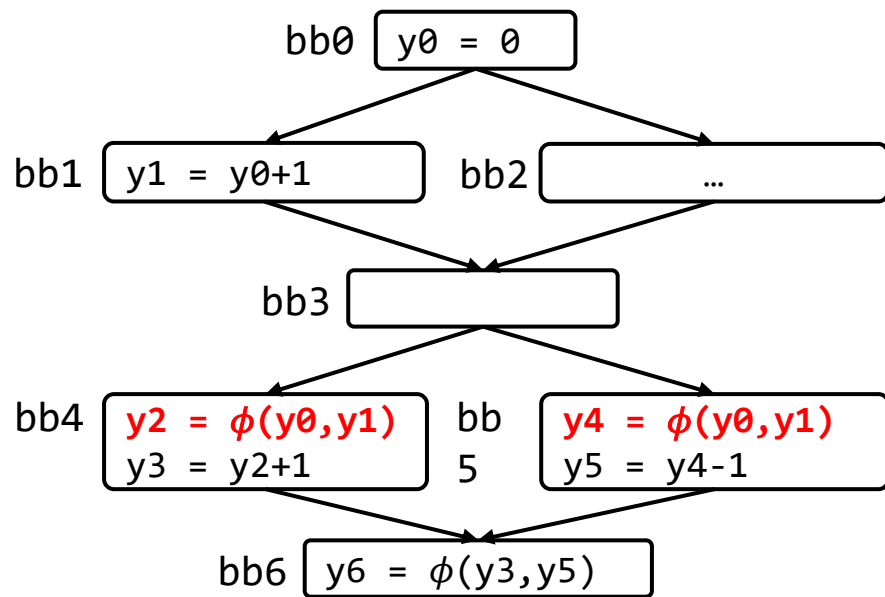
bb0:	S_x	S_y	S_z
%x = alloca i32	{}	{}	{}
%y = alloca i32	{}	{}	{}
%z = alloca i32	{}	{}	{}
store i32 0, i32* %x	{0}	{0}	{}
store i32 0, i32* %y	{0}	{0}	{}
%x0 = load i32, i32* %x	{0}	{0}	{}
%y0 = load i32, i32* %y	{0}	{0}	{}
%z0 = add i32 %x0, %y0	{0}	{0}	{}
%t0 = icmp sgt i32 %z0, 0	{0}	{0}	{}
br i1 %t0, label %bb1, label %bb2	{0}	{0}	{}
	{0}	{0}	{}
bb1:			
%y2 = add i32 %y0, 1	{0}	{0}	
store i32 %y2, i32* %y	{0}	{y2}	{}
br label %bb2	{0}	{y2}	{}
	{0}	{y2}	{}
bb2:			
%y3 = load i32, i32* %y	{0}	{0, y2}	
%y4 = add i32 %y3, 1	{0}	{0, y2}	{}
store i32 %y4, i32* %y	{0}	{0, y2}	{}
%z2 = add i32 %x0, %y4	{0}	{y4}	{}
store i32 %z2, i32* %z	{0}	{y4}	{}
ret i32 %z2	{}	{}	{z}

纯寄存器表示



四、Phi指令优化

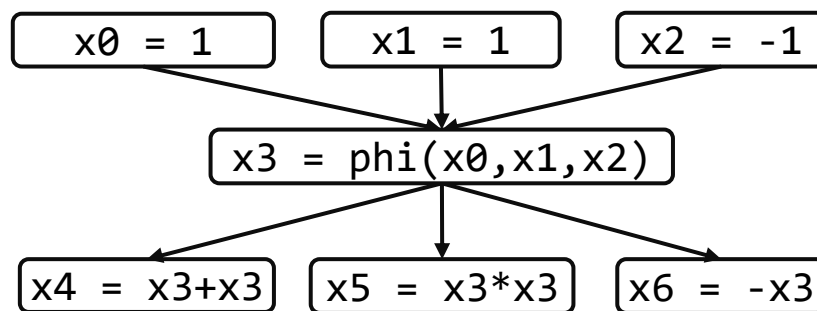
哪个Phi指令方案更优？



SSA简化def-use关系

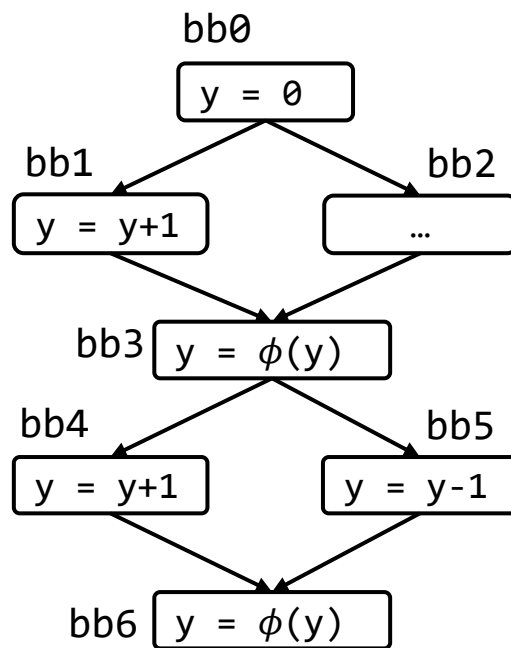
- 原始程序的def-use关系数量是 $O(m \times n)$;
- SSA的def-use数量减少为 $O(m + n)$ 。

```
match v1:
  0 => { x = 0; }
  1 => { x = 1; }
  _ => { x = -1; }
...
match v2:
  0 => { x = x + x; }
  1 => { x = x * x; }
  _ => { x = -x; }
```



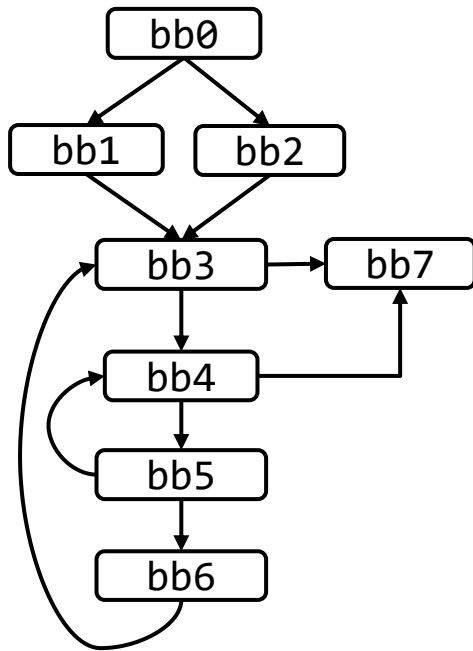
优化思路：基于支配边界优化Phi指令

- bb0支配bb2，bb1和bb2的支配边界都是bb3
- 如果bb1和bb2中都没有def(x)，bb3不需要phi(x)，可直接使用bb0中的def(x)
- 如果bb1中有def(y)，bb3中一定需要phi(y)



支配的基本概念

- 给定有向图 $G(V, E)$ 与起点 v_0 ，如果从 v_0 到某个点 v_j 均需要经过点 v_i ，则称 v_i 支配 v_j 或 v_i 是 v_j 的一个支配点
 - $v_i \in Dom(v_j)$
- 如果 $v_i \neq v_j$ ，则称 v_i 严格支配 v_j



控制流图

$$Dom(bb_0) = \{bb_0\}$$

$$Dom(bb_1) = \{bb_0, bb_1\}$$

$$Dom(bb_2) = \{bb_0, bb_2\}$$

$$Dom(bb_3) = \{bb_0, bb_3\}$$

$$Dom(bb_4) = \{bb_0, bb_3, bb_4\}$$

$$Dom(bb_5) = \{bb_0, bb_3, bb_4, bb_5\}$$

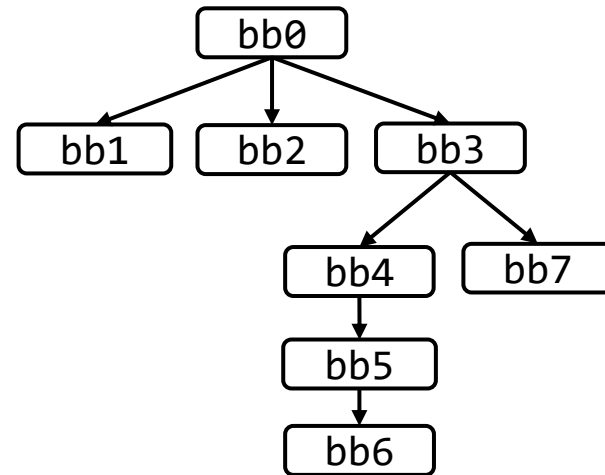
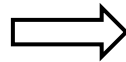
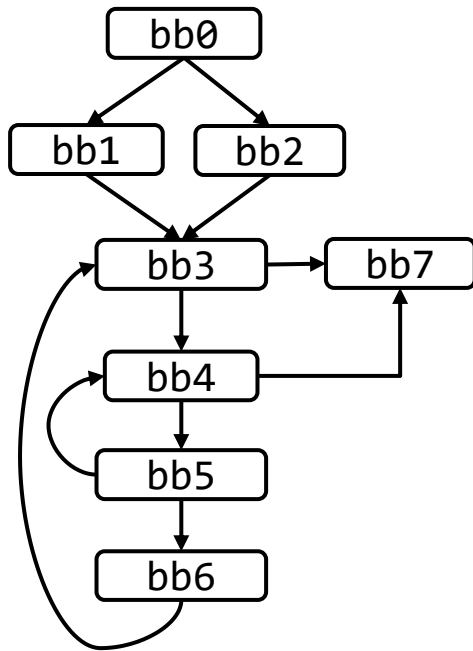
$$Dom(bb_6) = \{bb_0, bb_3, bb_4, bb_5, bb_6\}$$

$$Dom(bb_7) = \{bb_0, bb_3\}$$

$$Dom(v) = \begin{cases} \{v\}, & \text{if } v = v_0 \\ \{v\} \cup \left(\bigcap_{p \in pred(v)} Dom(p) \right), & \text{if } v \neq v_0 \end{cases}$$

支配树的基本概念

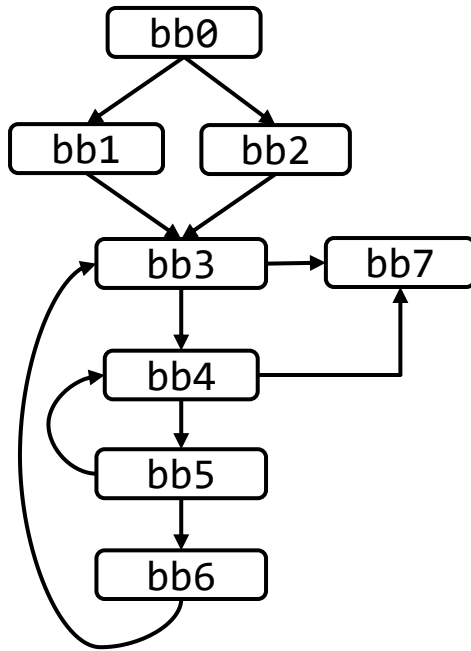
- 所有 v_j 的严格支配点中与 v_j 最接近的点成为 v_j 的最近支配点
 - $Idom(v_j) = v_i$, v_j 的其它严格支配点均严格支配 v_i
- 连接接所有的最近支配关系, 形成一棵支配树
 - 根节点外的每一点均存在唯一的最近支配点



支配树

支配边界Dominance Frontier

- v_i 的支配边界是所有满足条件的 v_j 的集合
 - v_i 支配 v_j 的一个前序节点
 - v_i 并不严格支配 v_j



$$DF(bb_0) = \{\}$$

$$DF(bb_1) = \{bb_3\}$$

$$DF(bb_2) = \{bb_3\}$$

$$DF(bb_3) = \{bb_3\}$$

$$DF(bb_4) = \{bb_4, bb_7\}$$

$$DF(bb_5) = \{bb_4\}$$

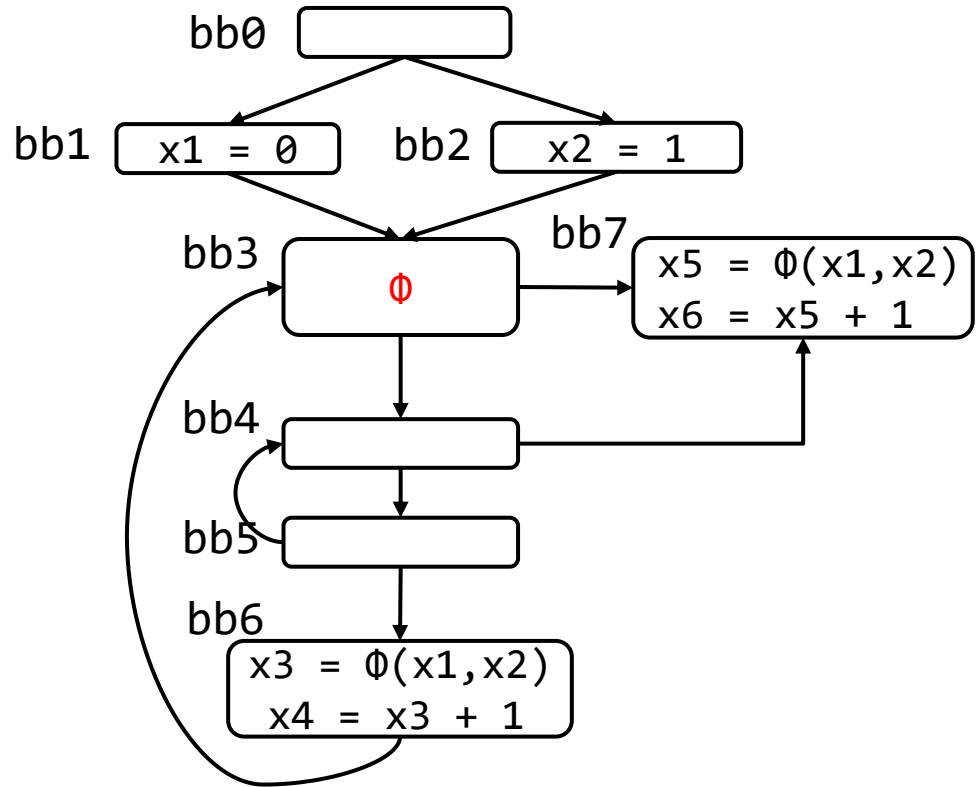
$$DF(bb_6) = \{bb_3\}$$

$$DF(bb_7) = \{\}$$

利用支配边界设置Phi指令

- 初始化：枚举所有变量的def-sites
 - $\text{def-sites}(x) = \{bb_1, bb_2, bb_6, bb_7\}$
- 为每个变量在 bb_j 增加phi节点：
 - $bb_i \in \text{def-sites}(x)$
 - $bb_j \in \text{DF}(bb_i)$
- 在 bb_3 增加phi指令的 $\phi(x)$

$DF(bb_0) = \{\}$
 $DF(bb_1) = \{bb_3\}$
 $DF(bb_2) = \{bb_3\}$
 $DF(bb_3) = \{bb_3\}$
 $DF(bb_4) = \{bb_4, bb_7\}$
 $DF(bb_5) = \{bb_4\}$
 $DF(bb_6) = \{bb_3\}$
 $DF(bb_7) = \{\}$



优化结果

- 重新编号
- 删除只有一个元素的phi指令

