

COMP130014 编译

# 第八讲：静态单赋值

徐辉

xuh@fudan.edu.cn



# 大纲

一、消除冗余load/store

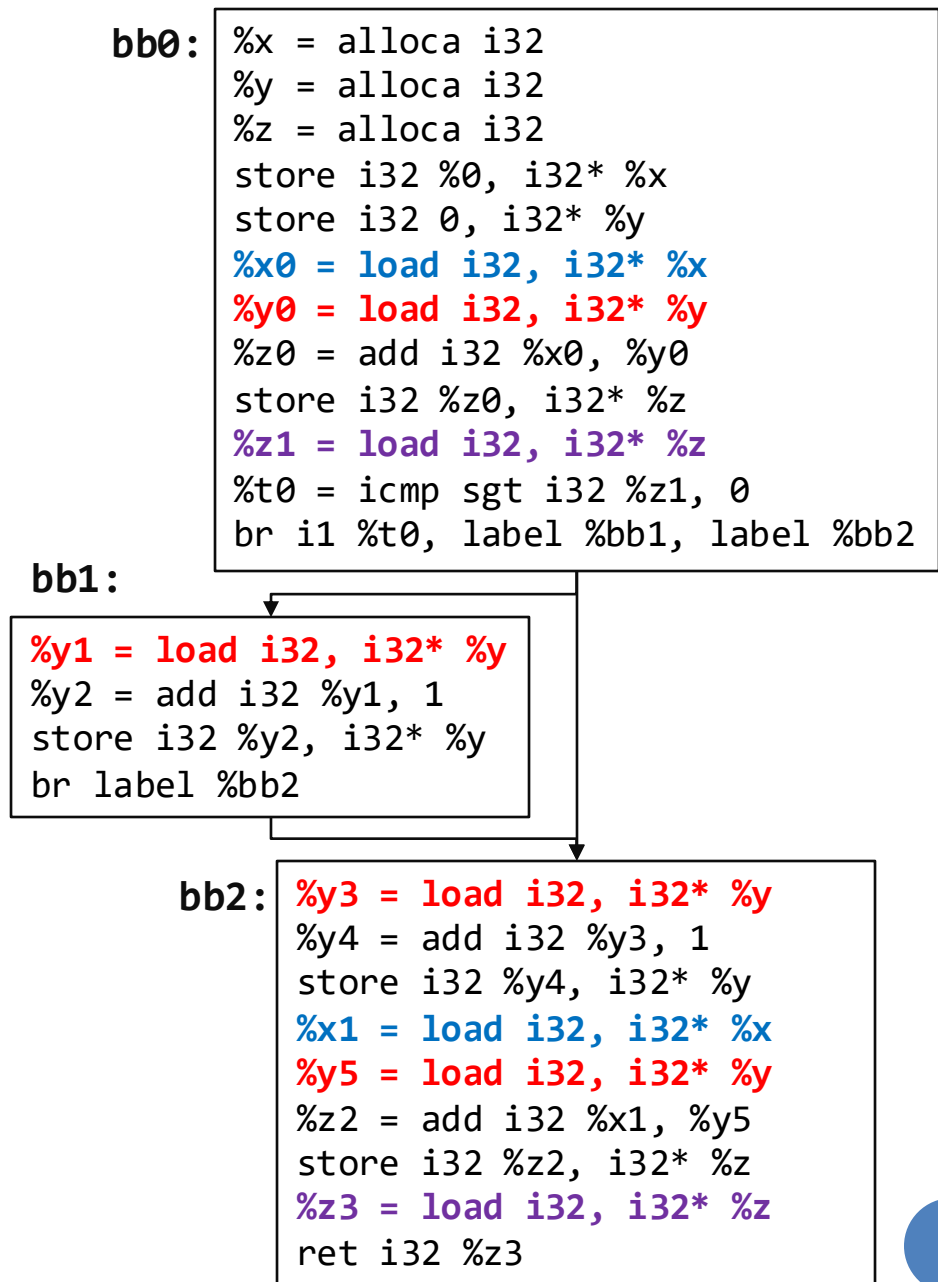
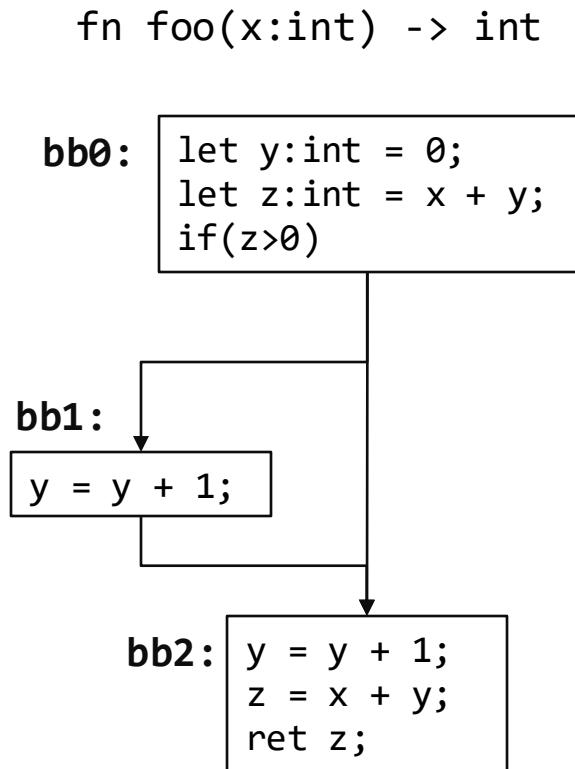
二、纯寄存器表示

三、Phi指令优化

# 一、消除冗余load/store

---

# 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 = load i32, i32\* %x

- $S_x = S_x \cup \{t\}$

- 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}

{}

{x0}

{y0}

{z0}

{x0}

{y0}

{z0,z1}

{x0}

{y0}

{z0,z1}

{x0}

{y0,y1}

{z0,z1}

{x0}

{y0,y1}

{z0,z1}

{x0}

{y2}

{z0,z1}

{x0}n{x0}

{y0}n{y2}

{z0,z1}n{z0,z1}

{x0}

{y3}

{z0,z1}

{x0}

{y3}

{z0,z1}

{x0}

{y4}

{z0,z1}

{x0,x1}

{y4}

{z0,z1}

{x0,x1}

{y4,y5}

{z0,z1}

{x0,x1}

{y4,y5}

{z0,z1}

{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}

{}

{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}

{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}n{x0}

{y0}n{y2}

{z0,z1}n{z0,z1}

{x0}

{y3}

{z0,z1}

{x0}

{y3}

{z0,z1}

{x0}

{y4}

{z0,z1}

{x0,x1}

{y4}

{z0,z1}

{x0,x1}

{y4,y5}

{z0,z1}

{x0,x1}

{y4,y5}

{z0,z1}

{x0,x1}

{y4,y5}

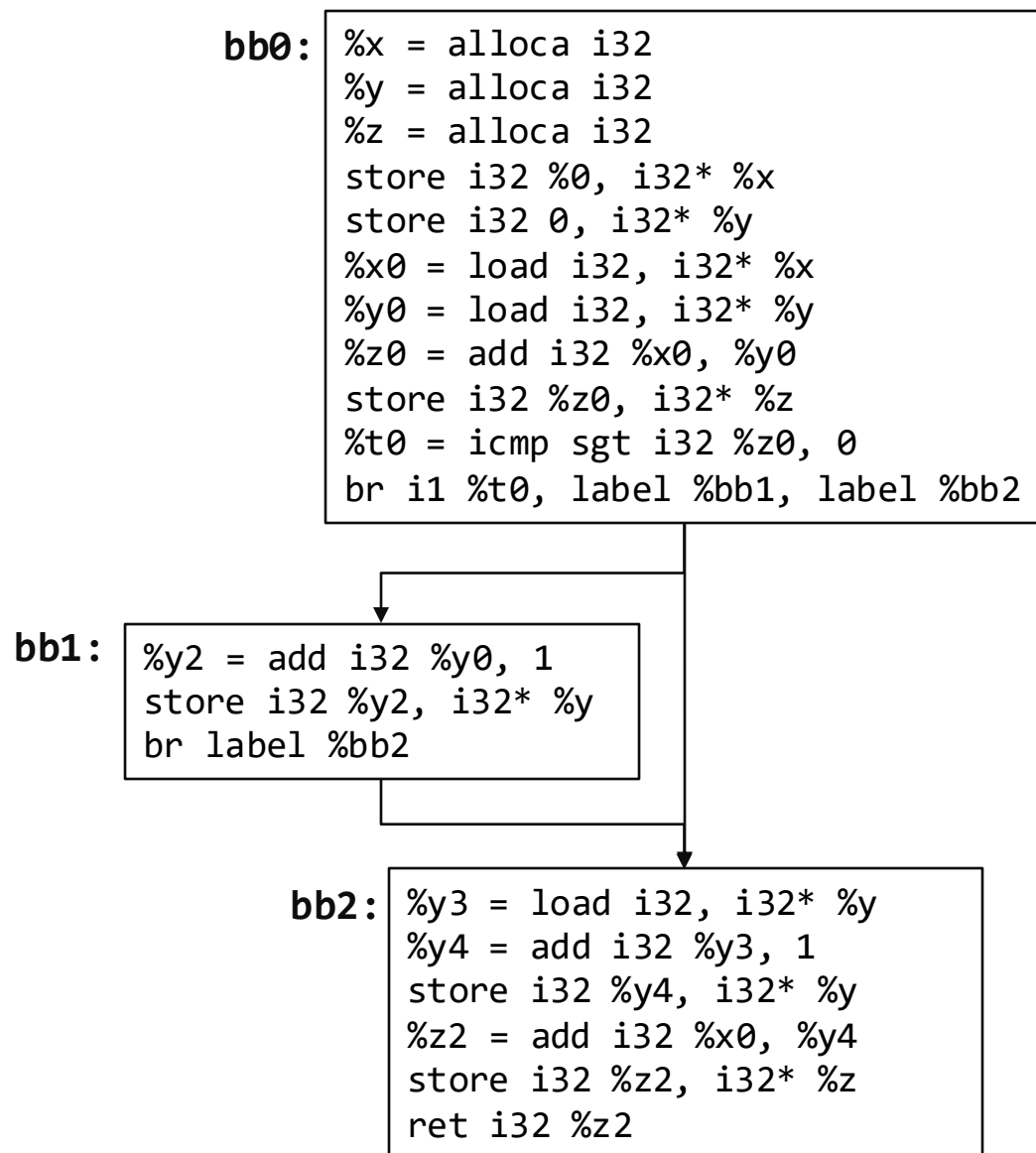
{z2}

{x0,x1}

{y4,y5}

{z2,z3}

# 优化结果





# 遇到循环：基于循环迭代的数据流分析

```
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]  $\cap$  OUT[p]  
            OUT[n] = TRANSFER(n)  
Until IN[n] and OUT[n] stops changing for all n
```

# 问题：算法是否一定会终止？

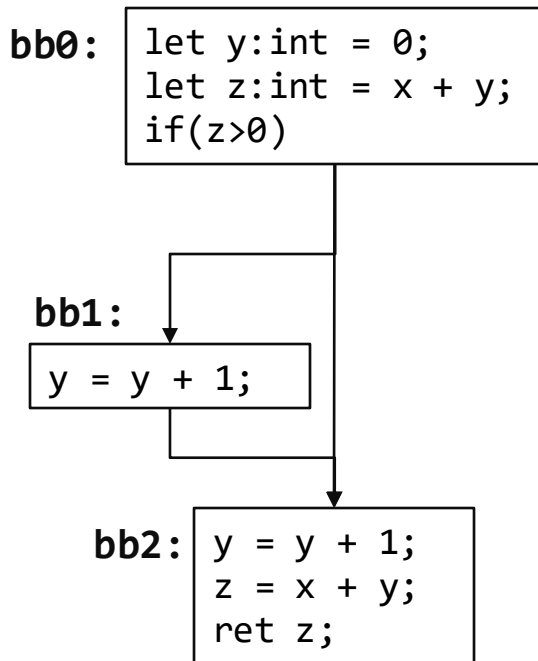
- 每个程序节点的可用寄存器数目单调递减

```
fn fac(n: int) -> int {  
    let r = 1;  
    while (n>0) {  
        r = r * n;  
        n = n-1;  
    }  
    ret r;  
}
```

```
define i32 @fac(i32 %0) {  
bb0:  
    %n = alloca i32  
    %r = alloca i32  
    store i32 %0, i32* %n  
    store i32 1, i32* %r  
    br label %bb1  
  
bb1:  
    %t1 = load i32, i32* %n  
    %t2 = icmp sgt i32 %t1, 0  
    br i1 %t2, label %bb2, label %bb3  
  
bb2:  
    %t3 = load i32, i32* %r  
    %t4 = load i32, i32* %n  
    %t5 = mul i32 %t3, %t4  
    store i32 %t5, i32* %r  
    %t6 = load i32, i32* %n  
    %t7 = sub i32 %t6, 1  
    store i32 %t7, i32* %n  
    br label %bb1  
  
bb3:  
    %t8 = load i32, i32* %r  
    ret i32 %t8  
}
```

# 线性IR中的store冗余

fn foo(x:int) -> int



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
  - $S = S \setminus \{x\}$
- %x = alloca i32
  - $S = S \setminus \{x\}$

- 遇到合并节点

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

# 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}

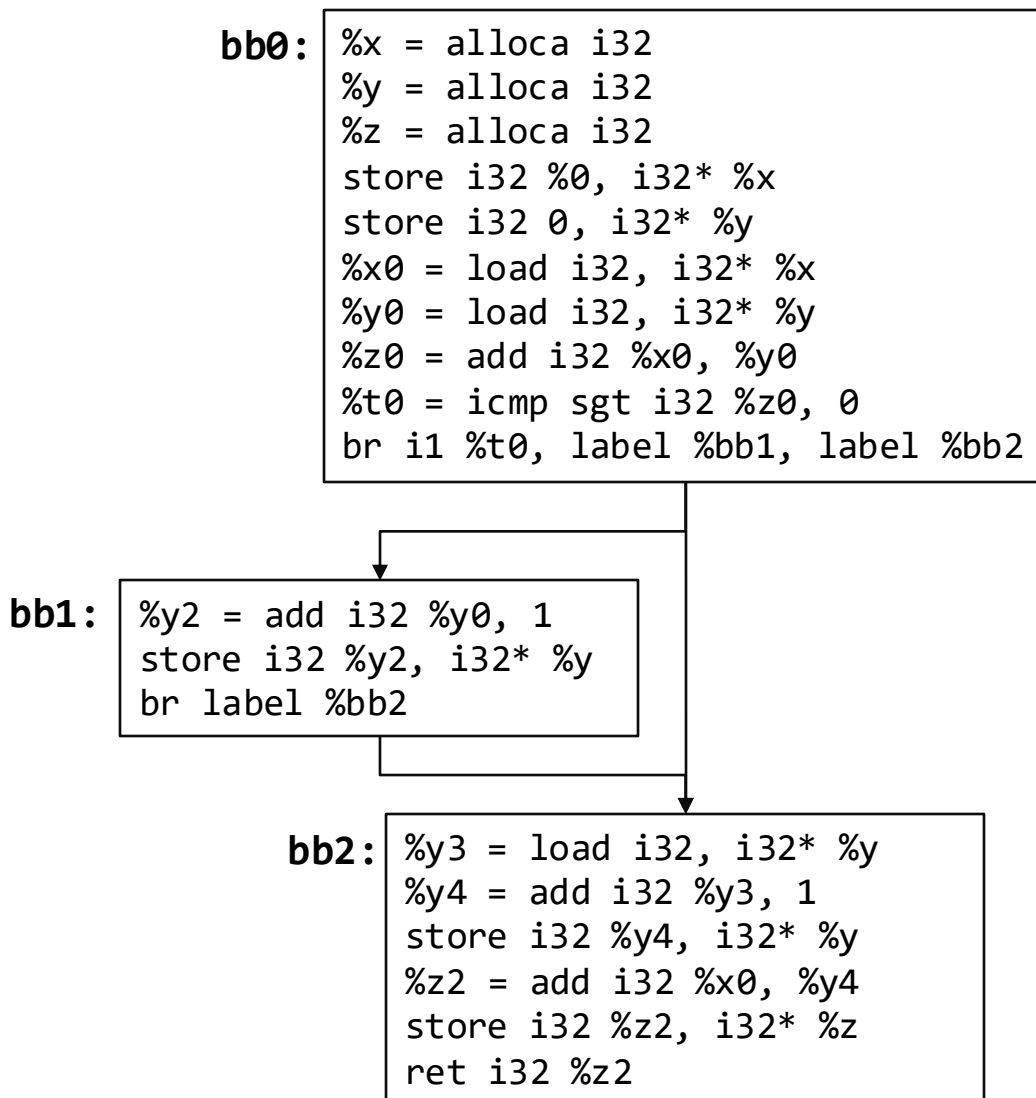
{z}

{z}

{}

{}

# 优化结果

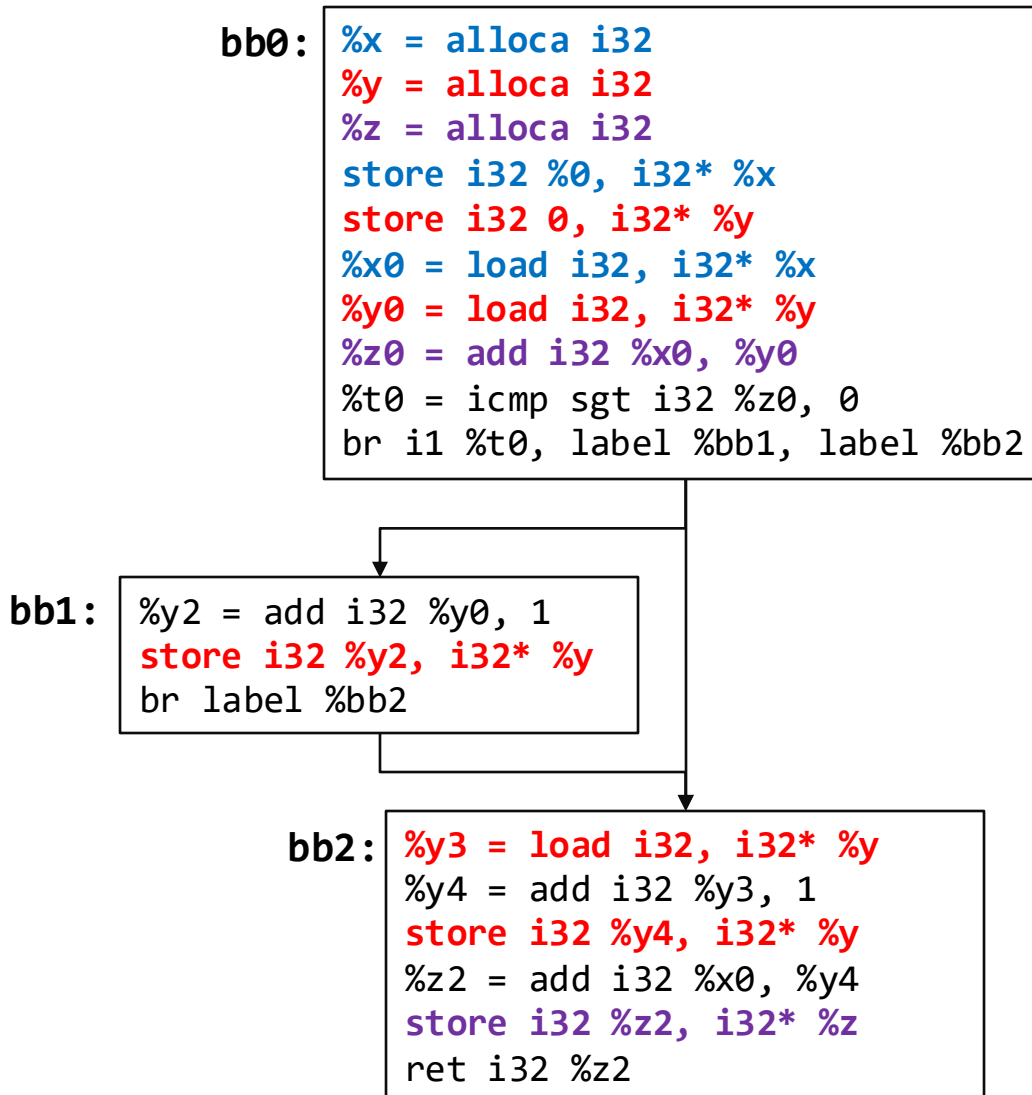


## 二、纯寄存器表示

---



# 消除数据存取



# 分析方法：内存数值流分析

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:

- `store i32 %t, i32* %x`

- $S_x = \{t\}$

- 遇到合并节点

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

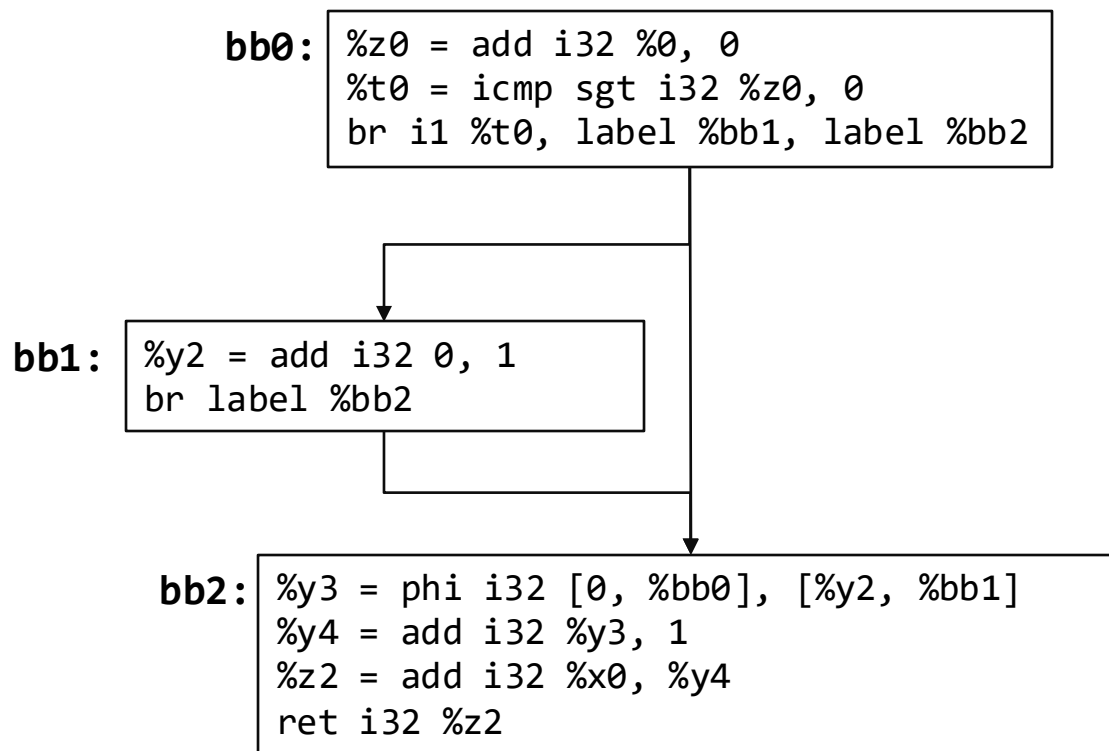
## 分析过程

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

## 分析结果

	$S_x$	$S_y$	$S_z$
<b>bb0:</b> <del>%x = alloca i32</del>			
<del>%y = alloca i32</del>			
<del>%z = alloca i32</del>			
store i32 %0, i32* %x	{}	{}	{}
<del>store i32 0, i32* %y</del>	{%0}	{}	{}
<del>%x0 = load i32, i32* %x</del>	{%0}	{0}	{}
<del>%y0 = load i32, i32* %y</del>	{%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}	{}
<b>bb1:</b>			
%y2 = add i32 %y0, 1	{%0}	{0}	{}
<del>store i32 %y2, i32* %y</del>	{%0}	{y2}	{}
br label %bb2	{%0}	{y2}	{}
<b>bb2:</b> %y3 = load i32, i32* %y	{%0}	{0, y2}	
%y4 = add i32 %y3, 1	{%0}	{0, y2}	{}
<del>store i32 %y4, i32* %y</del>	{%0}	{0, y2}	{}
%z2 = add i32 %x0, %y4	{%0}	{y4}	{}
<del>store i32 %z2, i32* %z</del>	{%0}	{y4}	{}
ret i32 %z2	{%0}	{y4}	{z2}

# 纯寄存器表示



# 练习：将下列代码转化为纯寄存器表示

```
define i32 @fac(i32 %0) {
bb0:
    %n = alloca i32
    %r = alloca i32
    store i32 %0, i32* %n
    store i32 1, i32* %r
    br label %bb1
bb1:
    %t1 = load i32, i32* %n
    %t2 = icmp sgt i32 %t1, 0
    br i1 %t2, label %bb2, label %bb3
bb2:
    %t3 = load i32, i32* %r
    %t4 = load i32, i32* %n
    %t5 = mul i32 %t3, %t4
    store i32 %t5, i32* %r
    %t6 = load i32, i32* %n
    %t7 = sub i32 %t6, 1
    store i32 %t7, i32* %n
    br label %bb1
bb3:
    %t8 = load i32, i32* %r
    ret i32 %t8
}
```

# 数据流分析方法小结

	May Analysis ( $\cup$ )	Must Analysis ( $\cap$ )
前向分析	纯寄存器表示	精简load
逆向分析	活跃性分析	精简store

# 数据流分析方法小结：框架


  $[n] = \text{Transfer}$    $[n]$

  $[n] =$     $[n']$

$n' \in$    $(n)$

---

  $= \text{IN or OUT}$

  $= \cup \text{ (may) or } \cap \text{ (must)}$

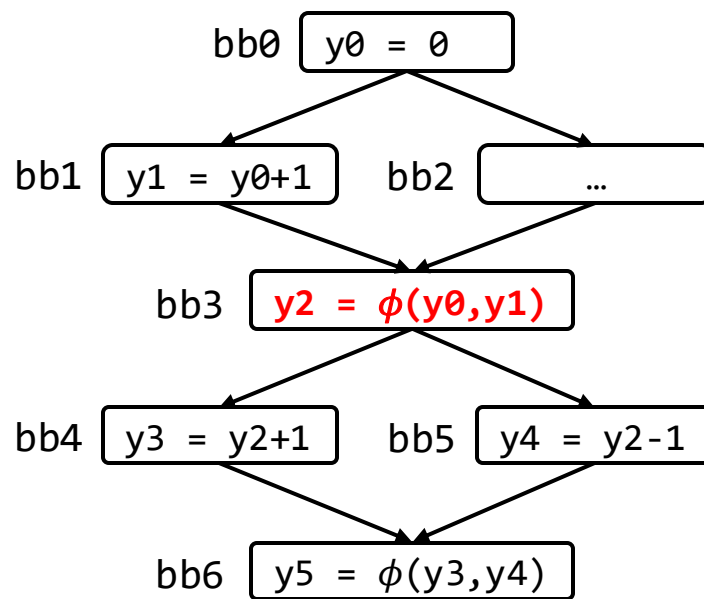
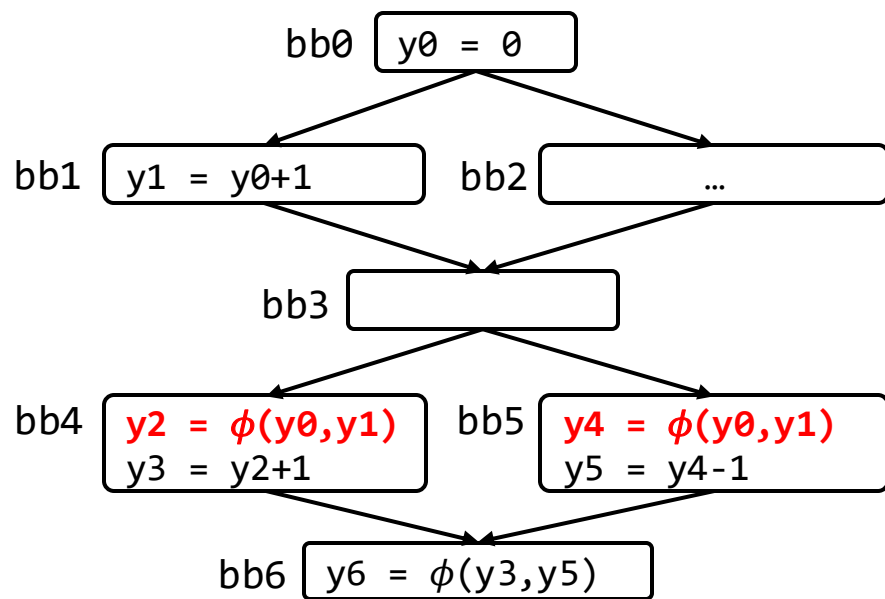
   $= \text{predecessors or successors}$



### 三、Phi指令优化

---

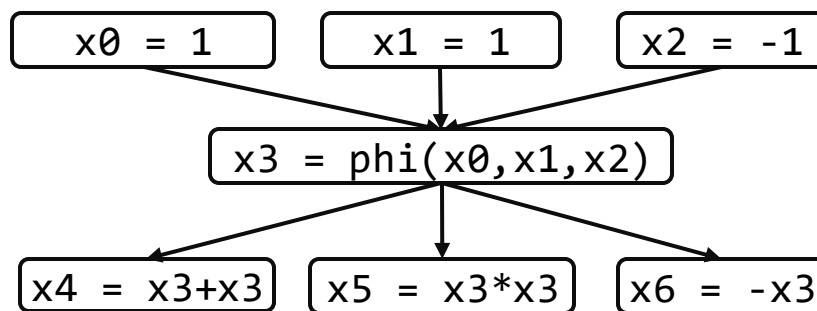
# 哪个phi指令方案更优？



# SSA简化def-use关系

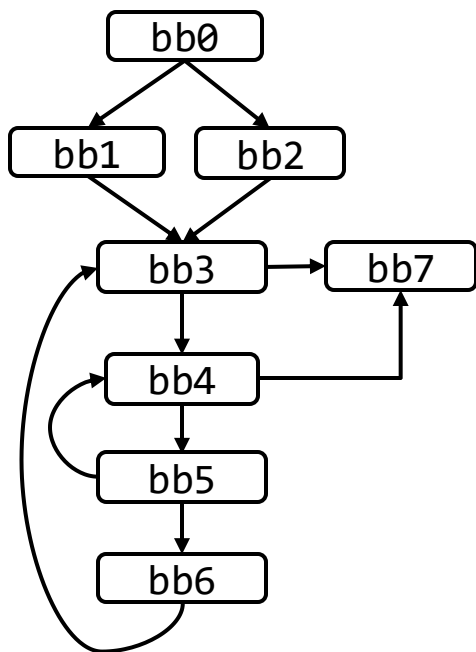
- 原始程序的def-use关系数量是 $O(m \times n)$ ;
- SSA的def-use数量减少为 $O(m + n)$ 。

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



# 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 $v_j$ ， $v_i \in IDom(v_j)$



控制流图

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

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

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

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

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

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

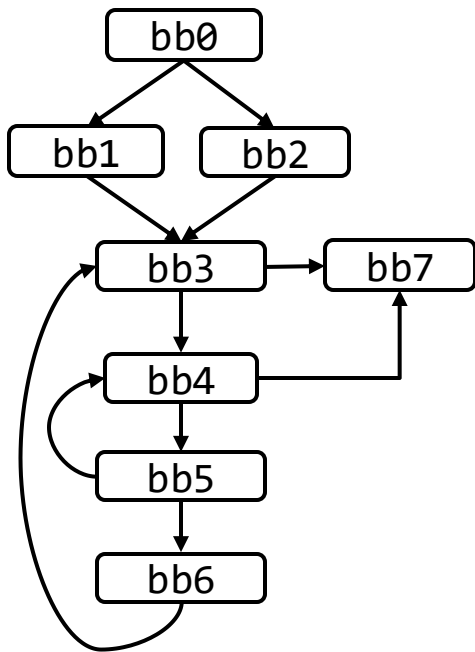
$$Dom(bb_6) = \{bb_0, bb_3, bb_4, bb_5, bb_6\}$$

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

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

# 支配边界

- $v_i$  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  $v_j$



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

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

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

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

$$DF(bb_4) = \{bb_3, bb_4, bb_7\}$$

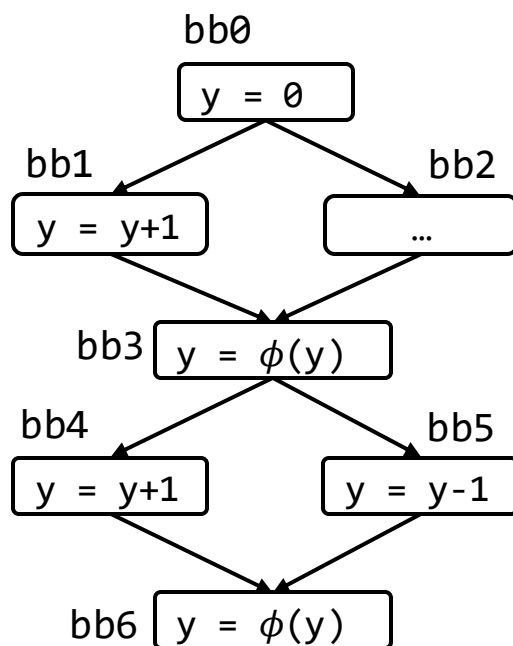
$$DF(bb_5) = \{bb_3, bb_4\}$$

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

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

# 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)



# 利用支配边界设置phi指令

- 初始化：枚举所有变量的def-sites

- def-sites(x) = {bb1, bb2, bb6, bb7}

- 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bb3增加phi指令的phi(x)

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

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

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

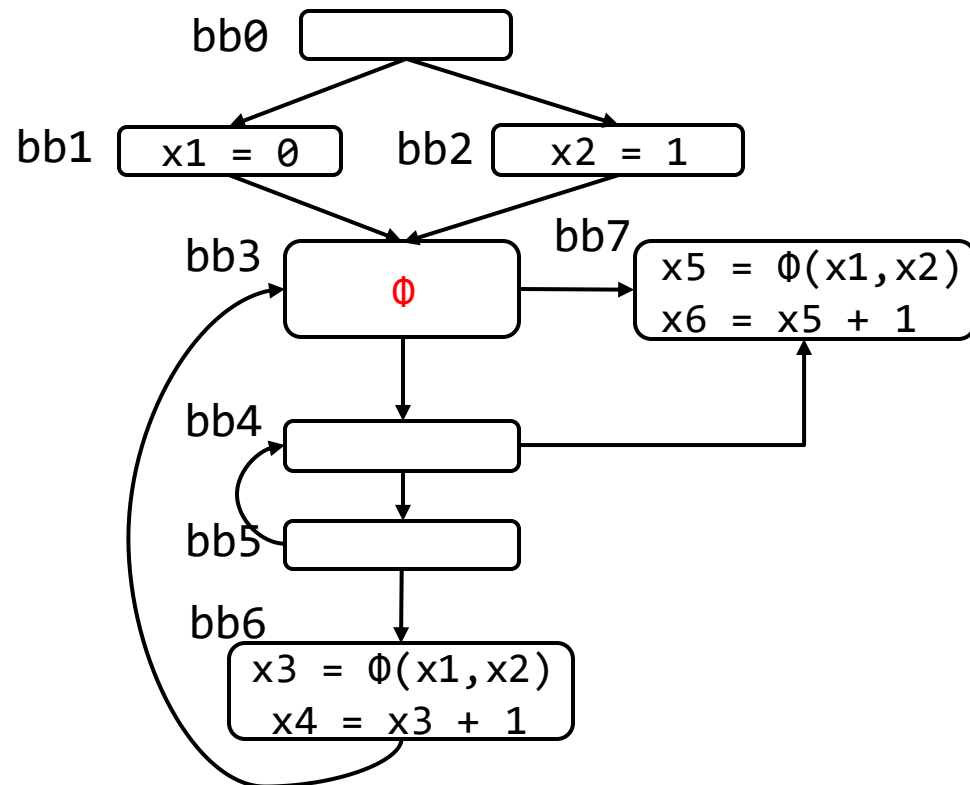
$$DF(bb_3) = \{bb_3\}$$

$$DF(bb_4) = \{bb_3, bb_4, bb_7\}$$

$$DF(bb_5) = \{bb_3, bb_4\}$$

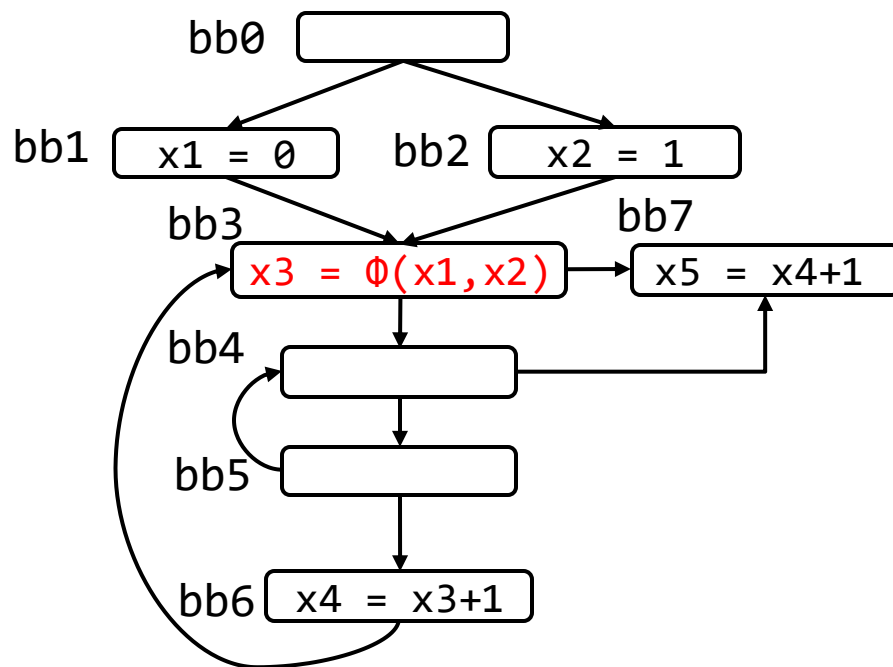
$$DF(bb_6) = \{bb_3\}$$

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



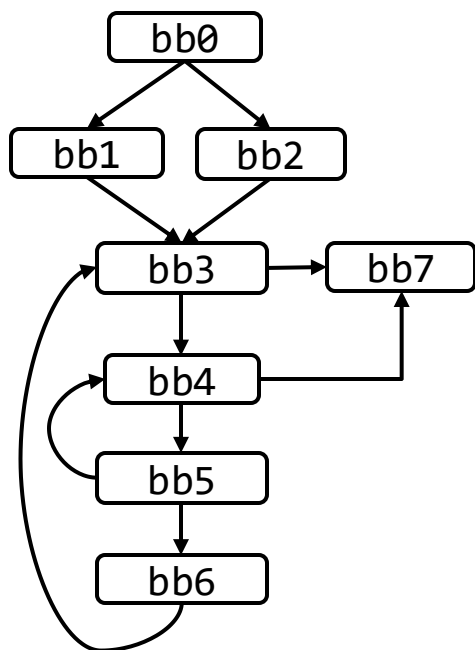
# 优化结果

- 重新编号
- 删除只有一个元素的phi指令





# 如何计算支配边界？



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

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

# 如何计算支配边界？

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

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

$\forall bb_i \in Dom(bb_0) - IDom(bb_1),$   
 $bb_1 \in DF(bb_i)$

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

# 练习

- 分析右侧代码：

- 1) 计算支配树
- 2) 计算支配边界
- 3) 修改为SSA

