#### 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

# bb0: let x:int = 0; let y:int = 0; int z:int = x + y; if(z>0) bb1: y = y + 1; y = y + 1; z = x + y; ret z;

```
%x = alloca i32
    bb0:
         %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 %73
```

#### 优化思路: 可用寄存器分析

#### 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

ret i32 %73

%z3 = load i32, i32\* %z

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

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

# 分析过程

ret i32 %73

,				
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———	(,,0)	()	<u> </u>
	%y0 = load i32, i32* %y————	{x0}		{}
	%z0 = add i32 %x0, %y0 ————	{x0}	.,,,	{}
	store i32 %z0, i32* %z ————	{x0}	.,,	{z0}
	%z1 = load i32, i32* %z	{x0}		{z0}
	%t0 = icmp sgt i32 %z1, 0	{x0}	{y0}	{z0,z1}
	br i1 %t0, label %bb1, label %b	b2		
bb1:				
0/1 1.00	<b>▼</b>			
_	d i32, i32* %y	{x0}	{y0,y1}	{z0,z1}
_	i32 %y1, 1 %y2, i32* %y	{x0}	1, 1, 1	{z0,z1}
br label		{x0}		{z0,z1}
DI TADET	7600Z	( - )		
	<del></del>	1 {x0}	$n\{x0\}$ {y0}n{y2}	$\{z0,z1\}\cap\{z0,z1\}$
bb	)2:  %y3 = load i32, i32* %y	{x0}		{z0,z1}
	%y4 = add i32 %y3, 1	{x0}	{y4}	{z0,z1}
	store i32 %y4, i32* %y %x1 = load i32, i32* %x		{y4}	{z0,z1}
			x1} {y4}	{z0,z1}
	%y5 = load i32, i32* %y	{x0,	x1} {y4,y5}	{z0,z1}
	%z2 = add i32 %x1, %y5	{x0,	x1} {y4,y5}	{z2}
	store i32 %z2, i32* %z	{x0,	x1} {y4,y5}	{z2}
	%z3 = load i32, i32* %z	{x0,	x1} {y4,y5}	{z2,z3}

# 分析结果

bb0:	%x = al	loca i32					
	%y = al	loca i32					
	%z = al	loca i32					
	store i	.32 0, i32	.* %x				
	store i	.32 0, i32	.* %y				
	%x0 = 1	oad i32,	i32* %x		(0)	()	()
	%y0 = load i32, i32* %y				{x0}	{}	{}
	%z0 = add i32 %x0, %y0				{x0}	{y0}	{}
	store i32 %z0, i32* %z				{x0}	{y0}	{z0}
	<del>%z1 = 1</del>	oad i32,	i32* %z		{x0}	{y0}	{z0}
			.32 <mark>%z1</mark> , 0		{x0}	{y0}	{z0, <mark>z1</mark> }
		. •	. %bb1 <b>,</b> label %	bb2			
bb1:			<u> </u>				
			7				
<del>%y1 = loa</del>					( 0)	( 0 4)	( 0 1)
%y2 = add	i32	l, 1			{x0}	{y0, <mark>y1</mark> }	{z0,z1}
store i32 %y2, i32* %y				{x0}	{y2}	{z0,z1}	
br label %bb2				{x0}	{y2}	{z0,z1}	
					6 03	63	( 0 4)
hl	2. %/3	- load i3	32, i32* %y		{x0}	{}	{z0,z1}
Di		= add i32			{x0}	{y3}	{z0,z1}
	_		i32* %y ——		{x0}	{y4}	{z0,z1}
	l l	-	32* %x		{x0}	{y4}	{z0,z1}
	l l				{x0, <mark>x1</mark> }	{y4}	{z0,z1}
			32, i32* %y		{x0,x1}	{y4, <mark>y5</mark> }	{z0,z1}
	l l		2 <mark>%x1</mark> , <mark>%y5</mark> ———		$\{x0,x1\}$	{y4,y5}	{z2}
			2, i32* %z ——		{x0,x1}	1,5 - 5	{z2}
	l l	= 10aa 13 i32 <mark>%z3</mark>	32, i32* %z		$\{x0,x1\}$	{y4,y5}	{z2, <mark>z3</mark> }
	l rei	13/ <mark>6/3</mark>					

#### 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 node n):
    IN[n] = {<v: Ø>: v is a program variable}
    OUT[n] = {<v: Ø>}
Repeat:
    For(each node n):
        For(each n's predecessor p)
            IN[n] = IN[n] n 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: let x:int = 0; let y:int = 0; int z:int = x + y; if(z>0) bb1: y = y + 1; bb2: y = y + 1; z = x + y;

ret z;

```
%x = alloca i32
    bb0:
          %y = alloca i32
          %z = alloca i32
          store i32 0, i32* %x
          store i32 0, i32* %v
          %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

```
%x = alloca i32
    bb0:
          %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

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

#### 分析过程

```
%x = alloca i32
    hh0:
          %y = alloca i32
                                                    {}
          %z = alloca i32
          store i32 0, i32* %x
                                                    {z}
          store i32 0, i32* %y
                                                    {z}
          %x0 = load i32, i32* %x -
                                                    {z}
          %y0 = load i32, i32* %y -
                                                    {z}
          %z0 = add i32 %x0, %y0
                                                    {z}
          store i32 %z0, i32* %z
                                                    {z}
          %t0 = icmp sgt i32 %z0, 0-
                                                    {z}
          br i1 %t0, label %bb1, label %bb2-
                                                    {y,z}\cap{z}
bb1:
                                                   {y,z}
%y2 = add i32 %y0, 1
                                                   {y,z}
store i32 %y2, i32* %y -
                                                    {z}
br label %bb2
                                                    {z}
       bb2: | %y3 = load i32, i32* %y -
                                                    \{y,z\}
             %y4 = add i32 %y3, 1
                                                    {y,z}
             store i32 %y4, i32* %y
                                                    {z}
             %z2 = add i32 %x0, %y4
                                                    {z}
             store i32 %z2, i32* %z
             ret i32 %z2
```

#### 分析结果

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

#### 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
             %y3 = load i32, i32* %y
       bb2:
             %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
            %y3 = load i32, i32* %y
       bb2:
             %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:
  - store i32 %t, i32\* %x
    - $S_{\mathbf{x}} = \{t\}$
- 遇到合并节点

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

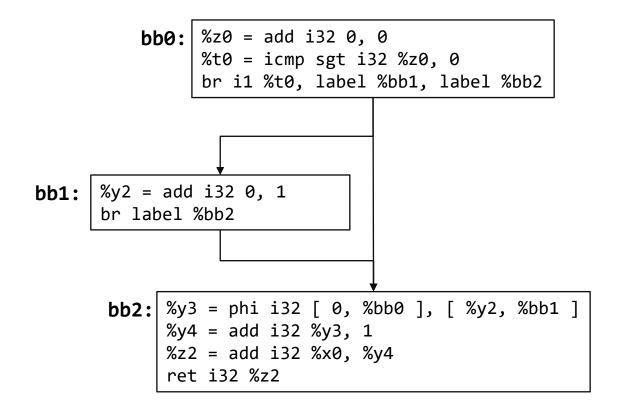
# 分析过程

		1		
bb0:	%x = alloca i32			
	%y = alloca i32	ſι	ſl	{}
	%z = alloca i32	{ } { }	{}	{}
	store i32 0, i32* %x		{}	
	store i32 0, i32* %y	{0}	{0}	{}
		{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:		(-)	(-)	
0/2	1 : 22 % 0 . 4	{0}	{0}	
	i32 %y0, 1	{0}	{y2}	{}
1	store i32 %y2, i32* %y		{y2}	{}
br label	%bb2 —	{0} {0}	{y2}	{}
		ίοι	\ <b>y</b> \ \ <b>j</b>	()
		(0),,(0)	(0)(-2)	
hl	o2: %y3 = load i32, i32* %y	{0}∪{0}	{0}∪{y2}	()
	%y4 = add i32 %y3, 1	{0}	{0}∪{y2}	{}
	store i32 %y4, i32* %y	{0}	{0}∪{y2}	{}
		{0}	{y4}	{}
	%z2 = add i32 %x0, %y4	{0}	{y4}	{}
	store i32 %z2, i32* %z	{0}	{y4}	{}
	ret i32 %z2	{}	{}	{z}

### 分析结果

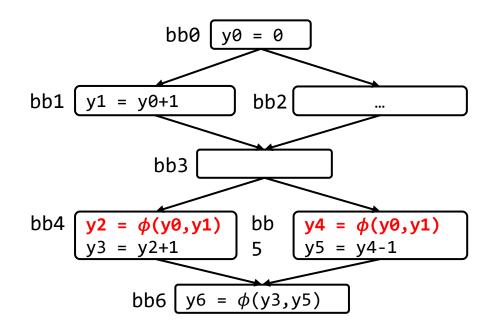
	0/ 11 :22			
bb0:	%x = alloca i32			
	<del>%y = alloca i32</del>	{}	{}	{}
	<del>%z = alloca i32</del>	{}	{}	{}
	<del>store i32 0, i32* %x</del>	{0}	{0}	{}
	<del>store i32 0, i32* %y</del>	{0}	{0}	{}
	%x0 = load i32, i32* %x	{0}	{0}	{}
	<del>%y0 = load i32, i32* %y</del>	{0}	{0}	{}
	%z0 = add i32	` '	• • •	{}
	%t0 = icmp sgt i32 %z0, 0	{0}	{0}	{}
	br i1 %t0, label %bb1, label %bb2	{0}	{0}	{}
		{0}	{0}	
bb1:		{0}	{0}	{}
001.		4 - 5		
%y2 = add	l i32 <mark>%y0</mark> , 1	{0}	{0}	()
	2 %y2, i32* %y	{0}	{y2}	{}
br label		{0}	{y2}	{}
		{0}	{y2}	{}
	0/2	{0}	{0,y2}	
DI	<b>62:</b>	{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}

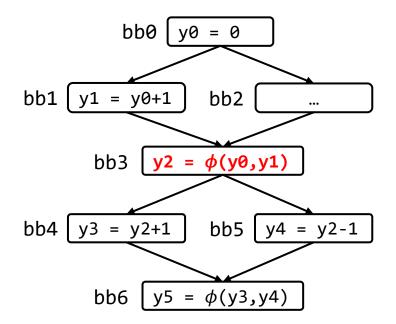
#### 纯寄存器表示



# 四、Phi指令优化

#### 哪个Phi指令方案更优?

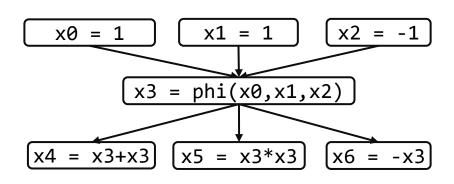




#### SSA简化def-use关系

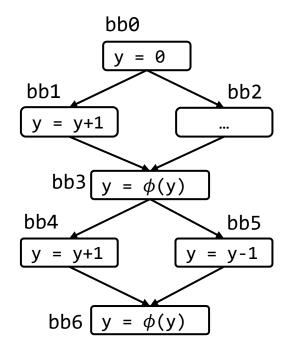
- 原始程序的def-use关系数量是 $O(m \times n)$ ;
- SSA的def-use数量减少为O(m+n)。

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



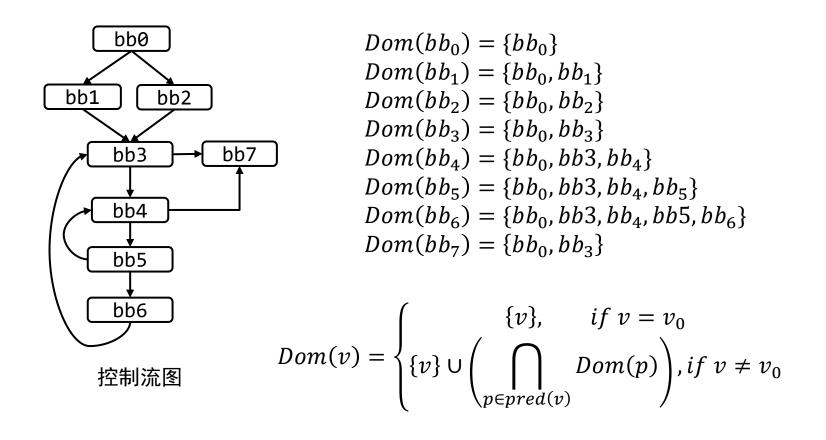
#### 优化思路:基于支配边界优化Phi指令

- 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)



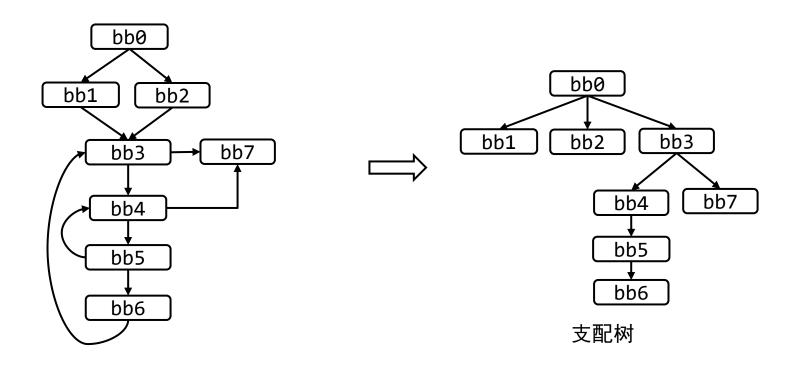
#### 支配的基本概念

- 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
  - $v_i \in Dom(v_i)$
- 如果 $v_i \neq v_j$ ,则称 $v_i$ 严格支配 $v_j$



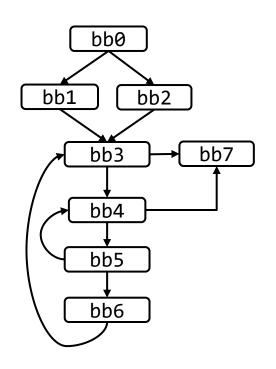
#### 支配树的基本概念

- 所有 $v_i$ 的严格支配点中与 $v_i$ 最接近的点成为 $v_i$ 的最近支配点
  - $Idom(v_i) = v_i$ ,  $v_i$ 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



#### 支配边界Dominance Frontier

- $v_i$ 的支配边界是所有满足条件的 $v_i$ 的集合
  - $v_i$  支配 $v_i$ 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
  - def-sites(x) = {bb1,bb2,bb6,bb7}
- 为每个变量在bb<sub>i</sub>增加phi节点:
  - $bb_i \in def\text{-sites}(x)$
  - bb<sub>i</sub> ∈ DF(bb<sub>i</sub>)
- 在bb3增加phi指令的phi(x)

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

DF(bb_1) = \{bb_3\}

DF(bb_2) = \{bb_3\}

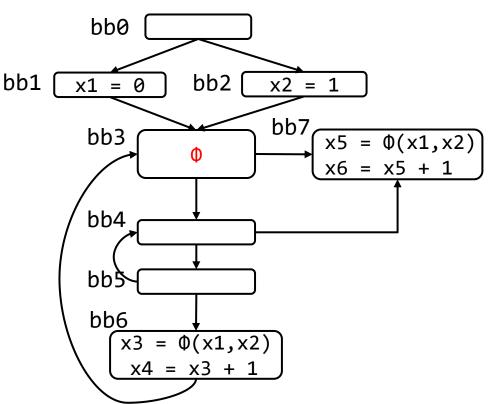
DF(bb_3) = \{bb_3\}

DF(bb_4) = \{bb_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

