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

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

bb2: | y = y + 1;

ret z;

z = x + y;

bb0:

```
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 %73
```

优化思路: 可用寄存器分析

```
%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
```

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

bb0:		= alloca i32					
	%y	%y = alloca i32					
	%z	%z = alloca i32					
	st	store i32 0, i32* %x			C	C	C
	st	ore i32 0, i32	2* %y		S_x	S_y	S_z
	%x	%x0 = load i32, i32* %x———			(0)	()	()
	%y	0 = load i32,	i32* %y———		{x0}	{}	{}
	%z0 = add i32 %x0, %y0 —				{x0}	{y0}	{}
	store i32 %z0, i32* %z				{x0}	{y0}	{z0}
	l .	1 = load i32,			{x0}	{y0}	{z0}
		0 = icmp sgt :			{x0}	{y0}	{z0,z1}
		. •	l %bb1, label s	%bb2			↓
bb1:			1				
DD1.							
%y1 = loa	d i	32, i32* %y					
%y2 = add	i 3	2 %y1, 1			{x0}	{y0,y1}	{z0,z1}
store i32 %y2, i32* %y				{x0}	{y0,y1,y2}		
br label %bb2				{x0}	{y2}	{z0,z1}	
	L						
L.		%y3 = load i:	22 +22* %/		{x0}n{x0}	{y0}n{y2}	$\{z0,z1\}\cap\{z0,z1\}$
Di)2:				{x0}	{y3}	{z0,z1}
		%y4 = add i3:			{x0}	{y3,y4}	{z0,z1}
		store i32 %y			{x0}	{y4}	{z0,z1}
		%x1 = load i:			${x0,x1}$	{y4}	{z0,z1}
		%y5 = load i			${x0,x1}$	{y4,y5}	{z0,z1}
	%z2 = add i32 %x1, %y5			{x0,x1}	{y4,y5}	{z0,z1,z2}	
	store i32 %z2, i32* %z			{x0,x1}	{y4,y5}	{z2}	
	%z3 = load i32, i32* %z			{x0,x1}	{y4,y5}	{z2,z3}	
		ret i32 %z3			F		

分析结果

		٦		
bb0:	%x = alloca i32			
	%y = alloca i32			
	%z = alloca i32			
	store i32 0, i32* %x			
	store i32 0, i32* %y	S_x	$\boldsymbol{S_{v}}$	S_z
	%x0 = load i32, i32* %x————			
	%y0 = load i32, i32* %y—————	{x0}	{}	{}
	%z0 = add i32 %x0, %y0 ————	{x0}	{y0}	{}
	store i32 %z0, i32* %z —————	{x0}	{y0}	{z0}
	%z1 = load i32, i32* %z	{x0}	{y0}	{z0}
	%t0 = icmp sgt i32 %z1, 0	{x0}	{y0}	{z0, <mark>z1</mark> }
	br i1 %t0, label %bb1, label %bb2			
hh4 .	DI II %CO, IdDCI %DDI, IdDCI %DD2			
bb1:	<u> </u>			
%y1 = lo a	d i32, i32* %y			
%y2 = add	i32 <mark>%y1</mark> , 1	{x0}	{y0, <mark>y1</mark> }	{z0,z1}
store i32	%y2, i32* %y	{x0}	{y0,y1,y2}	{z0,z1}
br label		{x0}	{y2}	{z0,z1}
	0 0 0 1 1 100 100 100	{x0}	{}	{z0,z1}
bt	52: %y3 = load i32, i32* %y	{x0}	{y3}	{z0,z1}
	%y4 = add i32 %y3, 1	{x0}	{y3,y4}	{z0,z1}
	store i32 %y4, i32* %y	{x0}	{y4}	{z0,z1}
	%x1 = load i32, i32* %x	{x0, <mark>x1</mark> }	{y4}	{z0,z1}
	%y5 = load i32<mark>, i</mark>32<mark>* %</mark>y	$\{x0, x1\}$	{y4, <mark>y5</mark> }	{z0,z1}
	%z2 = add i32 <mark>%x1</mark> , <mark>%y5</mark>	{x0,x1}	{y4,y5}	{z0,z1,z2}
	store i32 %z2, i32* %z	{x0,x1}		{z2}
	% z3 = load_i32, i32* %z	{x0,x1}	{y4,y5}	{z2, <mark>z3</mark> }
	ret i32 <mark>%z3</mark>	(,	() .))-)	() <mark></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 instruction n):
    IN[n] = {<v: Ø>: v is a program variable}
    OUT[n] = {<v: Ø>}
Repeat:
    For(each instruction 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* %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

```
%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

```
bb0: | %x = alloca i32 |
                                                    S
          %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}n{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

```
%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
         %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

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

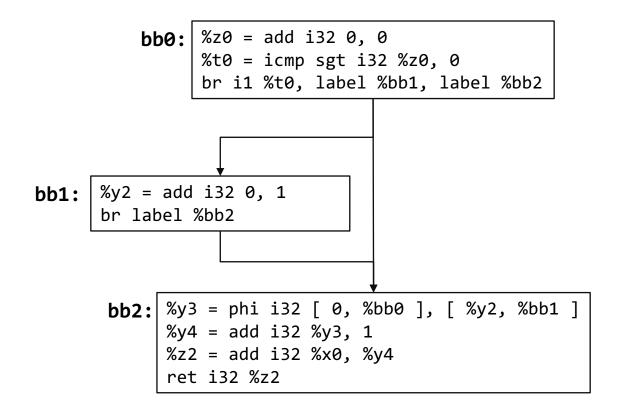
分析过程

bb0:	%x = alloca i32	S_x	S_{y}	S_z
	%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:				
%v2 = add	i32 %v0 1	{0}	{0}	
1 -	<pre>%y2 = add i32 %y0, 1 store i32 %y2, i32* %y br label %bb2</pre>		{y2}	{}
1			{y2}	{}
or raber %bb2		{0}	{y2}	{}
L	2: %y3 = load i32, i32* %y	{0}∪{0}	{0}∪{y2}	
Di	%y4 = add i32 %y3, 1	{0}	{0}∪{y2}	{}
		{0}	{0}∪{y2}	{}
	store i32 %y4, i32* %y	{0}	{y4}	{}
	%z2 = add i32 %x0, %y4 store i32 %z2, i32* %z	{0}	{y4}	{}
	ret i32 %z2	{0}	{y4}	{}
	160 132 /022	{}	{}	{z}

分析结果

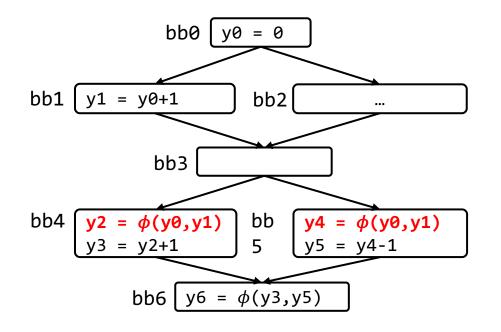
bb0:	%x = alloca i32	S_x	S_y	S_z
	%y = alloca i32	{}	{}	{}
	%z = alloca i32	{}	{}	{}
	store i32 0, i32* %x	{0}	{0}	{}
	store i32 0, i32* %y	{0}	{0}	{}
	%x0 = load i32, i32* %x	{0}	{0}	{}
	y0 = load i32, i32* %y z0 = add i32 <mark>%x0</mark> , <mark>%y0</mark>	{0}	{0}	{}
		{0}	{0}	{}
	%t0 = icmp sgt i32 %z0, 0	{0}	{0}	{}
	br i1 %t0, label %bb1, label %bb2	{0}	{0}	{}
		{0}	{0}	{}
bb1:	↓			
%v2 = adc	i32 <mark>%y0</mark> , 1	{0} {0}	{0}	
1 -	store i32 %y2, i32* %y		{y2}	{}
br label		{0}	{y2}	{}
		{0}	{y2}	{}
		(0)	(0, 0)	
bl	%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} {0}	{y4} {y4}	
	ret i32 %z2	{}	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	{z}
		U	()	

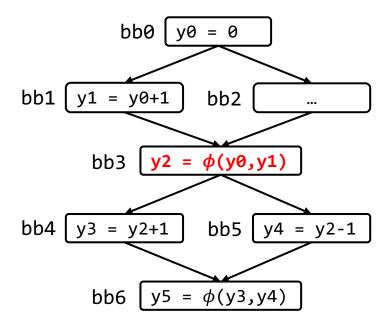
纯寄存器表示



四、Phi指令优化

哪个Phi指令方案更优?

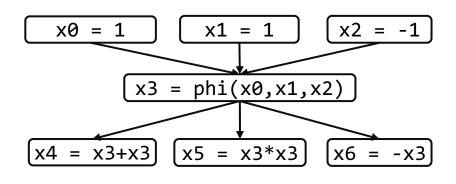




SSA简化def-use关系

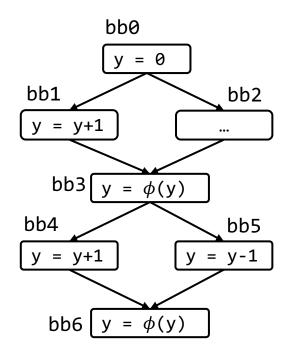
- 原始程序的def-use关系数量是 $O(m \times n)$;
- SSA的def-use数量减少为O(m+n)。

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



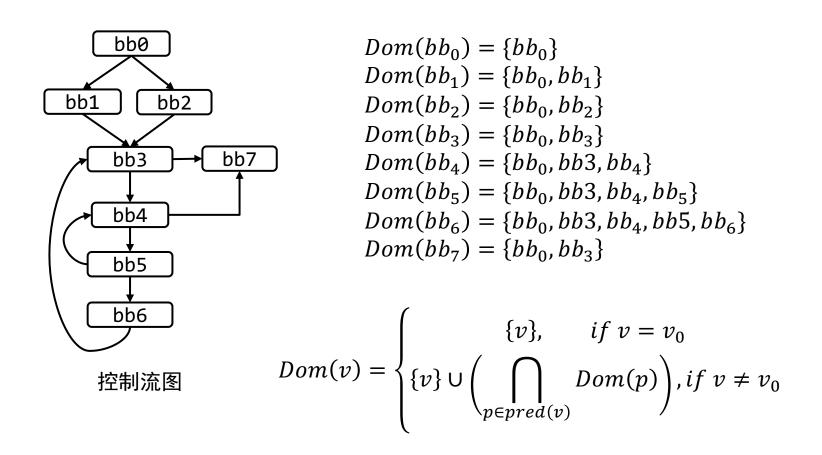
优化思路:基于支配边界优化Phi指令

- 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)



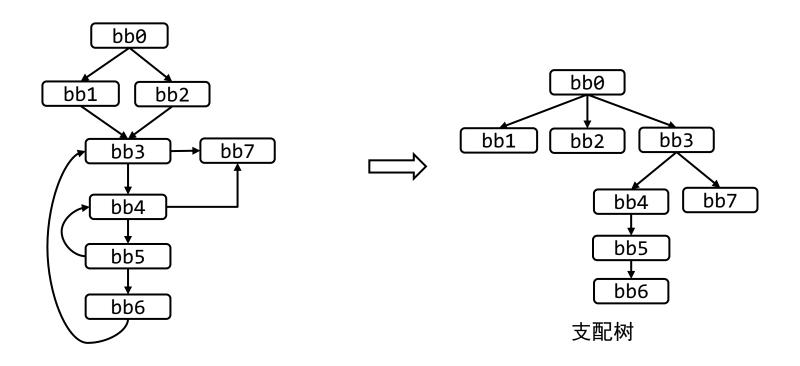
支配的基本概念

- 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_i 或 v_i 是 v_i 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



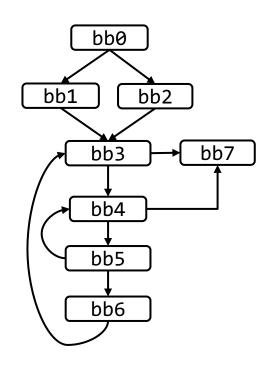
支配树的基本概念

- 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
 - $Idom(v_j) = v_i$, 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_3, bb_4, bb_7\}

DF(bb_5) = \{bb_3, 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_i增加phi节点:
 - $bb_i \in def\text{-sites}(x)$
 - $bb_j \in DF(bb_i)$
- 在bb3增加phi指令的phi(x)

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

DF(bb_1) = \{bb_3\}

DF(bb_2) = \{bb_3\}

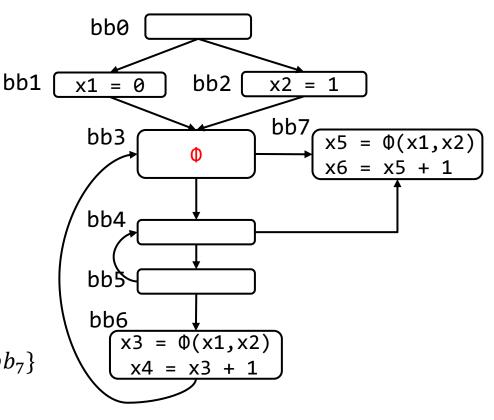
DF(bb_3) = \{bb_3\}

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

