#### 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

```
fn foo(x:int) -> int
 bb0:
        let y:int = 0;
        let 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
          %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

# 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
```

- 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

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

# 分析过程

	%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		\$\cdot S_\chi \{\times 0\} \{\times 0\}	<i>S</i> <sub>y</sub> {} {y0}	<i>S<sub>z</sub></i> {} {}
	%z0 = add i32 %x0, %y0		{x0}	{y0}	{}
	store i32 %z0, i32* %z		{x0}	{y0}	{z0}
	%z1 = load i32, i32* %z		{x0}	{y0}	{z0,z1}
	%t0 = icmp sgt i32 %z1, 0				
	or i1 %t0, label %bb1, label	%bb2			•
bb1:	▼				
%y1 = load	i32, i32* %y		{x0}	{y0}	{z0,z1}
_	i32 %y1, 1		{x0}	{y0,y1}	{z0,z1}
	%y2, i32* %y		{x0}	{y0,y1}	{z0,z1}
br label %	- I I		{x0}	{y2}	{z0,z1}
hh2	%y3 = load i32, i32* %y		{x0}n{x0}	{y0}n{y2}	$\{z0,z1\}\cap\{z0,z1\}$
002	%y4 = add i32 %y3, 1		{x0}	{y3}	{z0,z1}
	store i32 %y4, i32* %y —		{x0}	{y3}	{z0,z1}
	%x1 = load i32, i32* %x		{x0}	{y4}	{z0,z1}
	%x1 = 10dd 132, 132* %x  %y5 = load i32, i32* %y		{x0,x1}	{y4}	{z0,z1}
	%z2 = add i32 %x1, %y5		{x0,x1}	{y4,y5}	{z0,z1}
	store i32 %z2, i32* %z —		{x0,x1}	{y4,y5}	{z0,z1}
	%z3 = load i32, i32* %z—		{x0,x1}	{y4,y5}	{z2}
	ret i32 %z3		${x0,x1}$	{y4,y5}	{z2,z3} 6

# 分析结果

% % % % % % % % % % % % % % % % % % %	<pre>6x = alloca i32 6y = alloca i32 6z = alloca i32 6tore i32 %0, i32* %x 6tore i32 0, i32* %y 6x0 = load i32, i32* %y 6x0 = add i32, i32* %y 6x1 = add i32 %x0, %y0 6x1 = icmp sgt i32 %z1, 0 6x1 = icmp sgt i32 %z1, 0 6x1 = icmp sgt i32 %z1, 0 6x2 = icmp sgt i32 %z1, 0 6x3 = icmp sgt i32 %z1, 0 6x3 = icmp sgt i32 %z1, 0 6x4 = icmp sgt i32 %z1, 1abel %bb1</pre>	\$\cdot \x\0\} \{\times \0\}	\$\square\$ \{\footnote{y0}\} \{\footnote{y0}\} \{\footnote{y0}\} \{\footnote{y0}\} \{\footnote{y0}\}	<pre> {} {} {} {} {z0} {z0,z1}</pre>
	<u>▼</u>			
1 -	<del>_i32<sub>5</sub>_i</del> 32* %y	{x0}	[va v1]	[-0 -1]
1 -	i32 <mark>%y1</mark> , 1	{x0}	{y0,y1}	{z0,z1}
1	%y2, i32* %y	{x0}	{y0,y1} {y2}	{z0,z1} {z0,z1}
br label %b	Db2	(80)	(y 2 )	(20,21)
		{x0}n{x0}	{y0}∩{y2}	{z0,z1}∩{z0,z1}
bb2	:   %y3 = load i32, i32* %y	{x0}	{y3}	{z0,z1}
	%y4 = add i32 %y3, 1	{x0}	{y3}	{z0,z1}
	store i32 %y4, i32* %y	{x0}	{y4}	{z0,z1}
	%x1 = load i32, i32* %x	${x0,x1}$	{y4}	{z0,z1}
	%y5 = load i32, i32* %y	${x0,x1}$	{y4,y5}	{z0,z1}
	%z2 = add i32 %x1, %y5	{x0,x1}	{y4,y5}	{z0,z1}
	store i32 %z2, i32* %z <del>%z3 = load i32, i32* %z</del>	{x0,x1}	{y4,y5}	{z2}
	ret i32 <mark>%z3</mark>	{x0,x1}	{y4,y5}	$\{z2,z3\}$

#### 优化结果

```
%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
```

# 遇到循环: 基于循环迭代的数据流分析

```
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
```

#### 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
hh2:
    %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
hh3:
    %t8 = load i32, i32* %r
    ret i32 %t8
                                     10
```

# 线性IR中的store冗余

```
fn foo(x:int) -> int
bb0:
      let y:int = 0;
      let z:int = x + y;
      if(z>0)
 bb1:
 y = y + 1;
    bb2: | y = y + 1;
          z = x + y;
          ret z;
```

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

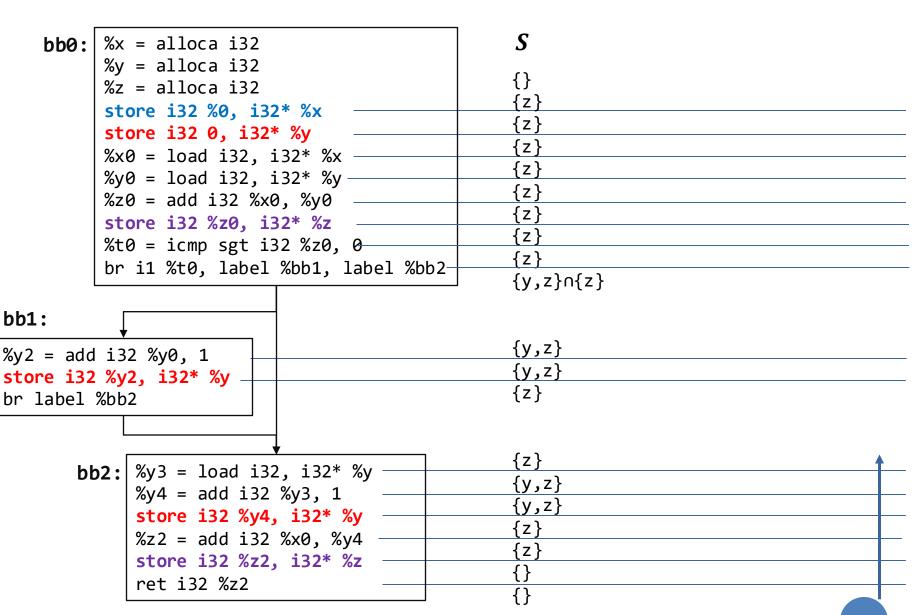
# 优化思路:可用store语句分析

```
%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
    - $S = S \setminus \{x\}$
  - %x = alloca i32
    - $S = S \setminus \{x\}$
- 遇到合并节点

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

# 分析过程



# 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
          %x0 = load i32, i32* %x -
          %y0 = load i32, i32* %y —
                                                   {z}
         %z0 = add i32 %x0, %y0 -----
                                                   {z}
          store i32 %z0, i32* %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
             store i32 %y4, i32* %y
                                                   {z}
             %z2 = add i32 %x0, %y4
             store i32 %z2, i32* %z
             ret i32 %z2
```

#### 优化结果

```
%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
      %y2 = add i32 %y0, 1
bb1:
      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
```

# 二、纯寄存器表示

# 消除数据存取

```
%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
      %y2 = add i32 %y0, 1
bb1:
      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: 内存数值流分析

```
%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
```

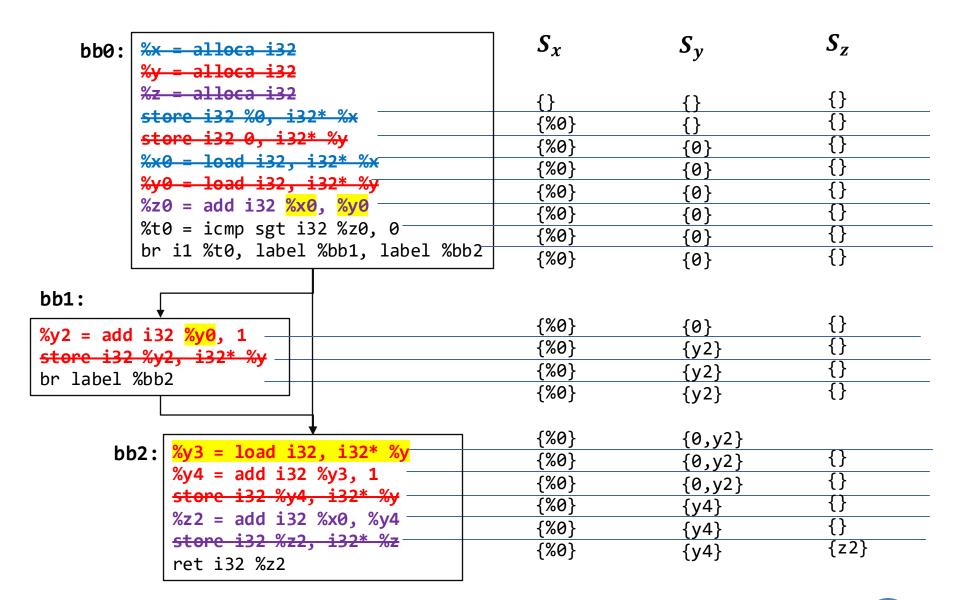
- 正向遍历控制流图
- Transfer函数定义:
  - store i32 %t, i32\* %x
    - $S_{x} = \{t\}$
- 遇到合并节点

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

# 分析过程

bb0:	%x = alloca i32	$S_{x}$	$S_{v}$	$S_z$
	%y = alloca i32		J	
	%z = alloca i32	{}	{}	{}
	store i32 %0, i32* %x	\ <i>\</i>		{}
	store i32 0, i32* %y		\\ {0}	{}
	%x0 = load i32, i32* %x		<u> </u>	{}
	%y0 = load i32, i32* %y	· · ·	{0}	
	%z0 = add i32 %x0, %y0	{%0} {%0}	{0}	{}
	%t0 = icmp sgt i32 %z0, 0	{%0}	{0}	{}
	br i1 %t0, label %bb1, label %bb2		{0} {0}	<del>{}</del> {}
		\%V)	(a)	ſ,
bb1:				
<b>O</b> 4 <b>O</b> 4	<del>*</del>	<b>{%0}</b>	{0}	{}
1 -	l i32 %y0, 1	{%0}	{y2}	{}
l .	2 %y2, i32* %y	{% <b>0</b> }	{y2}	{}
br label	%DD2	{%0}	{y2}	{}
		,		C
	<b>▼</b>	{%0}∪{%0}	{0}∪{y2}	
bl	b2:   %y3 = load i32, i32* %y	{%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	{}	{}	{z2}

#### 分析结果



# 纯寄存器表示

```
bb0: | %z0 = add i32 %0, 0
                %t0 = icmp sgt i32 %z0, 0
                br i1 %t0, label %bb1, label %bb2
      %y2 = add i32 0, 1
bb1:
      br label %bb2
       bb2: | %y3 = phi i32 [0, %bb0], [%y2, %bb1]
            %y4 = add i32 %y3, 1
            %z2 = add i32 %x0, %y4
             ret i32 %z2
```

# 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
```

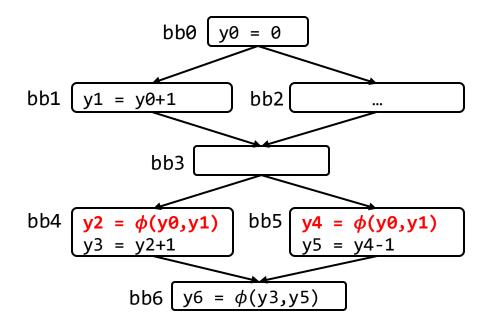
# 数据流分析方法小结

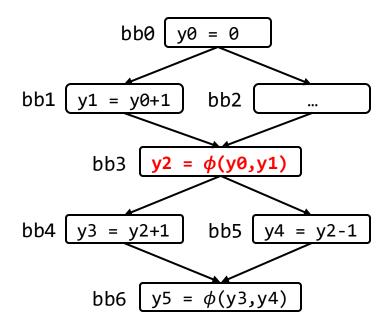
	May Analysis (U)	Must Analysis (∩)
前向分析	纯寄存器表示	精简load
逆向分析	活跃性分析	精简store

# 数据流分析方法小结: 框架

# 三、Phi指令优化

# 哪个phi指令方案更优?

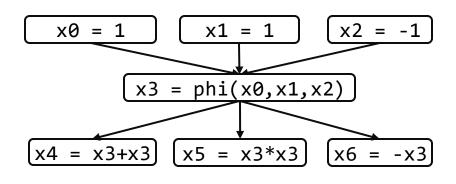




# SSA简化def-use关系

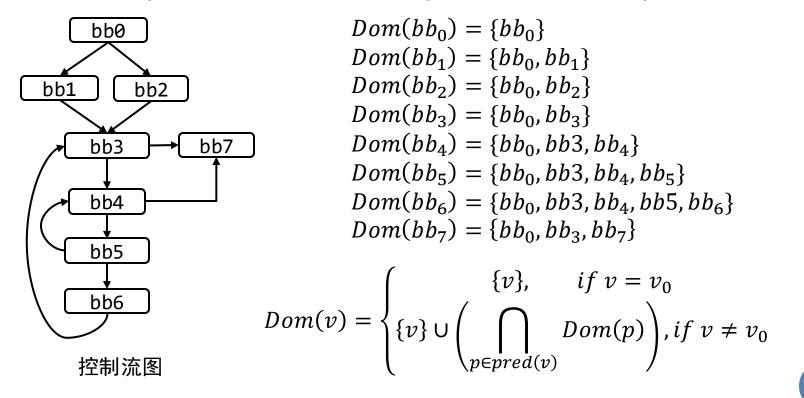
- 原始程序的def-use关系数量是 $O(m \times n)$ ;
- SSA的def-use数量减少为O(m+n)。

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



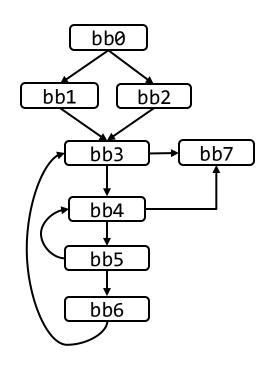
# 支配的基本概念

- 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 $v_i$ 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 $v_j$ , $v_i \in IDom(v_j)$



# 支配边界

- $v_i$ 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\}

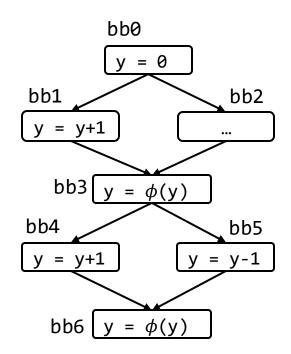
DF(bb_5) = \{bb_3, bb_4\}

DF(bb_6) = \{bb_3\}

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

# 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<sub>i</sub>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\}

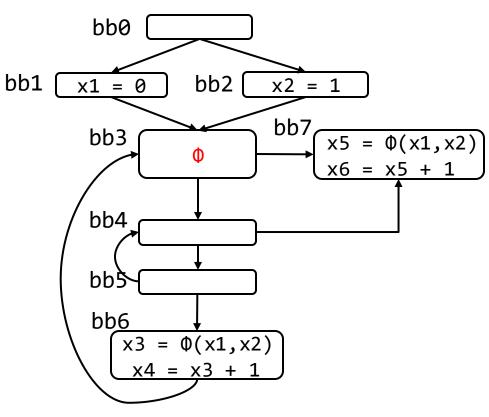
DF(bb_3) = \{bb_3\}

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

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

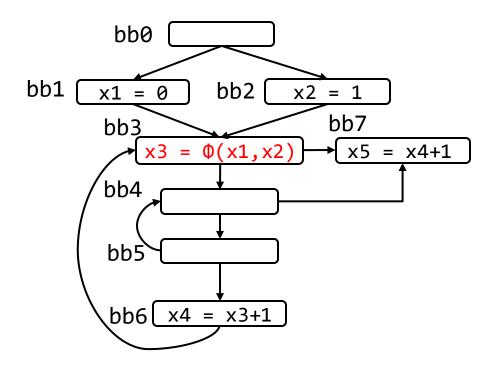
DF(bb_6) = \{bb_3\}

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

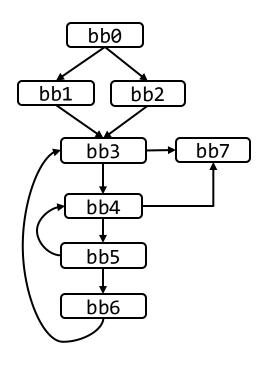


# 优化结果

- 重新编号
- 删除只有一个元素的phi指令



# 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) = \{bb3, 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) = \{\}
                                                     \forall bb_i \in Dom(bb_0) - IDom(bb_1),
Predecessor(bb_1) = \{bb_0\}
Predecessor(bb_2) = \{bb_0\}
                                                        bb_1 \in DF(bb_i)
Predecessor(bb_3) = \{bb_1, bb_2, bb_6\}
Predecessor(bb_4) = \{bb3, bb_5\}
Predecessor(bb_5) = \{bb_4\}
Predecessor(bb_6) = \{bb_5\}
                                                     DF(bb_0) = \{\}
Predecessor(bb_7) = \{bb_3, bb_4\}
                                                     DF(bb_1) = \{bb_3\}
                                                     DF(bb_2) = \{bb_3\}
                                                     DF(bb_3) = \{bb_3\}
Dom(bb_0) = \{bb_0\}
                                                     DF(bb_4) = \{bb3, bb_4, bb_7\}
Dom(bb_1) = \{bb_0, bb_1\}
                                                     DF(bb_5) = \{bb3, bb_4\}
Dom(bb_2) = \{bb_0, bb_2\}
                                                     DF(bb_6) = \{bb_3\}
Dom(bb_3) = \{bb_0, bb_3\}
                                                     DF(bb_{7}) = \{\}
Dom(bb_4) = \{bb_0, bb3, bb_4\}
Dom(bb_5) = \{bb_0, bb3, bb_4, bb_5\}
Dom(bb_6) = \{bb_0, bb3, bb_4, bb5, bb_6\}
Dom(bb_7) = \{bb_0, bb_3\}
```

#### 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

