EECS 583 – Class 7
Static Single Assignment Form

University of Michigan

September 17, 2025

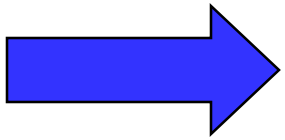
Announcements & Reading Material

- HW2 out today, due Oct 8
  - Spec and starting code are available on course webpage
  - Short lecture today by Naveen to go over the homework
  - Also check out piazza post by Naveen/Rishika
- Today's class
  - "Practical Improvements to the Construction and Destruction of Static Single Assignment Form," P. Briggs, K. Cooper, T. Harvey, and L. Simpson, *Software--Practice and Experience*, 28(8), July 1998, pp. 859-891.
- Next class – Optimization, Yay!
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988, 9.9, 10.2, 10.3, 10.7 Edition 1; 8.5, 8.7, 9.1, 9.4, 9.5 Edition 2

From Last Time: Converting to SSA Form

- Trivial for straight line code

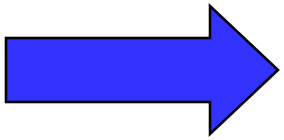
```
x = -1
y = x
x = 5
z = x
```



```
x0 = -1
y = x0
x1 = 5
z = x1
```

- More complex with control flow – Must use Phi nodes

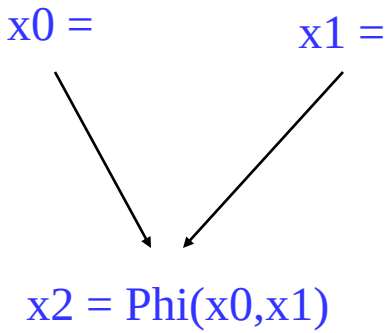
```
if ( ... )
   x = -1
else
   x = 5
y = x
```



```
if ( ... )
   x0 = -1
else
   x1 = 5
x2 = Phi(x0,x1)
y = x2
```

From Last Time: Phi Nodes (aka Phi Functions)

- Special kind of copy that selects one of its inputs
- Choice of input is governed by the CFG edge along which control flow reached the Phi node

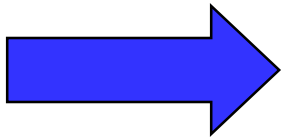


- Phi nodes are required when 2 non-null paths X→Z and Y→Z converge at node Z, and nodes X and Y contain assignments to V

From Last Time: Converting to SSA Form (2)

- What about loops?
  - No problem!, use Phi nodes again

```
i = 0
do {
   i = i + 1
}
while (i < 50)
```



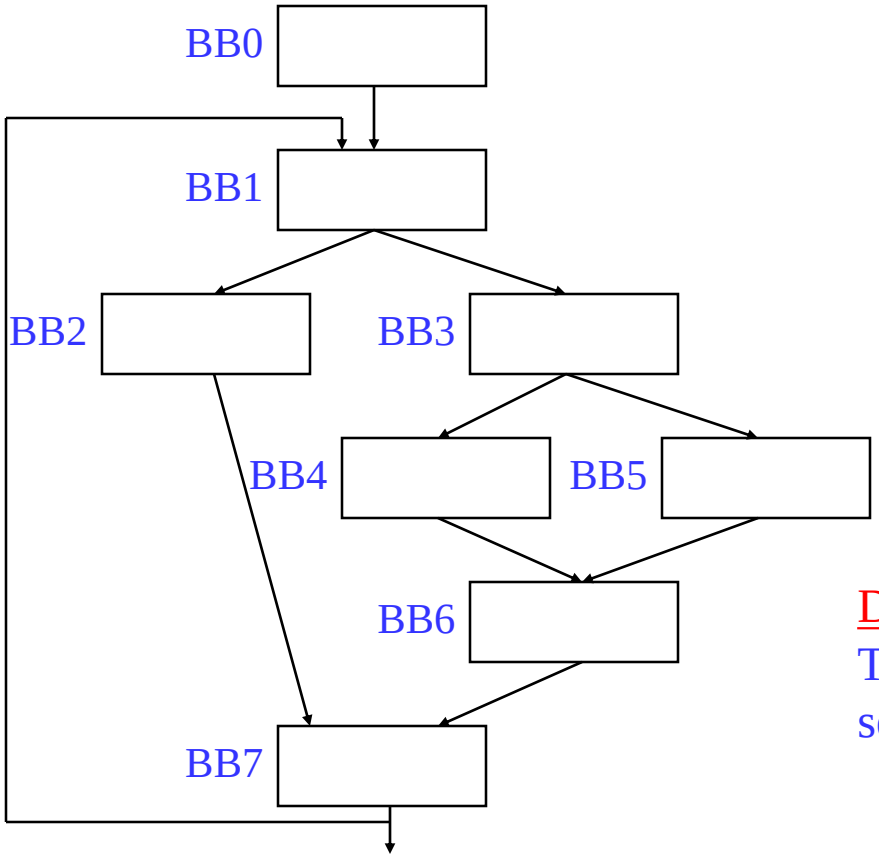
```
i0 = 0
do {
   i1 = Phi(i0, i2)
   i2 = i1 + 1
}
while (i2 < 50)
```

SSA Construction

- High-level algorithm
  1. Insert Phi nodes
  2. Rename variables
- A dumb algorithm
  - Insert Phi functions at every join for every variable
  - Solve reaching definitions
  - Rename each use to the def that reaches it (will be unique)
- Problems with the dumb algorithm
  - Too many Phi functions (precision)
  - Too many Phi functions (space)
  - Too many Phi functions (time)

Need Better Phi Node Insertion Algorithm

- A definition at n forces a Phi node at m iff n not in DOM(m), but n in DOM(p) for some predecessors p of m



def in BB4 forces Phi in BB6
def in BB6 forces Phi in BB7
def in BB7 forces Phi in BB1

Phi is placed in the block that
is just outside the dominated region
of the definition BB

Dominance frontier

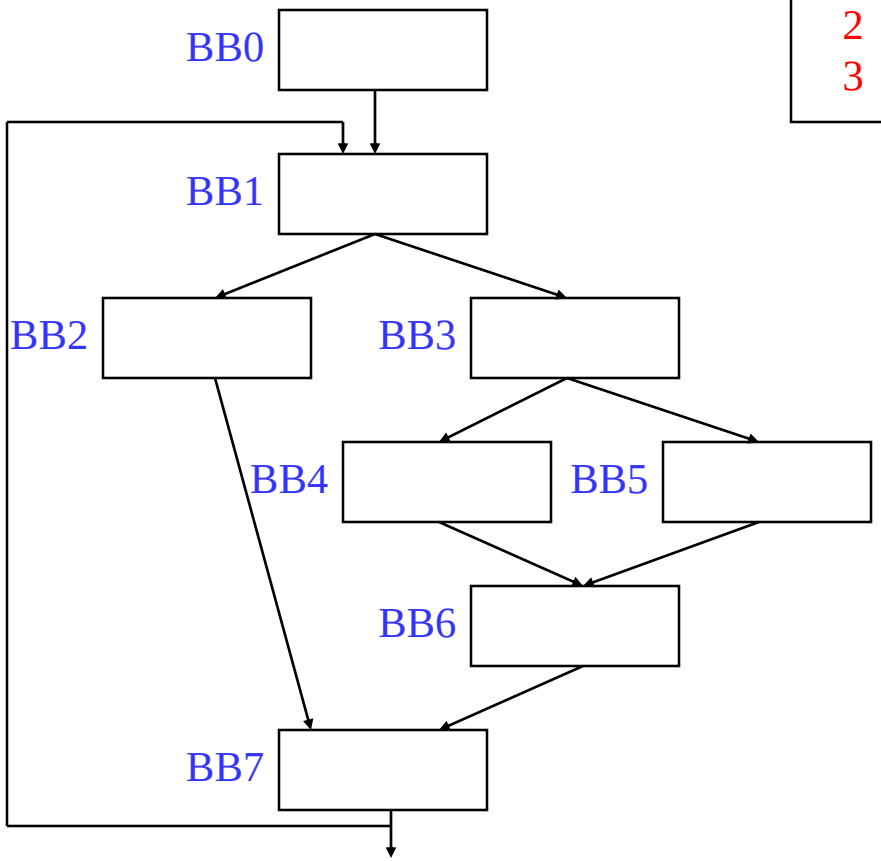
The dominance frontier of node X is the set of nodes Y such that

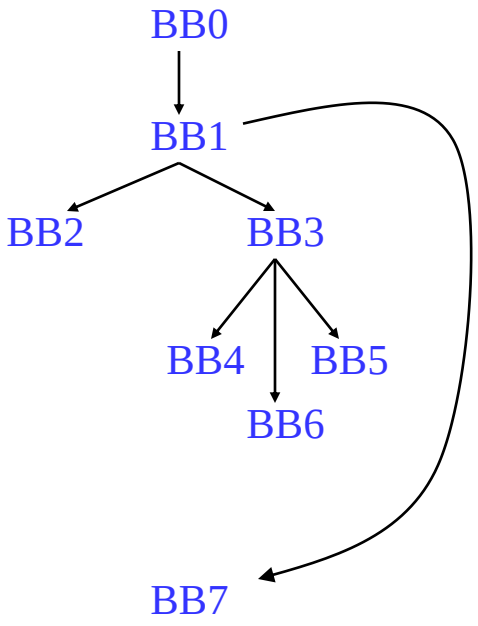
* X dominates a predecessor of Y, but
* X does not strictly dominate Y

Recall: Dominator Tree

First BB is the root node, each node dominates all of its descendants

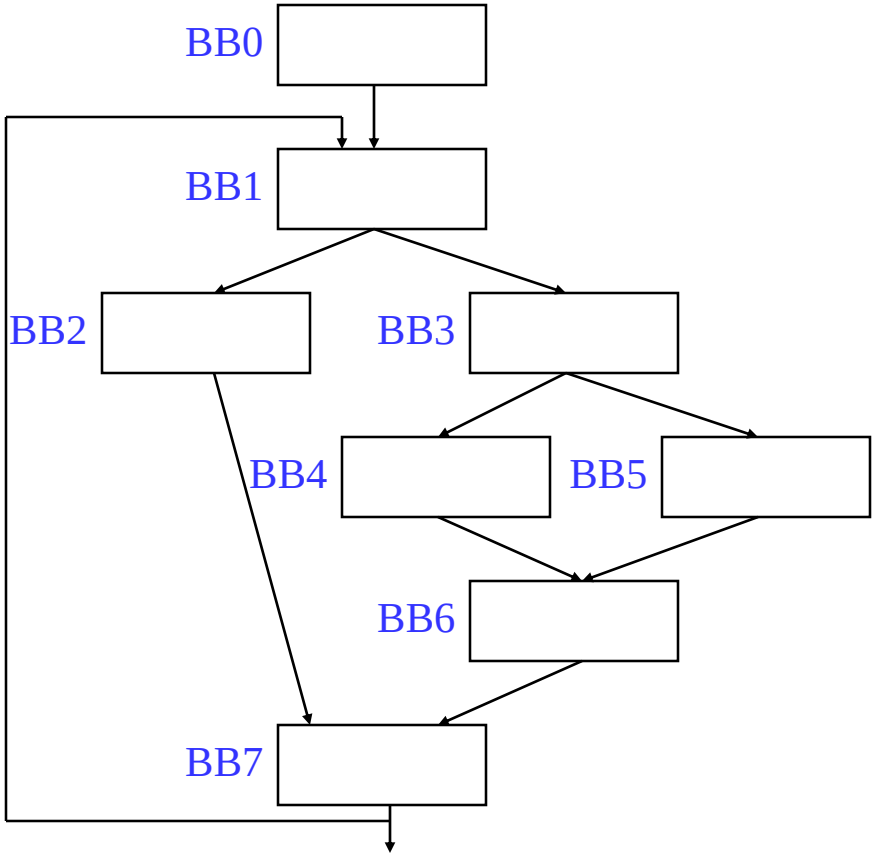
BB	DOM	BB	DOM
0	0	4	0,1,3,4
1	0,1	5	0,1,3,5
2	0,1,2	6	0,1,3,6
3	0,1,3	7	0,1,7

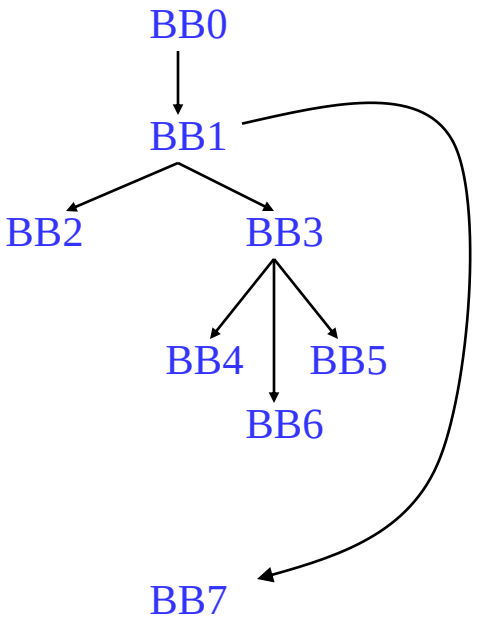




Dom tree

Computing Dominance Frontiers

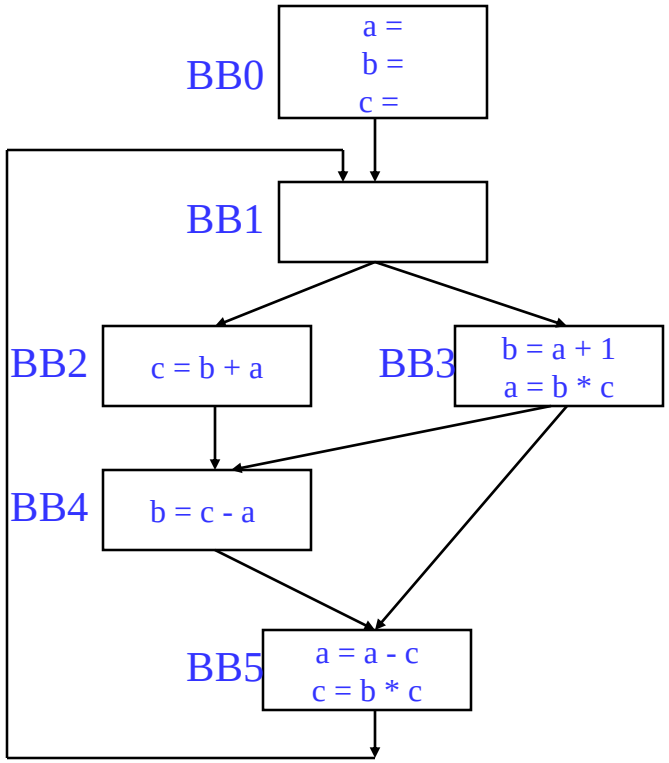




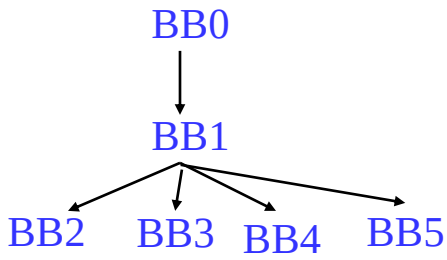
BB	DF
0	
1	
2	
3	
4	
5	
6	
7	

For each join point X in the CFG
  For each predecessor, Y, of X in the CFG
    Run up to the IDOM(X) in the dominator tree, adding X to DF(N) for each N between Y and IDOM(X) (or X, whichever is encountered first)

Class Problem – Compute DF for each BB



Dominator Tree



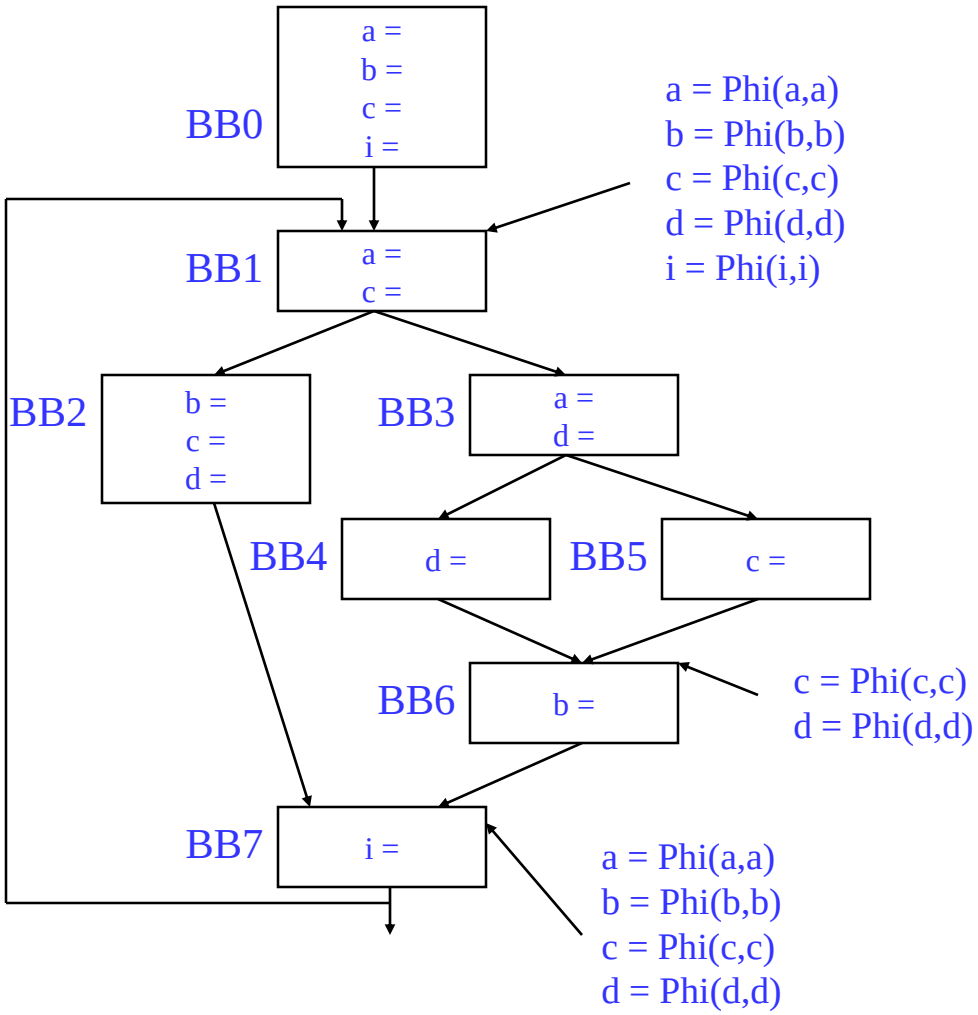
For each join point X in the CFG
  For each predecessor, Y, of X in the CFG
    Run up to the IDOM(X) in the dominator tree, adding X to DF(N) for each N between Y and IDOM(X) (or X, whichever is encountered first)

SSA Step 1 - Phi Node Insertion

- Compute dominance frontiers
- Find global names (aka virtual registers)
  - Global if name live on entry to some block
  - For each name, build a list of blocks that define it
- Insert Phi nodes
  - For each global name n
    - For each BB b in which n is defined
      - For each BB d in b's dominance frontier
        - Insert a Phi node for n in d
        - Add d to n's list of defining BBs

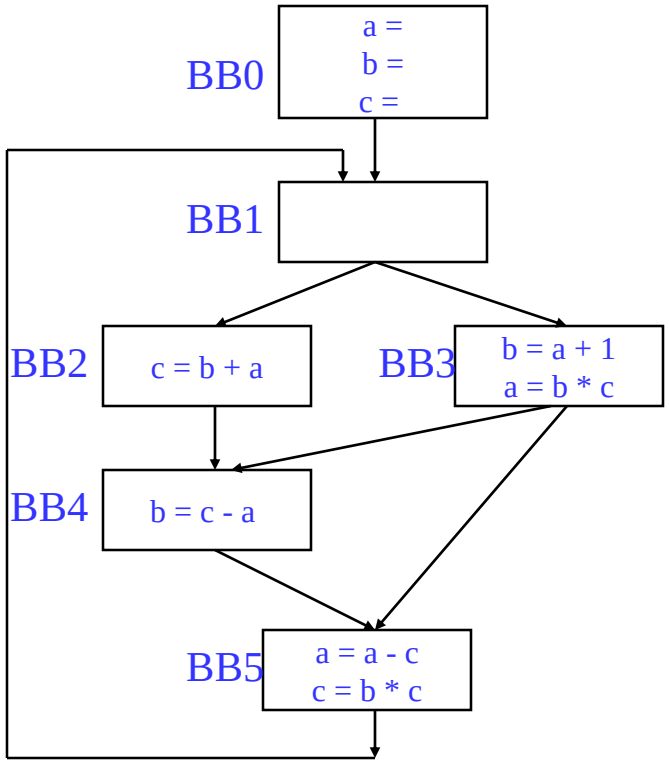
Phi Node Insertion - Example

BB	DF
0	-
1	-
2	7
3	7
4	6
5	6
6	7
7	1

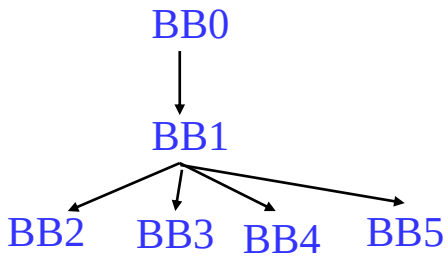


a is defined in 0,1,3
need Phi in 7
then a is defined in 7
need Phi in 1
b is defined in 0, 2, 6
need Phi in 7
then b is defined in 7
need Phi in 1
c is defined in 0,1,2,5
need Phi in 6,7
then c is defined in 7
need Phi in 1
d is defined in 2,3,4
need Phi in 6,7
then d is defined in 7
need Phi in 1
i is defined in BB7
need Phi in BB1

Class Problem – Insert the Phi Nodes



Dominator tree



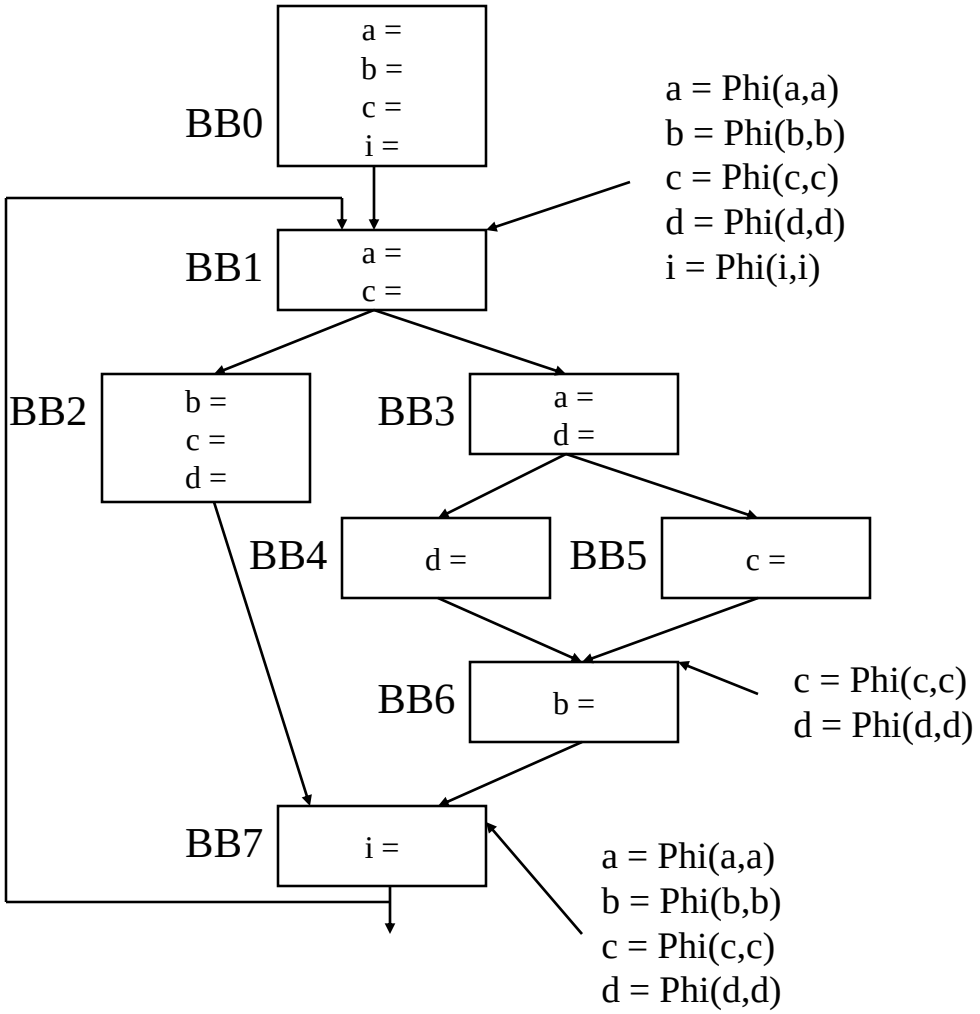
Dominance frontier

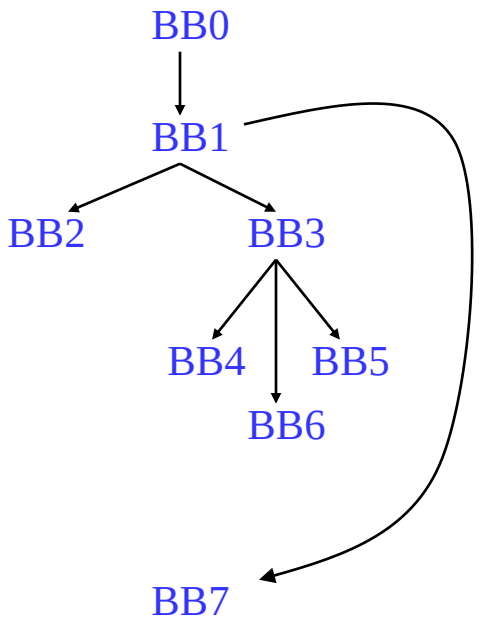
BB	DF
0	-
1	-
2	4
3	4, 5
4	5
5	1

SSA Step 2 – Renaming Variables

- Use an array of stacks, one stack per global variable (VR)
- Algorithm sketch
  - For each BB b in a preorder traversal of the dominator tree
    - Generate unique names for each Phi node
    - Rewrite each operation in the BB
      - Uses of global name: current name from stack
      - Defs of global name: create and push new name
    - Fill in Phi node parameters of successor blocks
    - Recurse on b's children in the dominator tree
    - <on exit from b> pop names generated in b from stacks

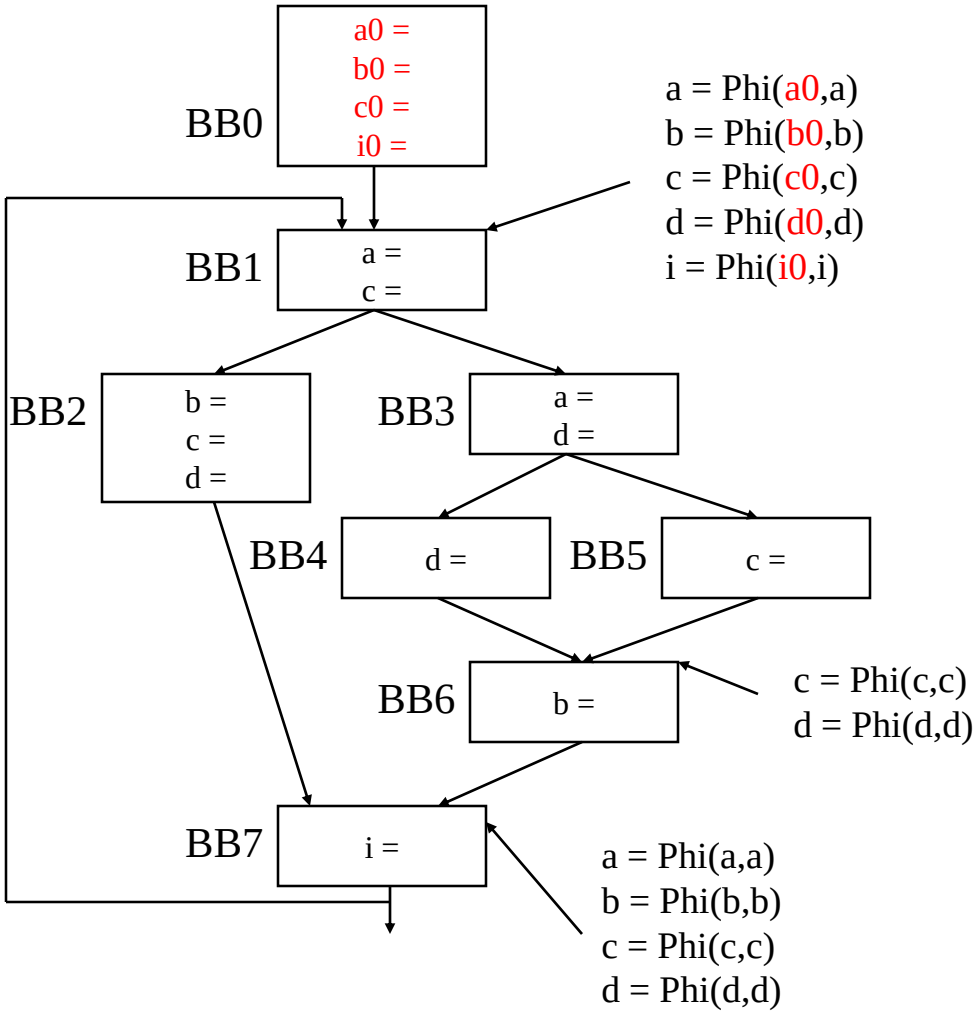
Renaming – Example (Initial State)

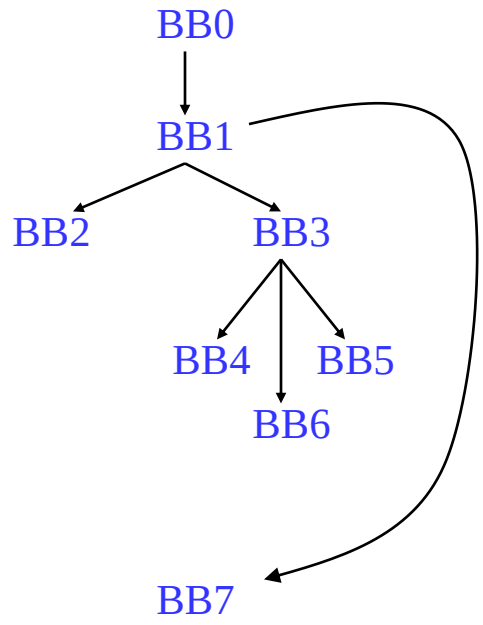




var:	a	b	c	d	i
ctr:	0	0	0	0	0
stk:	a0	b0	c0	d0	i0

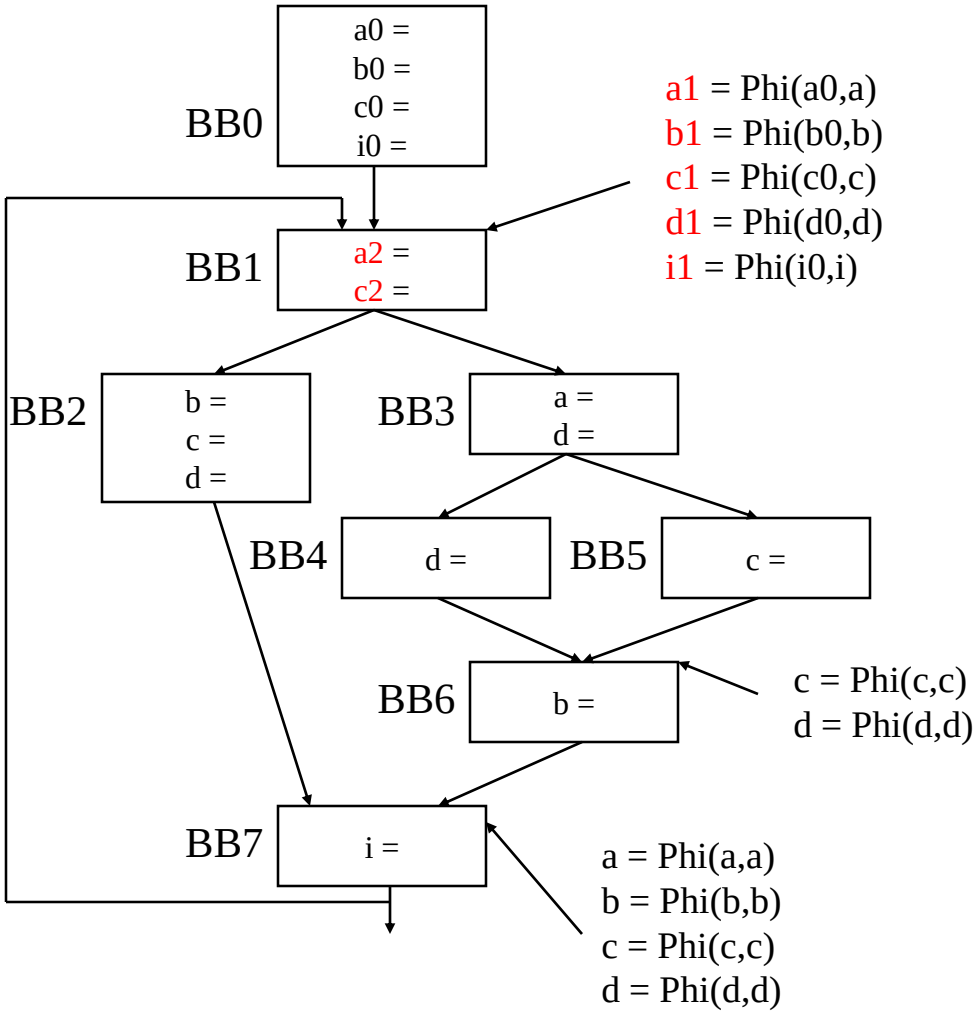
Renaming – Example (After BB0)

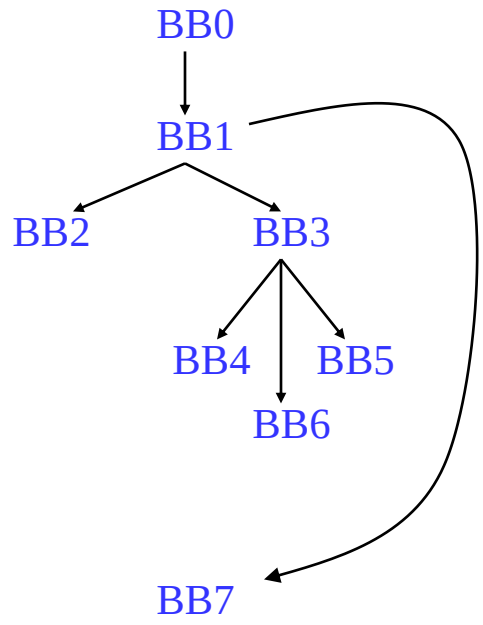




var:	a	b	c	d	i
ctr:	1	1	1	1	1
stk:	a0	b0	c0	d0	i0

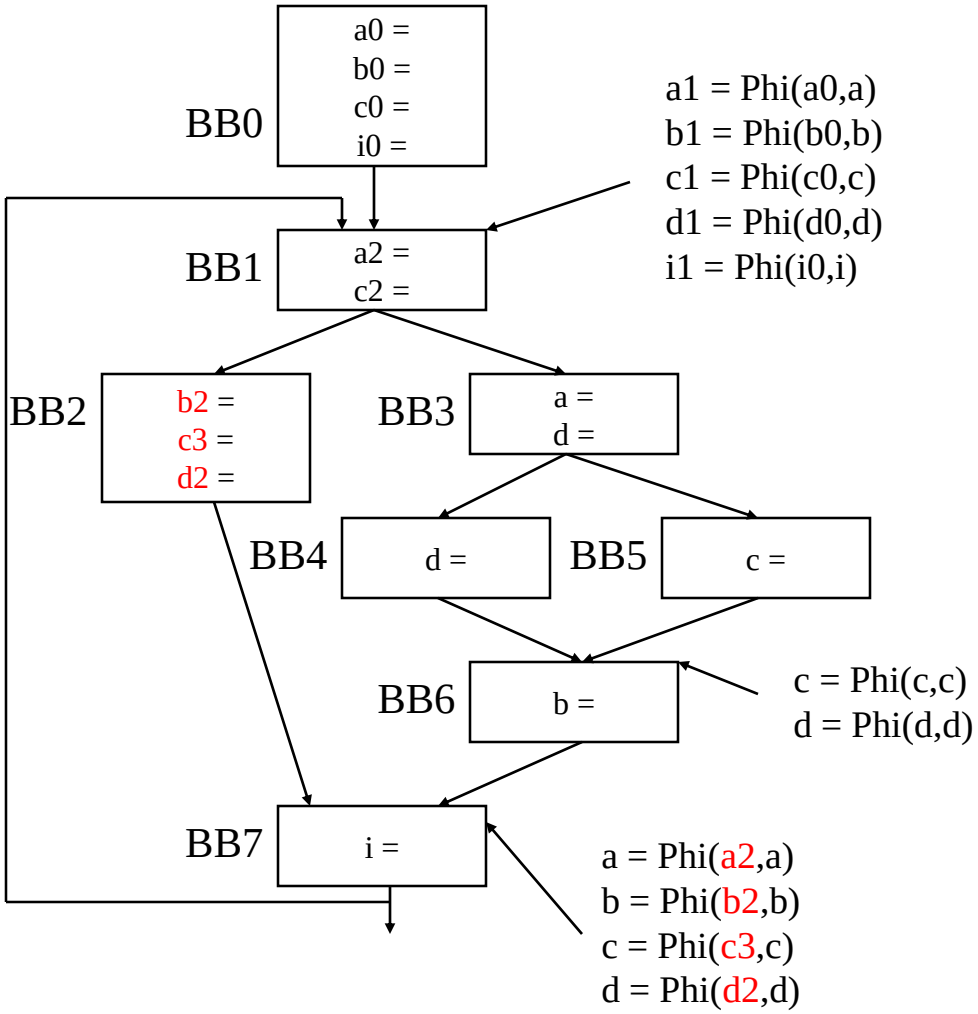
Renaming – Example (After BB1)

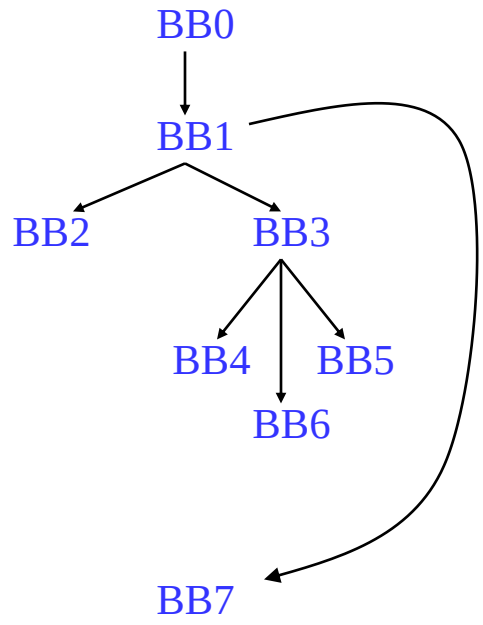




var:	a	b	c	d	i
ctr:	3	2	3	2	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2		

Renaming – Example (After BB2)

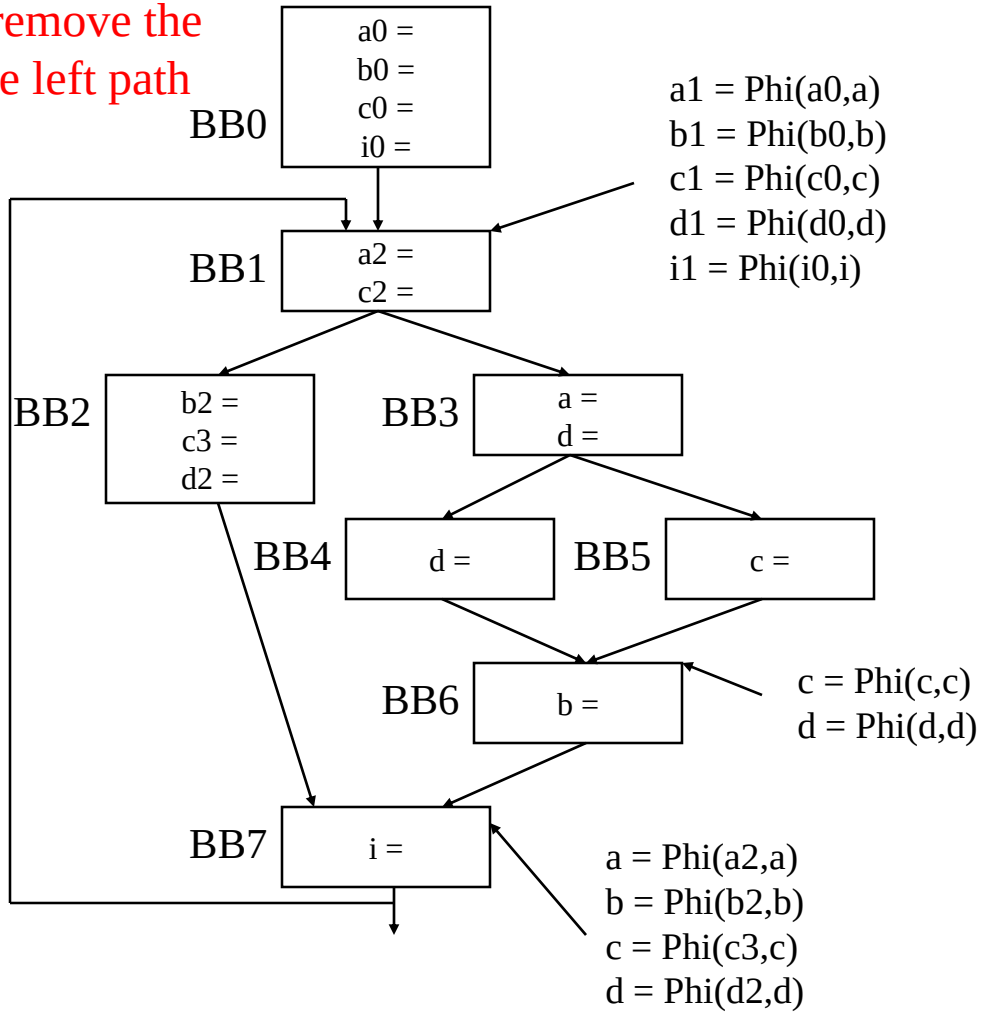


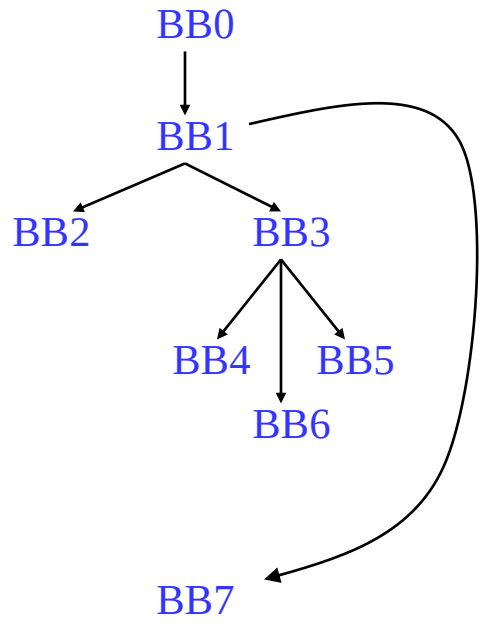


var:	a	b	c	d	i
ctr:	3	3	4	3	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b2	c2	d2	
			c3		

Renaming – Example (Before BB3)

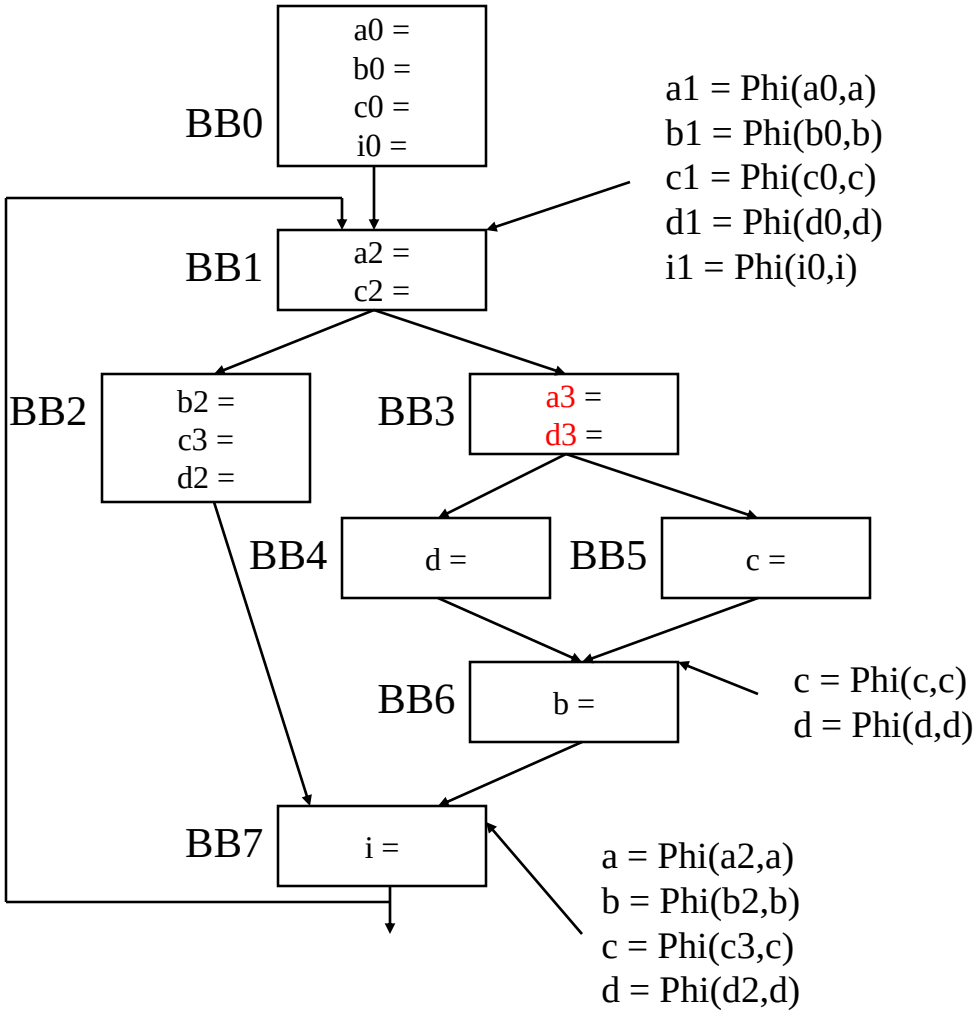
This just updates the stack to remove the stuff from the left path out of BB1

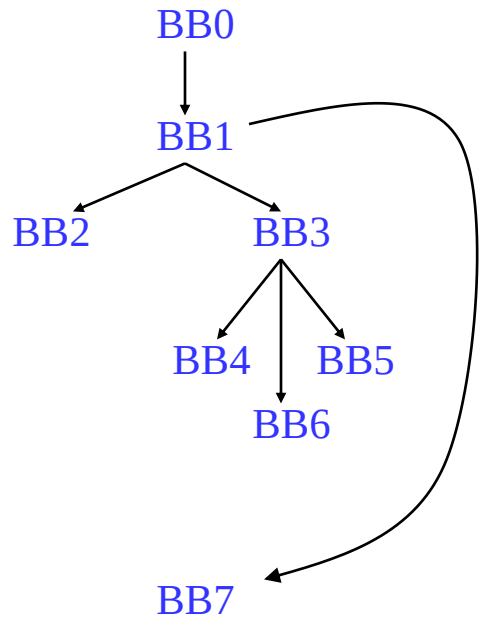




var:	a	b	c	d	i
ctr:	3	3	4	3	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2		

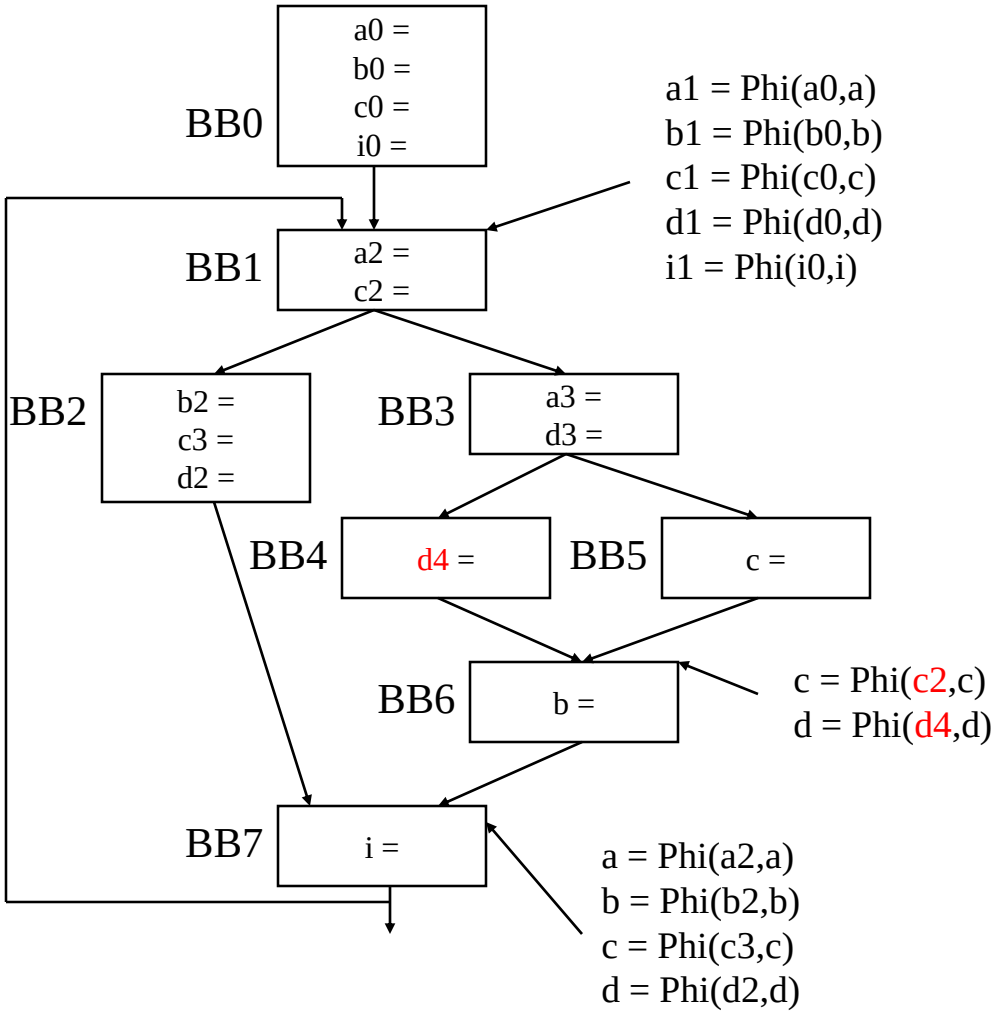
Renaming – Example (After BB3)

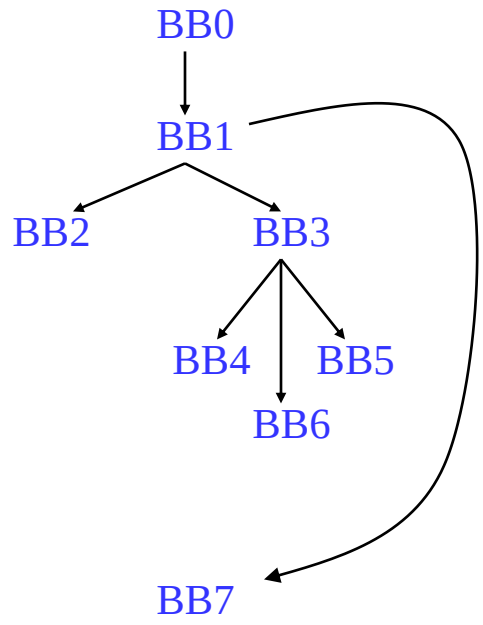




var:	a	b	c	d	i
ctr:	4	3	4	4	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3				

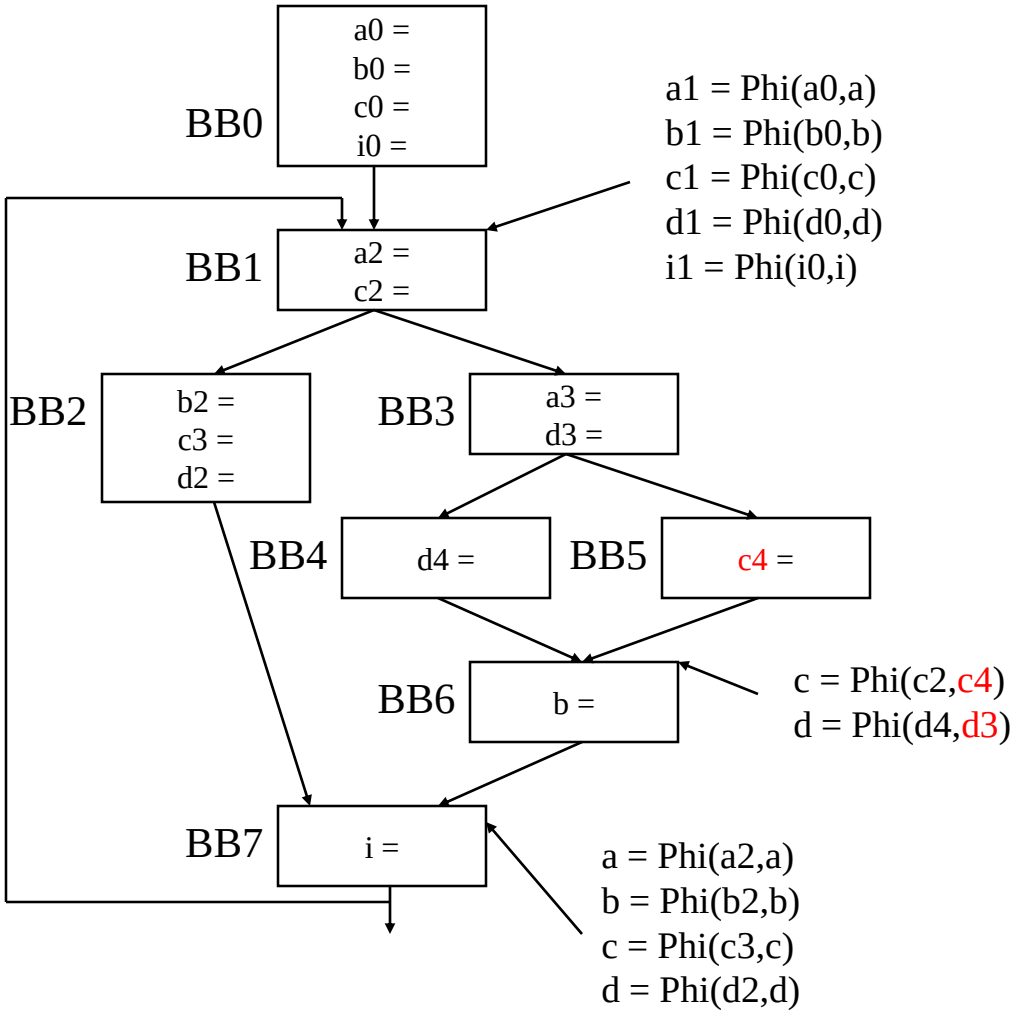
Renaming – Example (After BB4)

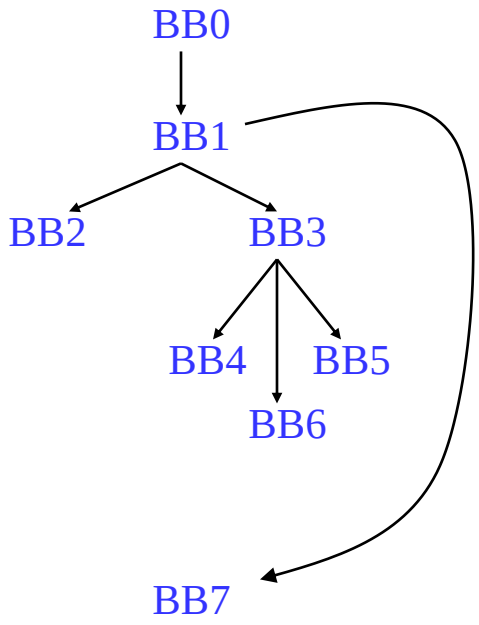




var:	a	b	c	d	i
ctr:	4	3	4	5	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3			d4	

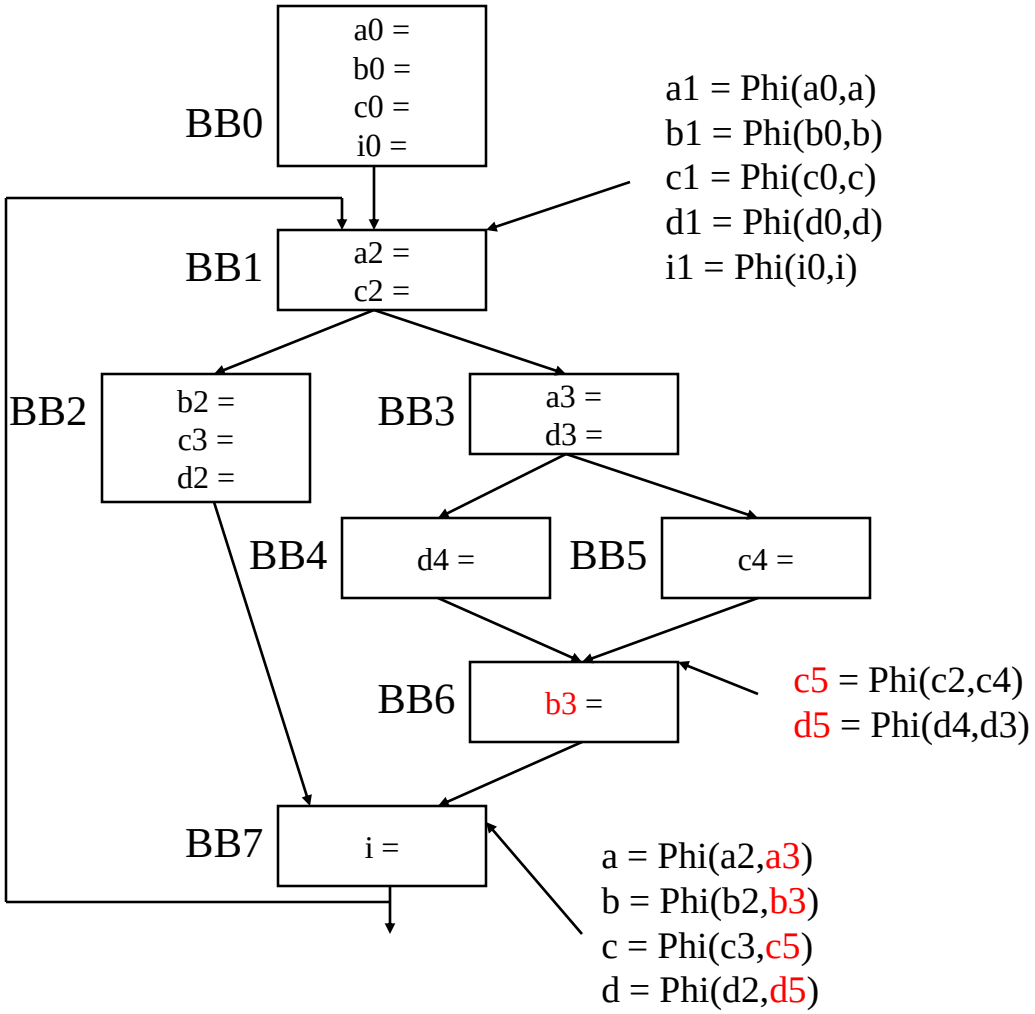
Renaming – Example (After BB5)

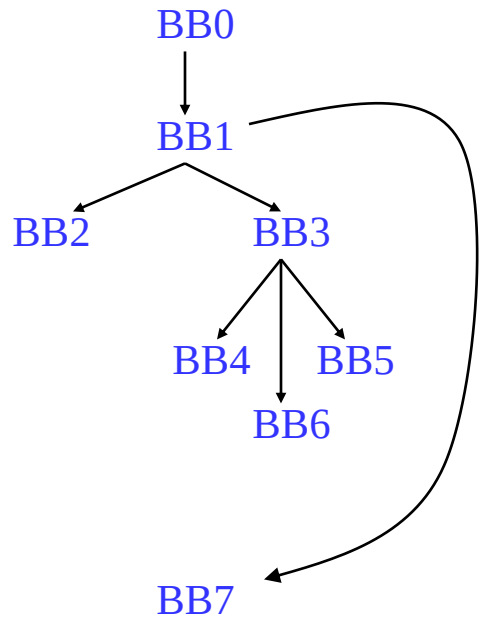




var:	a	b	c	d	i
ctr:	4	3	5	5	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2		c2	d3	
	a3		c4		

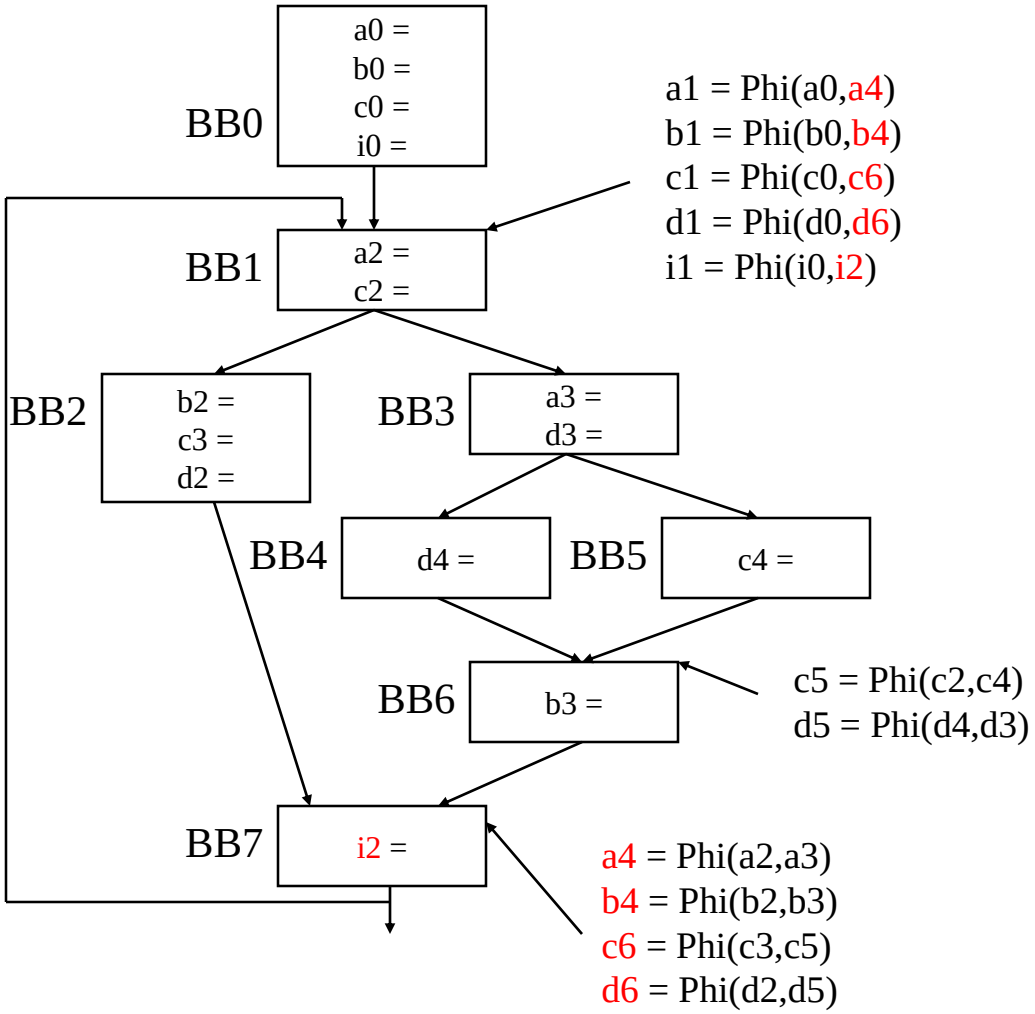
Renaming – Example (After BB6)

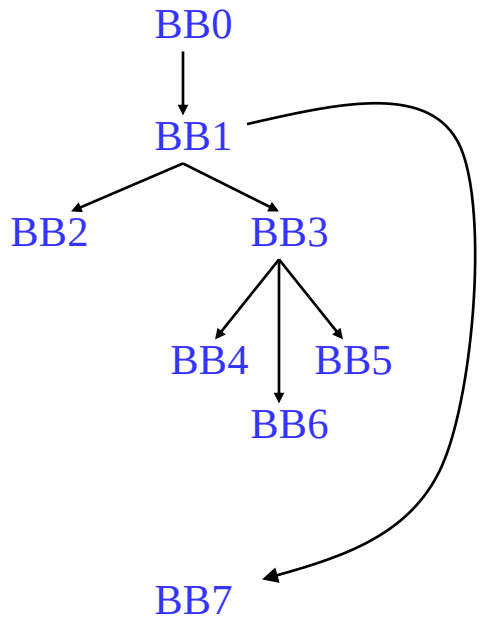




var:	a	b	c	d	i
ctr:	4	4	6	6	2
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b3	c2	d3	
	a3		c5	d5	

Renaming – Example (After BB7)

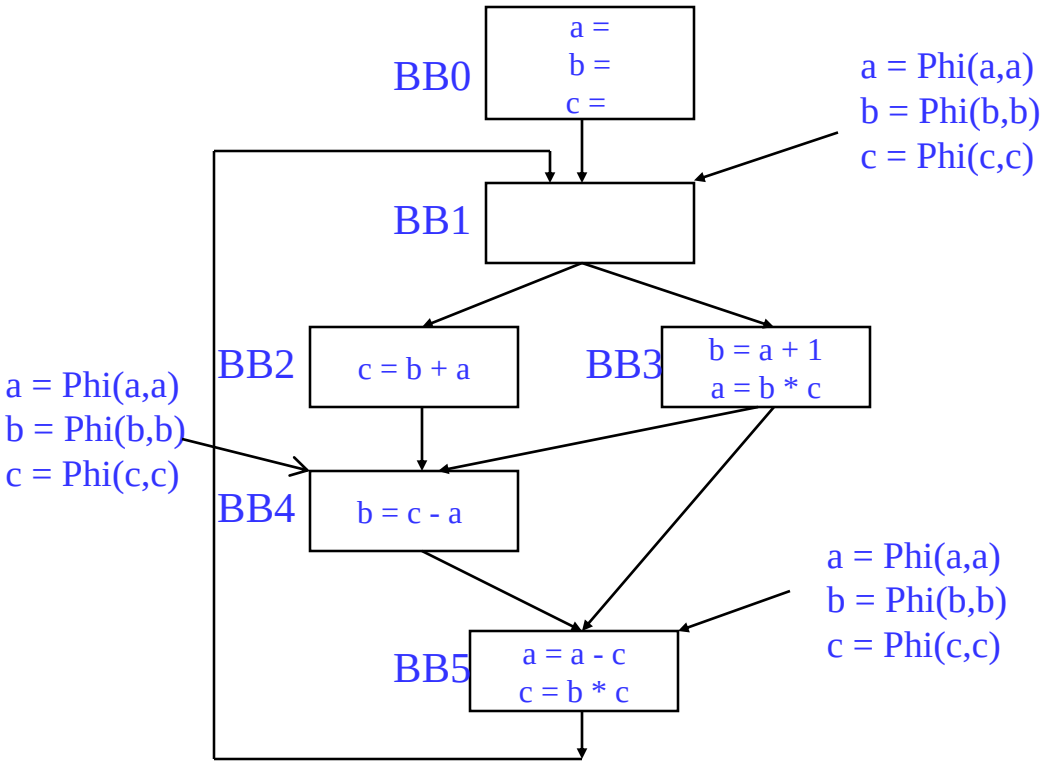




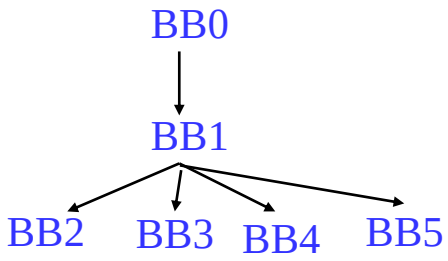
var:	a	b	c	d	i
ctr:	5	5	7	7	3
stk:	a0	b0	c0	d0	i0
	a1	b1	c1	d1	i1
	a2	b4	c2	d6	i2
	a4		c6		

Fin!

Homework Problem – Rename the Variables

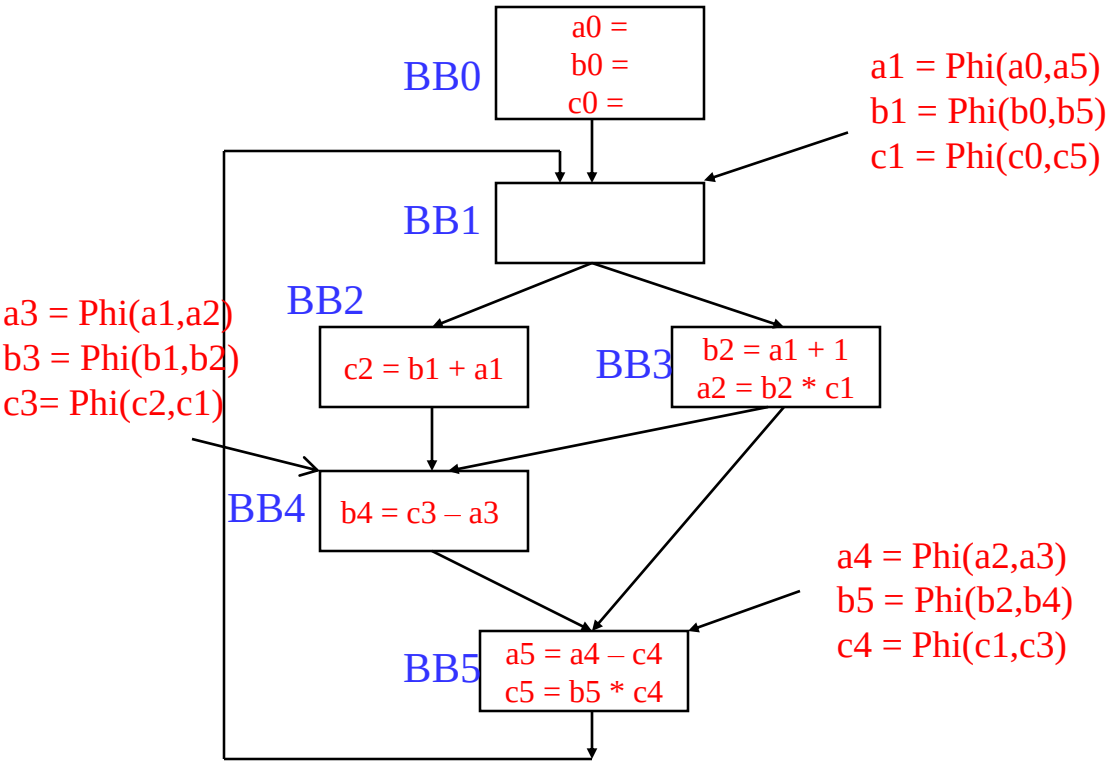


Dominator tree

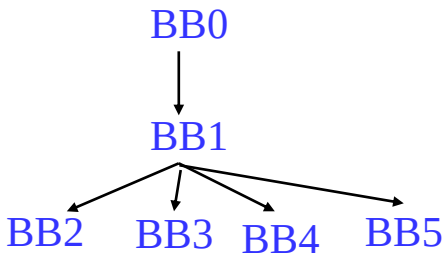


Homework Problem –Answer

Rename the variables



Dominator tree



Dominance frontier

BB	DF
0	-
1	-
2	4
3	4, 5
4	5
5	1

Next Topic: Code Optimization

- Make the code run faster on the target processor
  - My (Scott's) favorite topic !!
  - Other objectives: Power, code size
- Classes of optimization
  - 1. Classical (machine independent, done at IR level)
    - Reducing operation count (redundancy elimination)
    - Simplifying operations
    - Generally good for any kind of machine
  - 2. Machine specific (done in llc)
    - Peephole optimizations
    - Take advantage of specialized hardware features
  - 3. Parallelism enhancing (IR level often, but sometimes llc)
    - Increasing parallelism (ILP or TLP)
    - Possibly increase instructions

A Tour Through the Classical Optimizations

- For this class – Go over concepts of a small subset of the optimizations
  - What it is, why its useful
  - When can it be applied (set of conditions that must be satisfied)
  - How it works
  - Give you the flavor but don't want to beat you over the head
- Challenges
  - Register pressure?
  - Parallelism verses operation count

First Optimization: Dead Code Elimination

- Remove any operation who's result is never consumed
- Rules
  - X can be deleted
    - no stores or branches
  - DU chain empty or dest register not live
- This misses some dead code!!
  - Especially in loops
- Better Algorithm
  - Critical operation
    - store or branch operation
  - Any operation that does not directly or indirectly feed a critical operation is dead
  - Trace UD chains backwards from critical operations
  - Any op not visited is dead