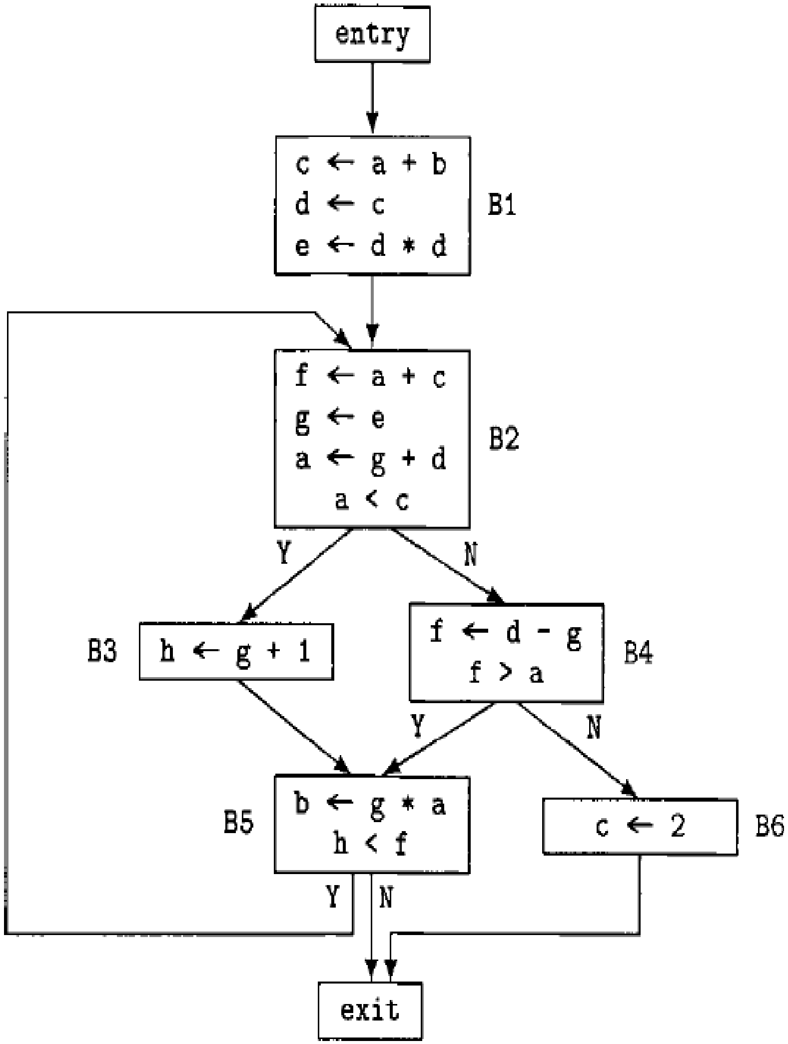
CC Lecture 8

Prepared for: 7th Sem, CE, DDU
Prepared by: Niyati J. Buch

Copy Propagation

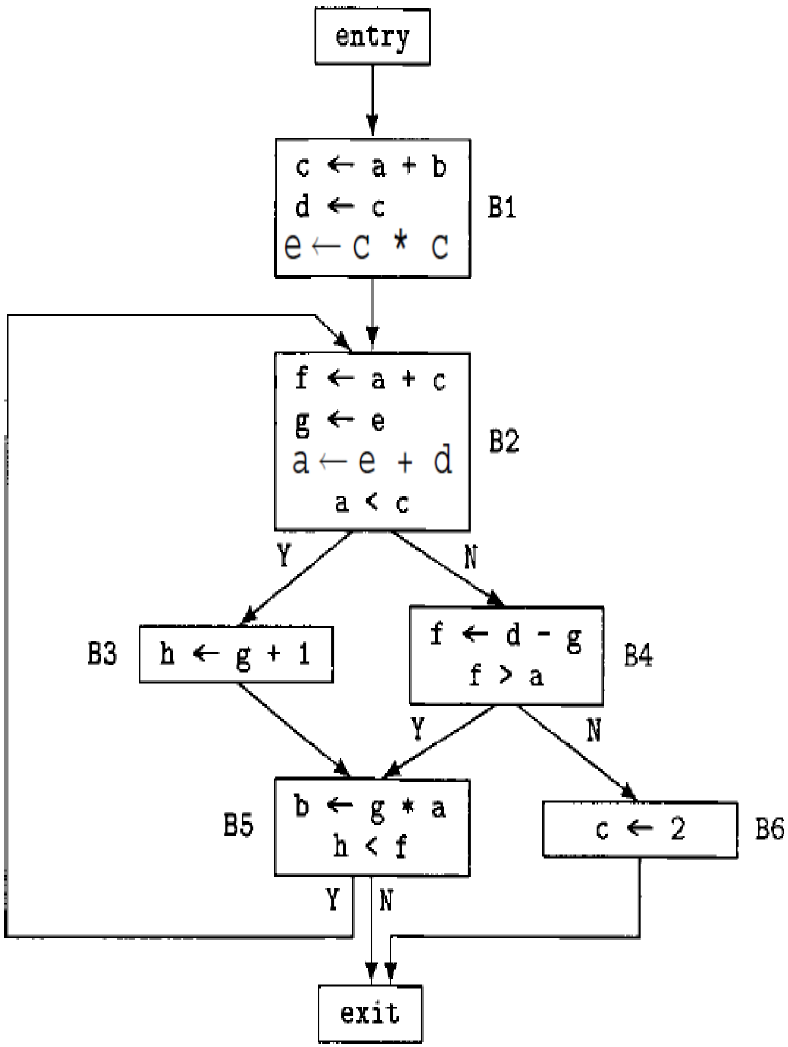
- Copy propagation is a transformation that, given an assignment **x ← y** for some variables x and y, replaces later uses of x with uses of y, as long as intervening instructions have not changed the value of **either x or y**.

Before local copy propagation



- This is the flow graph **before** local copy propagation.

After local copy propagation



- This is the flow graph **after** local copy propagation.

Global Copy Propagation

- To perform global copy propagation, we first do a data-flow analysis to determine which copy assignments reach uses of their left-hand variables unimpaired, i.e., without having either variable redefined in between.

- We define the set **COPY(i)** to consist of the instances of copy assignments occurring in block i that reach the end of block i.

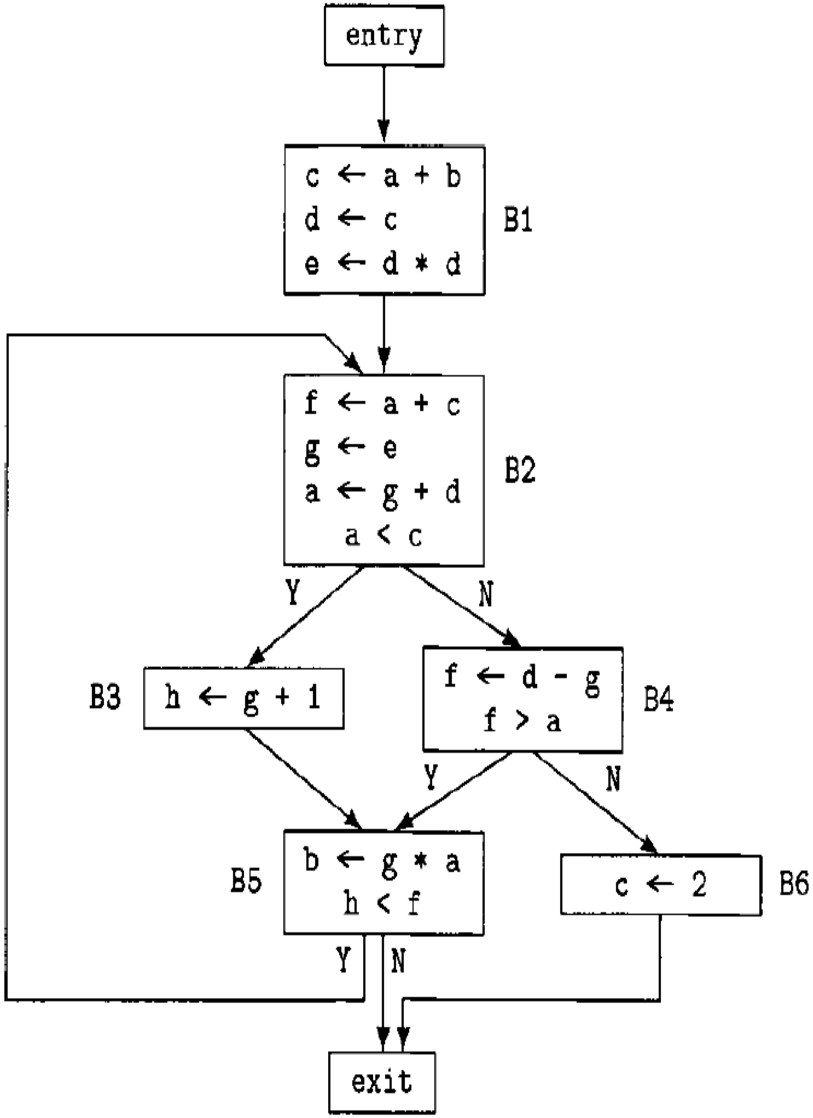
- We define **KILL(i)** to be the set of copy assignment instances killed by block i.

COPY(i) and KILL(i)

- COPY(i) is a set of quadruples **(u, v, i, pos)**,
  - such that **u ← v** is a copy assignment
  - and **pos** is the position in block **i** where the assignment occurs,
  - and neither **u** nor **v** is assigned to later in block **i**.

- KILL(i) is the set of quadruples **(u, v, blk, pos)**
  - such that **u ← v** is a copy assignment occurring at position **pos** in block **blk ≠ i**.

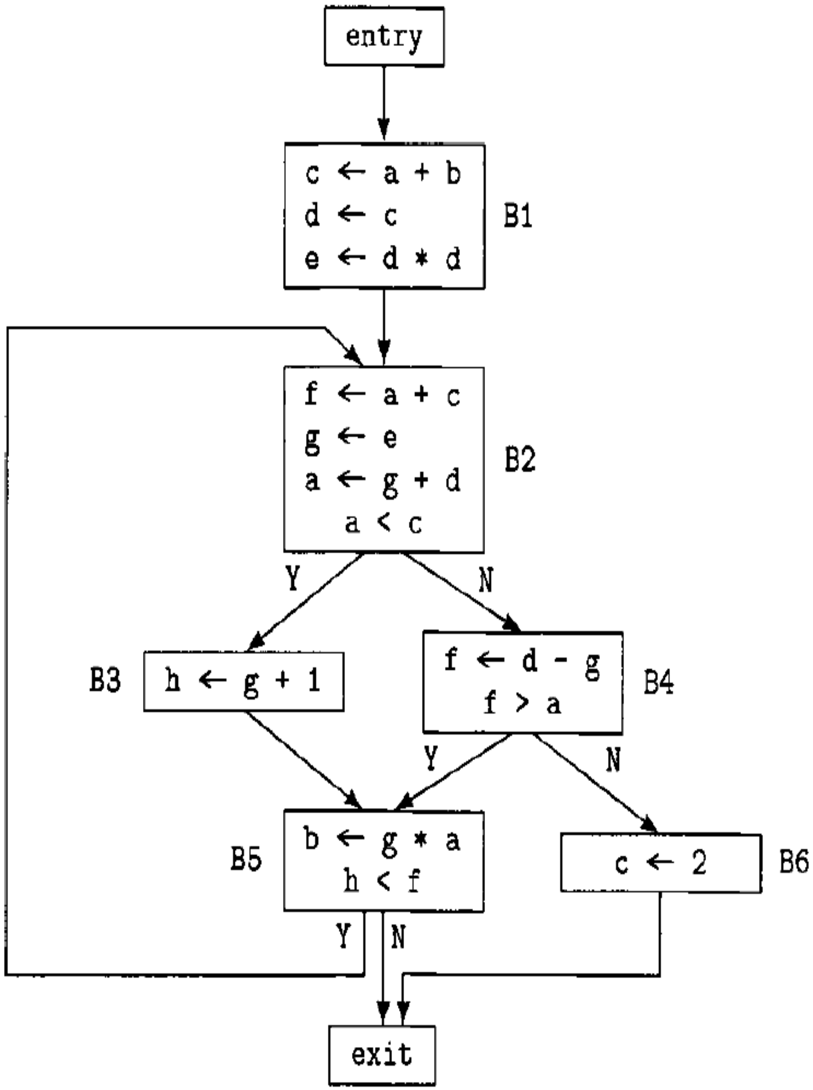
COPY(i) using vector representation



- COPY(entry) = <00>
- COPY(B1) = <10>
- COPY(B2) = <01>
- COPY(B3) = <00>
- COPY(B4) = <00>
- COPY(B5) = <00>
- COPY(B6) = <00>
- COPY(exit) = <00>

Bit position	COPY
1	{(d, c, B1, 2)}
2	{(g, e, B2, 2)}

KILL(i) using vector representation



- KILL(entry) = <00>
- KILL(B1) = <01>
- KILL(B2) = <00>
- KILL(B3) = <00>
- KILL(B4) = <00>
- KILL(B5) = <00>
- KILL(B6) = <10>
- KILL(exit) = <00>

Bit position	COPY
1	{(d, c, B1, 2)}
2	{(g, e, B2, 2)}

Initialize CPin

- CPin(x) = Ø if x = entry
- CPin(x) = U otherwise, where U = U COPY(i) for all i

CPin for all blocks (Pass 1)

- CPin(entry) = Ø | <00>
- CPin(B1) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(B2) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(B3) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(B4) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(B5) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(B6) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>
- CPin(exit) = {(d, c, B1, 2),(g, e, B2, 2)} | <11>

Data-flow equations for CPin(i) and CPout(i)

- Next, we define data-flow equations for CPin(i) and CPout(i) that represent the sets of copy assignments that are available for copy propagation on entry to and exit from block i, respectively.

- A copy assignment is **available on entry** to block i if it is available on exit from all predecessors of block i, so the path-combining operator is intersection.

- A copy assignment is **available on exit** from block j if it is either in COPY(j) or it is available on entry to block j and not killed by block j, i.e., if it is in CPin(j) and not in KILL(j)

Data-flow equations

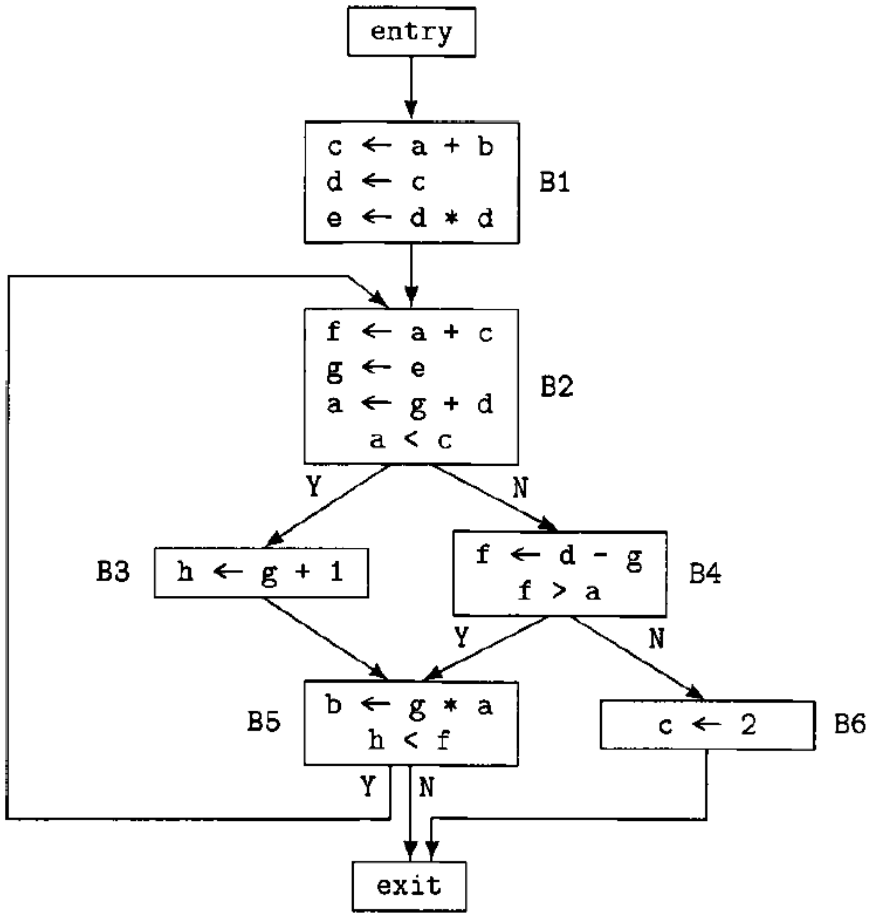
- CPin(i) = ∩ CPout(j) where j ∈ pred(i)
- CPout(i) = COPY(i) ∪ (CPin(i) - KILL(i))
- Equivalent:

$$CPout(i) = COPY(i) \bigcup (CPin(i) \bigcap \overline{KILL(i)})$$

- Substituting CPout into CPin, we obtain:

$$\boxed{CPin(i) = \bigcap_{j \in pred(i)} COPY(j) \bigcup (CPin(j) \bigcap \overline{KILL(j)})}$$

Our work-list order



- Since this is a forward problem, we manage our work-list in a reverse post-order (i.e. preorder means each block before its successors) order.

- One such order is **entry, B1, B2, B4, B6, B3, B5, exit**.

Pass 2: Applying iterative analysis for block entry

- CPin(entry) = <00>

- as per the equation as no predecessor is available.

Pass 2: Applying iterative analysis for block i = B1

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- entry is predecessor of B1
- CPin(B1) = COPY(entry) U (CPin(entry) - KILL(entry))
- CPin(B1) = <00> U (<00> - <00>)
- **CPin(B1) = <00>**

Pass 2: Applying iterative analysis for block i = B2

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B1 and B5 are predecessors of B2

- CPin(B2) = (COPY(B1) ∪ (CPin(B1) - KILL(B1)))
  ∩ (COPY(B5) ∪ (CPin(B5) - KILL(B5)))
- CPin(B2) = (<10> ∪ (<11> - <01>))
  ∩ (<00> ∪ (<11> - <00>))
  = (<10> ∪ <10>) ∩ (<00> ∪ <11>)
  = <10> ∩ <11>
- **CPin(B2) = <10>**

Pass 2: Applying iterative analysis for block i = B4

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B2 is predecessor of B4

- CPin(B4) = COPY(B2) ∪ (CPin(B2) - KILL(B2))

- CPin(B4) = <01> ∪ (<11> - <00>)

  = <01> ∪ <11>

- **CPin(B4) = <11>**

Pass 2: Applying iterative analysis for block i = B6

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B4 is predecessor of B6
- CPin(B6) = COPY(B4) ∪ (CPin(B4) - KILL(B4))
- CPin(B6) = <00> ∪ (<11> - <00>)
  = <00> ∪ <11>
- **CPin(B6) = <11>**

Pass 2: Applying iterative analysis for block i = B3

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B2 is predecessor of B3
- CPin(B3) = COPY(B2) ∪ (CPin(B2) - KILL(B2))
- CPin(B3) = <01> ∪ (<11> - <00>)

  = <01> ∪ <11>
- **CPin(B3) = <11>**

Pass 2: Applying iterative analysis for block i = B5

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B3 and B4 are predecessors of B5

- CPin(B5) = (COPY(B3) ∪ (CPin(B3) - KILL(B3)))
  ∩ (COPY(B4) ∪ (CPin(B4) - KILL(B4)))

- CPin(B5) = (<00> ∪ (<11> - <00>)) ∩ (<00> ∪ (<11> - <00>))
  = (<00> ∪ <11>) ∩ (<00> ∪ <11>)
  = <11> ∩ <11>
- **CPin(B5) = <11>**

Pass 2: Applying iterative analysis for block i = exit

$$CP\,in(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CP\,in(j) \cap \overline{KILL(j)})$$

- B5 and B6 are predecessors of exit

- CPin(exit) = (COPY(B5) ∪ (CPin(B5) - KILL(B5)))
  ∩ (COPY(B6) ∪ (CPin(B6) - KILL(B6)))
- CPin(exit) = (<00> ∪ (<11> - <00>))
  ∩ (<00> ∪ (<11> - <10>))
  = (<00> ∪ <11>) ∩ (<00> ∪ <01>)
  = <11> ∩ <01>
- **CPin(exit) = <01>**

CPin(i)

	Pass 1	Pass 2
CPin(entry)	<00>	**<00>**
CPin(B1)	<11>	**<00>**
CPin(B2)	<11>	**<10>**
CPin(B3)	<11>	**<11>**
CPin(B4)	<11>	**<11>**
CPin(B5)	<11>	**<11>**
CPin(B6)	<11>	**<11>**
CPin(exit)	<11>	**<01>**

Pass 3: Applying iterative analysis for block entry

- CPin(entry) = <00>

- as per the equation as no predecessor is available.

Pass 3: Applying iterative analysis for block i = B1

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- entry is predecessor of B1
- CPin(B1) = COPY(entry) U (CPin(entry) - KILL(entry))
- CPin(B1) = <00> U (<00> - <00>)
- **CPin(B1) = <00>**

Pass 3: Applying iterative analysis for block i = B2

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B1 and B5 are predecessors of B2

- CPin(B2) = (COPY(B1) ∪ (CPin(B1) - KILL(B1)))
  ∩ (COPY(B5) ∪ (CPin(B5) - KILL(B5)))
- CPin(B2) = (<10> ∪ (<00> - <01>)) ∩ (<00> ∪ (<11> - <00>))
  = (<10> ∪ <00>) ∩ (<00> ∪ <11>)
  = <10> ∩ <11>

- **CPin(B2) = <10>**

Pass 3: Applying iterative analysis for block i = B4

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \bigcup (CPin(j) \bigcap \overline{KILL(j)})$$

- B2 is predecessor of B4
- CPin(B4) = COPY(B2) ∪ (CPin(B2) - KILL(B2))
- CPin(B4) = <01> ∪ (<10> - <00>)
  = <01> ∪ <10>
- **CPin(B4) = <11>**

Pass 3: Applying iterative analysis for block i = B6

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B4 is predecessor of B6

- CPin(B6) = COPY(B4) ∪ (CPin(B4) - KILL(B4))

- CPin(B6) = <00> ∪ (<11> - <00>)
  = <00> ∪ <11>

- **CPin(B6) = <11>**

Pass 3: Applying iterative analysis for block i = B3

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B2 is predecessor of B3
- CPin(B3) = COPY(B2) ∪ (CPin(B2) - KILL(B2))
- CPin(B3) = <01> ∪ (<10> - <00>)

  = <01> ∪ <10>
- **CPin(B3) = <11>**

Pass 3: Applying iterative analysis for block i = B5

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B3 and B4 are predecessors of B5

- CPin(B5) = (COPY(B3) ∪ (CPin(B3) - KILL(B3)))
  ∩ (COPY(B4) ∪ (CPin(B4) - KILL(B4)))

- CPin(B5) = (<00> ∪ (<11> - <00>)) ∩ (<00> ∪ (<11> - <00>))
  = (<00> ∪ <11>) ∩ (<00> ∪ <11>)
  = <11> ∩ <11>
- **CPin(B5) = <11>**

Pass 3: Applying iterative analysis for block i=exit

$$CPin(i) = \bigcap_{j \in pred(i)} COPY(i) \cup (CPin(j) \cap \overline{KILL(j)})$$

- B5 and B6 are predecessors of exit

- CPin(exit) = (COPY(B5) ∪ (CPin(B5) - KILL(B5)))
  ∩ (COPY(B6) ∪ (CPin(B6) - KILL(B6)))
- CPin(exit) = (<00> ∪ (<11> - <00>)) ∩ (<00> ∪ (<11> - <10>))
  = (<00> ∪ <11>) ∩ (<00> ∪ <01>)
  = <11> ∩ <01>

- **CPin(exit) = <01>**

This completes one more iteration of iterative data flow analysis. There is no change during Pass 3, so we can stop as shown below.

	Pass 1	Pass 2	Pass 3	CPin() sets
CPin(entry)	<00>	<00>	<00>	Ø
CPin(B1)	<11>	<00>	<00>	Ø
CPin(B2)	<11>	<10>	<10>	{(d, c, B1, 2)}
CPin(B3)	<11>	<11>	<11>	{(d, c, B1, 2), (g, e, B2, 2)}
CPin(B4)	<11>	<11>	<11>	{(d, c, B1, 2), (g, e, B2, 2)}
CPin(B5)	<11>	<11>	<11>	{(d, c, B1, 2), (g, e, B2, 2)}
CPin(B6)	<11>	<11>	<11>	{(d, c, B1, 2), (g, e, B2, 2)}
CPin(exit)	<11>	<01>	<01>	{(g, e, B2, 2)}

Global Copy Propagation

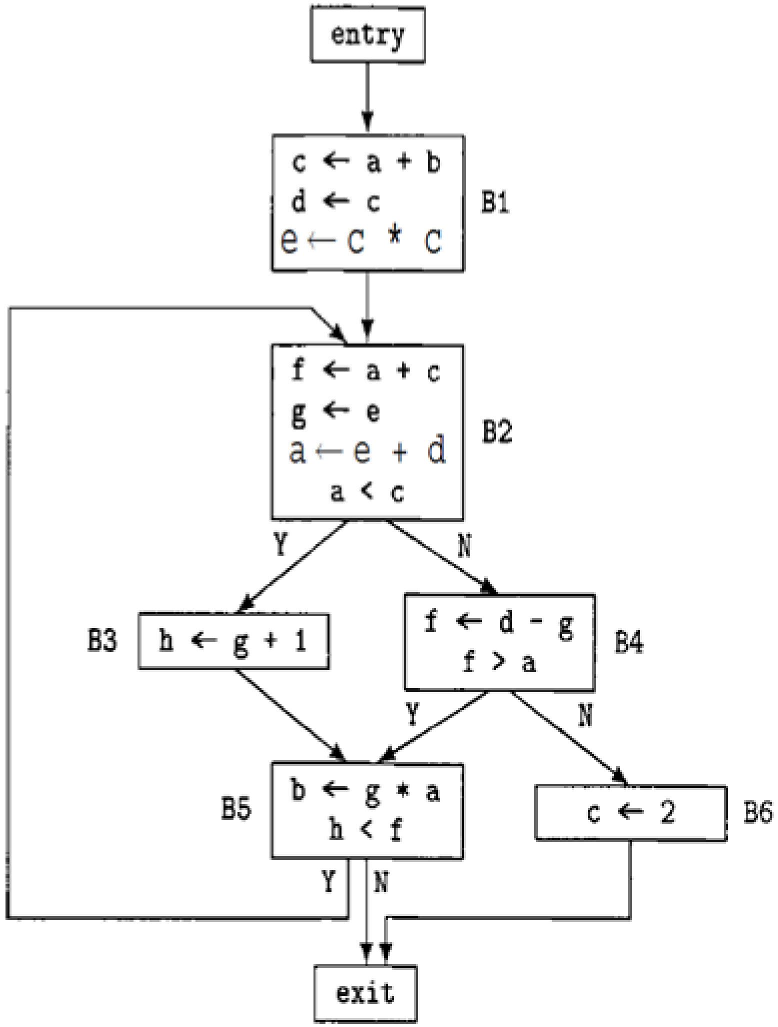
- Given the data-flow information CPin( ) and assuming that we have already done local copy propagation, we perform global copy propagation as follows:

1. For each basic block B,

   set ACP = {a Є Var x Var where ∃w Є integer such that <a@1,a@2,B,w> Є CPin(B)}.

2. For each basic block B, perform the local copy-propagation algorithm.

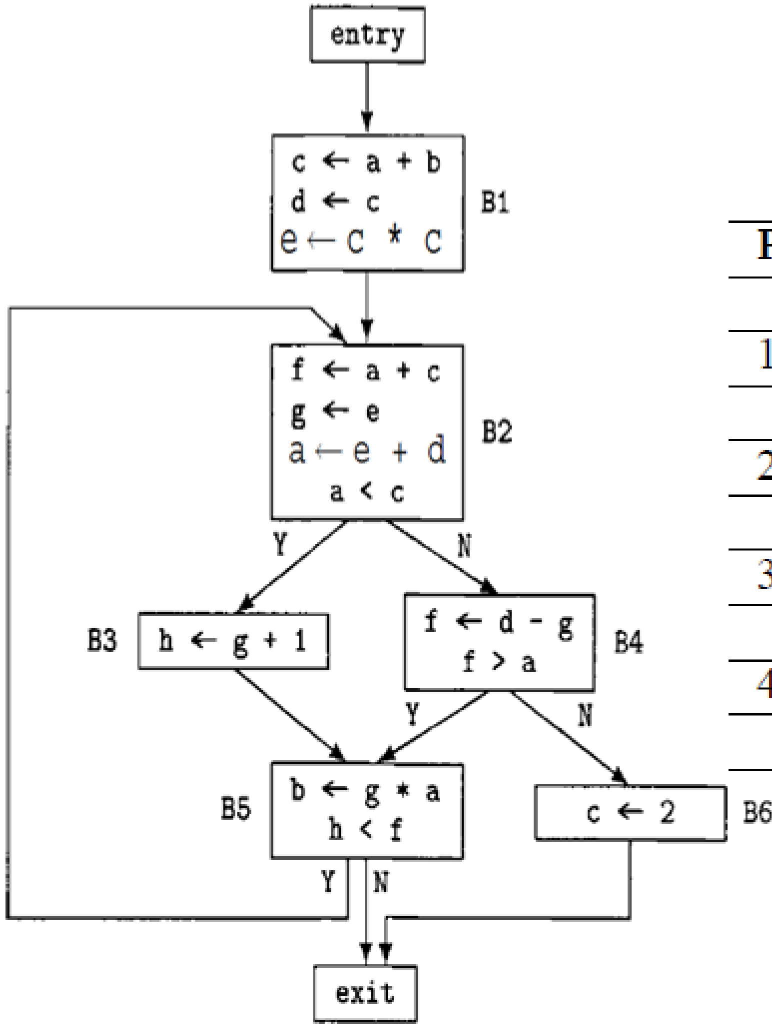
For block B1



- CPin(B1) = Ø

Position	Code Before	ACP	Code After
		$\emptyset$	
1	c ← a + b		c ← a + b
		$\emptyset$	
2	d ← c		d ← c
		$\{\langle d, c\rangle\}$	
3	e ← c * c		e ← c * c
		$\{\langle d, c\rangle\}$	

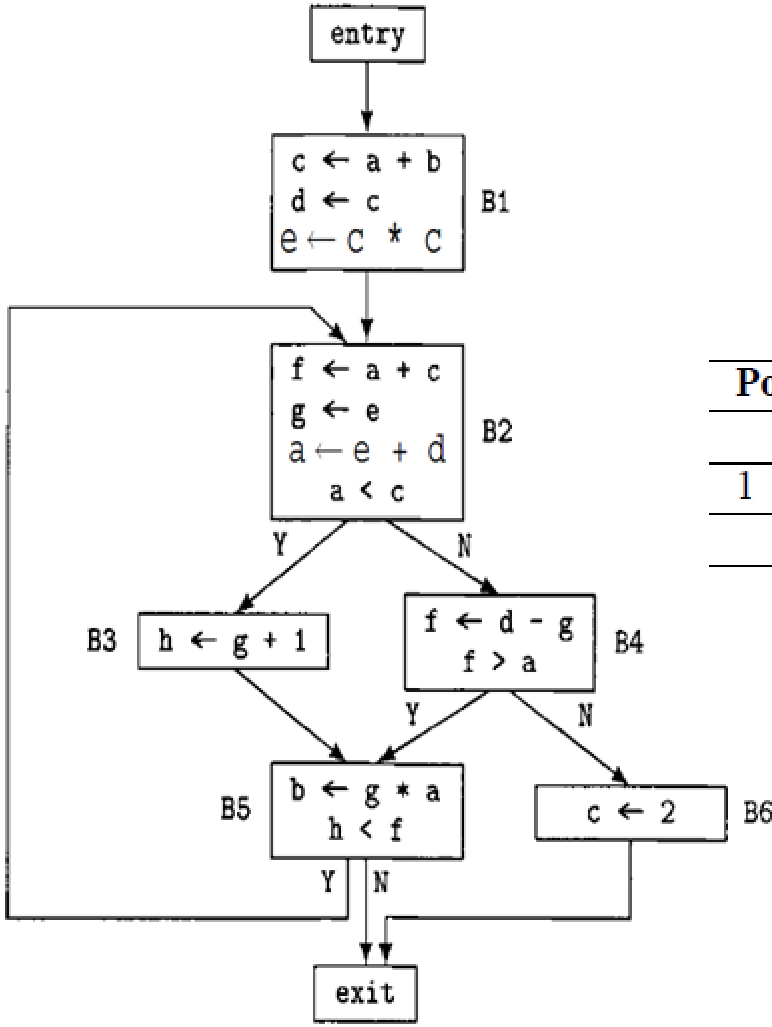
For block B2



- CPin(B2) = {(d, c, B1, 2)}

Position	Code Before	ACP	Code After
		$\{\langle d, c\rangle\}$	
1	f ← a + c		f ← a + c
		$\{\langle d, c\rangle\}$	
2	g ← e		g ← e
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	
3	a ← **e** + d		a ← e + c
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	
4	a < c		a < c
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	

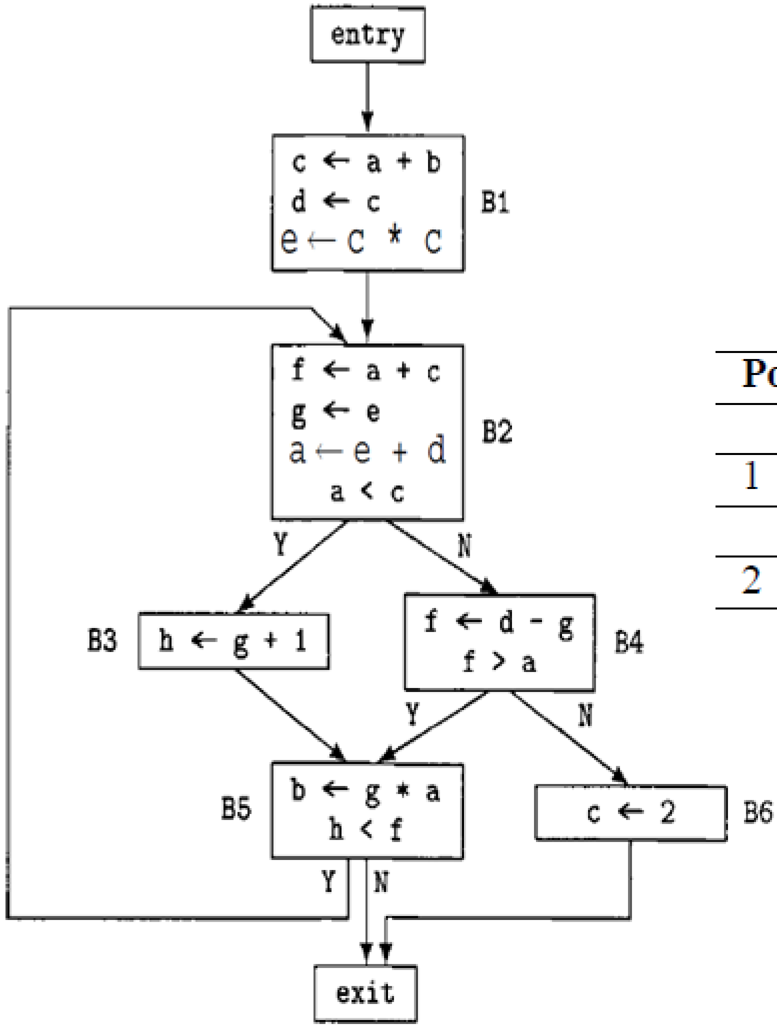
For block B3



- CPin(B3) = {(d, c, B1, 2), (g, e, B2, 2)}

Position	Code Before	ACP	Code After
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	
1	h ← g + 1		h ← e + 1
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	

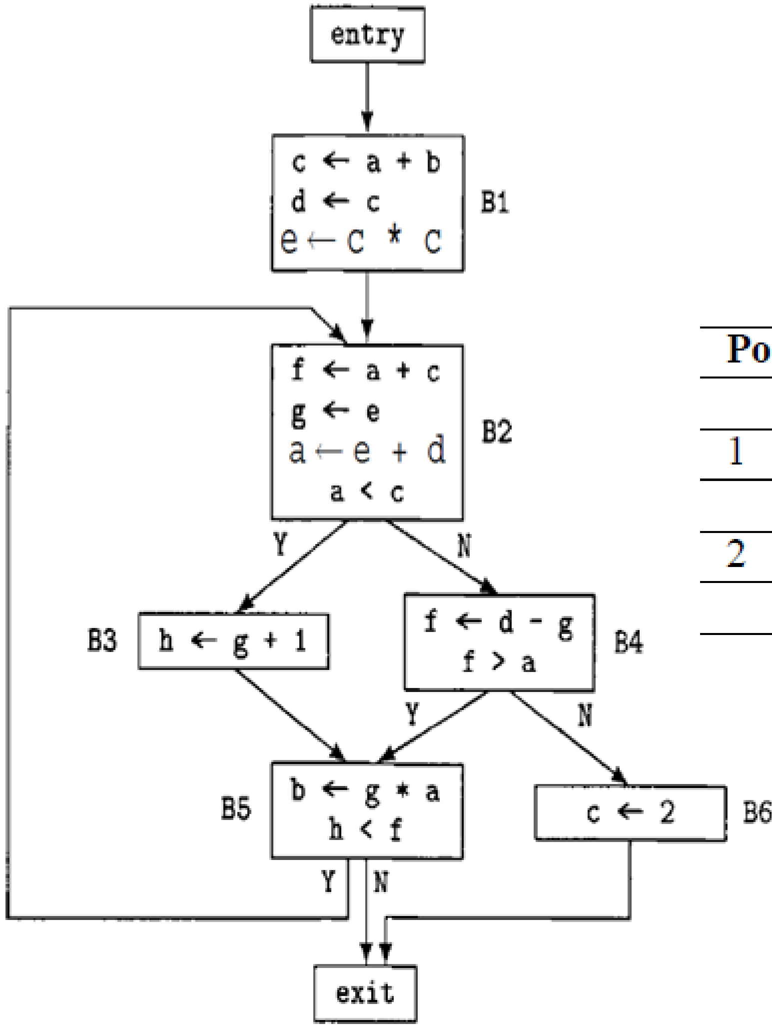
For block B4



- CPin(B4) = {(d, c, B1, 2), (g, e, B2, 2)}

Position	Code Before	ACP	Code After
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	
1	f ← d - g		f ← c - e
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	
2	f < a		f < a
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	

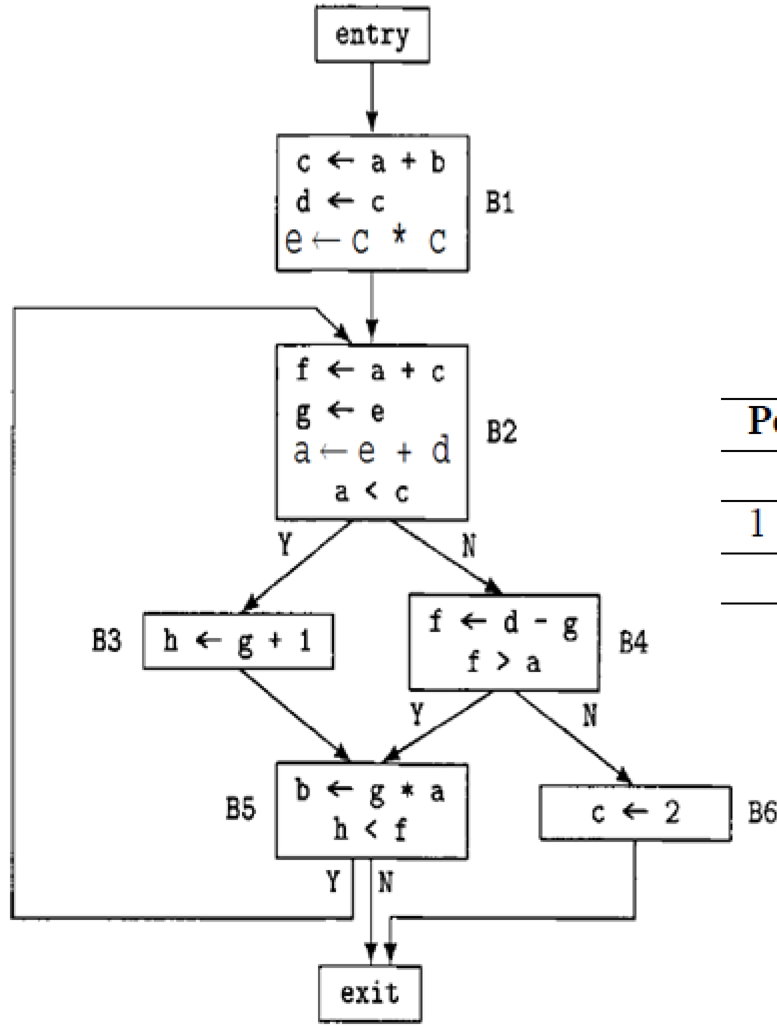
For block B5



- CPin(B5) = {(d, c, B1, 2), (g, e, B2, 2)}

Position	Code Before	ACP	Code After
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	
1	b ← g * a		b ← e * a
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	
2	h < f		h < f
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	

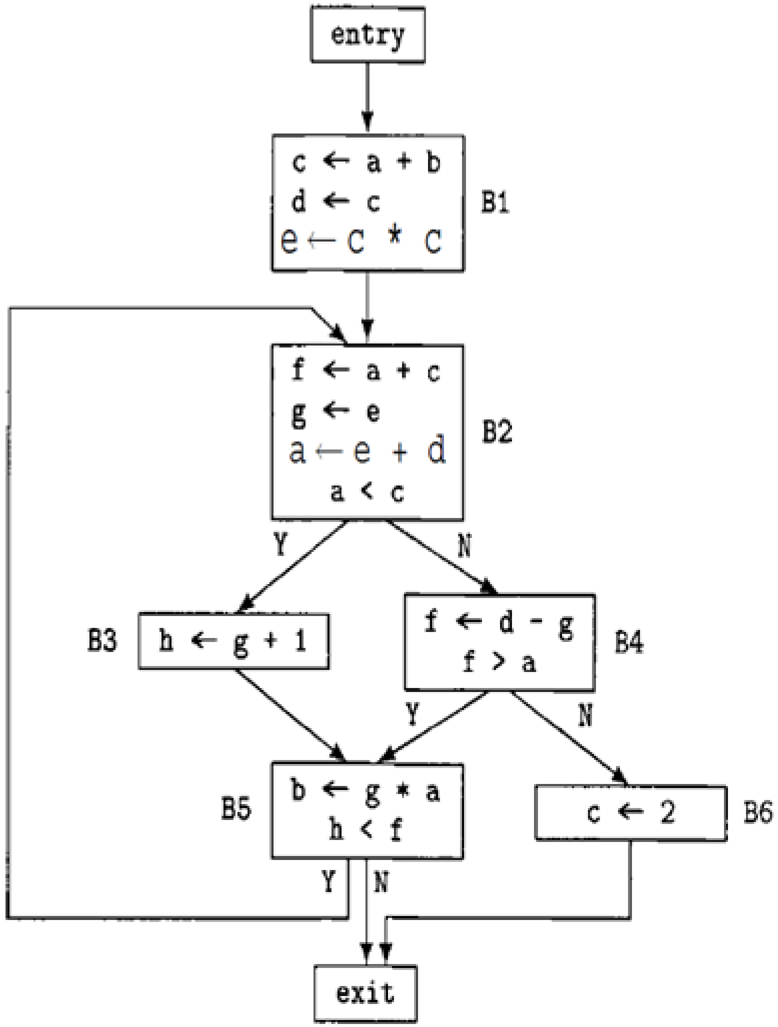
For block B6



- CPin(B6) = {(d, c, B1, 2), (g, e, B2, 2)}

Position	Code Before	ACP	Code After
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	
1	$c \leftarrow 2$		$c \leftarrow 2$
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	

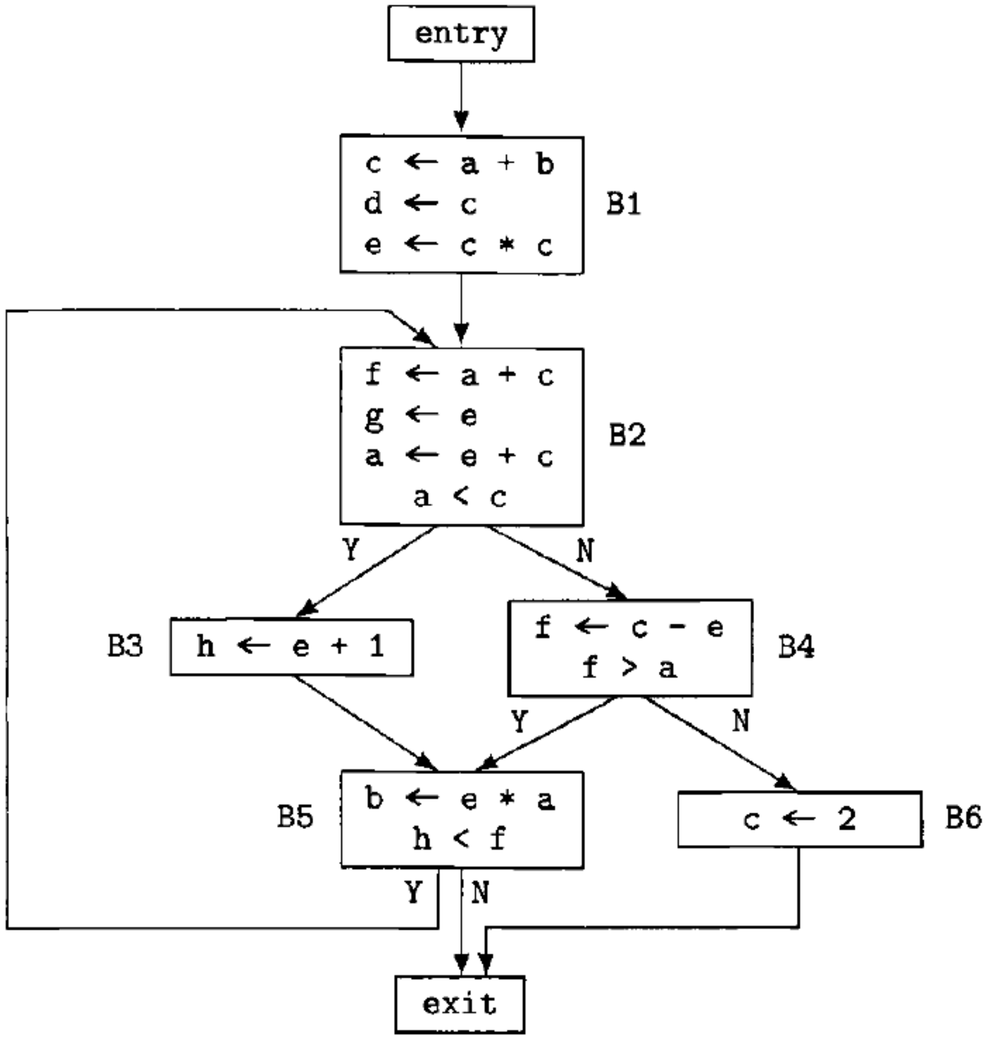
For block exit



- CPin(exit) = {(g, e, B2, 2)}

Position	Code Before	ACP	Code After
		$\{\langle g, e\rangle\}$	

Finally, we have



Control-flow Analysis

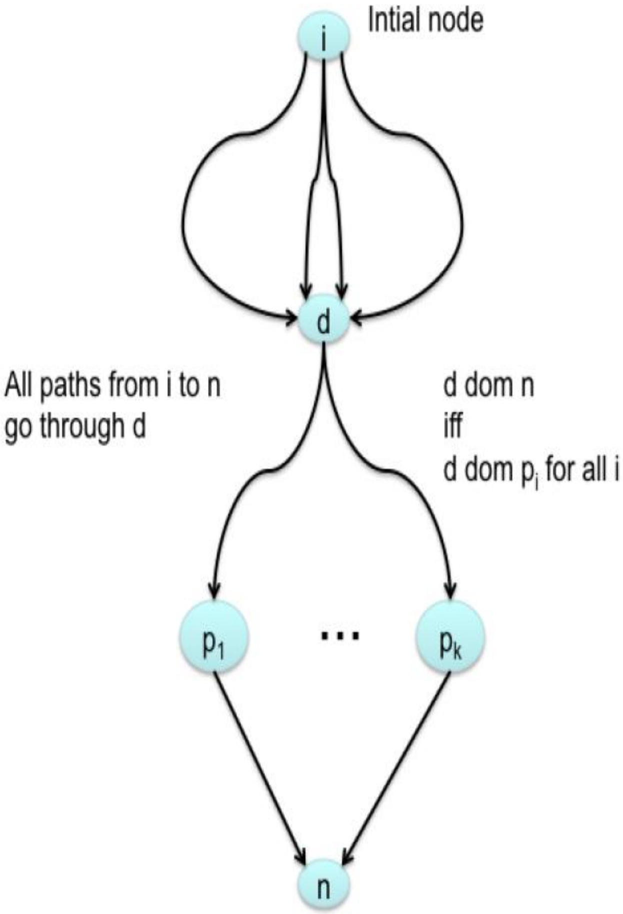
- **Control-flow analysis** (CFA) is a static-code-analysis technique for determining the **control flow** of a program.

- The **control flow** is expressed as a **control-flow graph** (CFG).

Control-flow graphs (CFG)

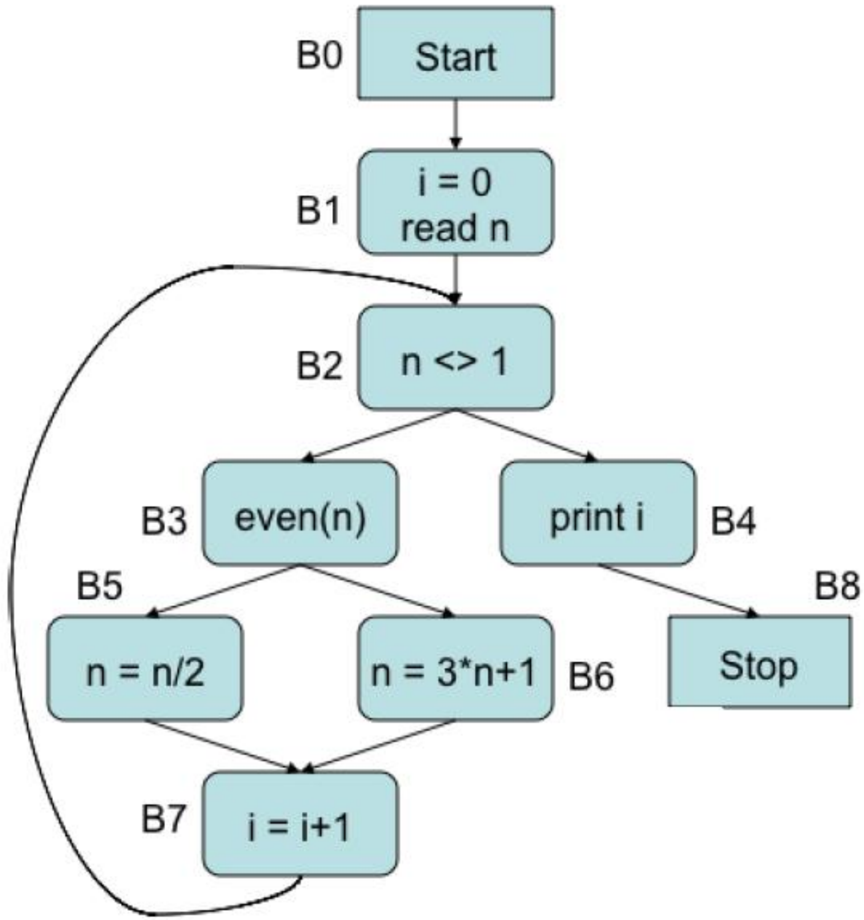
- Control-flow analysis (CFA) helps us to understand the structure of control-flow graphs (CFG)
  - To determine the loop structure of CFGs
  - To compute dominators - useful for code motion
  - To compute dominance frontiers - useful for the construction of the static single assignment form (SSA)
  - To compute control dependence - needed in parallelization

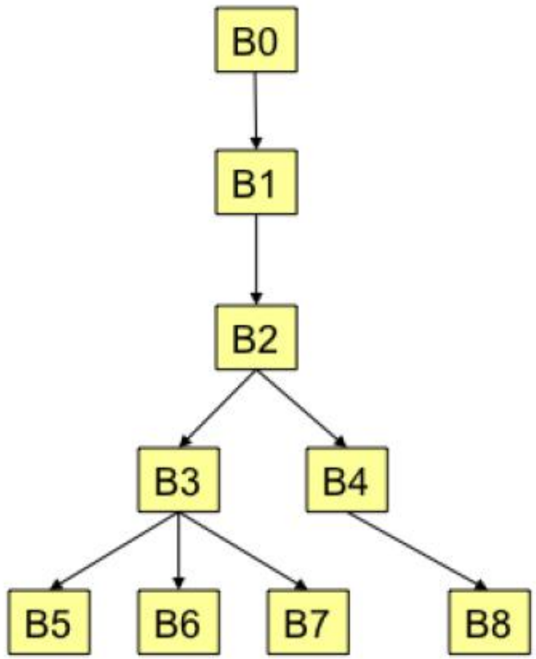
Dominators



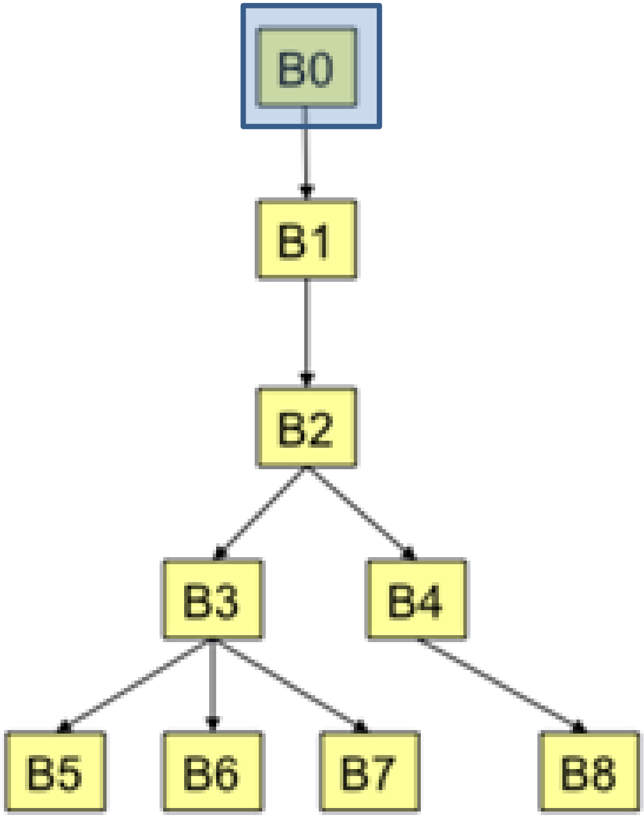
- A node d in a flow graph is said to dominates node n, written ***d dom n***, if every path from the initial node of the flow graph to n goes through d.
- Initial node is the root, and each node dominates only its descendents in the dominator tree (including itself).
- The node x strictly dominates y, if x dominates y and x ≠ y.
- x is the immediate dominator of y (denoted idom(y)), if x is the closest strict dominator of y.
- A **dominator tree** shows all the immediate dominator relationships not the transitive relationship.
- Principle of the dominator algorithm
  - If p1, p2, …, pk , are all the predecessors of n, and d ≠ n, then d dom n, iff d dom pi for each i.

Dominator Example 1:



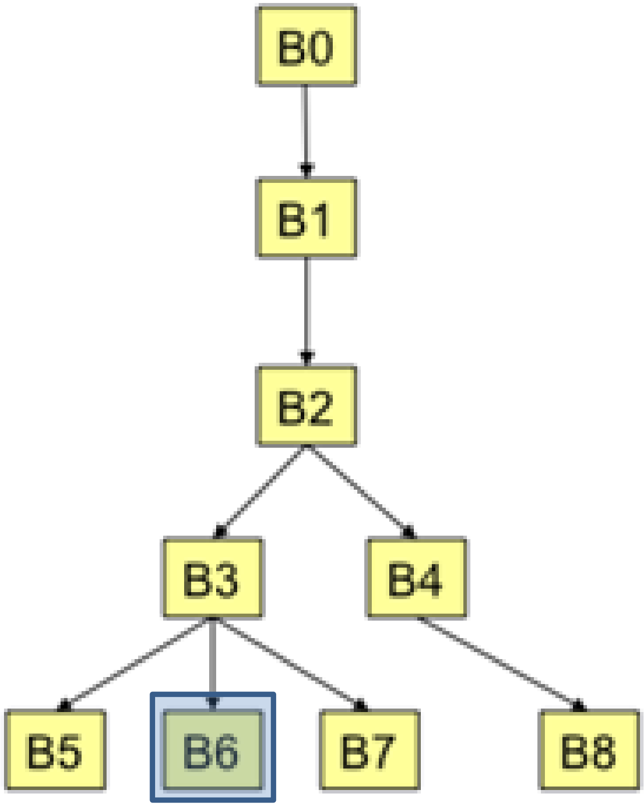


B0



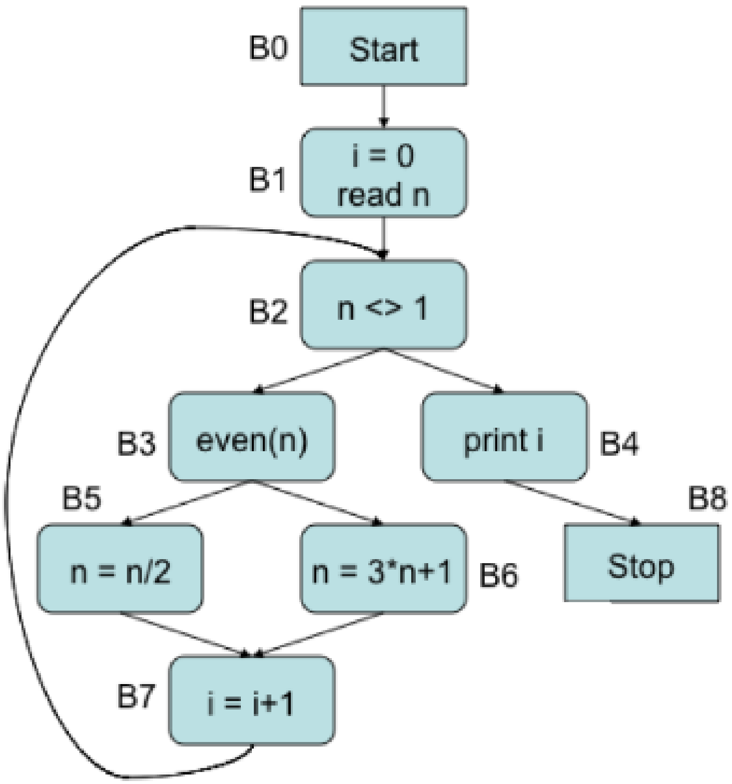
- Suppose we consider the initial node B0 it obviously dominates all the nodes in the flow graph.
- Because, every path starting from the initial node to any other node must actually pass through the start node.
- So that is why **B0** is the **root** of this dominator tree

B6



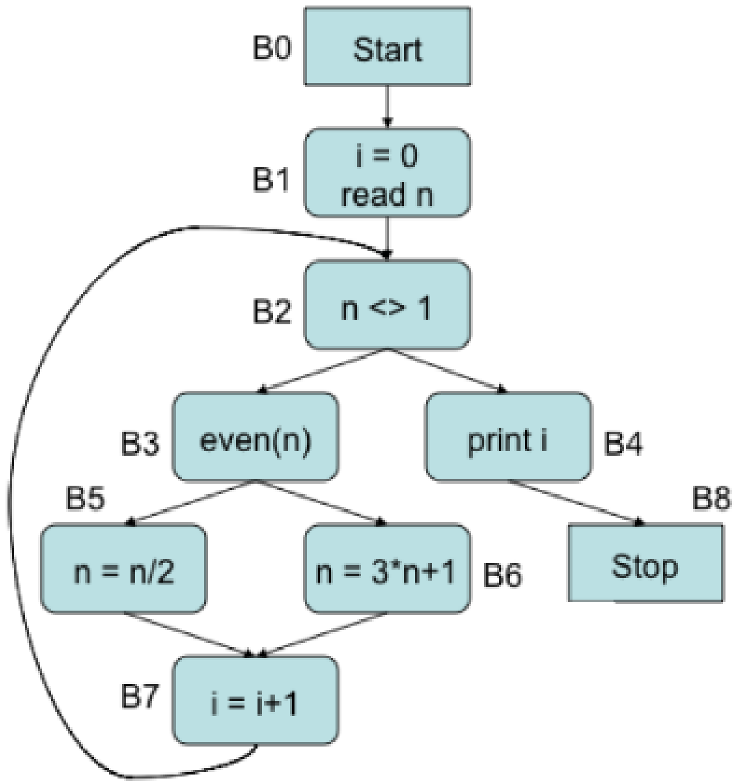
- B6 is a leaf node here in the dominance tree.
- The immediate dominator of B6 is B3.
- Immediate dominator of B3 is B2, immediate dominator of B2 is B1 and that of B1 is B0 .
- So, in other words the dominators of B6 are B3, B2, B1 and B0

B6 with CFG



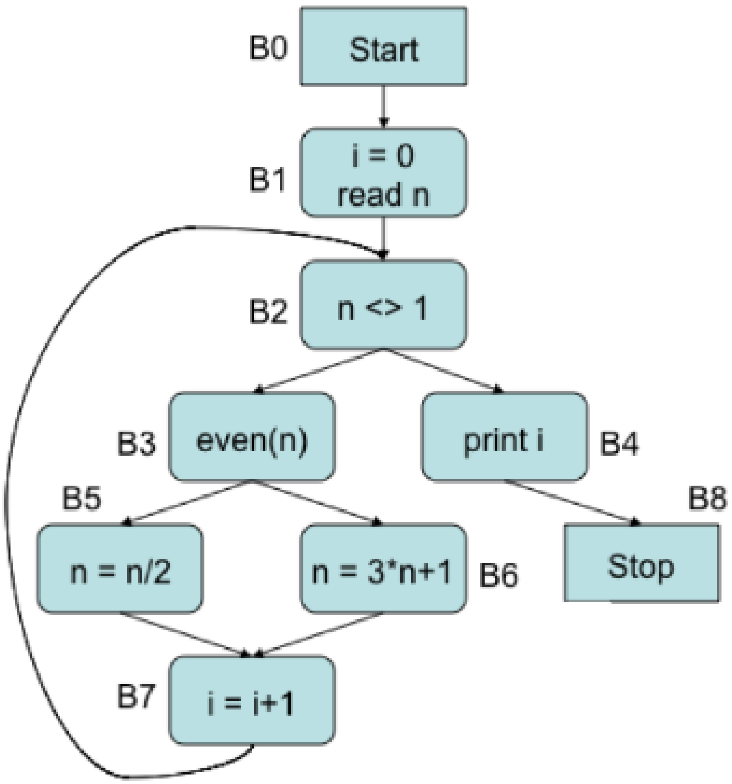
- B0 is very trivial and B1 is also trivial because there is only one path from start to B1. So, all paths starting from start to B6 will definitely have passed through B1.
- Then we have B2 again there is only one path from B1 to B2. So starting from B0 to goto B6 will also have to pass through B2.

B6 with CFG



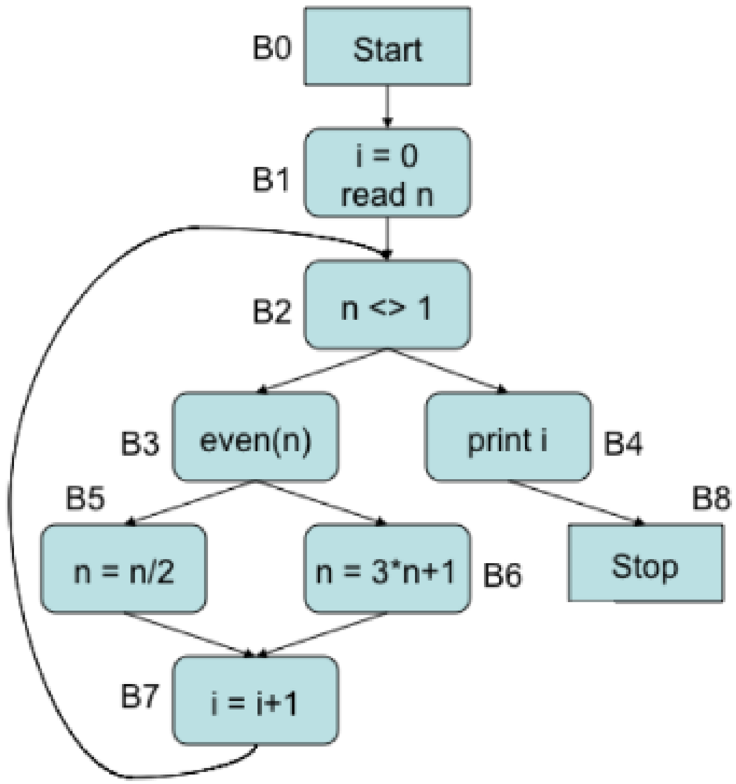
- Now from B2 how do we reach B6?
- So obviously, if we go this way (B2→ B4→ B8) we cannot reach B6 .
- So, we will have to go through B3; that means, again B3 will be a dominator of B6 .
- Then if we start from B0 . Then B1. Then B2. Then B3.

B6 with CFG

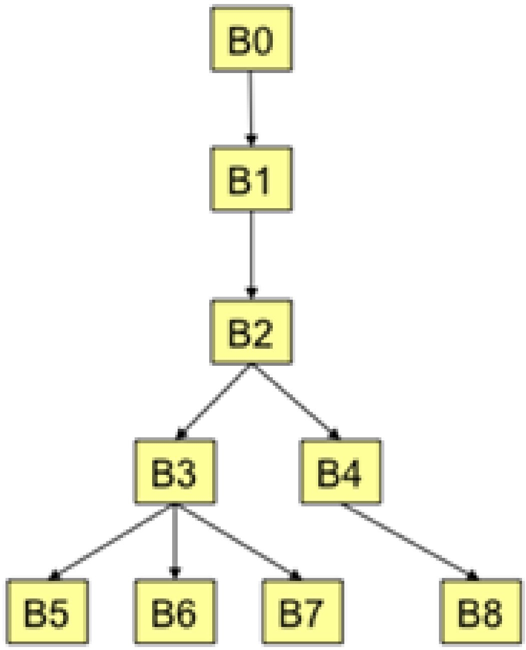


- Let us say we go to B5 , B7 and then we go back to B2 and then again B3 and B6. So this is not a compulsory path.
- Since every path from the start node to B6 does not contain B5 and B7 , B5 and B7 do not dominate B6.
- Only B3, B2, B1 and B0 dominate B6

B6 with CFG



- So, in other words the dominators of B6 are B3, B2, B1 and B0



An algorithm for finding dominator:

$D(n) = OUT[n]$ for all n in N (the set of nodes in the flow graph), after the following algorithm terminates

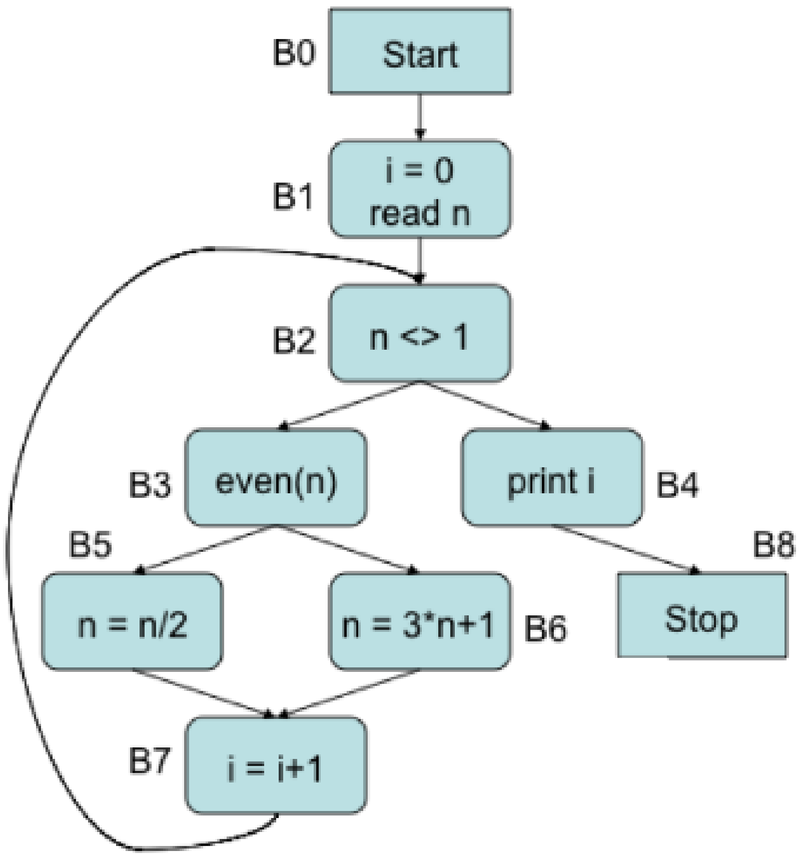
{ /* n_0 = initial node; N = set of all nodes; */
  $OUT[n_0] = \{n_0\}$;
  for n in $N - \{n_0\}$ do $OUT[n] = N$;
  while (*changes to any* $OUT[n]$ *or* $IN[n]$ *occur*) do
    for n in $N - \{n_0\}$ do

$$IN[n] = \bigcap_{P \text{ a predecessor of } n} OUT[P];$$

$$OUT[n] = \{n\} \cup IN[n]$$

}

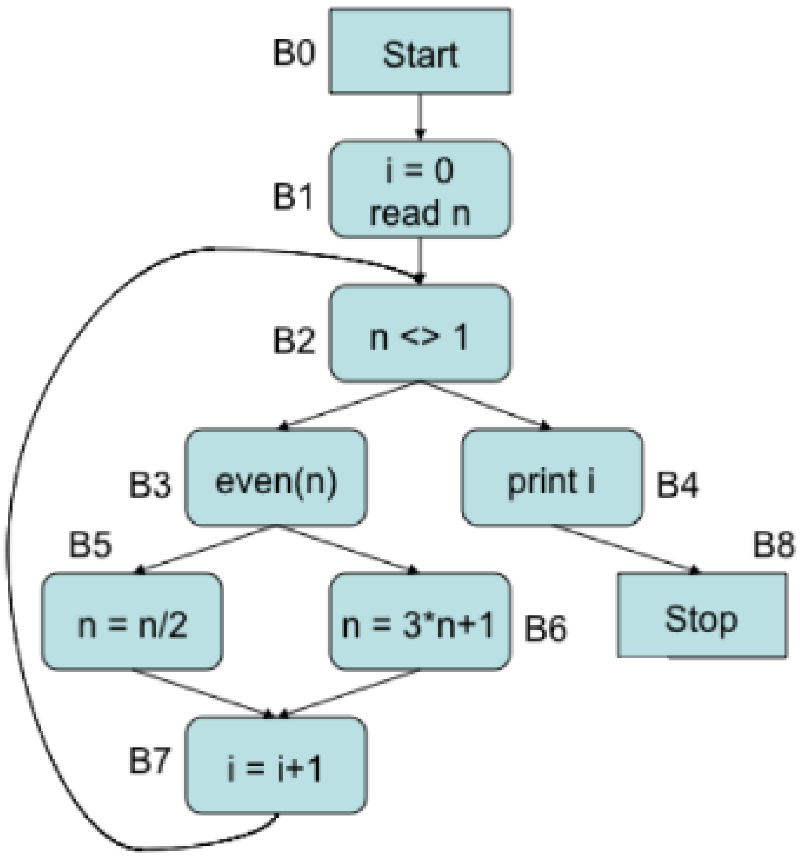
Applying the algorithm for finding dominator



- OUT[B0]={B0}
- OUT[B1] = OUT[B2] = OUT[B3] = OUT[B4] = OUT[B5] = OUT[B6] = OUT[B7] = OUT[B8] =

  {B0, B1, B2, B3, B4, B5, B6, B7, B8}

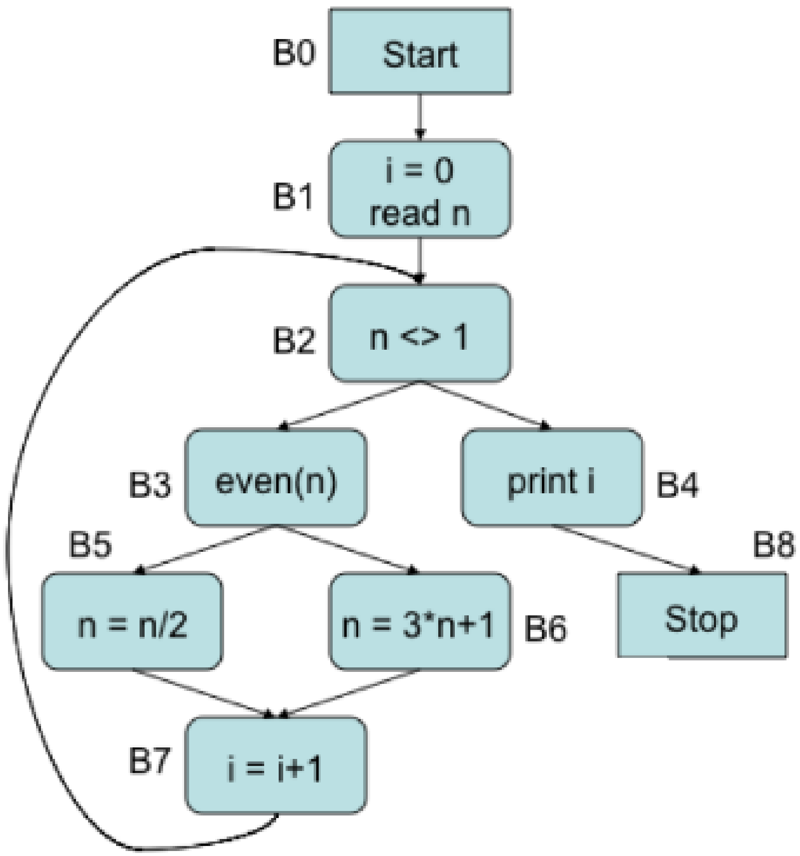
Applying the algorithm: B1



- IN[B1]
  = OUT[B0]
  = {B0}

- OUT[B1] = {B0, B1}

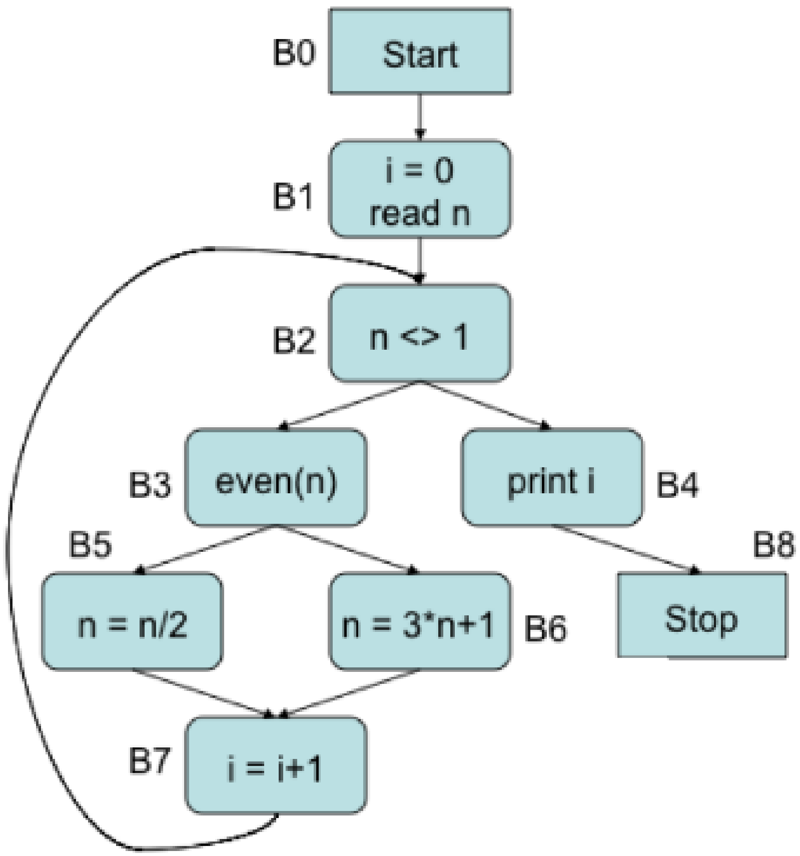
Applying the algorithm: B2



- IN[B2]
  = OUT[B1] ∩ OUT[B7]
  = {B0, B1}

- OUT[B2] = {B0, B1, B2}

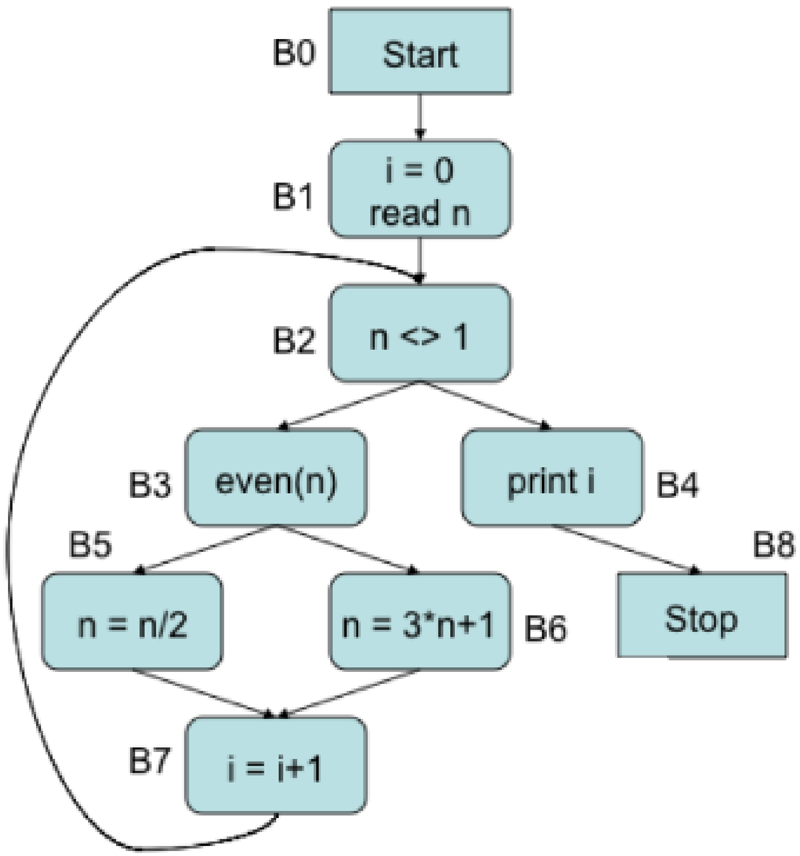
Applying the algorithm: B3



- IN[B3]
  = OUT[B2]
  = {B0, B1, B2}

- OUT[B3] = {B0, B1, B2, B3}

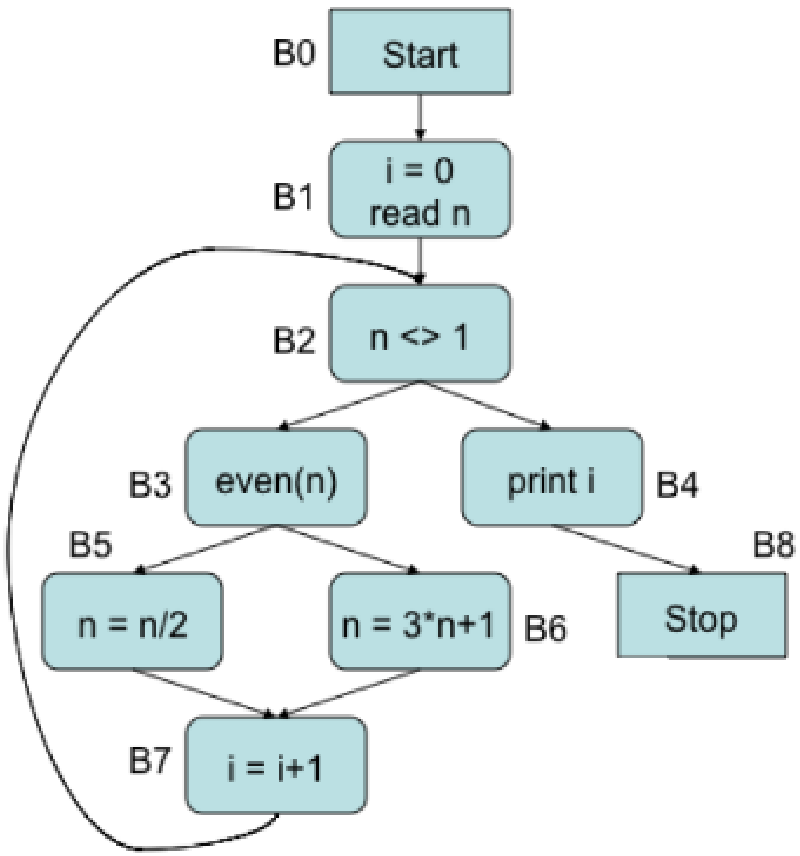
Applying the algorithm: B4



- IN[B4]
  = OUT[B2]
  = {B0, B1, B2}

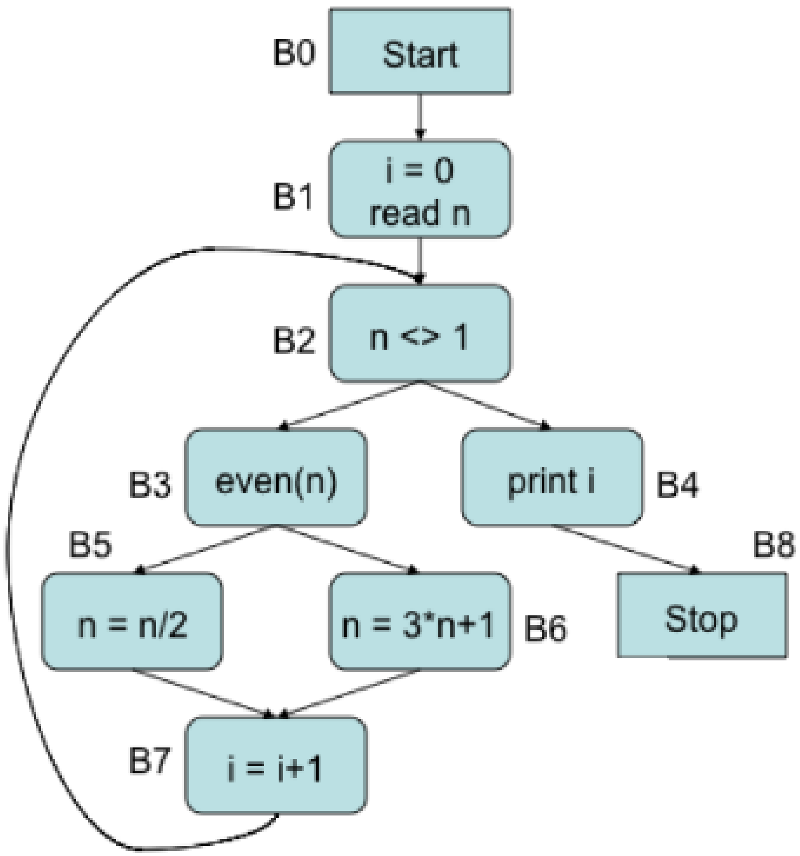
- OUT[B4] = {B0, B1, B2, B4}

Applying the algorithm: B5



- IN[B5]
  = OUT[B3]
  = {B0, B1, B2, B3}
- OUT[B5] = {B0, B1, B2, B3, B5}

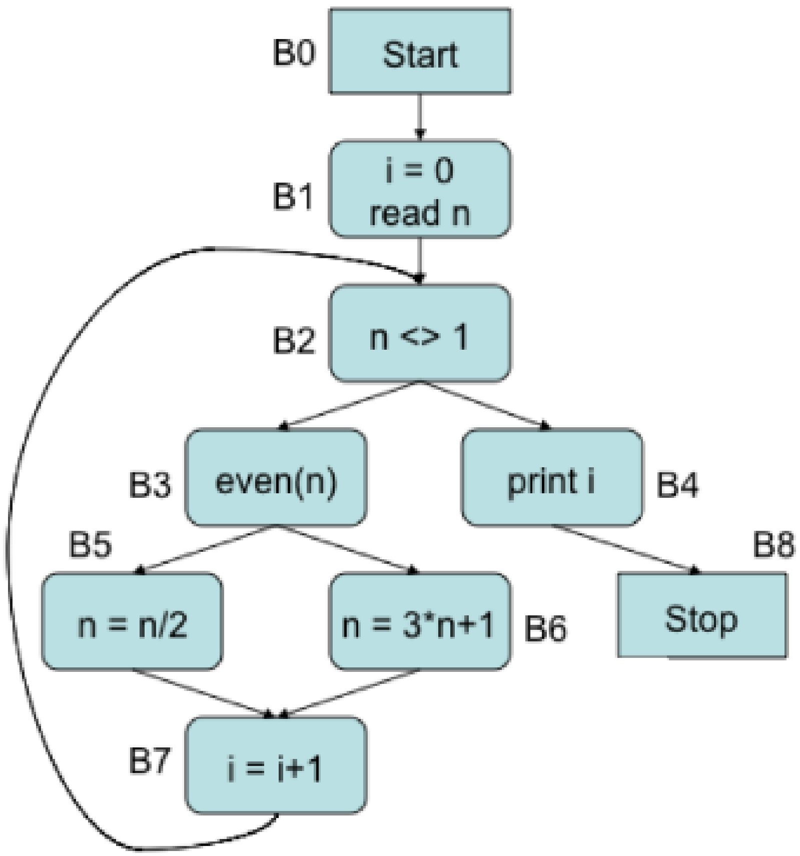
Applying the algorithm: B6



- IN[B6]
  = OUT[B3]
  = {B0, B1, B2, B3}

- OUT[B6] = {B0, B1, B2, B3, B6}

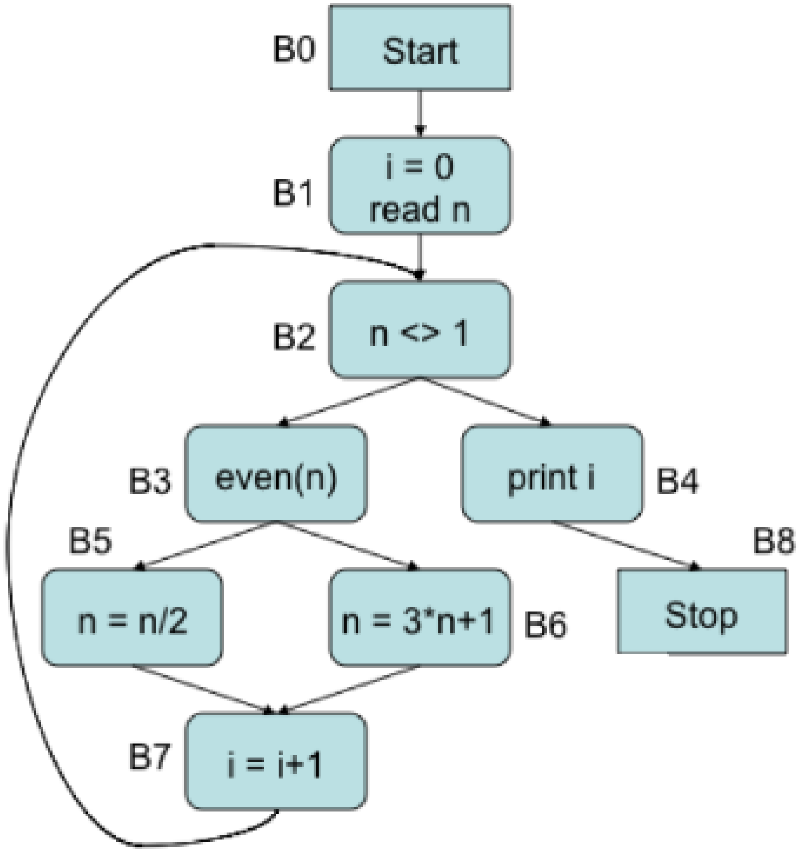
Applying the algorithm: B7



- IN[B7]
  = OUT[B5] ∩ OUT[B6]
  = {B0, B1, B2, B3}

- OUT[B7] = {B0, B1, B2, B3, B7}

Applying the algorithm: B8



- IN[B8]
  = OUT[B4]
  = {B0, B1, B2, B4}

  B8 is last node so OUT[B8] not required

Finally, the dominator tree