CC Week 5

Prepared for: 7th Sem, CE, DDU

Prepared by: Niyati J. Buch

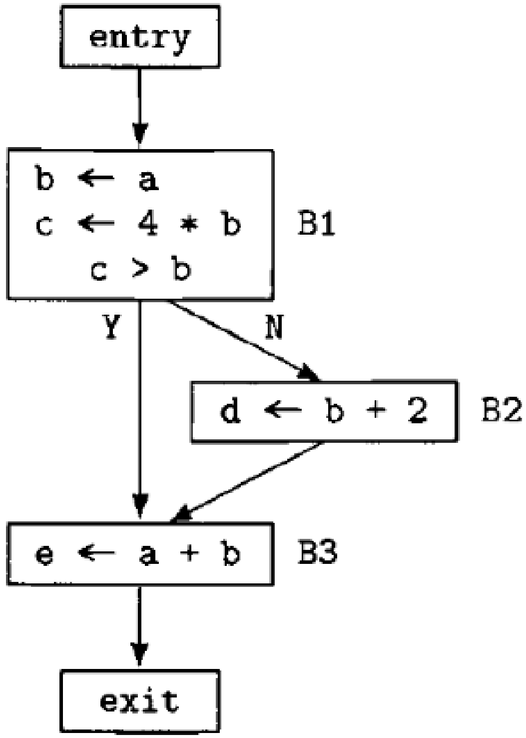
Ref. Book 1: Compiler: Principles, techniques and tools by Aho, Ullman and Sethi, 2nd Ed., Pearson Education
Ref. Book 2: Advanced Compiler Design & Implementation By Steven S Muchnick

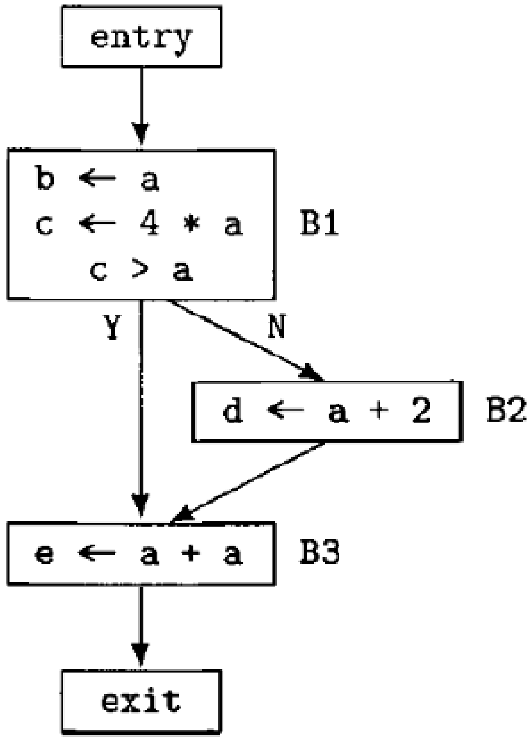
Contents

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**.

Example of Copy Propagation





(a) Example of a copy assignment to propagate, namely, b ← a in **B1**

(b) the result of doing copy propagation on it.

Phases of Copy Propagation

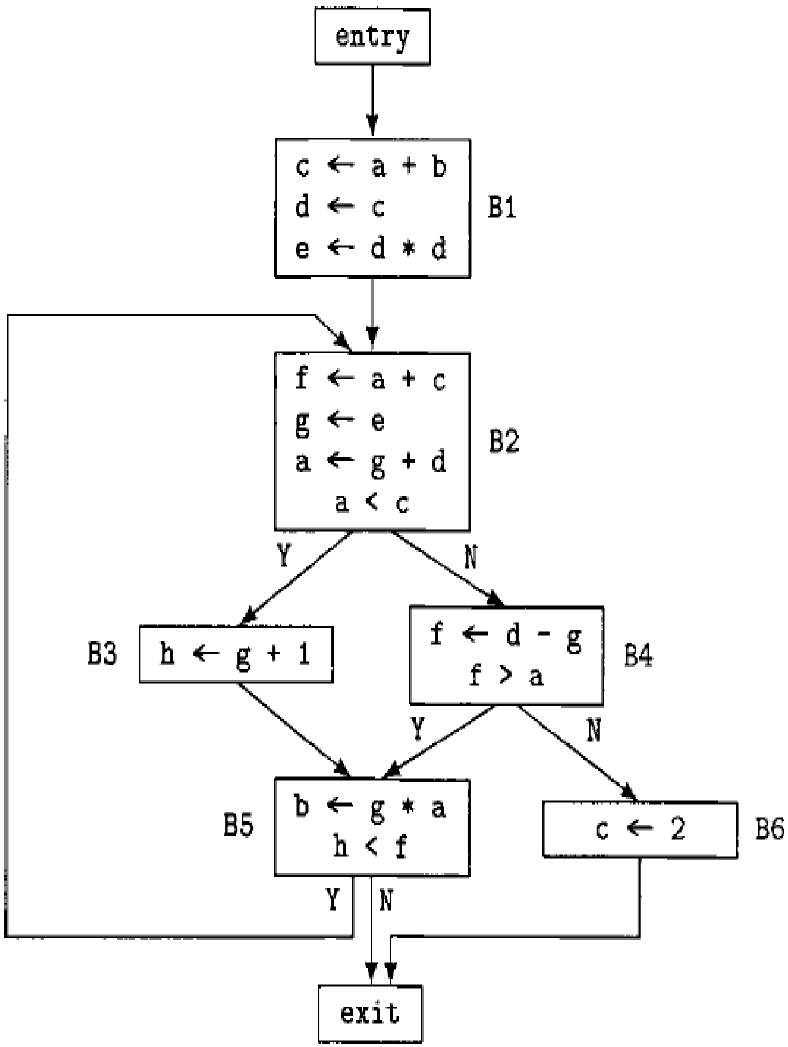
- Copy propagation can reasonably be divided into **local** and **global** phases,
  – the first operating within individual basic blocks and
  – the latter across the entire flow- graph,
- or it can be accomplished in a single global phase.

Example 1: Basic block of 5 instructions

Position	Code Before	ACP	Code After
		$\emptyset$	
1	b ← a		b ← a
		{⟨b,a⟩}	
2	c ← b + 1		c ← a + 1
		{⟨b,a⟩}	
3	d ← b		d ← a
		{⟨b,a⟩,⟨d,a⟩}	
4	b ← d + c		b ← a + c
		{⟨d,a⟩}	
5	b ← d		b ← a
		{⟨d,a⟩,⟨b,a⟩}	

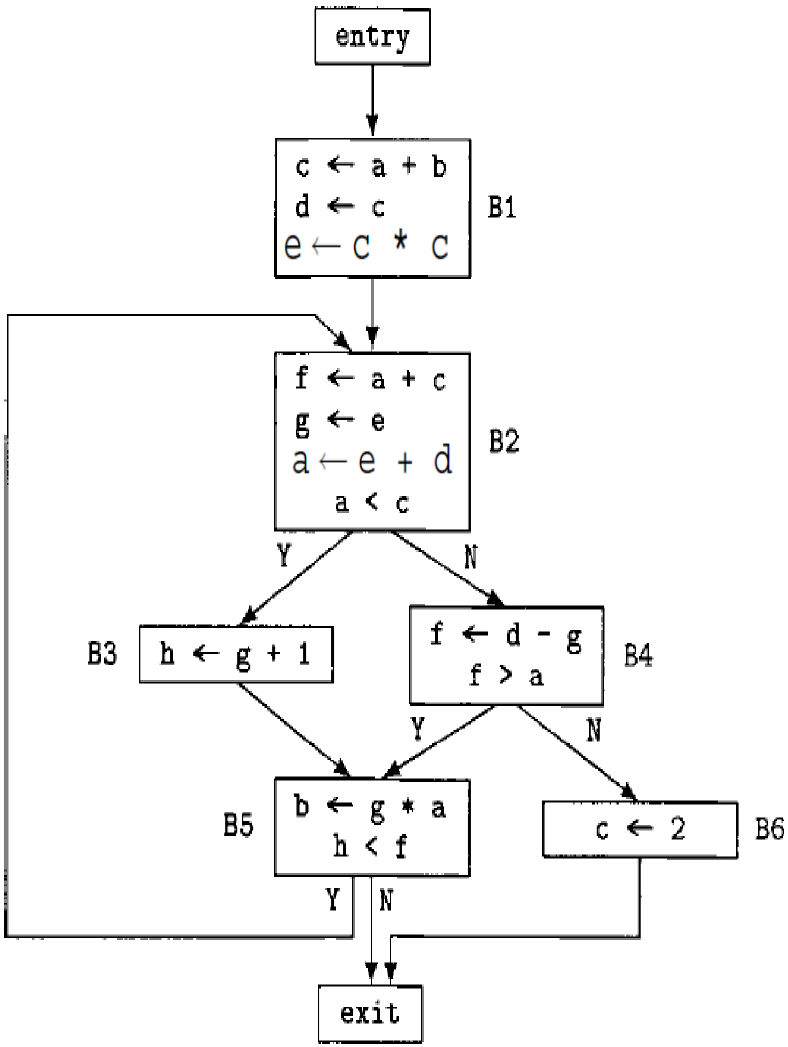
- The first column shows the position
- The second column shows a basic block of five instructions before applying the ACP algorithm
- The third column shows the value of ACP at each step
- The fourth column shows the result of applying ACP
- ACP = Available Copy Propagation

Example 2



- This is the flow graph **before** 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.

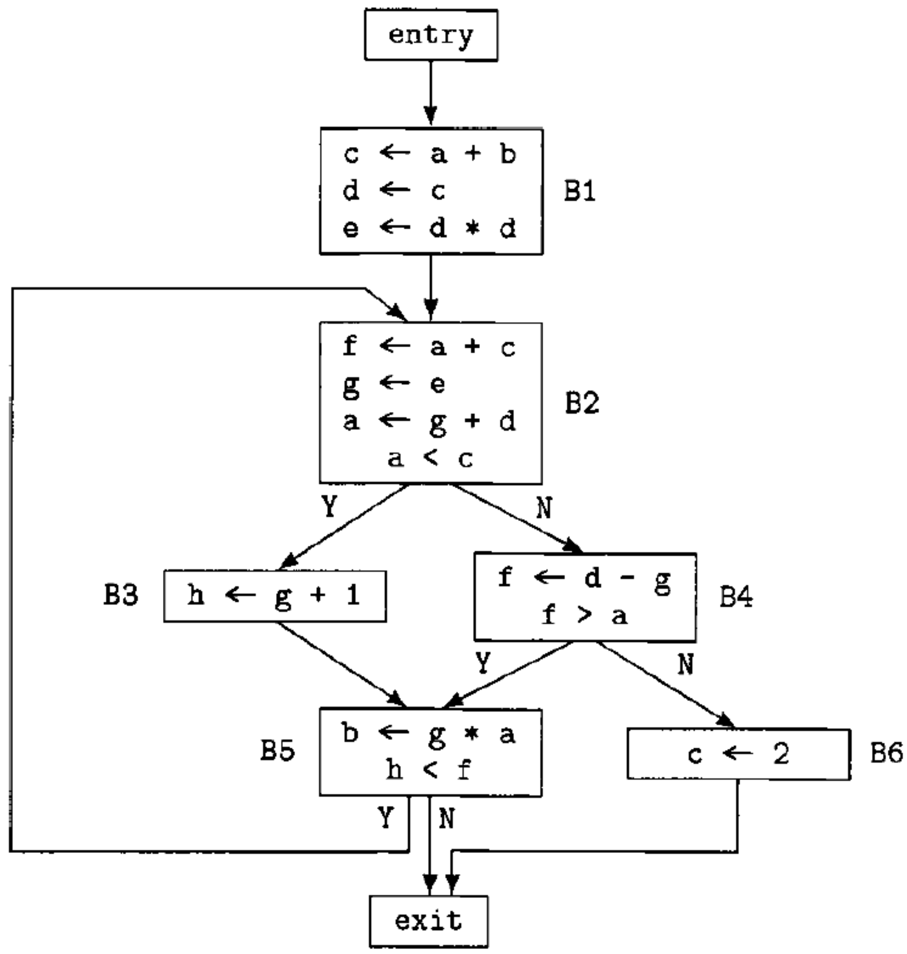
- We define **KILL(i)** to be the set of copy assignment instances killed by block i.

COPY(i) and KILL(i)

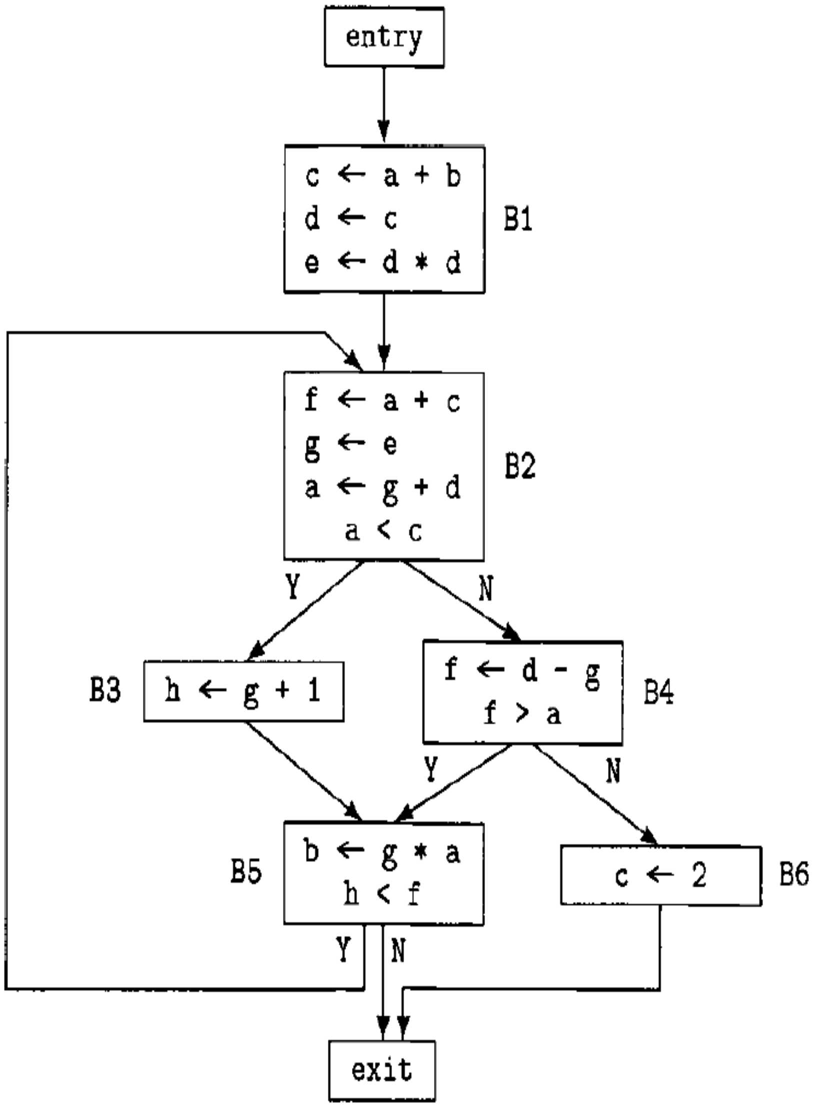
- COPY(i) is a set of quadruples **(u, v, i, pos)**,
  - such that **u ← v** is a copy assignment
  - and **pos** is the position in block **i** where the assignment occurs,
  - and neither **u** nor **v** is assigned to later in block **i**.

- 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**.

Find COPY(i) and KILL(i) for given flow graph

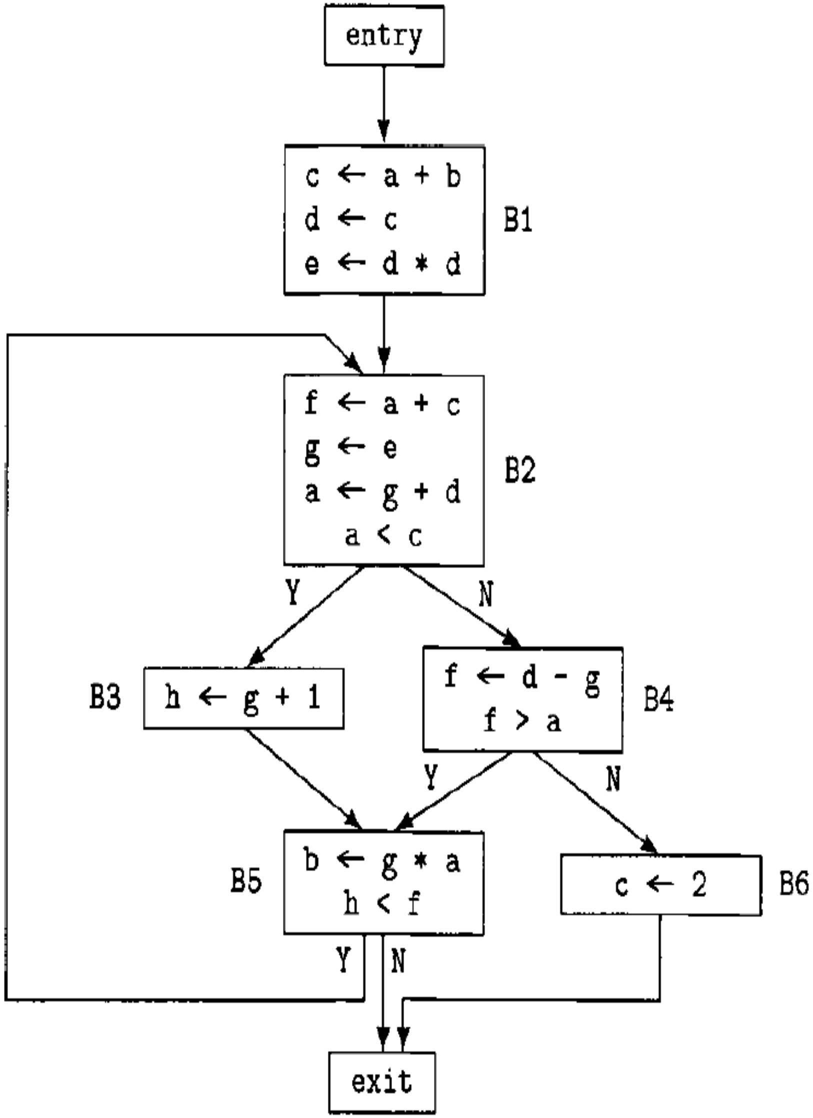


COPY(i) using set notation



- COPY(entry) = ∅
- COPY(B1) = {(d, c, B1, 2)}
- COPY(B2) = {(g, e, B2, 2)}
- COPY(B3) = ∅
- COPY(B4) = ∅
- COPY(B5) = ∅
- COPY(B6) = ∅
- COPY(exit) = ∅

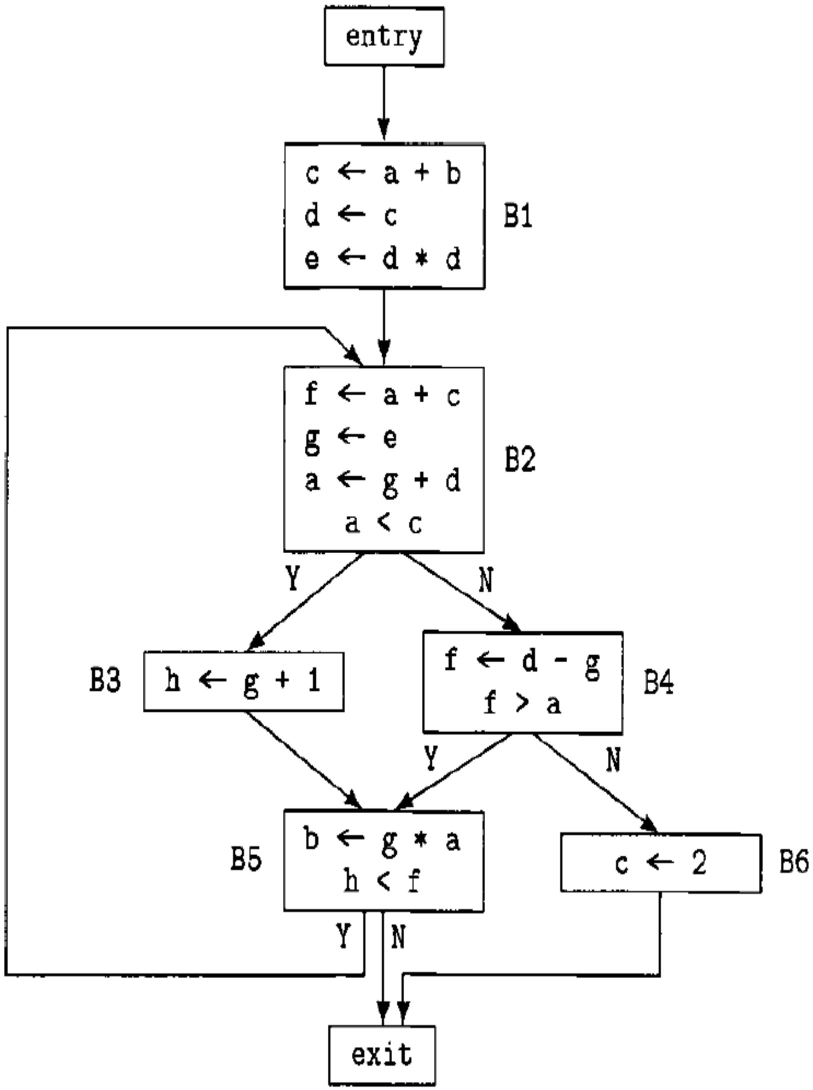
COPY(i) using vector representation



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

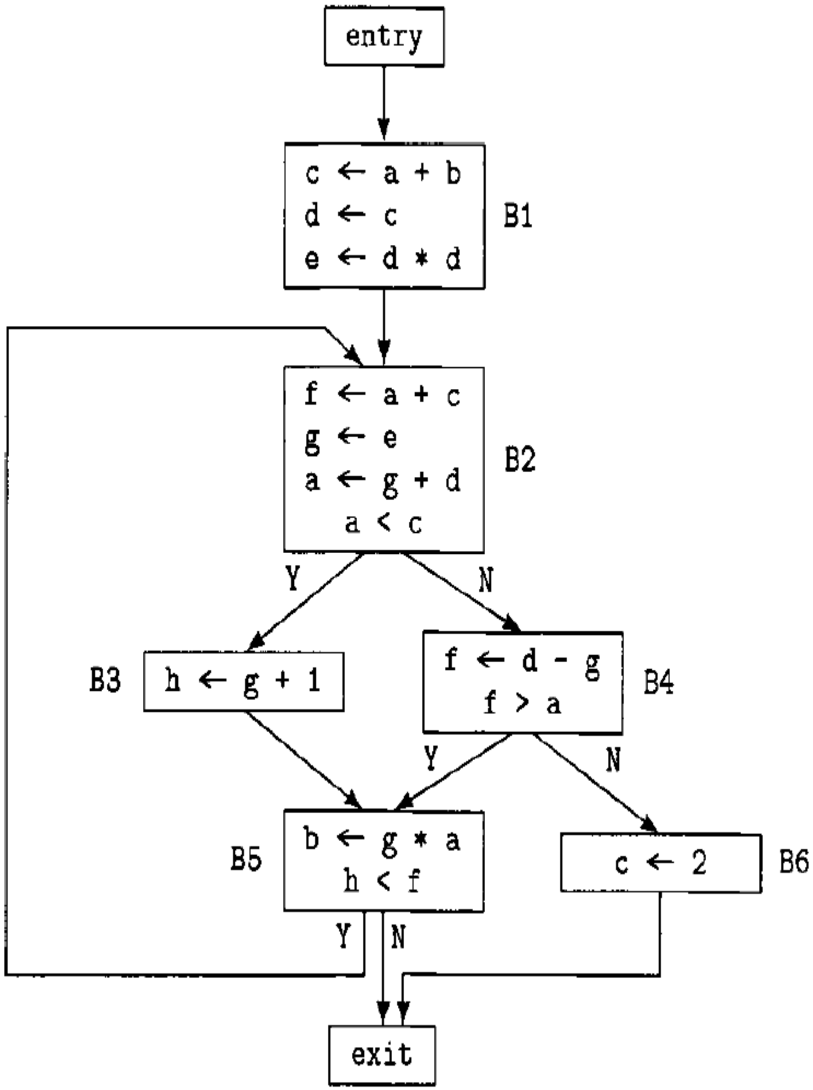
Bit position	COPY
1	{(d, c, B1, 2)}
2	{(g, e, B2, 2)}

KILL(i) using set notation



- KILL(entry) = ∅
- KILL(B1) = {(g, e, B2, 2)}
- KILL(B2) = ∅
- KILL(B3) = ∅
- KILL(B4) = ∅
- KILL(B5) = ∅
- KILL(B6) = {(d, c, B1, 2)}
- KILL(exit) = ∅

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 = ∪ COPY(i) for all i

CPin for all blocks

- 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

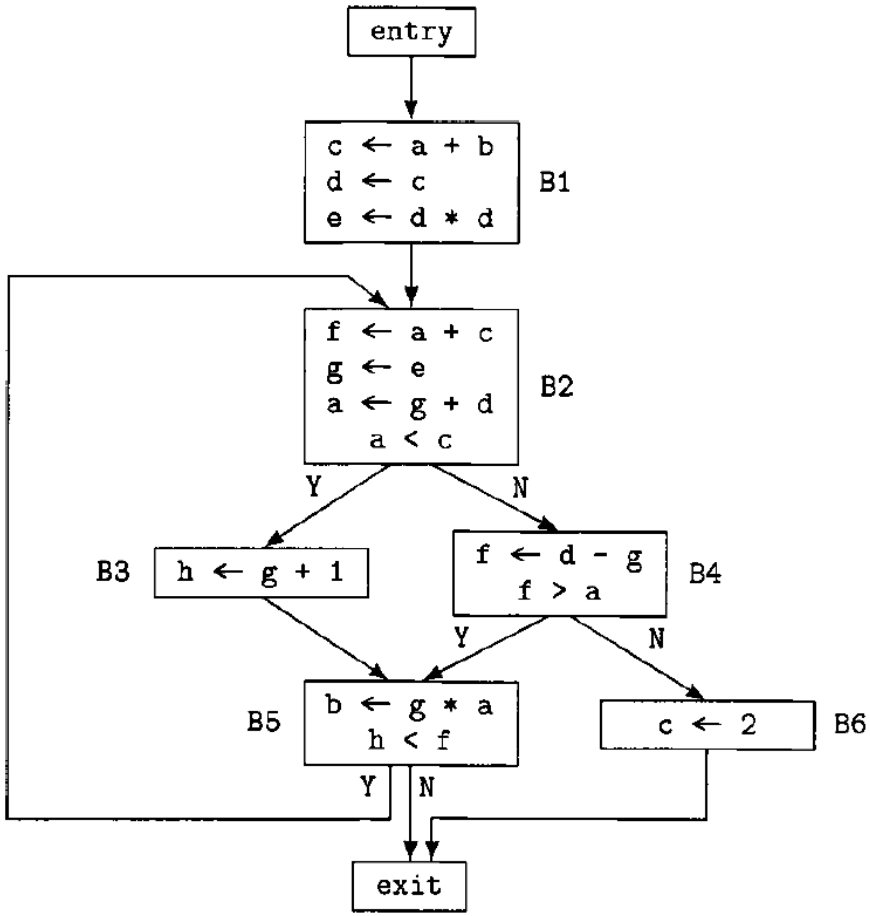
- CPin(i) = ∩ CPout(j) where j ∈ pred(i)
- CPout(i) = COPY(i) ∪ (CPin(i) - KILL(i))
- Equivalent:

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

- 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 **<u>reverse post-order</u>** (i.e. preorder means each block before its successors) order.

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

Applying iterative analysis for block entry

- CPin(entry) = <00>

- as per the equation as no predecessor is available.

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) ∪ (CPin(entry) - KILL(entry))
- CPin(B1) = <00> ∪ (<00> - <00>)
- **CPin(B1) = <00>**

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>**

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>**

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>**

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>**

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>**

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>**

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) ∪ (CPin(entry) - KILL(entry))
- CPin(B1) = <00> ∪ (<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

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

- 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

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

- 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

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

- 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)}

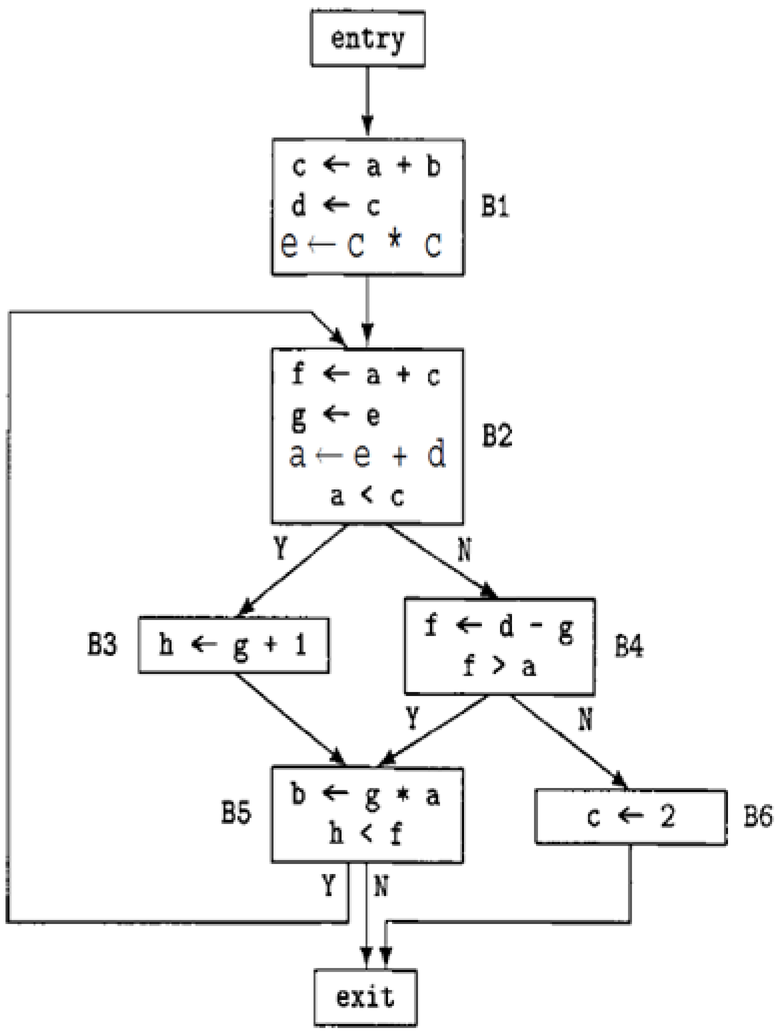
Global Copy Propagation

- 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.

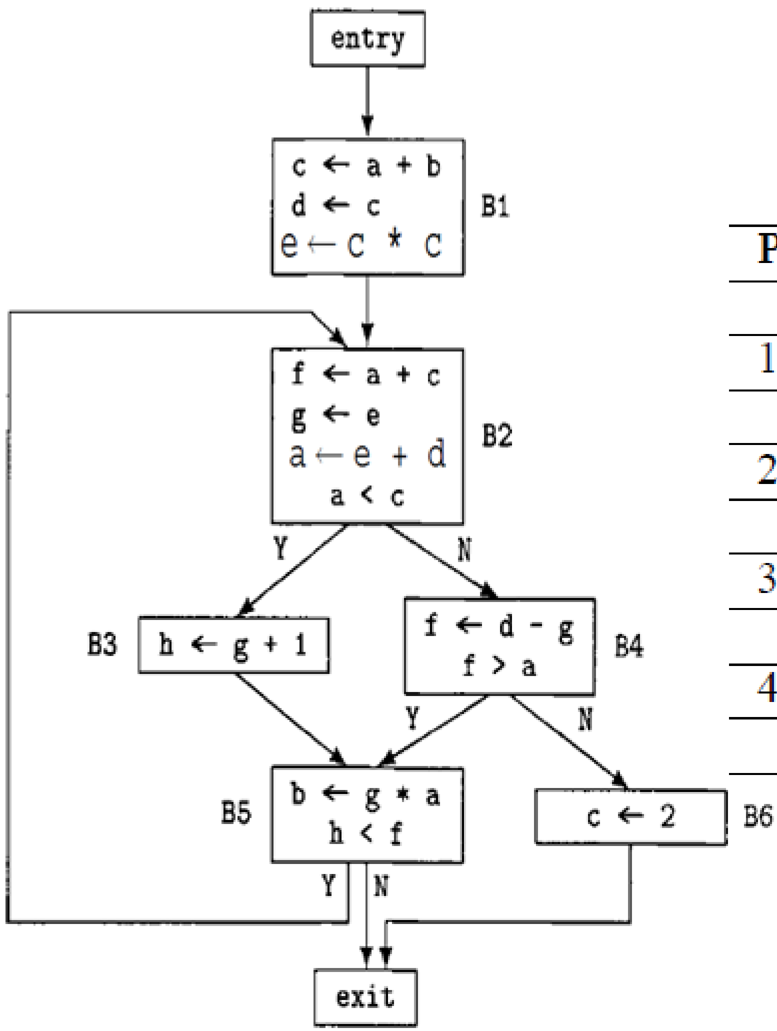
For block B1



- 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
		$\{\langle d, c \rangle\}$	

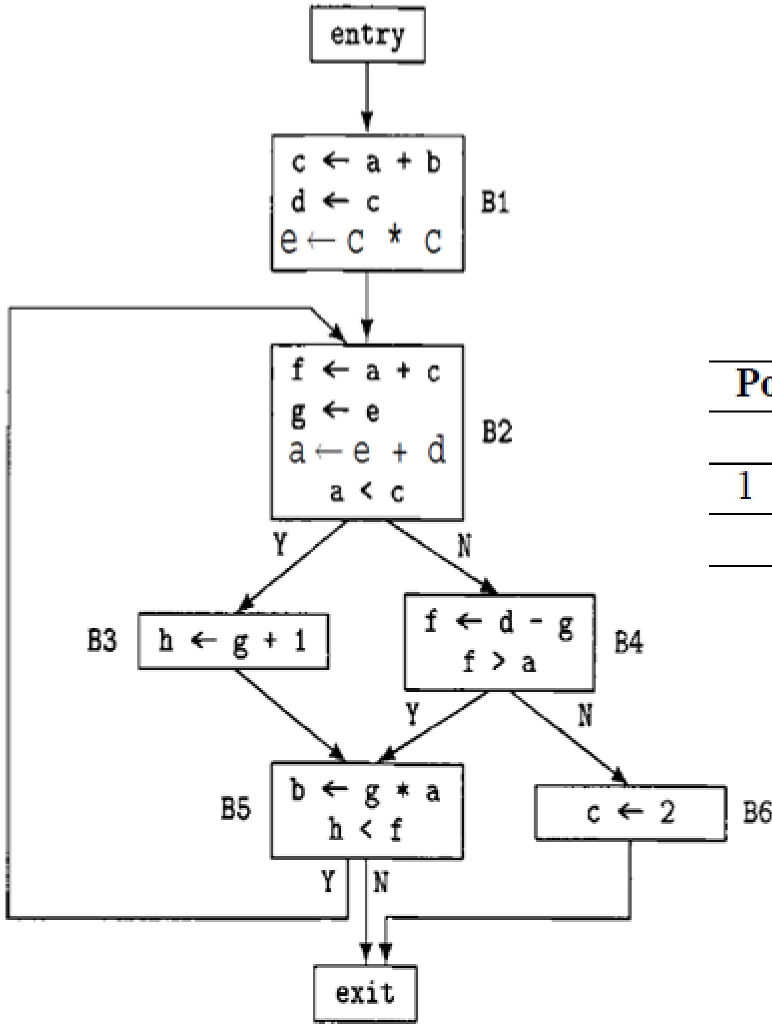
For block B2



- 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
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	

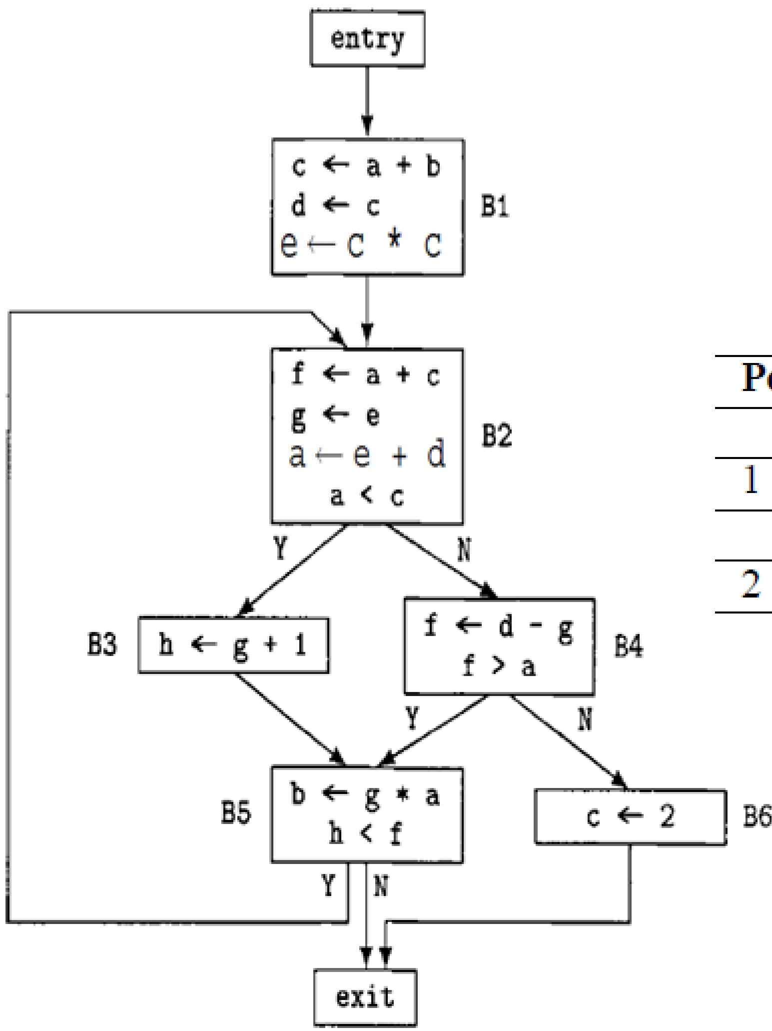
For block B3



- 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
		$\{\langle d, c\rangle, \langle g, e\rangle\}$	

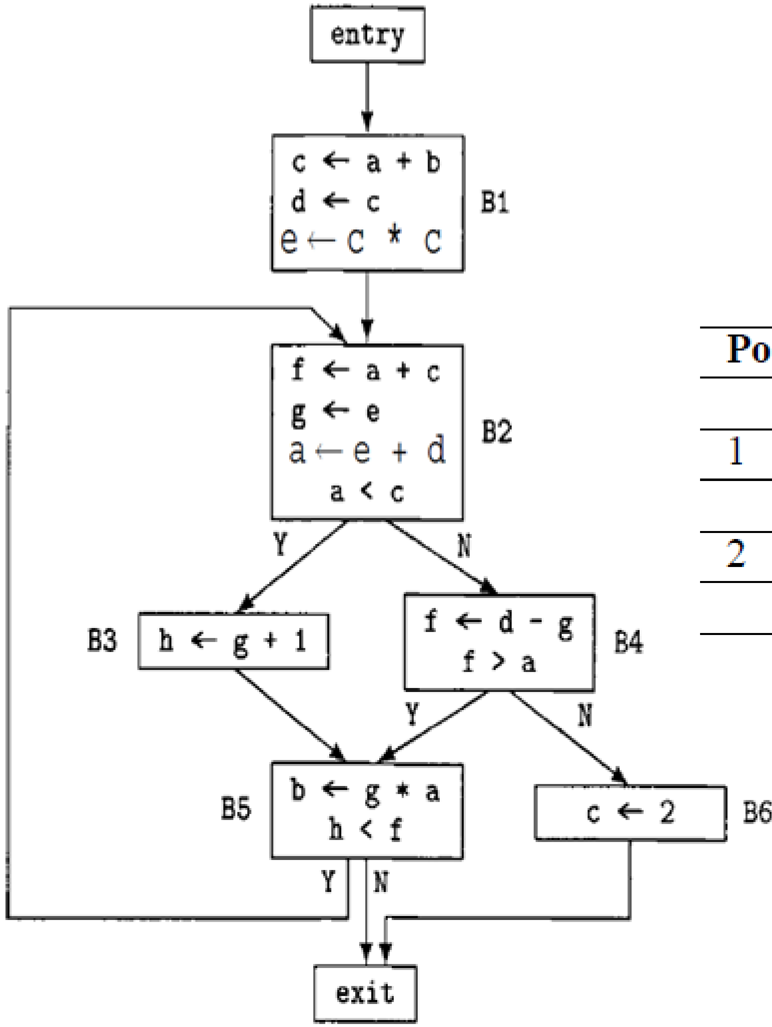
For block B4



- 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
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	

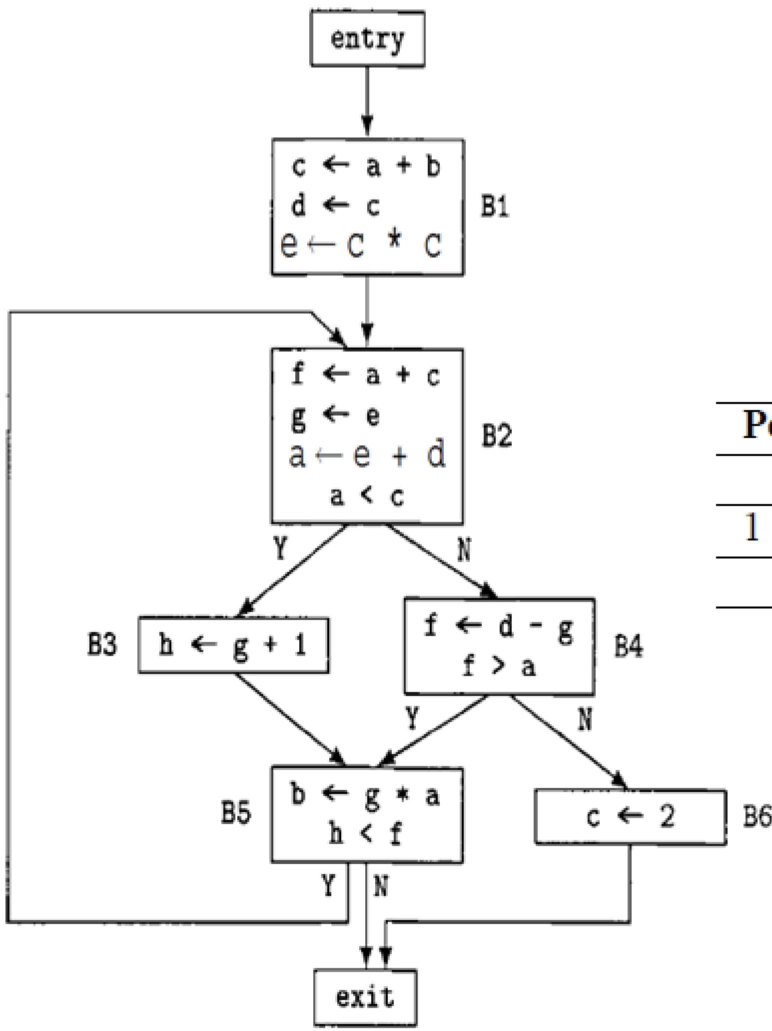
For block B5



- 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
		$\{\langle d,c\rangle,\langle g,e\rangle\}$	

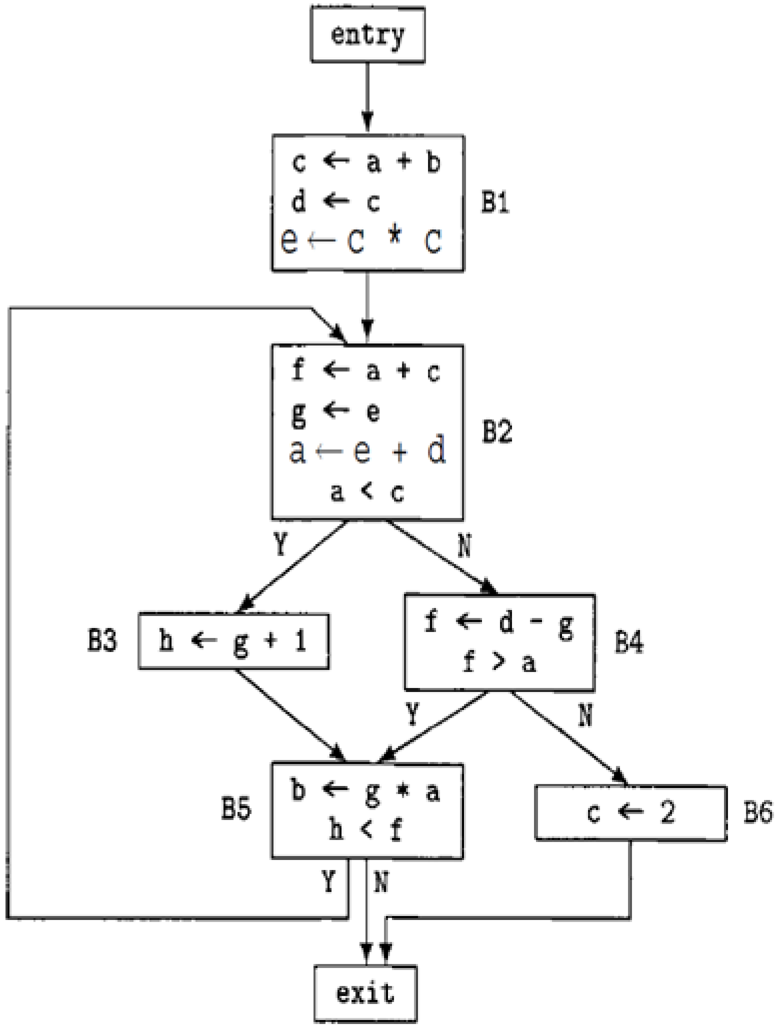
For block B6



- 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	$c \leftarrow 2$		$c \leftarrow 2$
		$\{\quad \langle g, e\rangle\}$	

For block exit



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

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

Finally, we have