EECS 583 – Class 6
Dataflow Analysis II

University of Michigan

September 15, 2025

Announcements & Reading Material

- HW 1 – Due tonight at midnight
  - See piazza/GSIs if you are stuck
  - Can use up to 2 late days (10% penalty per day)
- HW 2 coming out Wednesday
- Today's class
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988. (Sections: 10.5, 10.6, 10.9, 10.10 Edition 1; 9.2, 9.3 Edition 2)
- Next 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.

Recap: Liveness vs Reaching Defs

Liveness

OUT = Union(IN(succs))
IN = GEN + (OUT – KILL)

Bottom-up dataflow
Any path
Keep track of variables/registers
Uses of variables → GEN
Defs of variables → KILL

Reaching Definitions/DU/UD

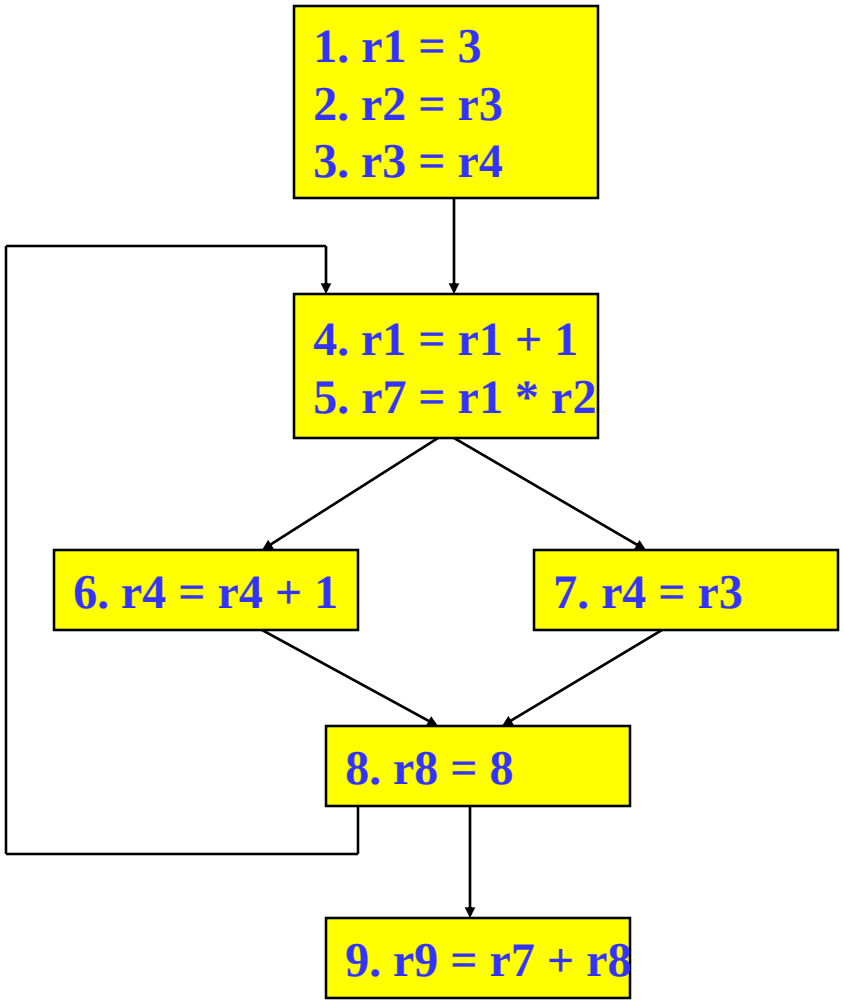
IN = Union(OUT(preds))
OUT = GEN + (IN – KILL)

Top-down dataflow
Any path
Keep track of instruction IDs
Defs of variables → GEN
Defs of variables → KILL

From Last Time: DU/UD Chains

- Convenient way to access/use reaching defs info
- Def-Use chains
  - Given a def, what are all the possible consumers of the operand produced
  - Maybe consumer
- Use-Def chains
  - Given a use, what are all the possible producers of the operand consumed
  - Maybe producer

Example – DU/UD Chains



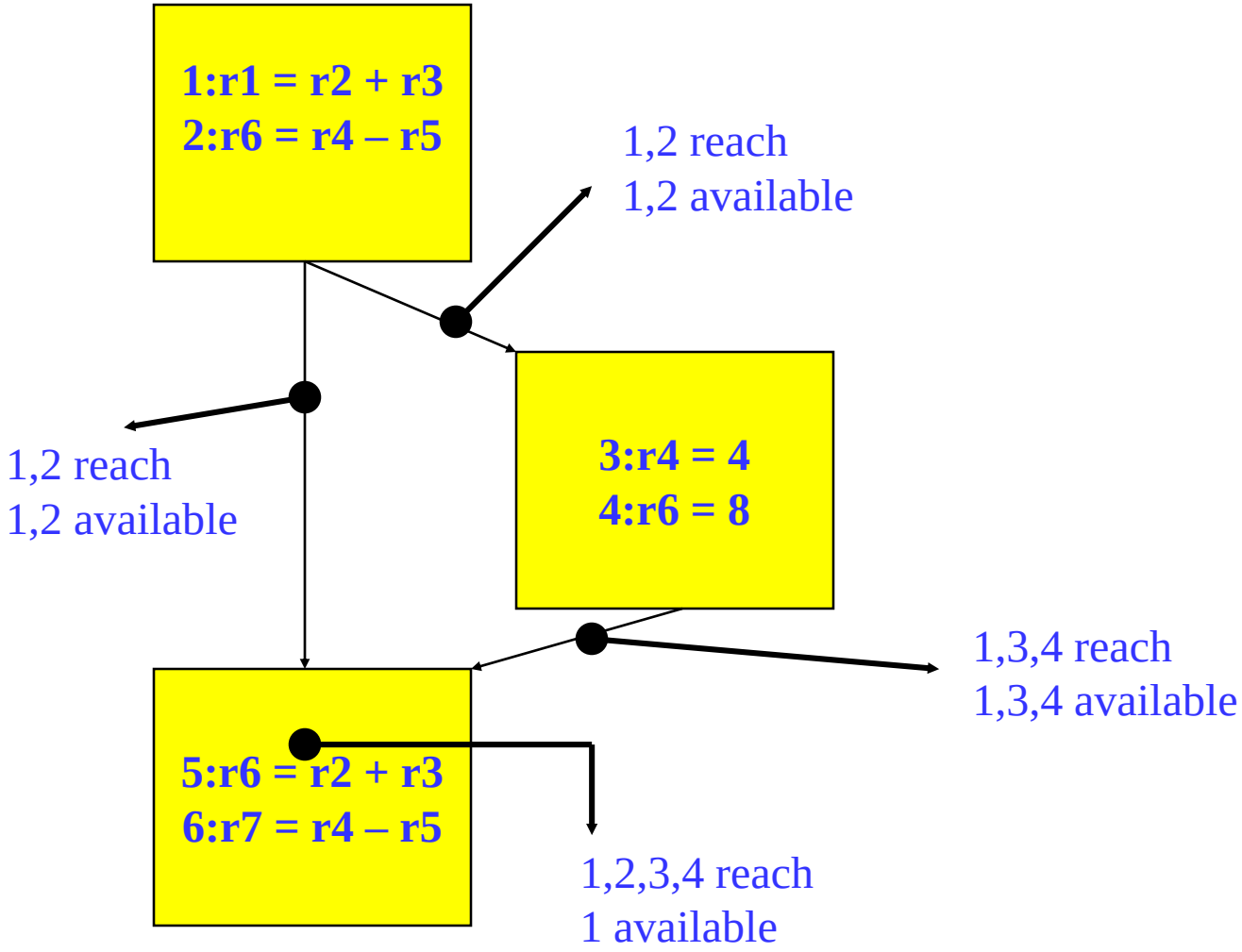
Generalizing Dataflow Analysis

- Transfer function
  - How information is changed by "something" (BB)
  - OUT = GEN + (IN – KILL) /* forward analysis, e.g., rdefs */
  - IN = GEN + (OUT – KILL) /* backward analysis, e.g., liveness */
- Meet function
  - How information from multiple paths is combined
  - IN = Union(OUT(predecessors)) /* forward analysis */
  - OUT = Union(IN(successors)) /* backward analysis */
- Generalized dataflow algorithm
  - while (change)
    - change = false
    - for each BB
      - apply meet function
      - apply transfer functions
      - if any changes → change = true

What About All Path Problems?

- Up to this point
  - Any path problems (maybe relations)
    - Definition reaches along some path
    - Some sequence of branches in which def reaches
    - Lots of defs of the same variable may reach a point
  - Use of Union operator in meet function
- All-path: Definition guaranteed to reach
  - Regardless of sequence of branches taken, def reaches
  - Can always count on this
  - Only 1 def can be guaranteed to reach
  - Availability (as opposed to reaching)
    - Available definitions
    - Available expressions (could also have reaching expressions, but not that useful)

Reaching vs Available Definitions



Available Definition Analysis (Adefs)

- A definition d is available at a point p if along all paths from d to p, d is not killed
- Remember, a definition of a variable is killed between 2 points when there is another definition of that variable along the path
  - r1 = r2 + r3 kills previous definitions of r1
- Algorithm
  - Forward dataflow analysis as propagation occurs from defs downwards
  - Use the Intersect function as the meet operator to guarantee the all-path requirement
  - GEN/KILL/IN/OUT similar to reaching defs
    - Initialization of IN/OUT is the tricky part

Compute GEN/KILL Sets for each BB (Adefs)

Exactly the same as reaching defs !!!

```
for each basic block in the procedure, X, do
   GEN(X) = 0
   KILL(X) = 0
   for each operation in sequential order in X, op, do
      for each destination operand of op, dest, do
         G = op
         K = {all ops which define dest – op}
         GEN(X) = G + (GEN(X) – K)
         KILL(X) = K + (KILL(X) – G)
      endfor
   endfor
endwhile
```

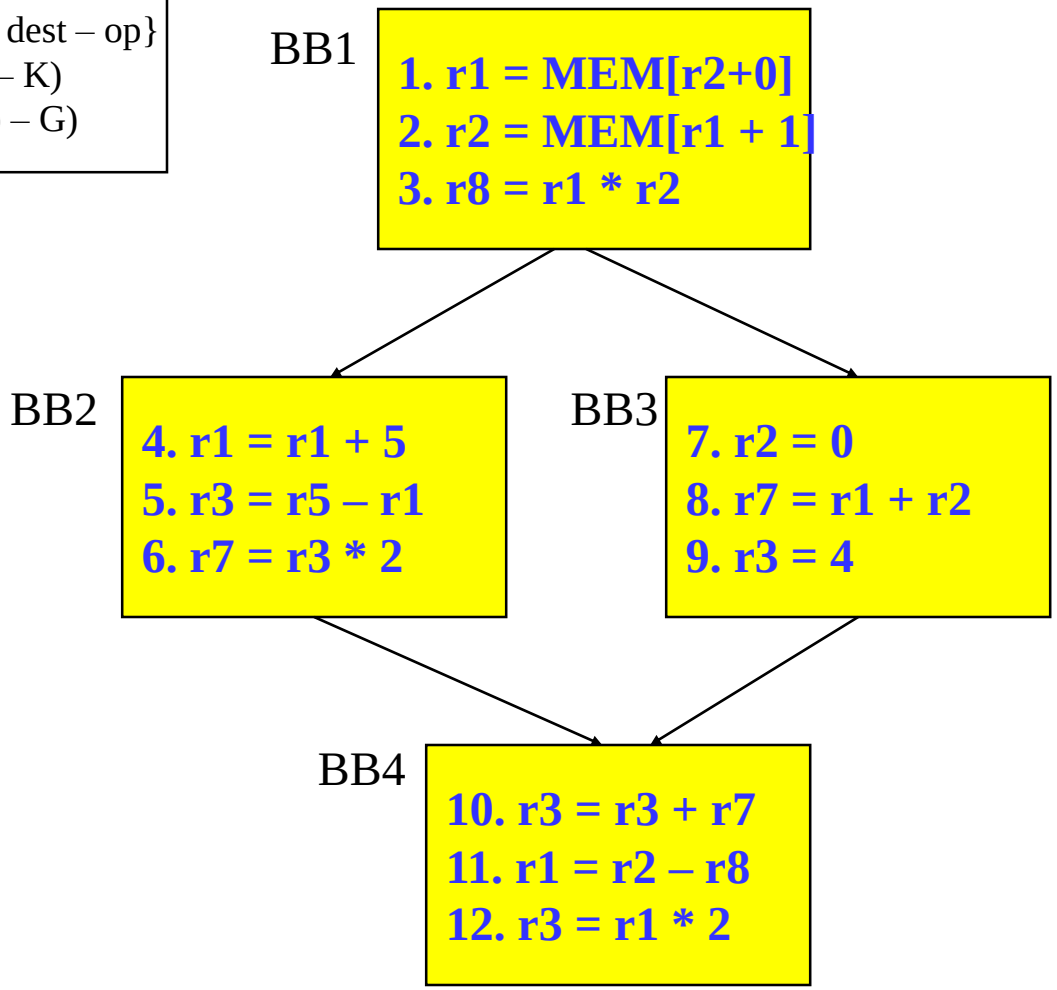
Compute IN/OUT Sets for all BBs (Adefs)

U = universal set of all operations in the Procedure
IN(0) = 0
OUT(0) = GEN(0)
for each basic block in procedure, W, (W != 0), do
    IN(W) = 0
    OUT(W) = U – KILL(W)

change = 1
while (change) do
    change = 0
    for each basic block in procedure, X, do
        old_OUT = OUT(X)
        IN(X) = **Intersect**(OUT(Y)) for all predecessors Y of X
        OUT(X) = GEN(X) + (IN(X) – KILL(X))
        if (old_OUT != OUT(X)) then
            change = 1
        endif
    endfor
endwhile

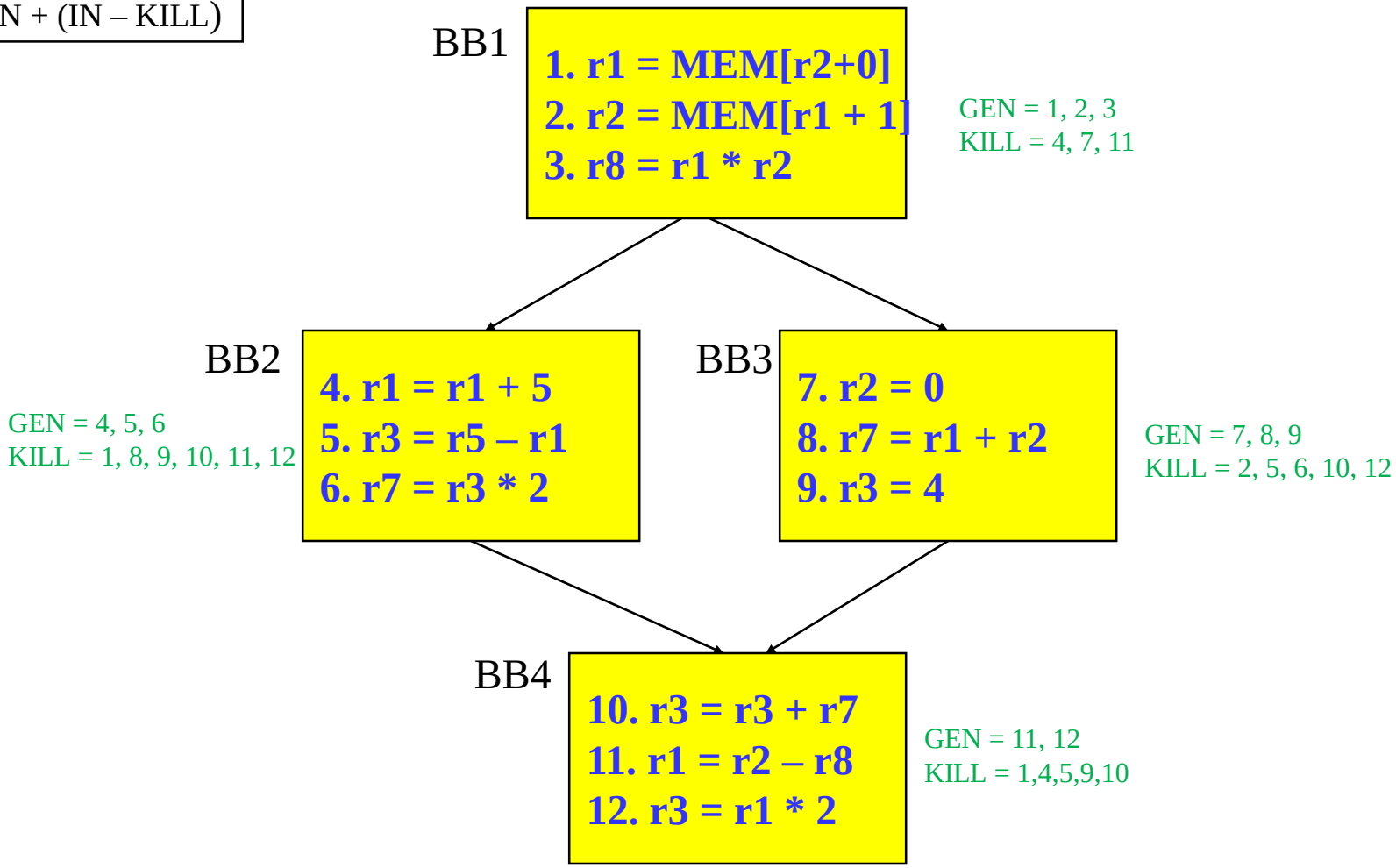
Example Adef Calculation

G = op
K = {all ops which define dest – op}
GEN(X) = G + (GEN(X) – K)
KILL(X) = K + (KILL(X) – G)



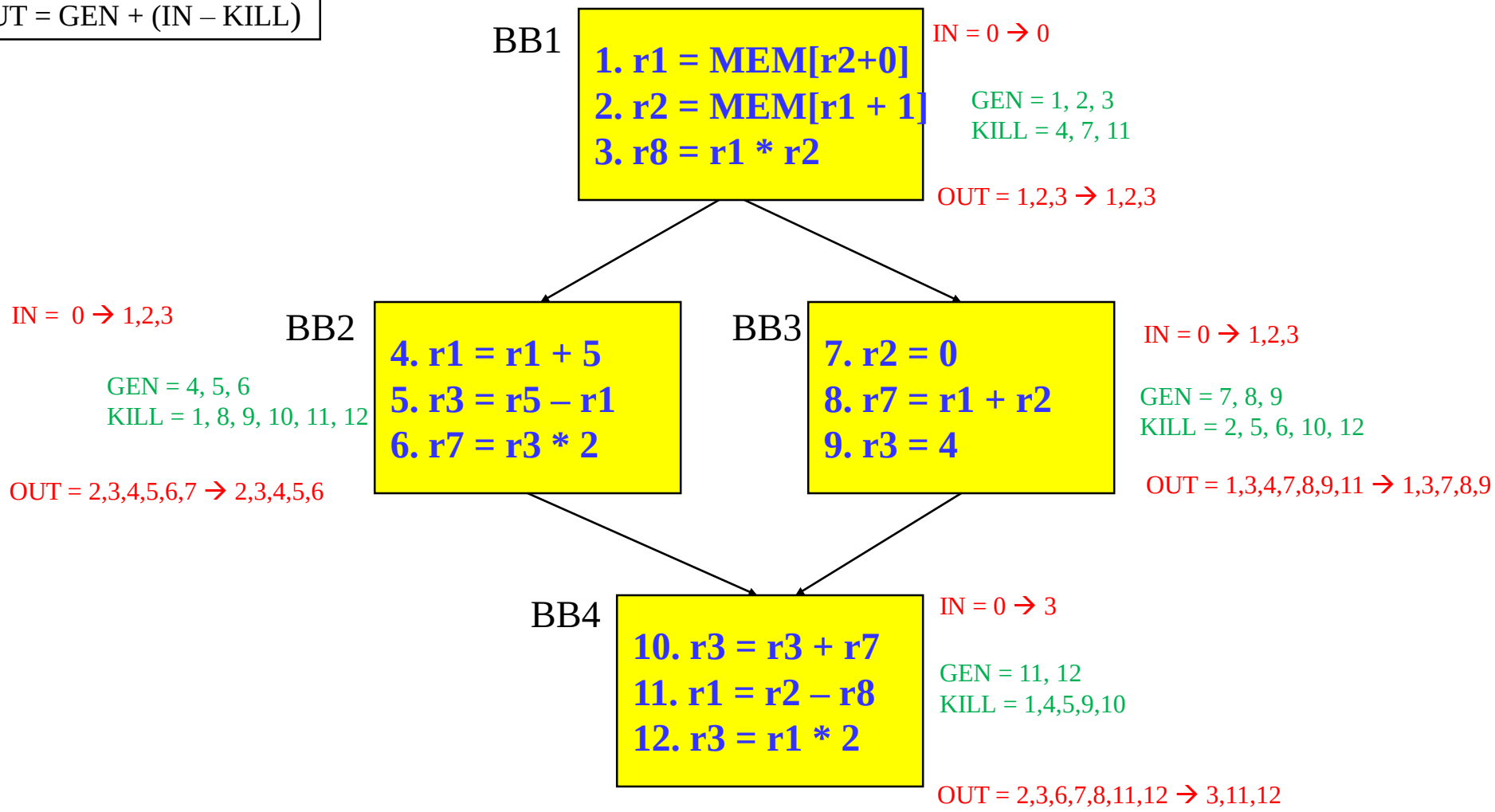
Example Adef Calculation - Continued

IN = Intersect(OUT(preds))
OUT = GEN + (IN – KILL)



Example Adef Calculation - Answer

IN = Intersect(OUT(preds))
OUT = GEN + (IN – KILL)



Available Expression Analysis (Aexprs)

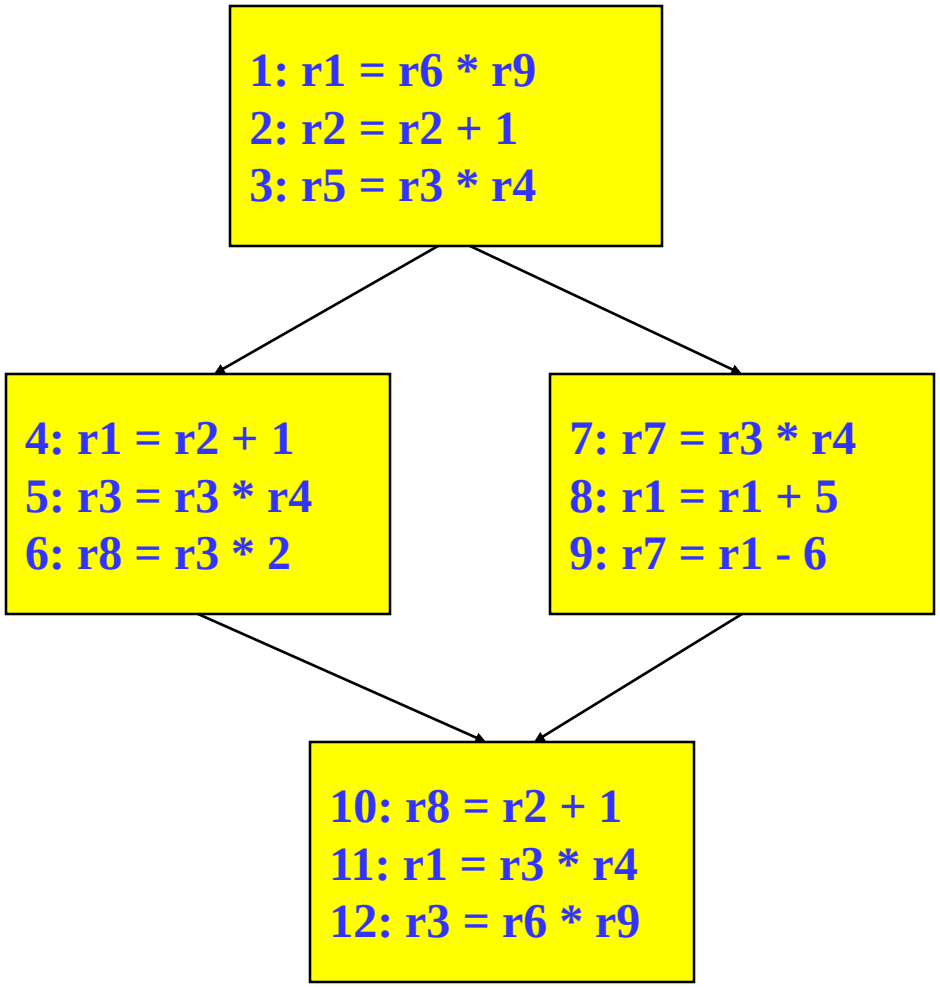
- An expression is a RHS of an operation
  - r2 = r3 + r4, r3+r4 is an expression
- An expression e is available at a point p if along all paths from e to p, e is not killed
- An expression is killed between 2 points when one of its source operands are redefined
  - r1 = r2 + r3 kills all expressions involving r1
- Algorithm
  - Forward dataflow analysis as propagation occurs from defs downwards
  - Use the Intersect function as the meet operator to guarantee the all-path requirement
  - Looks exactly like adefs, except GEN/KILL/IN/OUT are the RHS's of operations rather than the LHS's

Computation of Aexpr GEN/KILL Sets

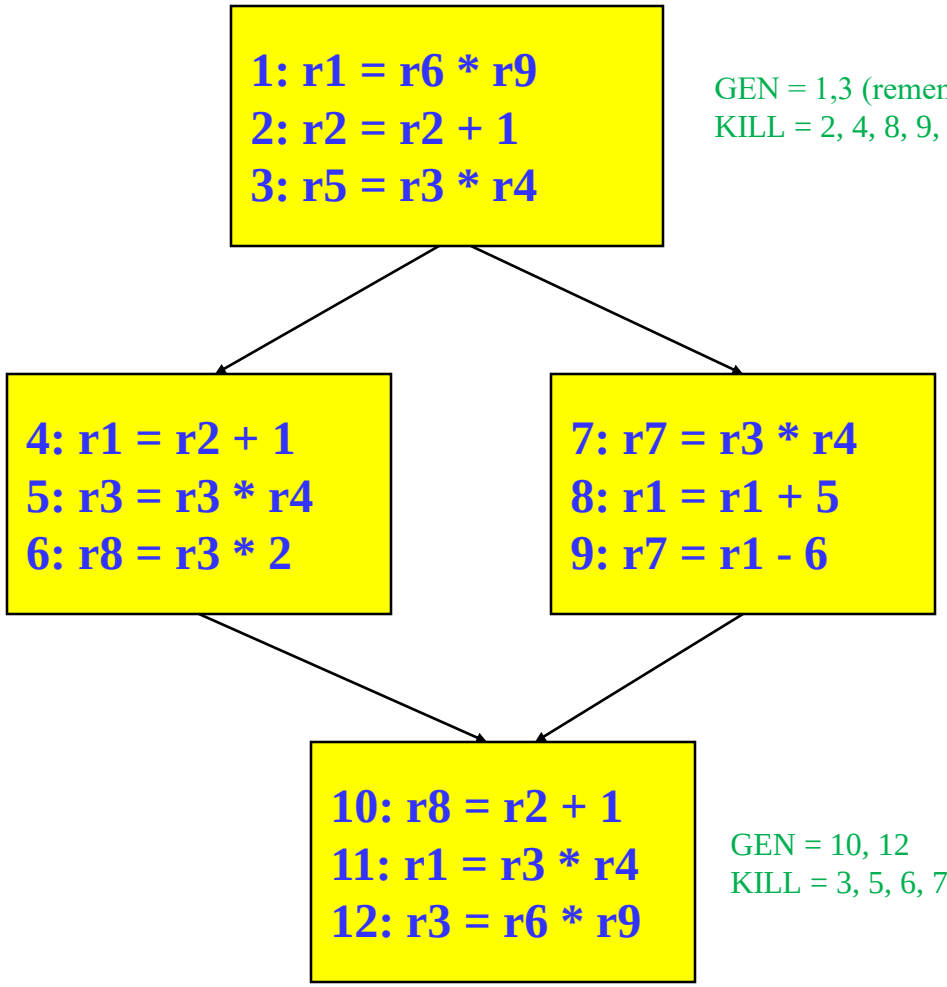
We can also formulate the GEN/KILL slightly differently so you do not need to break up instructions like "r2 = r2 + 1".

```
for each basic block in the procedure, X, do
   GEN(X) = 0
   KILL(X) = 0
   for each operation in sequential order in X, op, do
      K = 0
      for each destination operand of op, dest, do
         K += {all ops which use dest}
      endfor
      if (op not in K)
            G = op
      else
            G = 0
      GEN(X) = G + (GEN(X) – K)
      KILL(X) = K + (KILL(X) – G)
   endfor
endfor
```

Example Aexpr Calculation



Example Aexpr Calculation - Continued



GEN = 1,3 (remember {1, 3} means {“r6*r9”, “r3*r4”})
KILL = 2, 4, 8, 9, 10

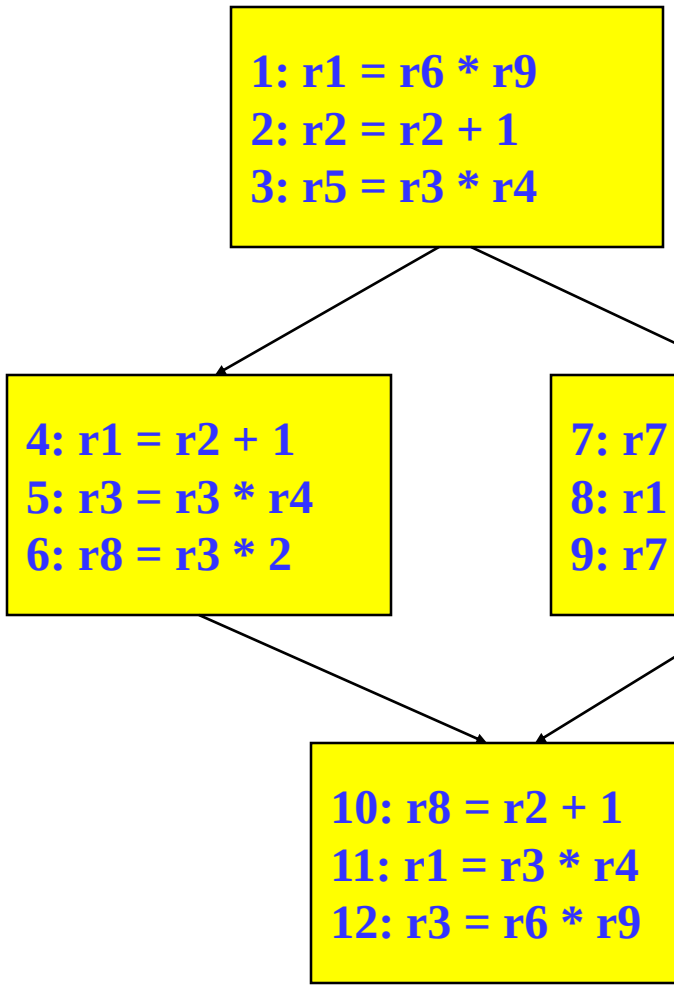
GEN = 4, 6
KILL = 3, 5, 7, 8, 9, 11

GEN = 7, 9
KILL = 8

GEN = 10, 12
KILL = 3, 5, 6, 7, 8, 9, 11

Example Aexpr Calculation - Answer

IN/OUT sets
A → B
A = initial state
B = after first iteration



IN = - → -

GEN = 1,3 (remember {1, 3} means {"r6*r9", "r3*r4"})
KILL = 2, 4, 8, 9, 10

OUT = 1,3,5,6,7,11,12 → 1,3

IN = - → 1,3

GEN = 4, 6
KILL = 3, 5, 7, 8, 9, 11

OUT = 1,2,4,6,10,12 → 1,4,6

IN = - → 1,3

GEN = 7, 9
KILL = 8

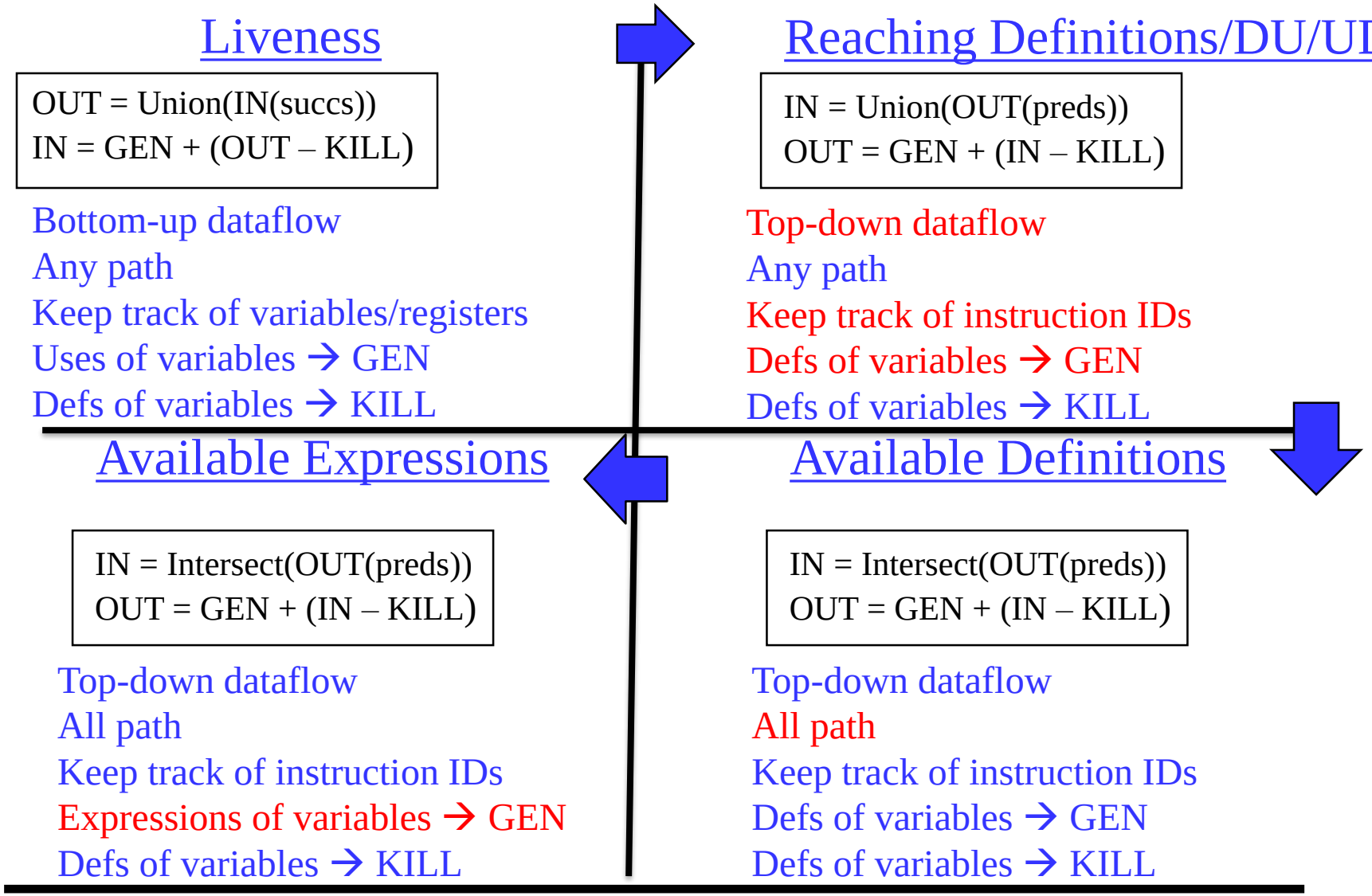
OUT = 1,2,3,4,5,6,7,9,10,11,12 → 1,3,7,9

IN = - → 1

GEN = 10, 12
KILL = 3, 5, 6, 7, 8, 9, 11

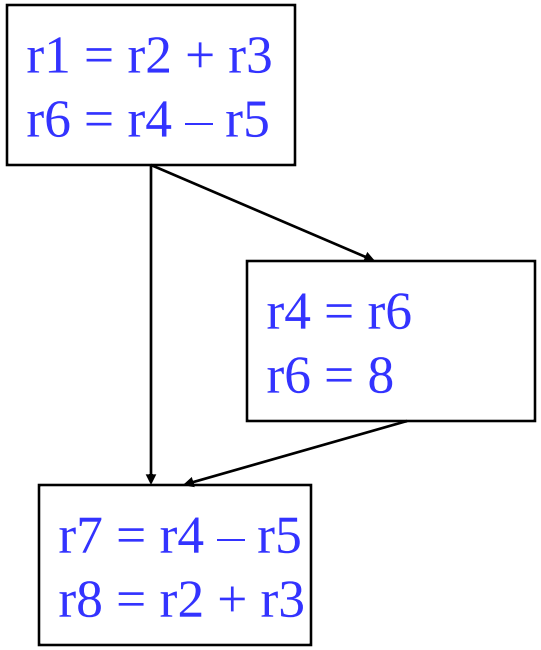
OUT = 1,2,4,10,12 → 1,10,12

Dataflow Summary



Static Single Assignment (SSA) Form

- Difficulty with optimization
  - Multiple definitions of the same register
  - Which definition reaches
  - Is expression available?

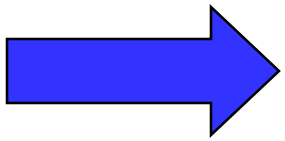


- Static single assignment
  - Each assignment to a variable is given a unique name
  - All of the uses reached by that assignment are renamed
  - DU chains become obvious based on the register name!

Converting to SSA Form

- Trivial for straight line code

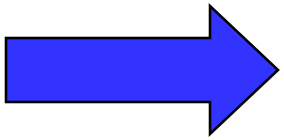
```
x = -1
y = x
x = 5
z = x
```



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

- More complex with control flow – Must use Phi nodes

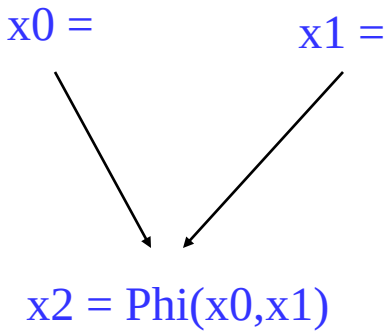
```
if ( ... )
   x = -1
else
   x = 5
y = x
```



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

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

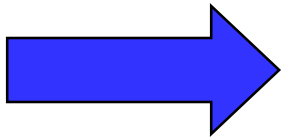


- Phi nodes are required when 2 non-null paths X→Z and Y→Z converge at node Z, and nodes X and Y contain assignments to V

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 Plusses and Minuses

- Advantages of SSA
  - Explicit DU chains – Trivial to figure out what defs reach a use
    - Each use has exactly 1 definition!!!
  - Explicit merging of values
  - Makes optimizations easier
- Disadvantages
  - When transform the code, must either recompute (slow) or incrementally update (tedious)

Generation of SSA Form

- 2 big parts
  - Determine where Phi nodes for each variable need to go
  - Rename the variables
- Algorithm next time!!