EECS 583 – Class 5
Dataflow Analysis

University of Michigan

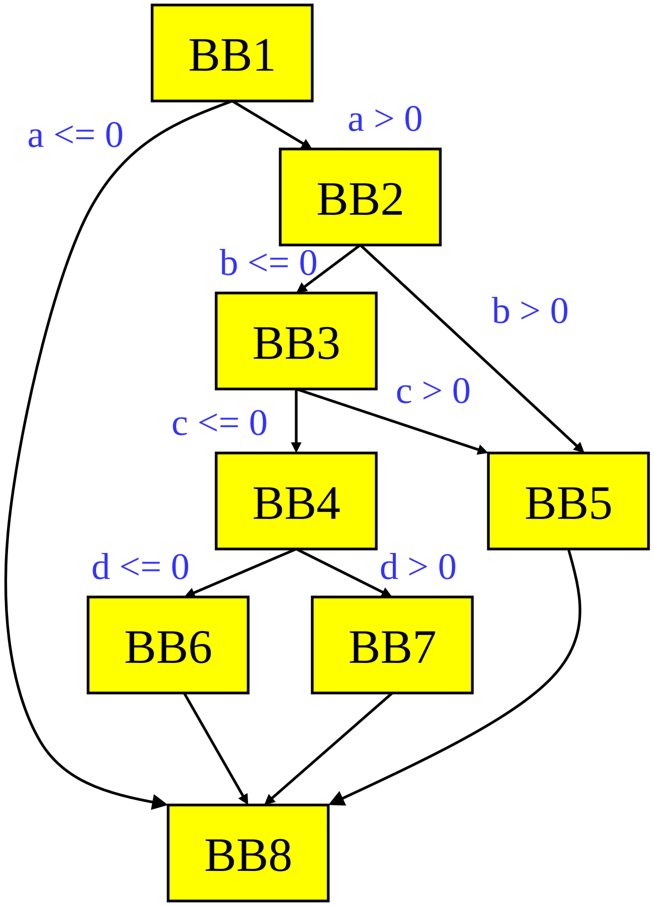
September 19, 2018

Reading Material + Announcements

- Reminder – HW 1 due next Monday at midnight
  - Submit uniquename_hw1.tgz file to:
    - eecs583a.eecs.umich.edu:/hw1_submissions
  - Before asking questions: 1) Read all threads on piazza, 2) Think a bit
    - Then, post question or talk to Ze if you are stuck
- Today's class
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988. (Chapters: 10.5, 10.6 Edition 1; Chapters 9.2 Edition 2)
- Material for next Monday
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988. (Chapters: 10.5, 10.6, 10.9, 10.10 Edition 1; Chapters 9.2, 9.3 Edition 2)

Class Problem From Last Time - Answer

```
if (a > 0) {
    r = t + s
    if (b > 0 || c > 0)
        u = v + 1
    else if (d > 0)
        x = y + 1
    else
        z = z + 1
}
```



BB	CD
1	-
2	1
3	-2
4	-3
5	2,3
6	-4
7	4
8	-

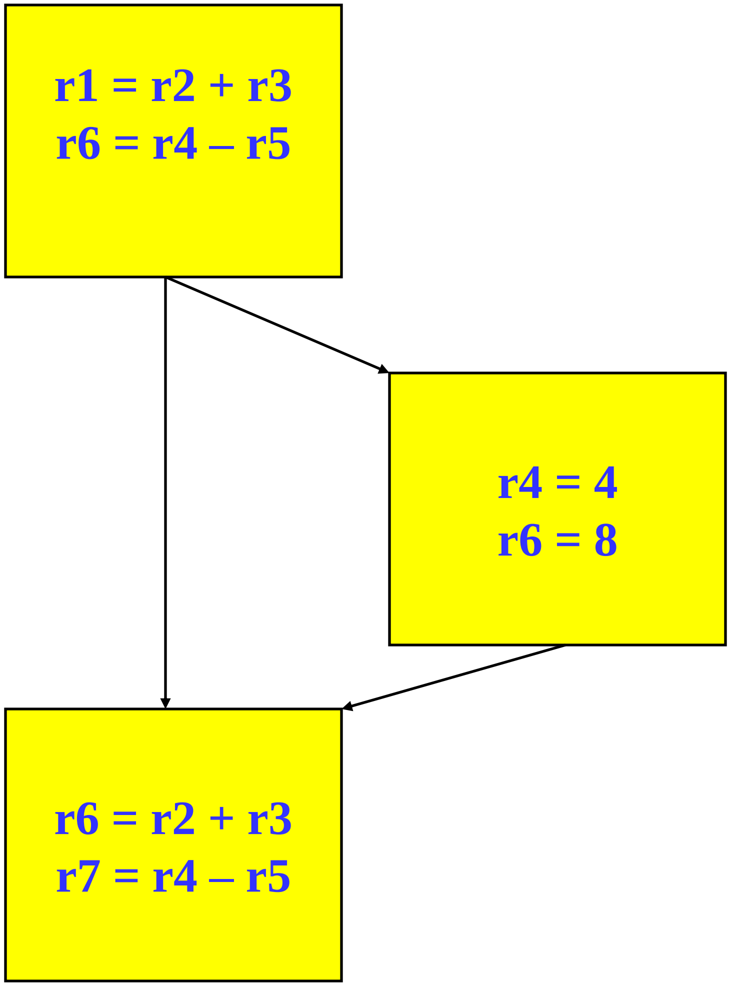
```
p3 = 0
p1 = CMPP.UN (a > 0) if T
r = t + s if p1
p2,p3 = CMPP.UC.ON (b > 0) if p1
p4,p3 = CMPP.UC.ON (c > 0) if p2
u = v + 1 if p3
p5,p6 = CMPP.UC.UN (d > 0) if p4
x = y + 1 if p6
z = z + 1 if p5
```

a. Draw the CFG

b. Compute CD

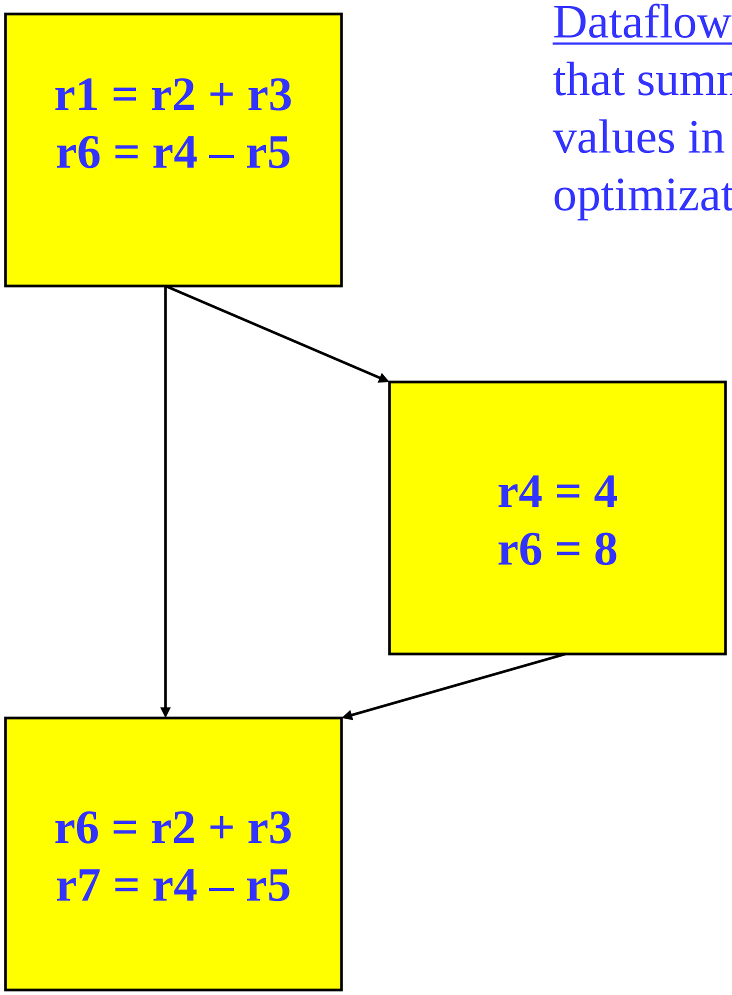
c. If-convert the code

Looking Inside the Basic Blocks: Dataflow Analysis + Optimization



- Control flow analysis
  - Treat BB as black box
  - Just care about branches
- Now
  - Start looking at ops in BBs
  - What’s computed and where
- Classical optimizations
  - Want to make the computation more efficient
- Ex: Common Subexpression Elimination (CSE)
  - Is r2 + r3 redundant?
  - Is r4 – r5 redundant?
  - What if there were 1000 BB’s
  - Dataflow analysis !!

Dataflow Analysis Introduction



r1 = r2 + r3
r6 = r4 – r5

r4 = 4
r6 = 8

r6 = r2 + r3
r7 = r4 – r5

Dataflow analysis – Collection of information that summarizes the creation/destruction of values in a program. Used to identify legal optimization opportunities.

Pick an arbitrary point in the program

Which VRs contain useful data values? (liveness or upward exposed uses)

Which definitions may reach this point? (reaching defns)

Which definitions are guaranteed to reach this point? (available defns)

Which uses below are exposed? (downward exposed uses)

Live Variable (Liveness) Analysis

- Defn: For each point p in a program and each variable y, determine whether y can be used before being redefined starting at p
- Algorithm sketch
  - For each BB, y is live if it is used before defined in the BB or it is live leaving the block
  - Backward dataflow analysis as propagation occurs from uses upwards to defs
- 4 sets
  - GEN = set of external variables consumed in the BB
  - KILL = set of external variable uses killed by the BB
    - equivalent to set of variables defined by the BB
  - IN = set of variables that are live at the entry point of a BB
  - OUT = set of variables that are live at the exit point of a BB

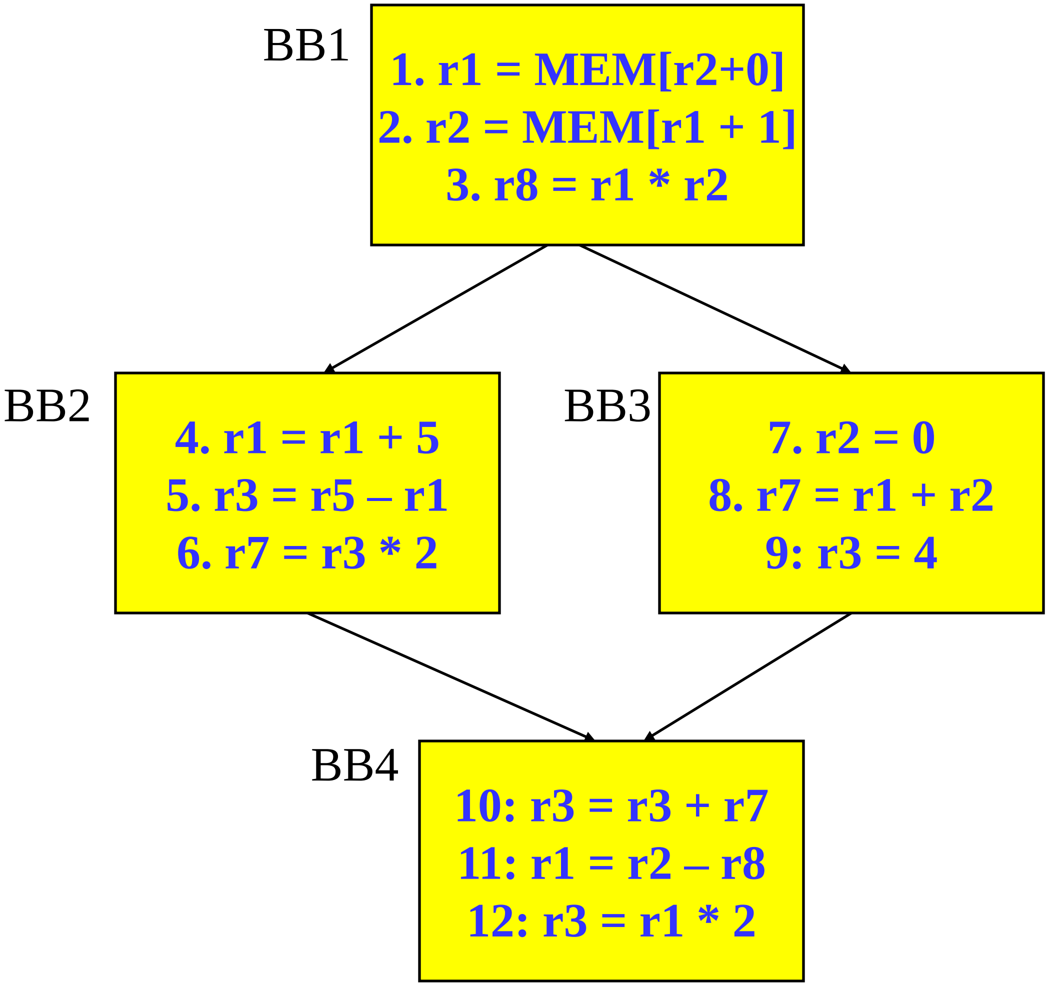
Computing GEN/KILL Sets For Each BB

```
for each basic block in the procedure, X, do
    GEN(X) = 0
    KILL(X) = 0
    for each operation in reverse sequential order in X, op, do
        for each destination operand of op, dest, do
            GEN(X) -= dest
            KILL(X)  += dest
        endfor
        for each source operand of op, src, do
            GEN(X) += src
            KILL(X) -= src
        endfor
    endfor
endfor
```

Example – GEN/KILL Liveness Computation

OUT = Union(IN(succs))

IN = GEN + (OUT – KILL)



BB1

1. r1 = MEM[r2+0]
2. r2 = MEM[r1 + 1]
3. r8 = r1 * r2

BB2

4. r1 = r1 + 5
5. r3 = r5 – r1
6. r7 = r3 * 2

BB3

7. r2 = 0
8. r7 = r1 + r2
9: r3 = 4

BB4

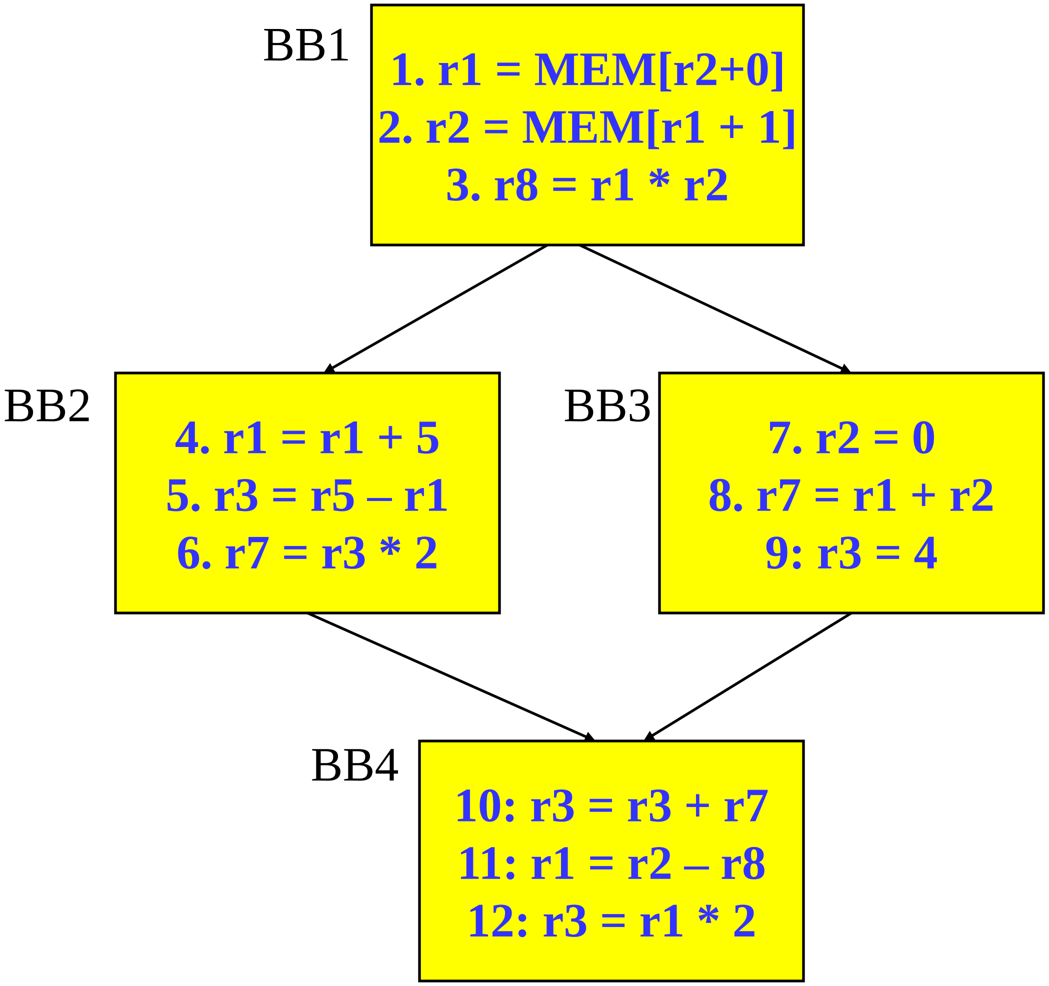
10: r3 = r3 + r7
11: r1 = r2 – r8
12: r3 = r1 * 2

Compute IN/OUT Sets for all BBs

```
initialize IN(X) to 0 for all basic blocks X
change = 1
while (change) do
    change = 0
    for each basic block in procedure, X, do
        old_IN = IN(X)
        OUT(X) = Union(IN(Y)) for all successors Y of X
        IN(X) = GEN(X) + (OUT(X) – KILL(X))
        if (old_IN != IN(X)) then
            change = 1
        endif
    endfor
endfor
```

Example – Liveness Computation

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



BB1

```
1. r1 = MEM[r2+0]
2. r2 = MEM[r1 + 1]
3. r8 = r1 * r2
```

BB2

```
4. r1 = r1 + 5
5. r3 = r5 – r1
6. r7 = r3 * 2
```

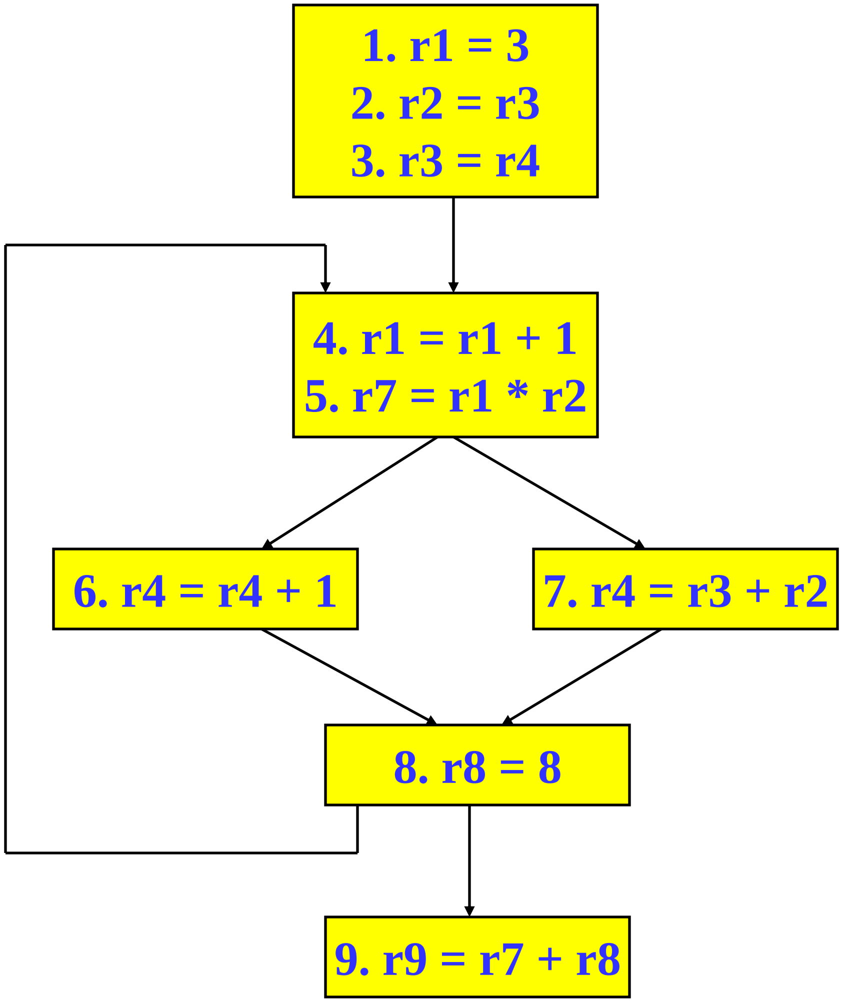
BB3

```
7. r2 = 0
8. r7 = r1 + r2
9: r3 = 4
```

BB4

```
10: r3 = r3 + r7
11: r1 = r2 – r8
12: r3 = r1 * 2
```

Class Problem



Compute liveness

- Calculate GEN/KILL for each BB
- Calculate IN/OUT for each BB

Reaching Definition Analysis (rdefs)

- A definition of a variable x is an operation that assigns, or may assign, a value to x
- A definition d reaches a point p if there is a path from the point immediately following d to p such that d is not “killed” along that path
- 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
- Liveness vs Reaching defs
  - Liveness → variables (e.g., virtual registers), don’t care about specific users
  - Reaching defs → operations, each def is different
  - Forward dataflow analysis as propagation occurs from defs downwards (liveness was backward analysis)

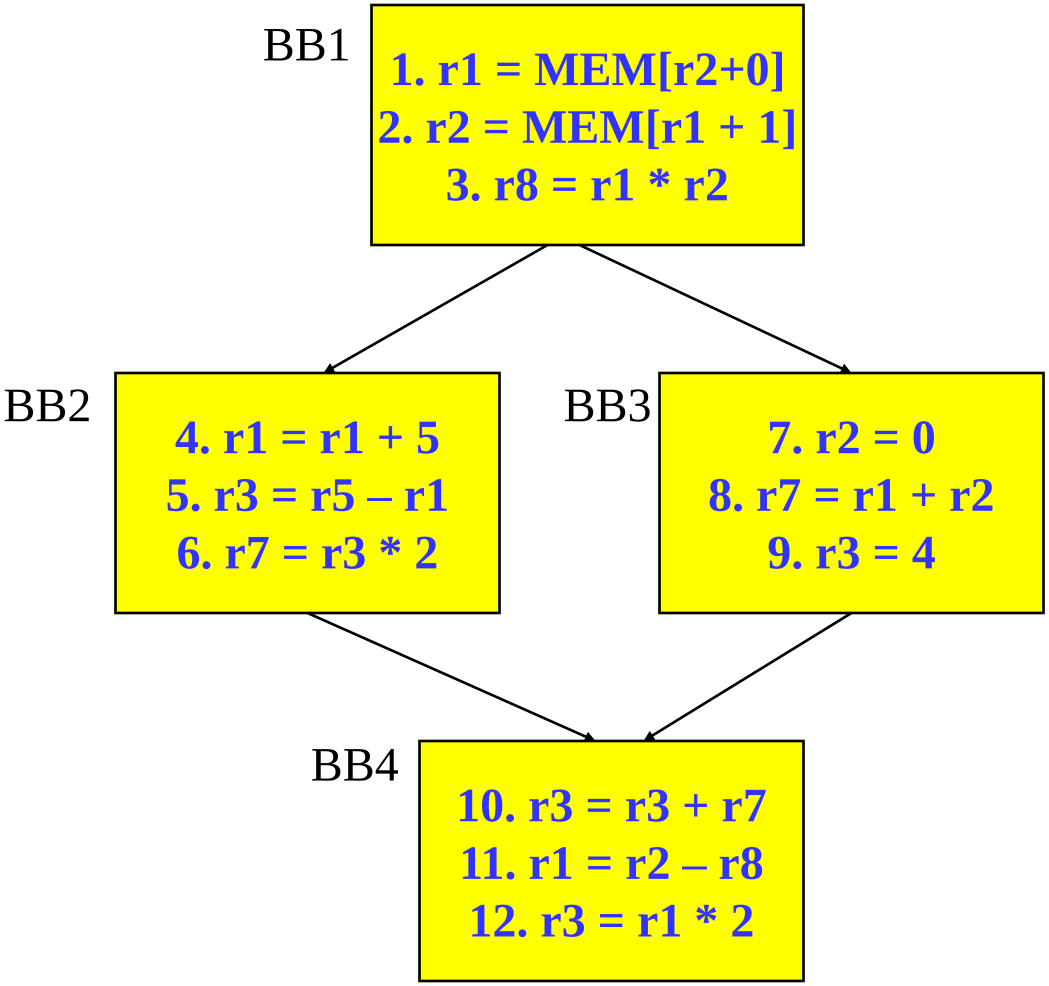
Compute Rdef GEN/KILL Sets for each BB

GEN = set of definitions created by an operation
KILL = set of definitions destroyed by an operation
- Assume each operation only has 1 destination for simplicity
  so just keep track of "ops"..

```
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
endfor
```

Example GEN/KILL Rdef Calculation

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



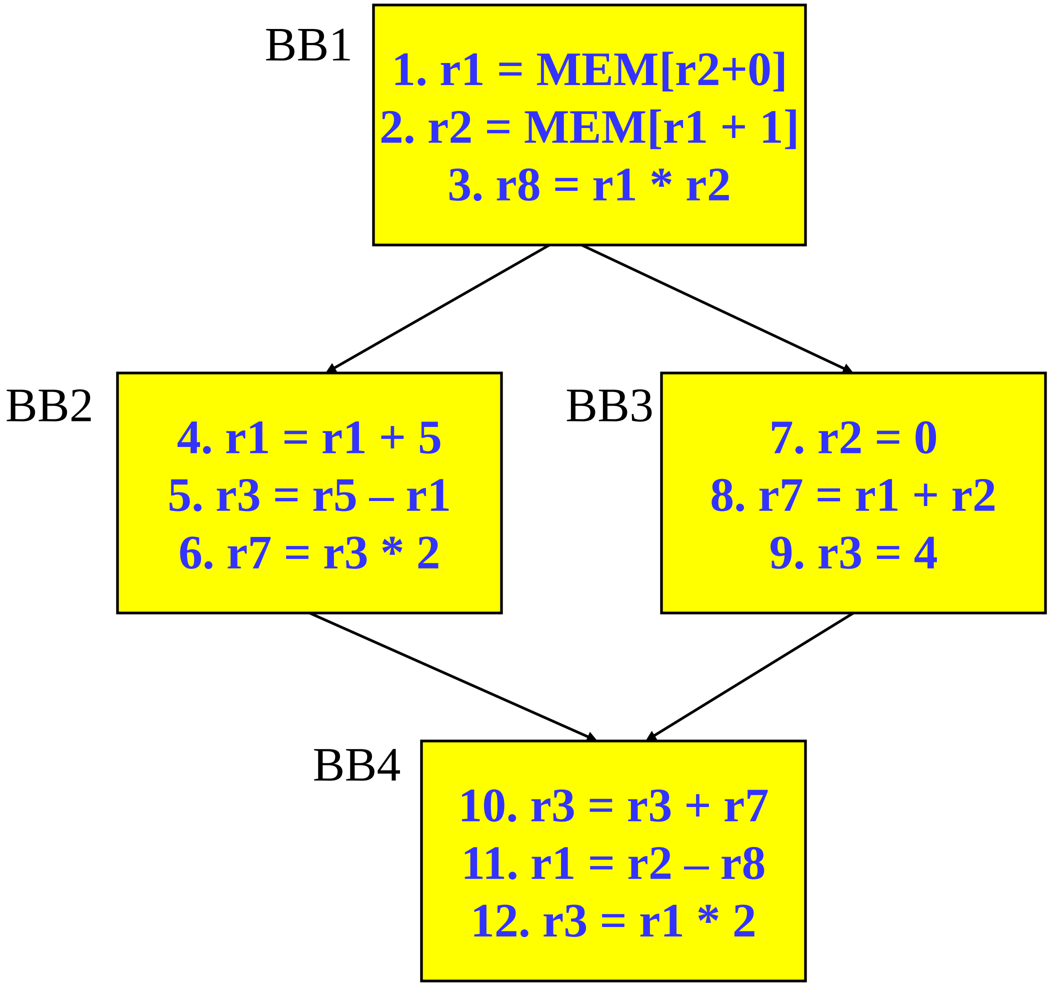
Compute Rdef IN/OUT Sets for all BBs

IN = set of definitions reaching the entry of BB
OUT = set of definitions leaving BB

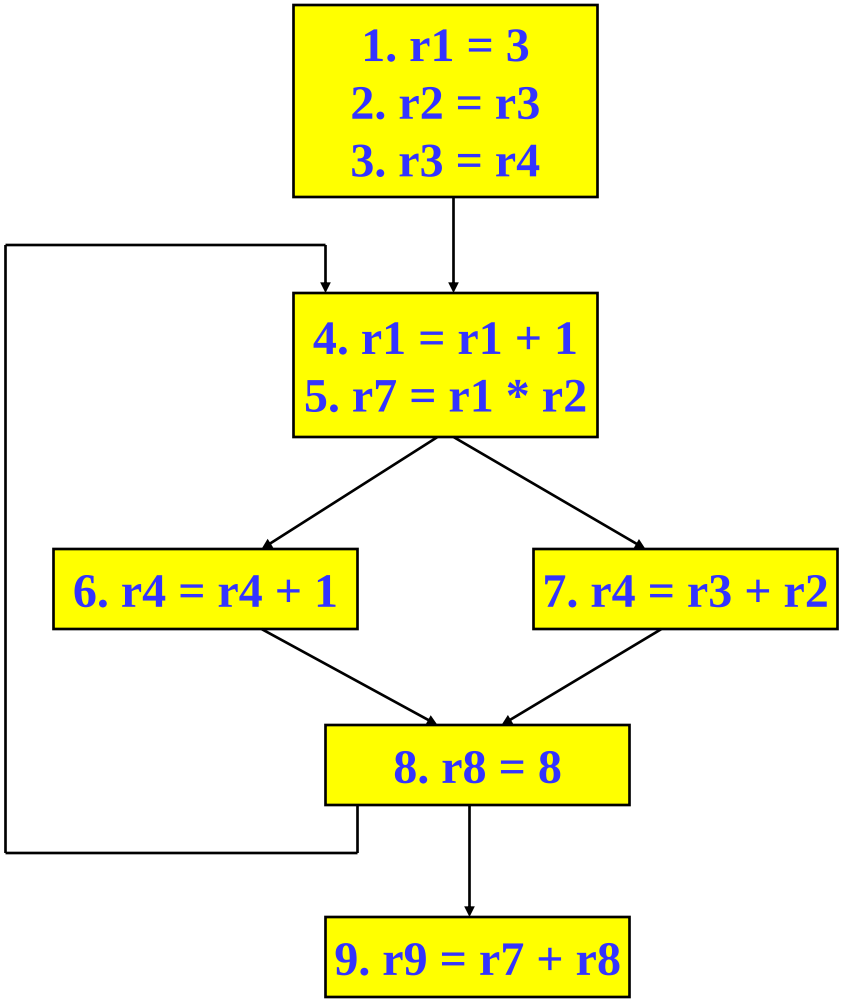
```
initialize IN(X) = 0 for all basic blocks X
initialize OUT(X) = GEN(X) for all basic blocks X
change = 1
while (change) do
    change = 0
    for each basic block in procedure, X, do
        old_OUT = OUT(X)
        IN(X) = Union(OUT(Y)) for all predecessors Y of X
        OUT(X) = GEN(X) + (IN(X)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

Example Rdef Calculation

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



Class Problem



Compute reaching defs

- Calculate GEN/KILL for each BB
- Calculate IN/OUT for each BB