

# EECS 583 – Class 6

## Dataflow Analysis

---

*University of Michigan*

*September 23, 2019*

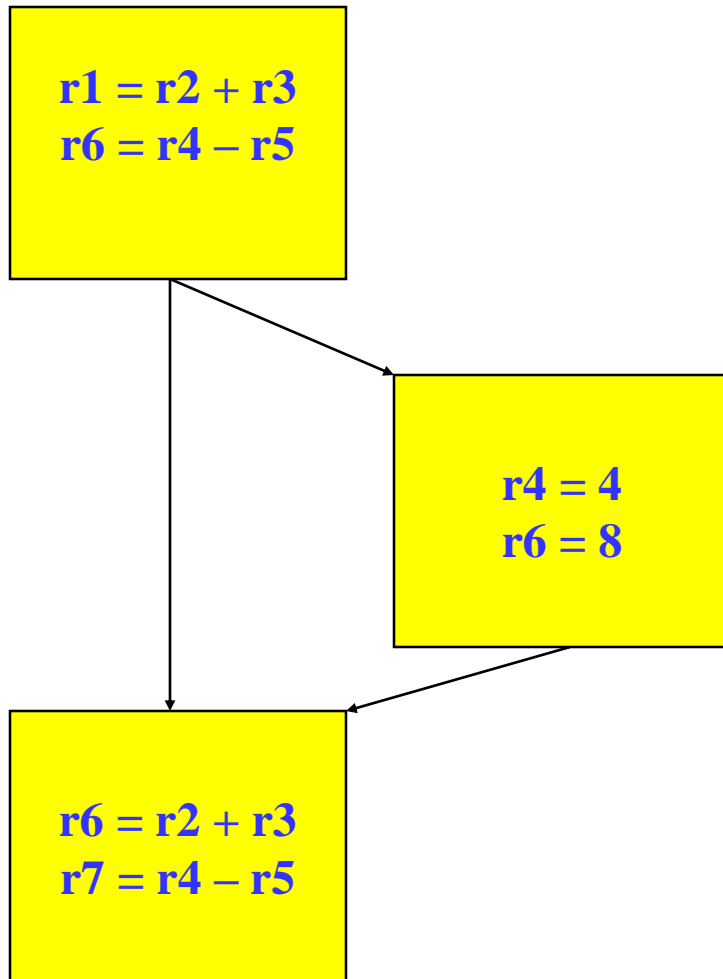
# Reading Material + Announcements

---

- ❖ Reminder – HW 1 due date extended to tonight midnight
  - » Submit `uniquename_hw1.tgz` file to:
    - `eeecs583a.eecs.umich.edu:/hw1_submissions`
- ❖ HW2 out on Wednes
  - » Sung will give overview today for last part of class
- ❖ 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 Wednesday
  - » *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)

# 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

---

**$r1 = r2 + r3$**   
 **$r6 = r4 - r5$**

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

**$r4 = 4$**   
 **$r6 = 8$**

**$r6 = r2 + r3$**   
 **$r7 = r4 - r5$**

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 defs)

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

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

---

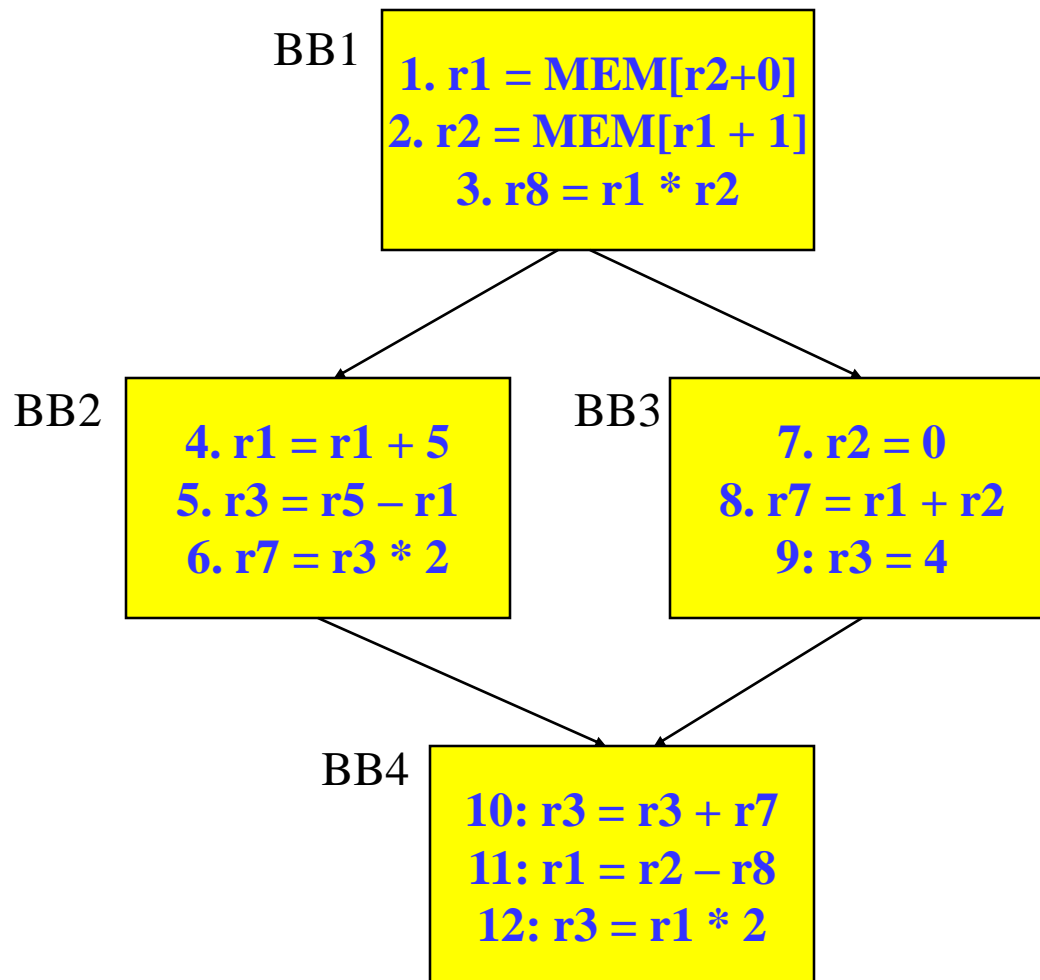
```
for each basic block in the procedure, X, do  
    GEN(X) = 0  
    KILL(X) = 0  
    for each operation in reverse sequential order in X, op, do  
        for each destination operand of op, dest, do  
            GEN(X) -= dest  
            KILL(X) += dest  
        endfor  
        for each source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



## Compute IN/OUT Sets for all BBs

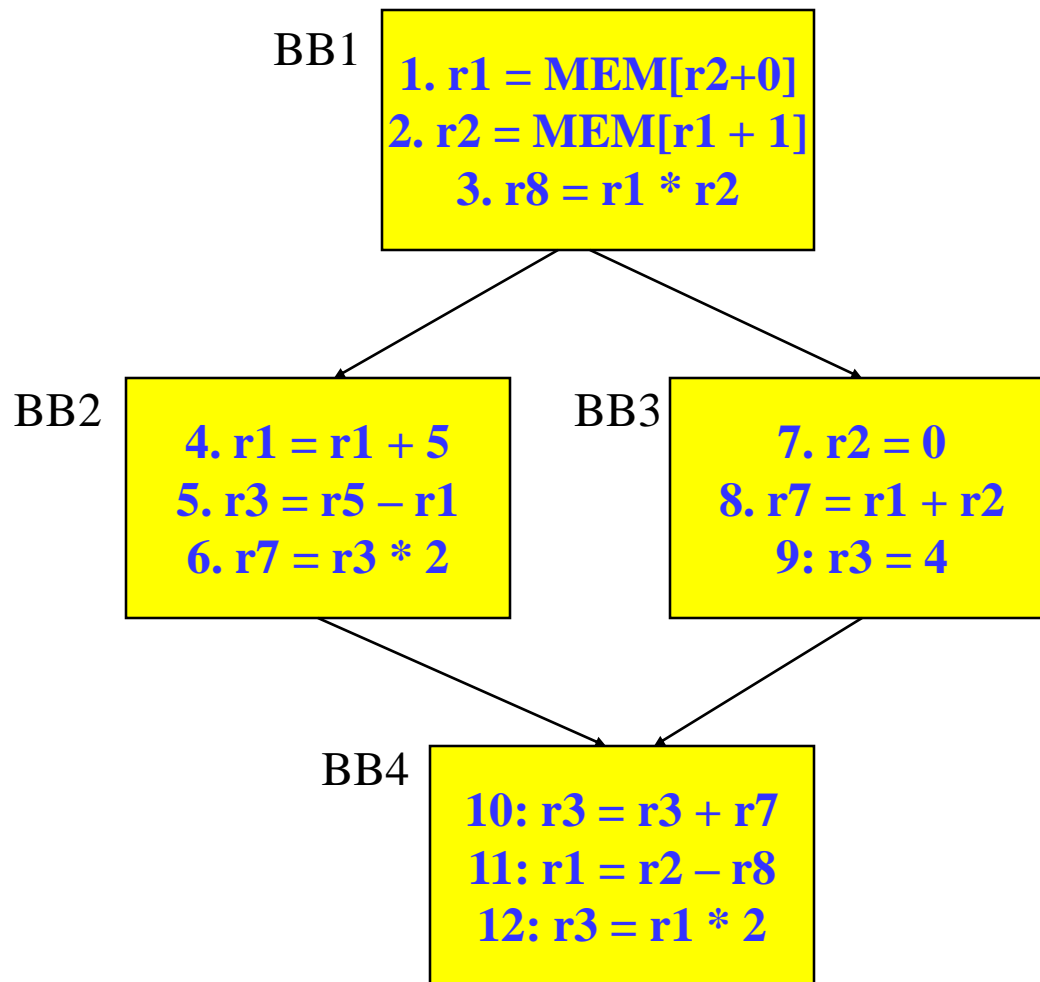
---

```
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)



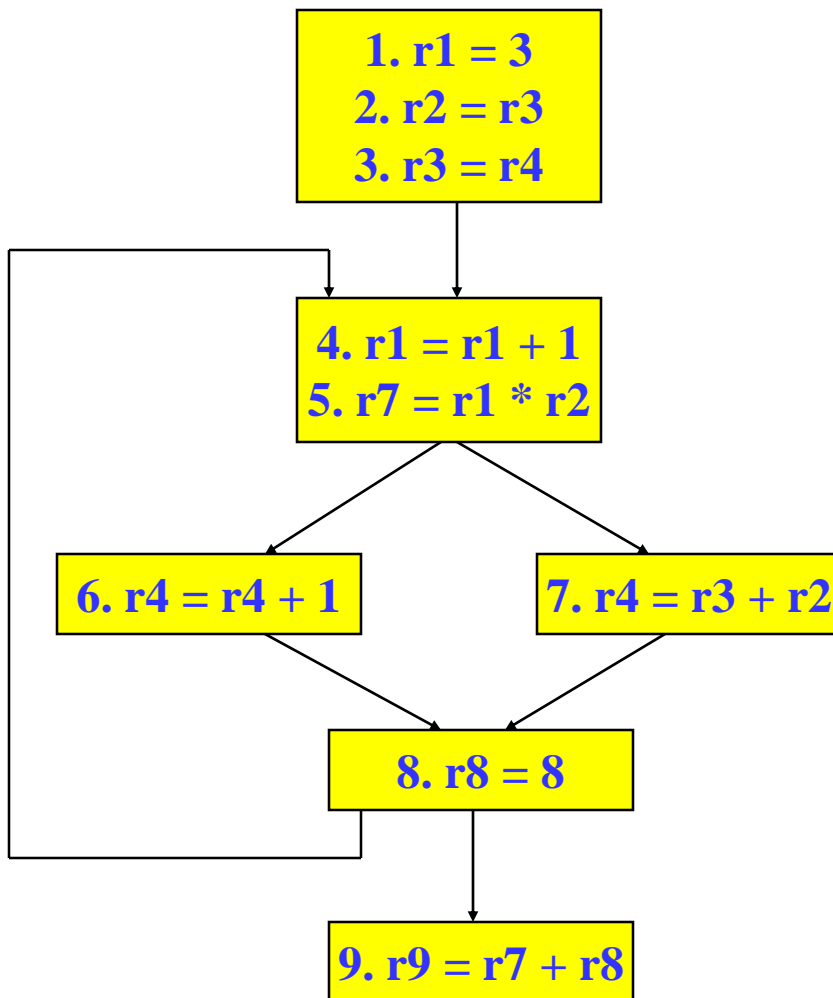
# Homework Problem

---

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  $\rightarrow$  variables (e.g., virtual registers), don't care about specific users
  - » Reaching defs  $\rightarrow$  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

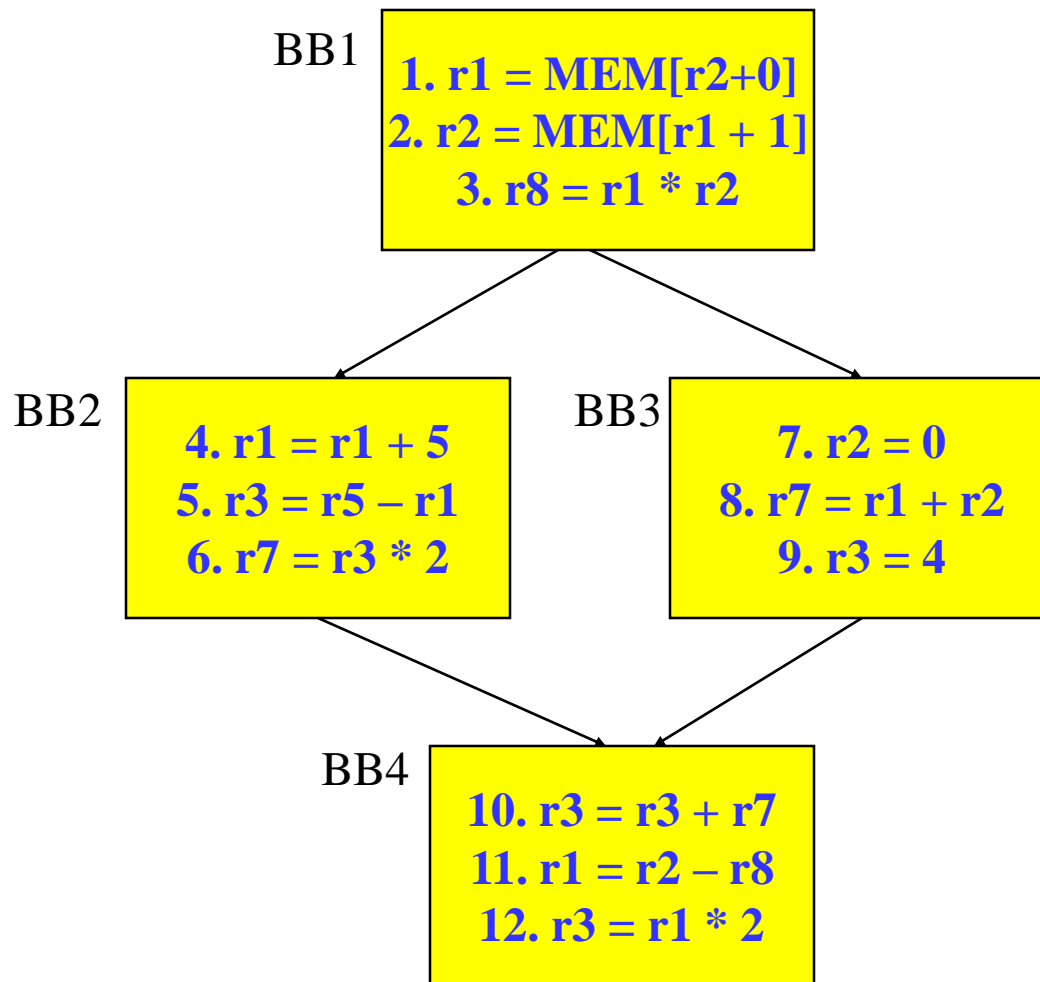
- Assume each operation only has 1 destination for simplicity  
so just keep track of “ops”..

```
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

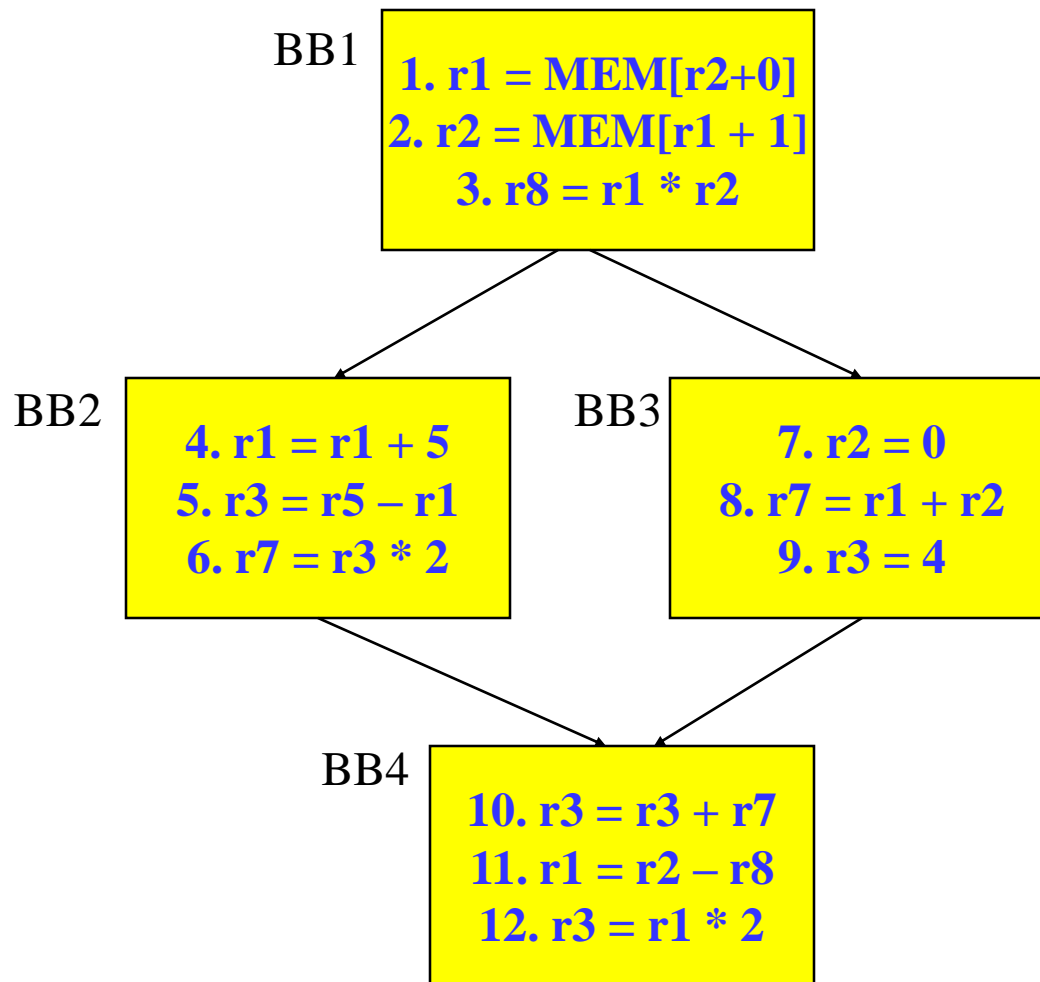
OUT = set of definitions leaving BB

```
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)



# Homework Problem

---

Compute reaching defs

Calculate GEN/KILL for each BB

Calculate IN/OUT for each BB

