EECS 583 – Class 5

Finish Control Flow Analysis, Dataflow Analysis Intro

University of Michigan

September 15, 2021

Reading Material + Announcements

- Reminder – HW 1 due tonight 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 Yunjie/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)

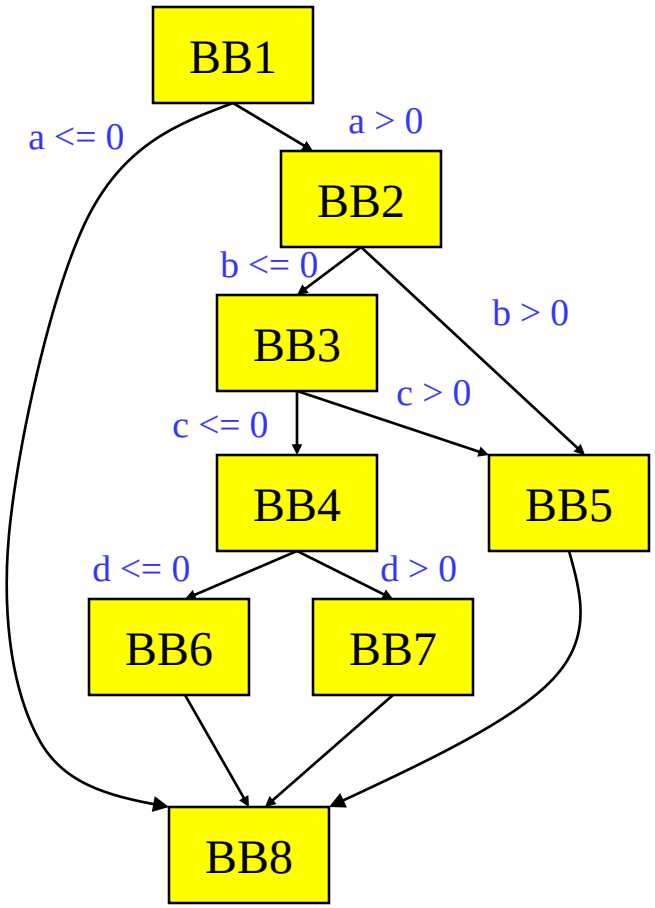
From Last Time: Homework Problem

```
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
}
```

a. Draw the CFG
b. Compute CD
c. If-convert the code

Homework Problem 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
}
```



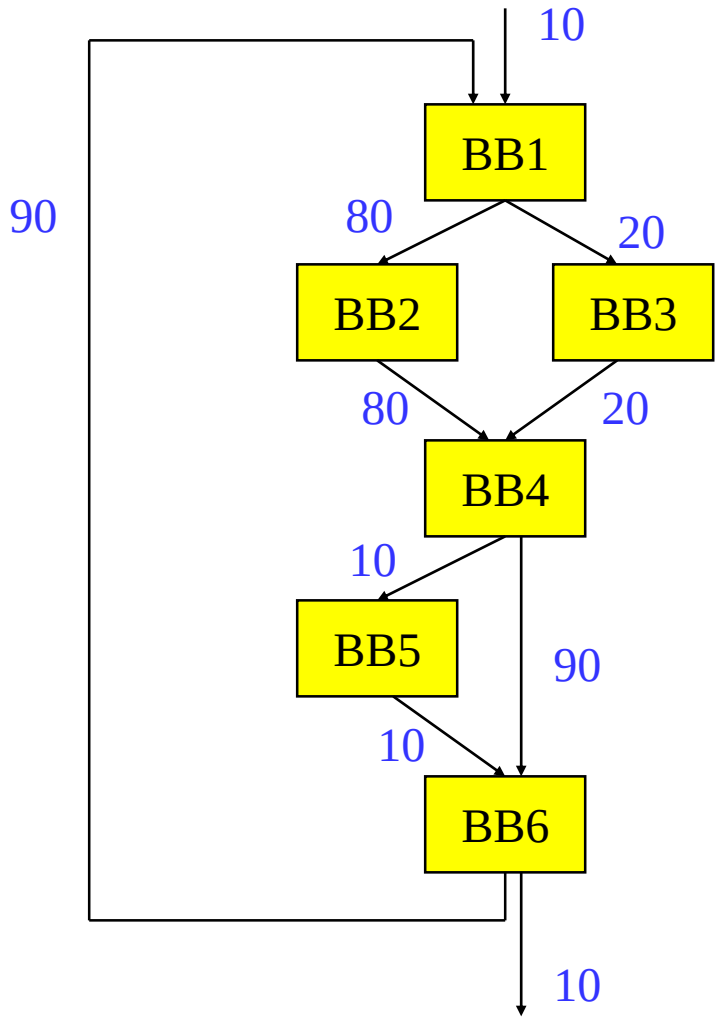
BB	CD
1	-
2	1
3	-2
4	-3
5	2,3
6	-4
7	4
8	-

a. Draw the CFG
b. Compute CD
c. If-convert the code

```
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
```

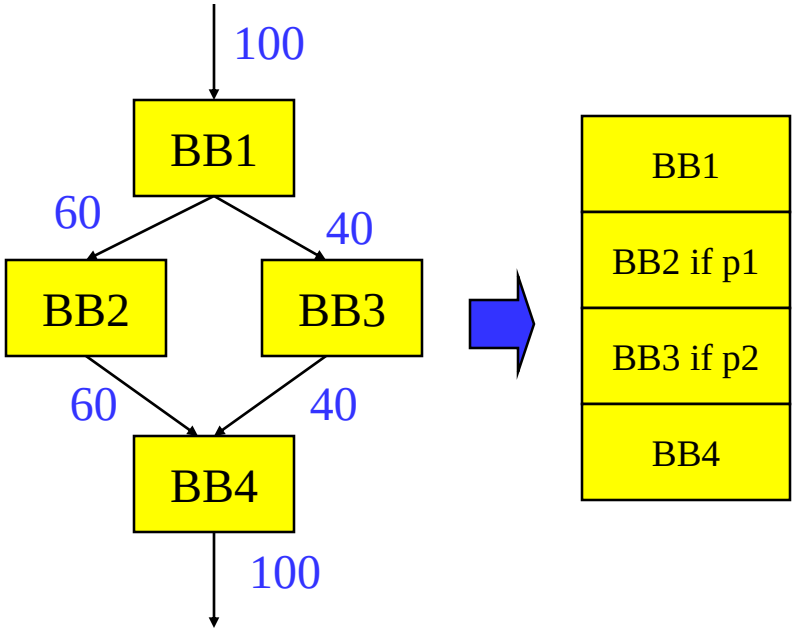
When to Apply If-conversion?

- Positives
  - Remove branch
    - No disruption to sequential fetch
    - No prediction or mispredict
    - No draining of pipeline for mispredict
    - No use of branch resource
  - Increase potential for operation overlap
    - Creates larger basic blocks
    - Convert control dependences into data dependences
  - Enable more aggressive compiler xforms
    - Software pipelining
    - Height reduction
- What about the negatives?



Negative 1: Resource Usage

Instruction execution is additive
for all BBs that are if-converted, thus
require more processor resources

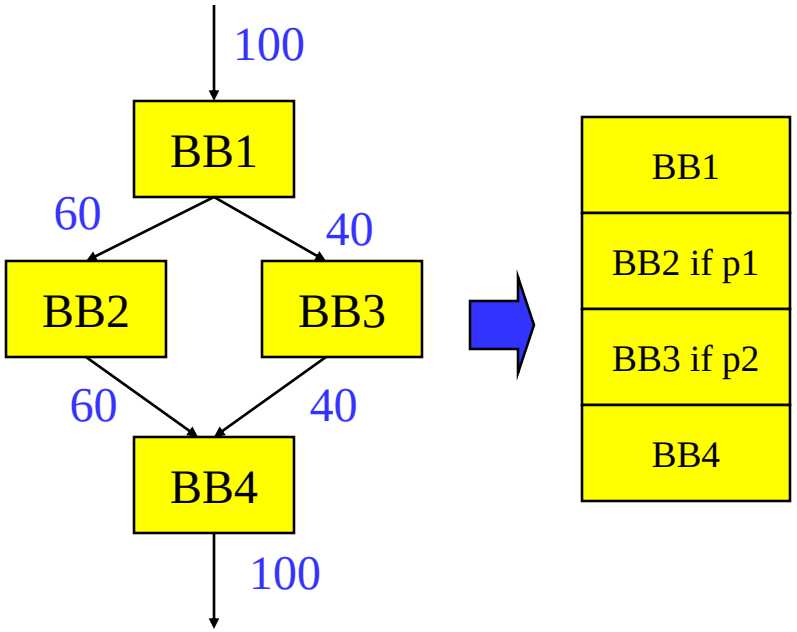


Be careful applying if-conversion too liberally
when processor resources constrained OR
blocks have large numbers of instructions

Negative 2: Dependence Height

Dependence height is max of
for all BBs that are if-converted
(dep height = schedule length
with infinite resources)

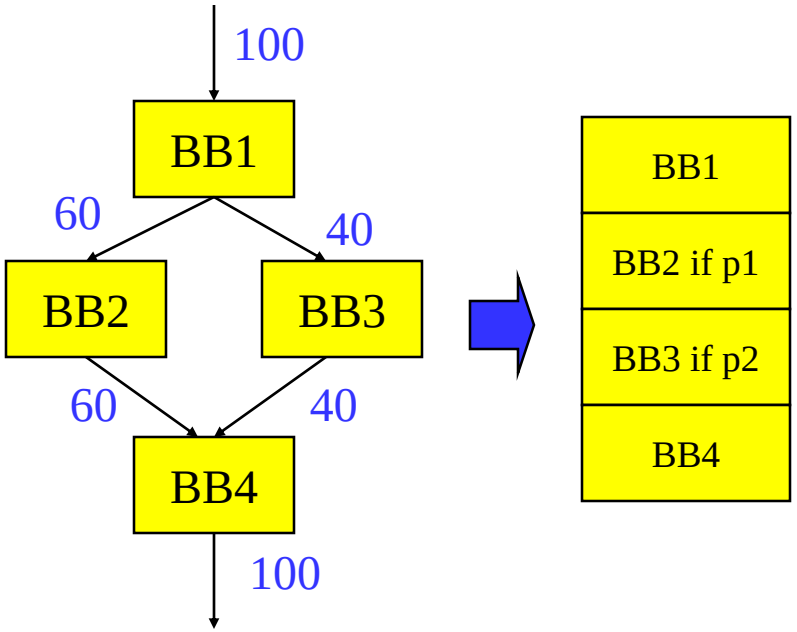
Be careful with if-converting blocks with mismatched dependence heights



Negative 3: Hazard Presence

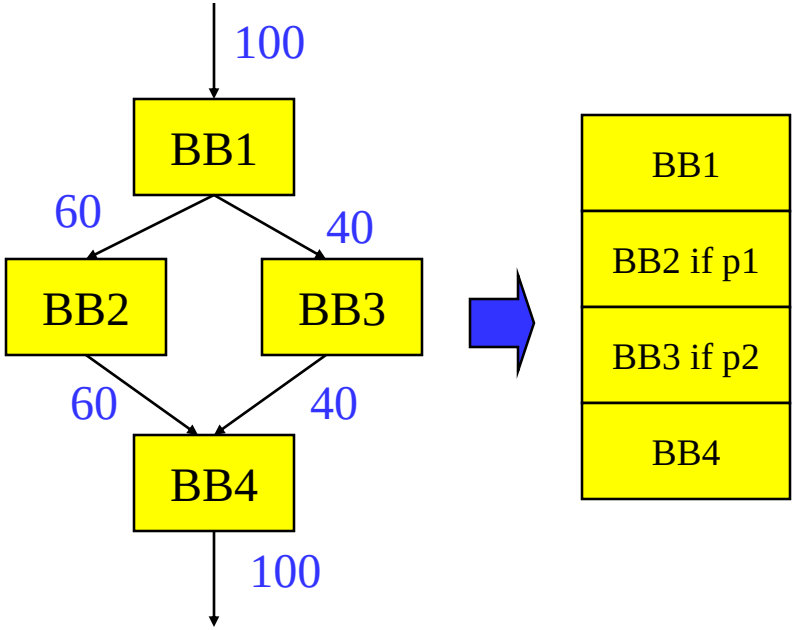
Hazard = operation that forces
the compiler to be conservative,
so limited reordering or optimization,
e.g., subroutine call, pointer store, …

Hazards should be avoided except
on the “main path”



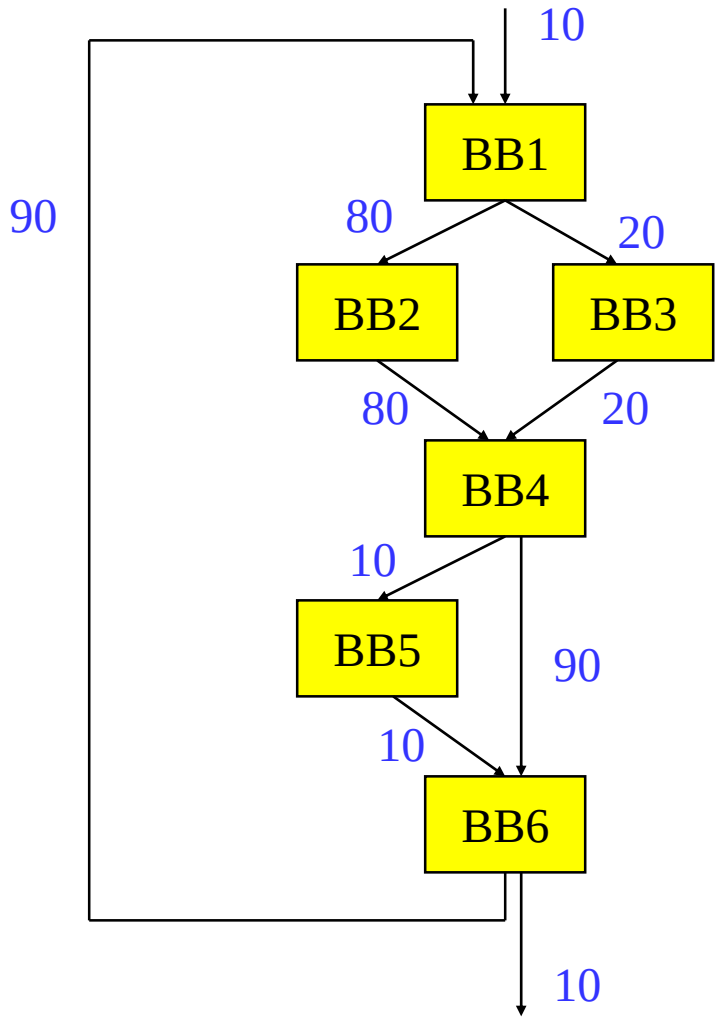
Deciding When/What To If-convert

- Resources
  - Small resource usage ideal for less important paths
- Dependence height
  - Matched heights are ideal
  - Close to same heights is ok
- Remember everything is relative for resources and dependence height !
- Hazards
  - Avoid hazards unless on most important path
- Estimate of benefit
  - Branches/Mispredicts removed
  - Increased instruction overlap

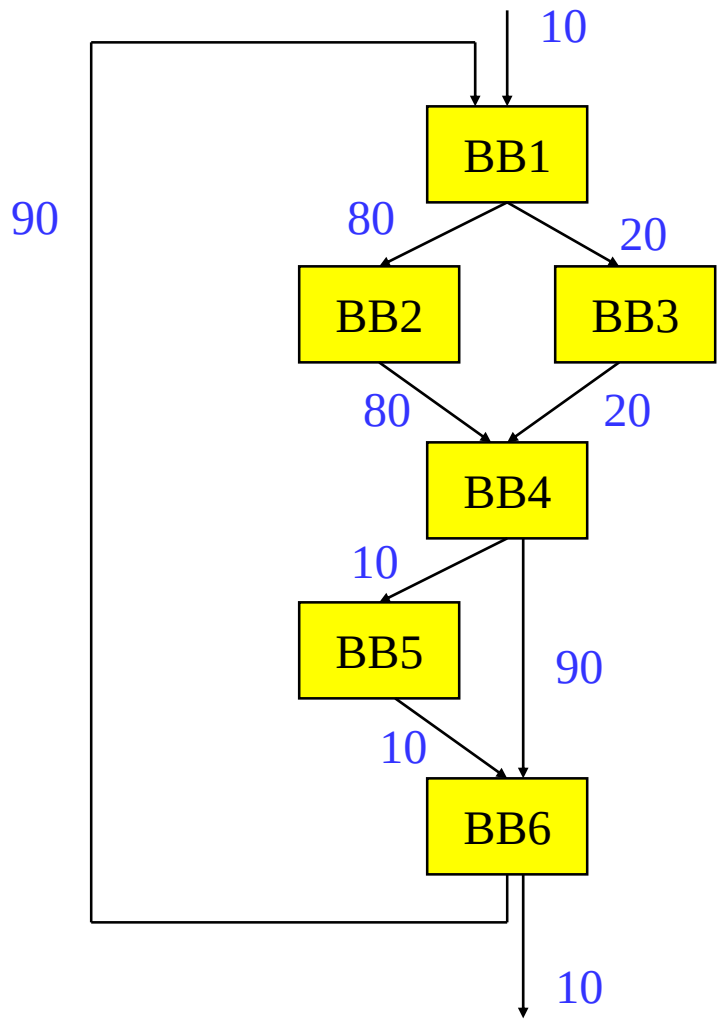


The Hyperblock

- Hyperblock - Collection of basic blocks in which control flow may only enter at the first BB. *All internal control flow is eliminated via if-conversion*
  - "Likely control flow paths"
  - Acyclic (outer backedge ok)
  - Multiple intersecting traces with no side entrances
  - Side exits still exist
- Hyperblock formation
  - 1. Block selection
  - 2. Tail duplication
  - 3. If-conversion

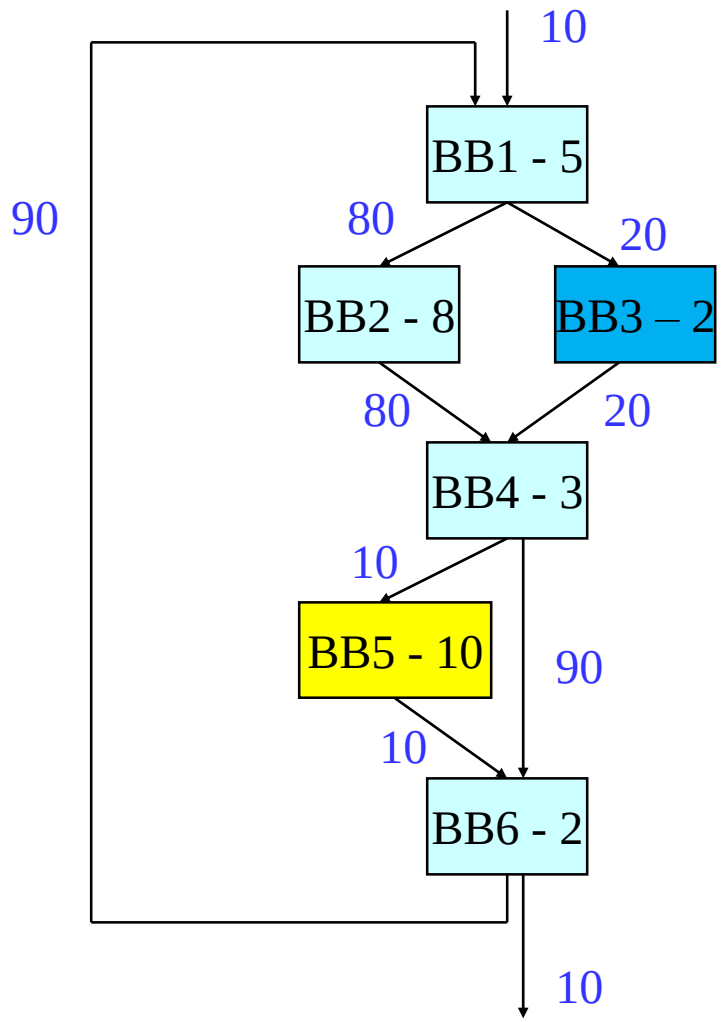


Block Selection



- Block selection
  - » Select subset of BBs for inclusion in HB
  - » Difficult problem
  - » Weighted cost/benefit function
    - Height overhead
    - Resource overhead
    - Hazard overhead
    - Branch elimination benefit
    - Weighted by frequency

Example - Step 1 - Block Selection

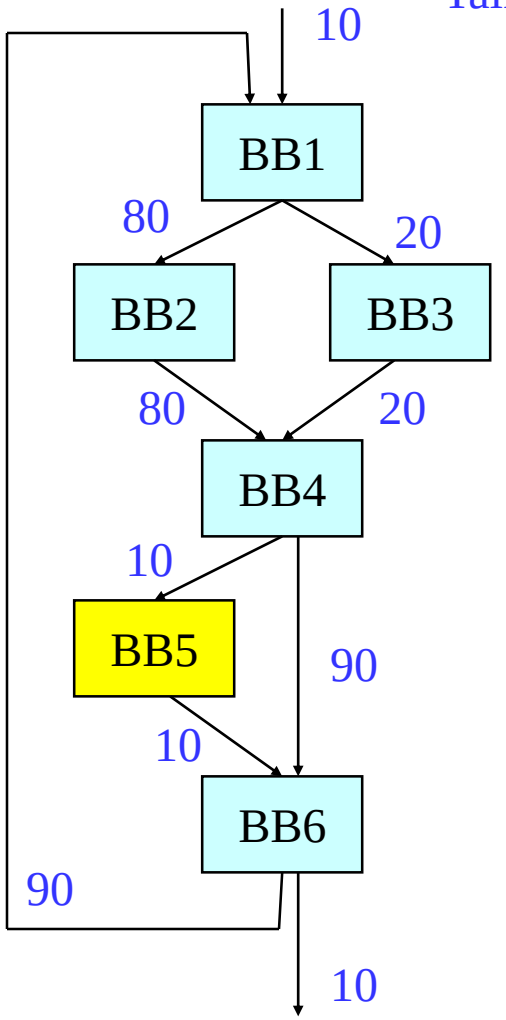


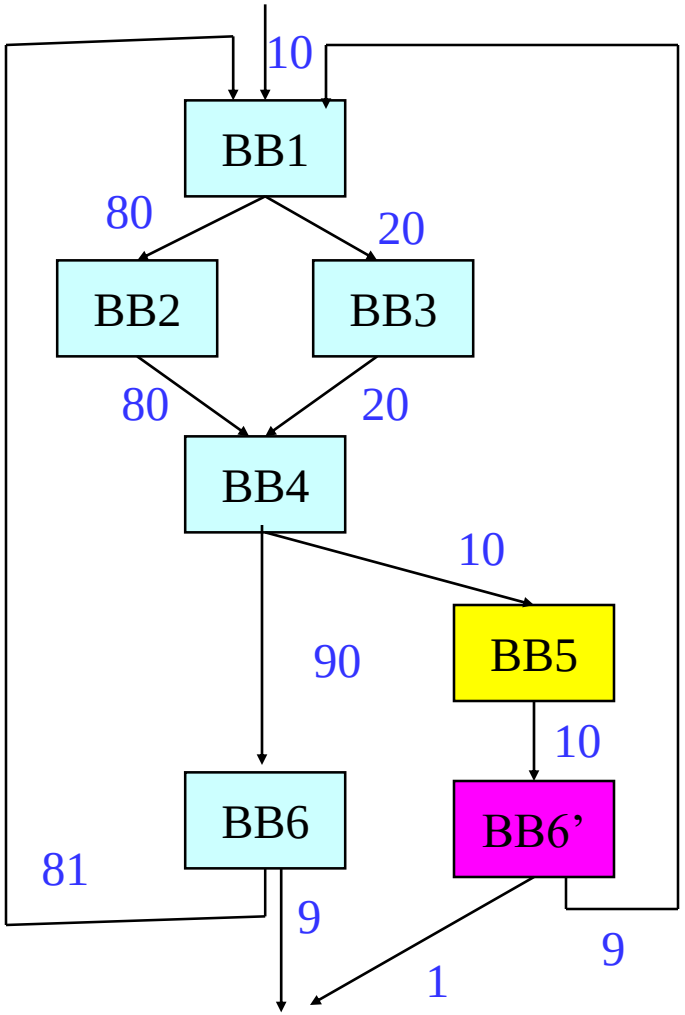
main path = BB1,BB2,BB4,BB6

Consider adding BB3 and BB5

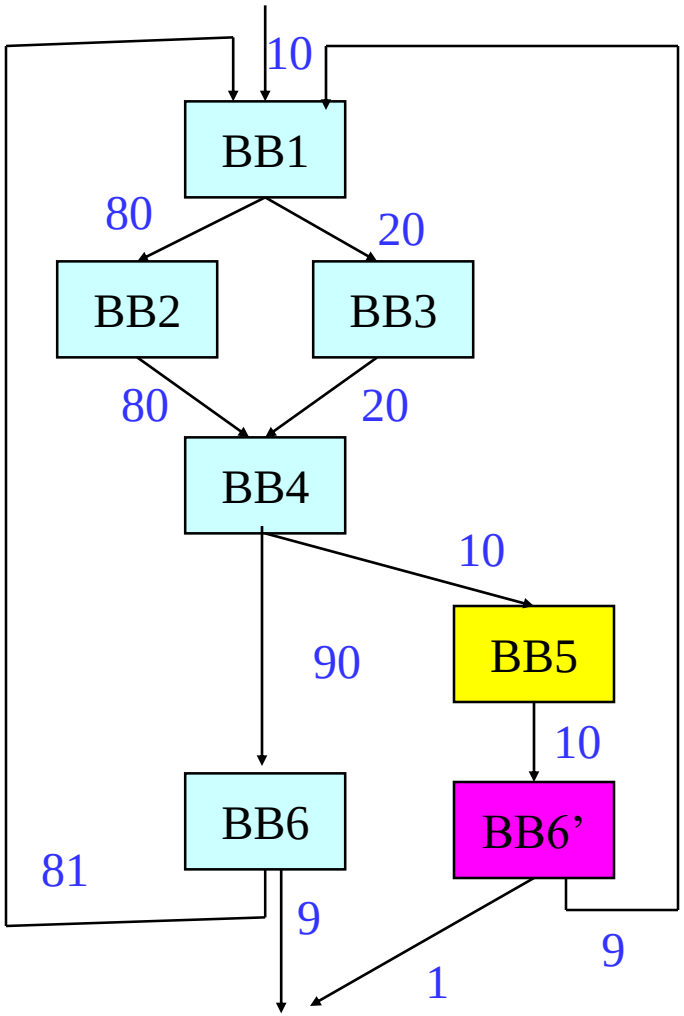
Example - Step 2 - Tail Duplication

Tail duplication same as with Superblock formation

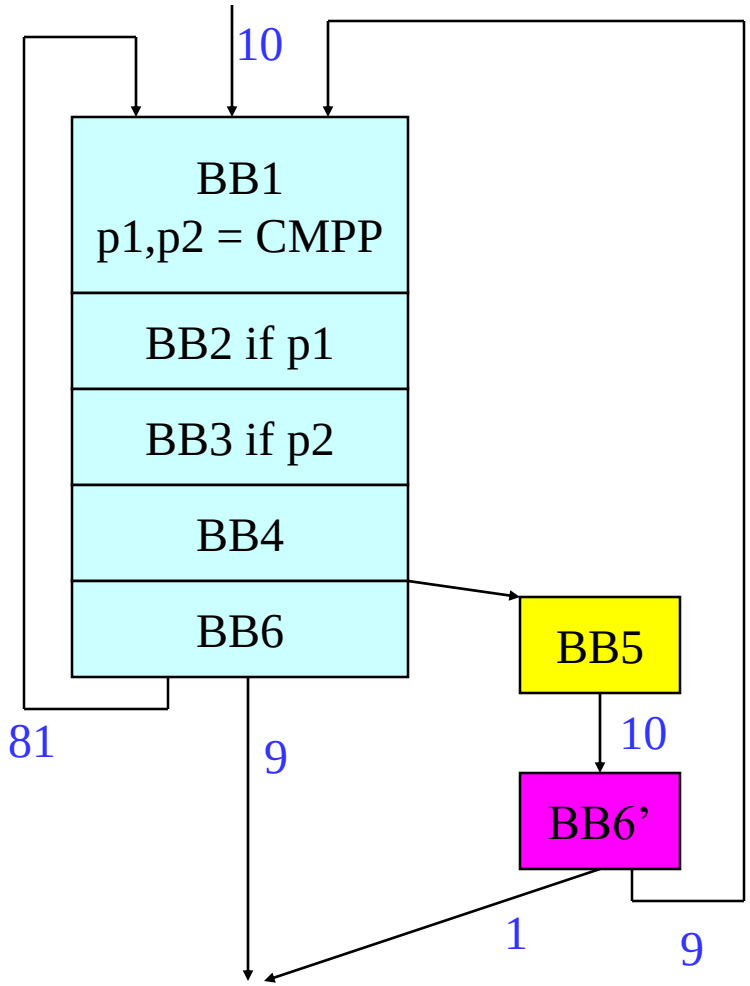




Example - Step 3 – If-conversion



If-convert intra-HB branches only!!

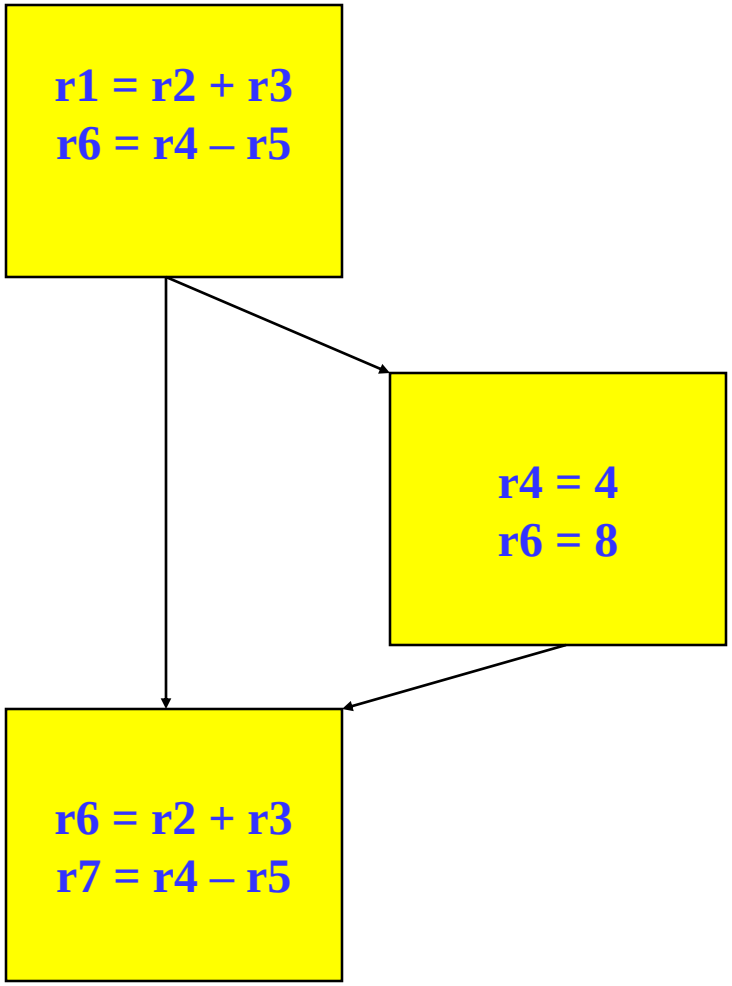


For More on Predicates/Hyperblocks

- See
  - "Effective Compiler Support for Predicated Execution using the Hyperblock", S. Mahlke et al., MICRO-25, 1992.
  - "Control CPR: A Branch Height Reduction Optimization for EPIC Processors", M. Schlansker et al., PLDI-99, 1999.

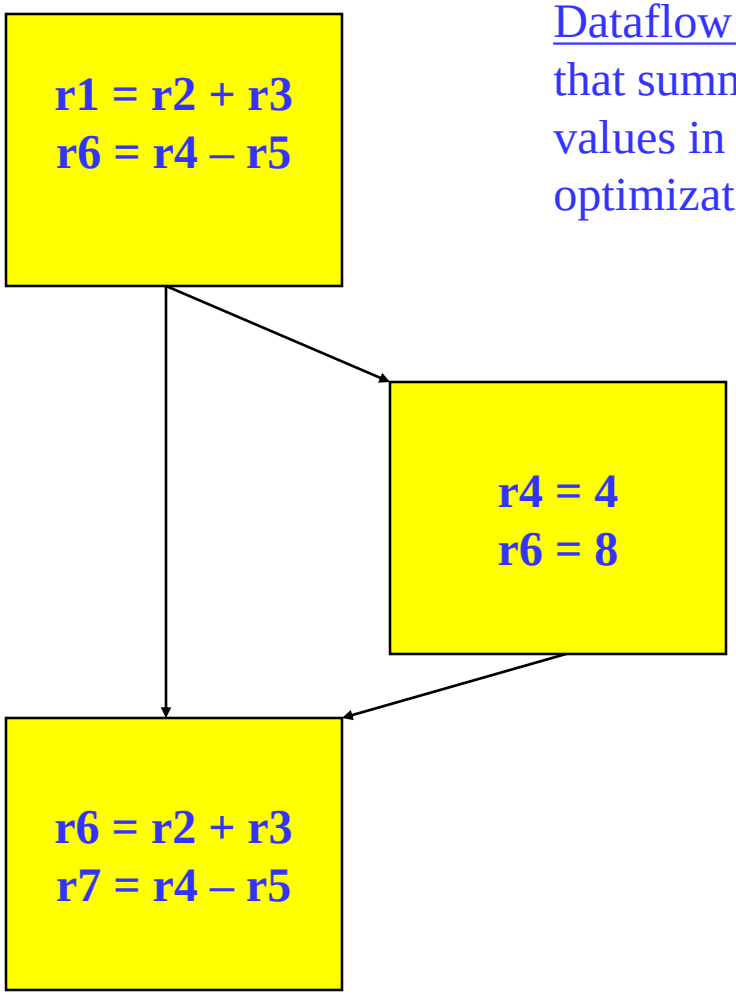
New Topic
Dataflow Analysis!

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



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)

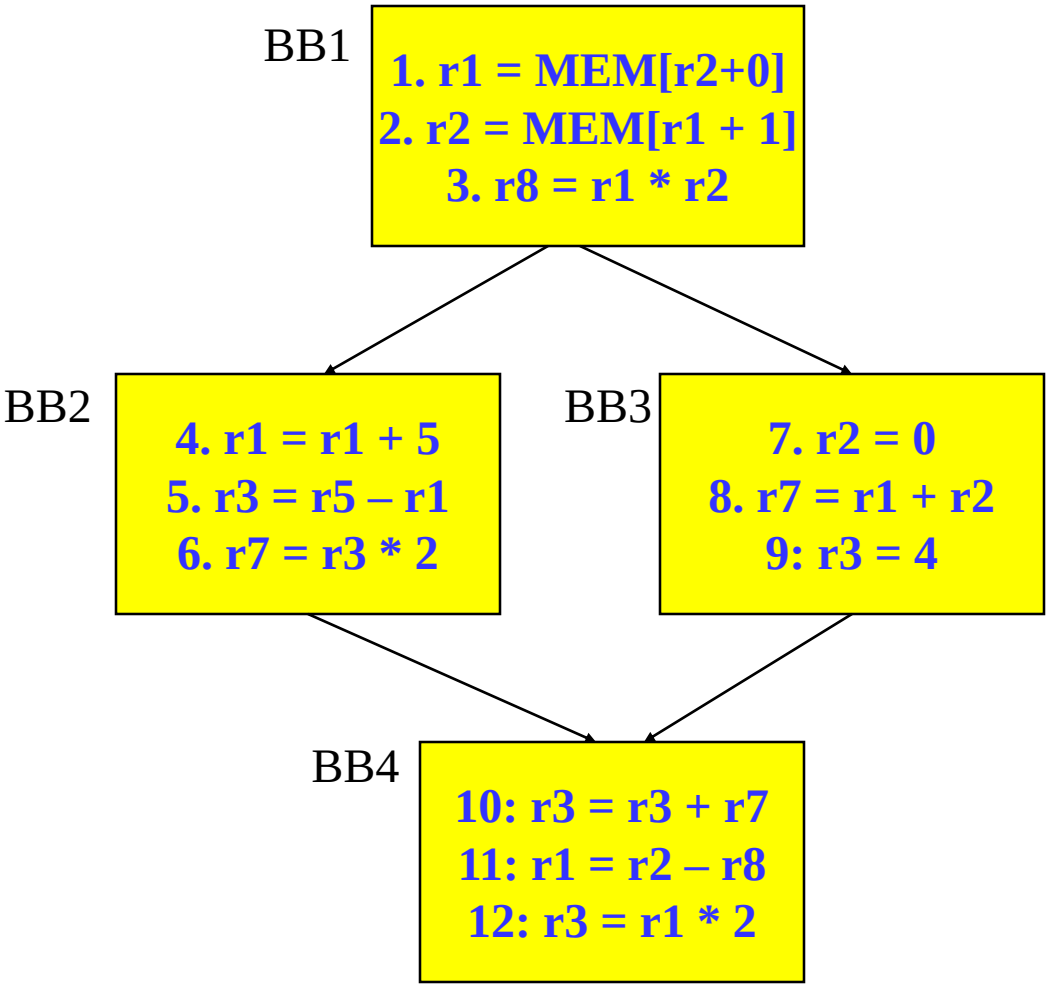
Live Variable (Liveness) Analysis

- 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

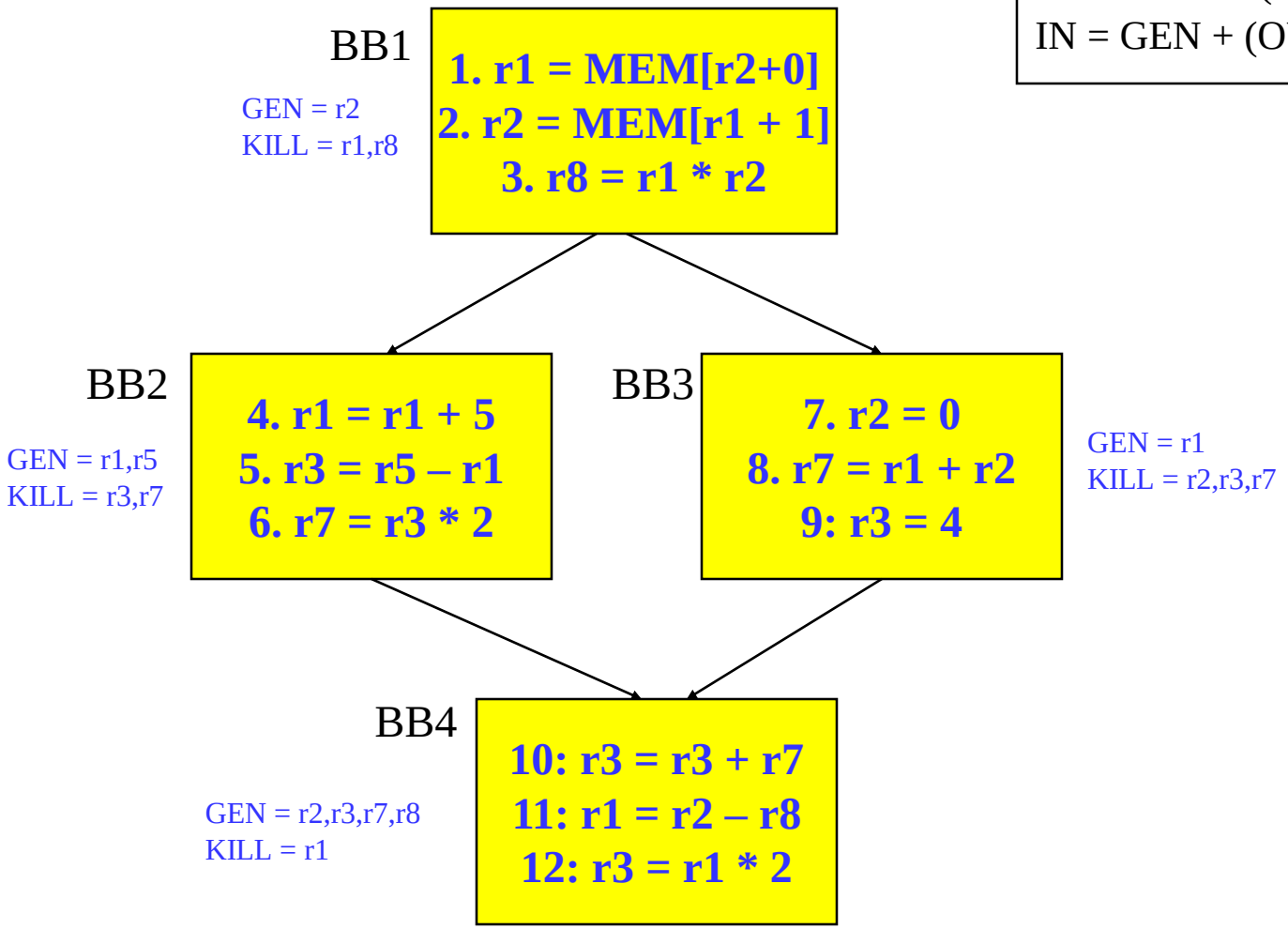


Compute IN/OUT Sets for all BBs

```
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)



Class Problem

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