EECS 583 – Class 5
Hyperblocks, Control Height Reduction

University of Michigan

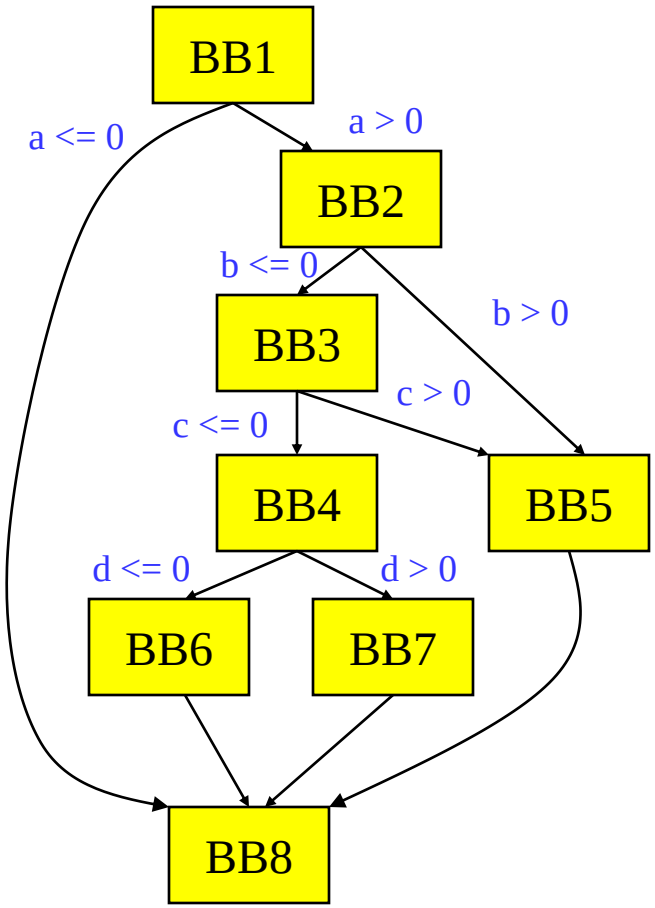
September 21, 2011

Reading + Announcements Material

- Reminder – HW 1 due Friday at midnight
  - Submit uniquename_hw1.tgz file to andrew.eecs.umich.edu:/y/submit/
  - Talk to Daya in office hours Thurs or Fri if having trouble
- My office hours today – cancelled due to industry visitors
- Today's class
  - "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.
- Material for next Monday
  - *Compilers: Principles, Techniques, and Tools*, A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988. (Sections: 10.5, 10.6, 10.9, 10.10)

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
}
```



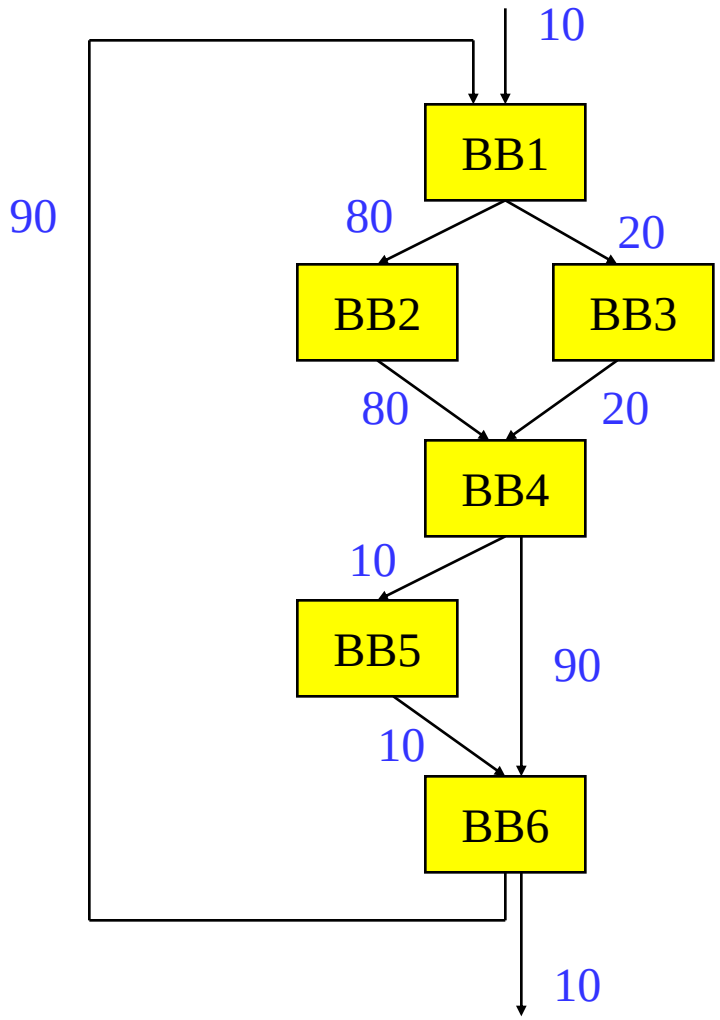
BB	CD
1	-
2	1
3	-2
4	-3
5	2,3
6	-4
7	4
8	-

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
```

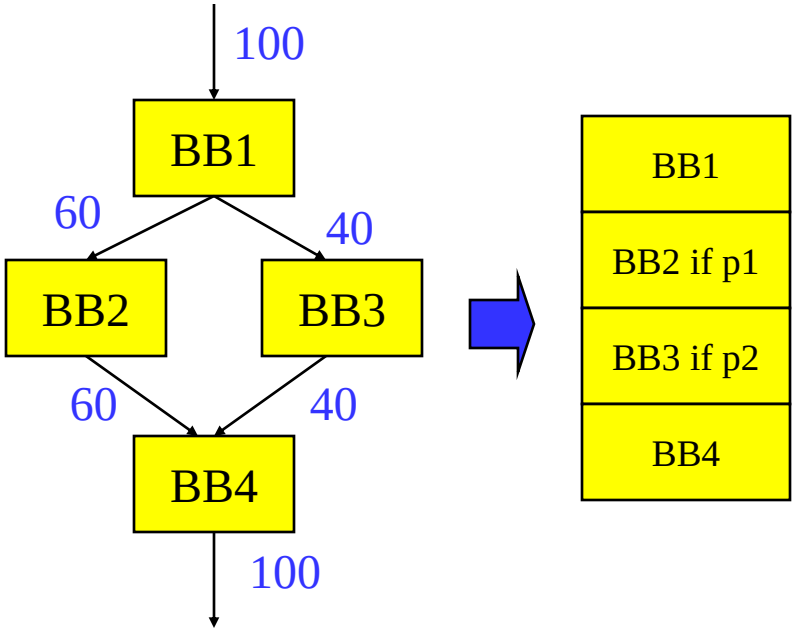
When to Apply If-conversion?

- Positives
  - Remove branch
    - No disruption to sequential fetch
    - No prediction or mispredict
    - No use of branch resource
  - Increase potential for operation overlap
  - Enable more aggressive compiler xforms
    - Software pipelining
    - Height reduction
- Negatives
  - Max or Sum function applied when overlap
    - Resource usage
    - Dependence height
    - Hazard presence
  - Executing useless operations



Negative 1: Resource Usage

Resource usage is additive
for all BBs that are if-converted



Case 1: Each BB requires 3 resources
Assume processor has 2 resources

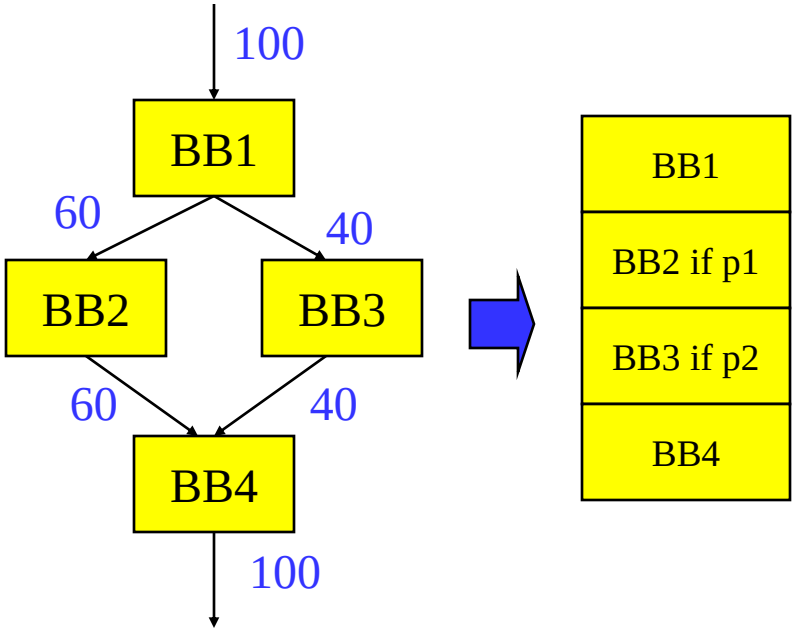
No IC: 1*3 + .6*3 + .4*3 + 1*3 = 9
9 / 2 = 4.5 = 5 cycles
IC: 1(3 + 3 + 3+ 3) = 12
12 / 2 = 6 cycles

Case 2: Each BB requires 3 resources
Assume processor has 6 resources

No IC: 1*3 + .6*3 + .4*3 + 1*3 = 9
9 / 6 = 1.5 = 2 cycles
IC: 1(3+3+3+3) = 12
12 / 6 = 2 cycles

Negative 2: Dependence Height

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



Case 1: height(bb1) = 1, height(bb2) = 3
Height(bb3) = **9**, height(bb4) = 2

No IC: 1*1 + .6*3 + .4*9 + 1*2 = 8.4

IC: 1*1 + 1*MAX(3,9) + 1*3 = 13

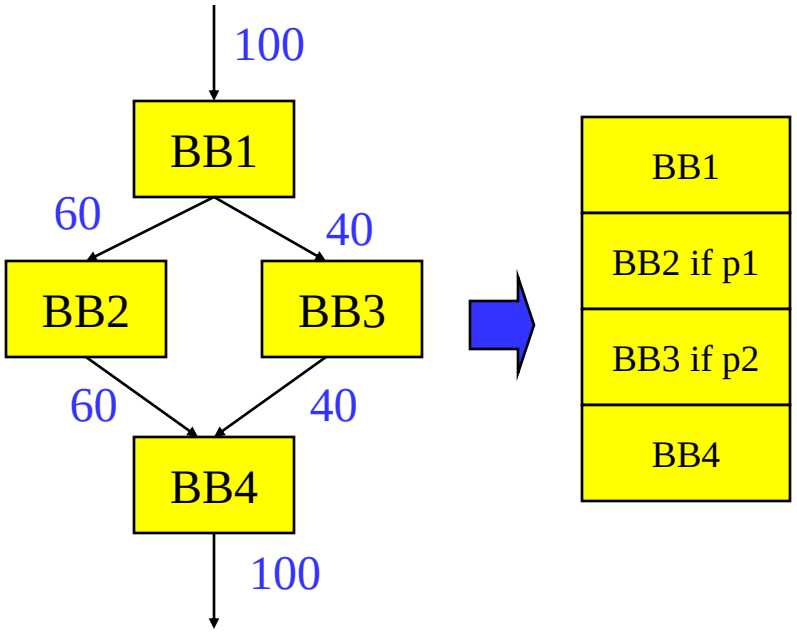
Case 2: height(bb1) = 1, height(bb2) = 3
Height(bb3) = **3**, height(bb4) = 2

No IC: 1*1 + .6*3 + .4*3 + 1*2 = 6

IC: 1*1 + 1*MAX(3,3) + 1*2 = 6

Negative 3: Hazard Presence

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



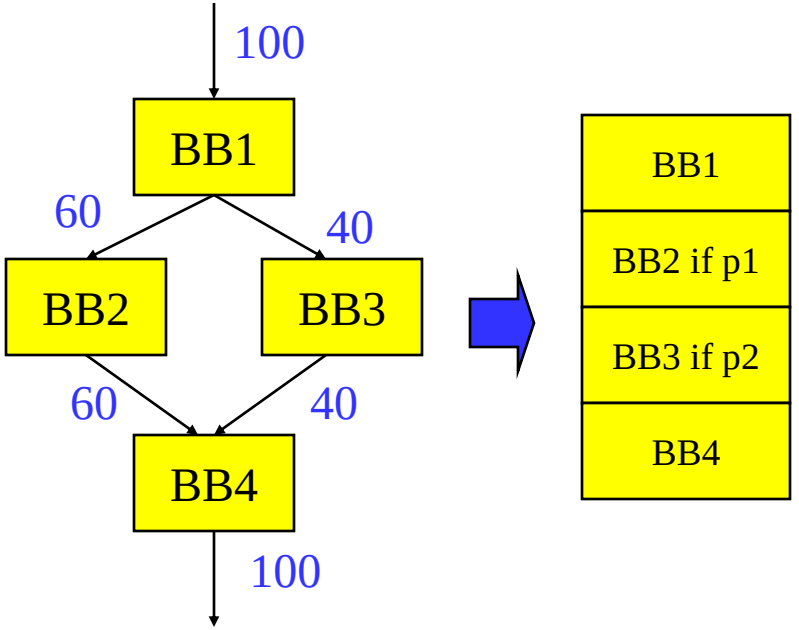
Case 1: Hazard in BB3

No IC : SB out of BB1, 2, 4, operations
In BB4 free to overlap with those in
BB1 and BB2

IC: operations in BB4 cannot overlap
With those in BB1 (BB2 ok)

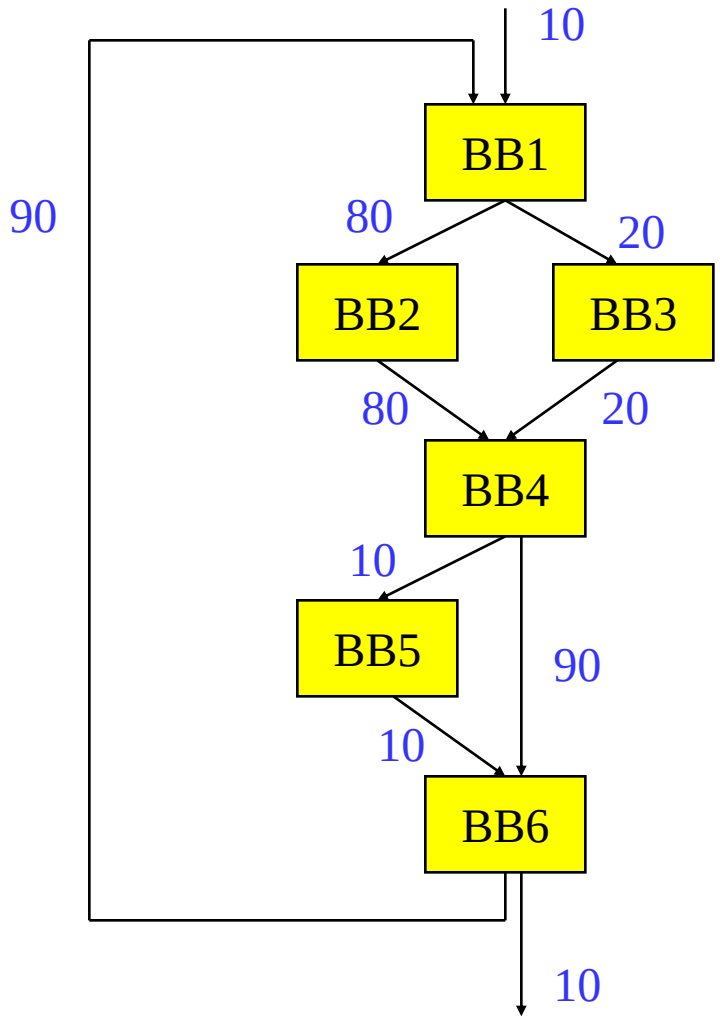
When To If-convert?

- 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
  - » Fudge factor

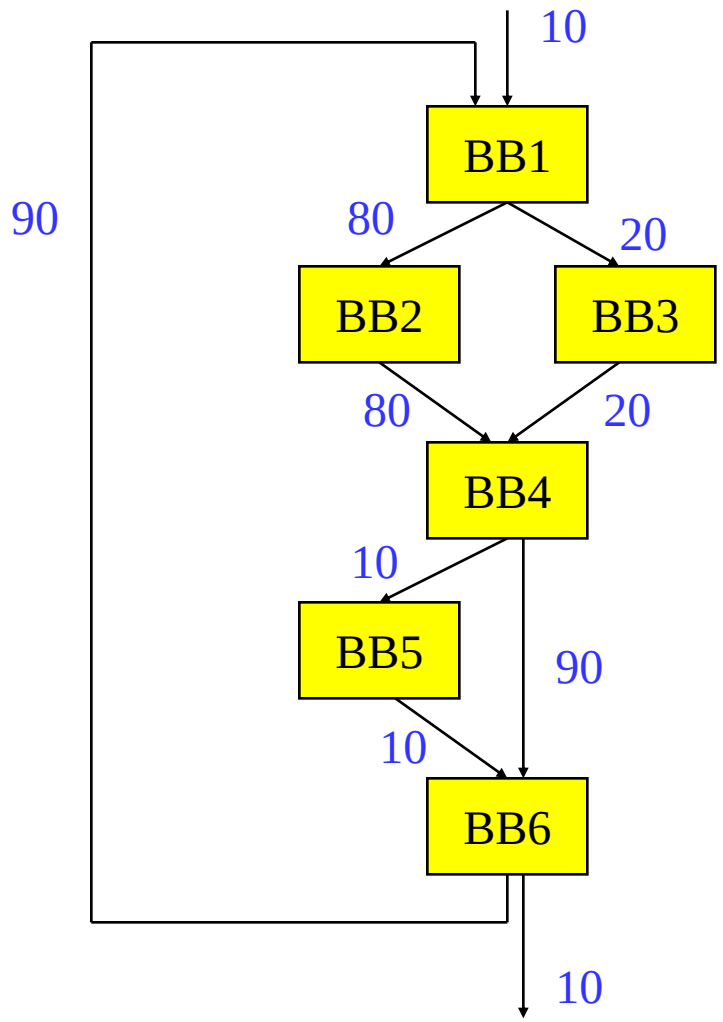


The Hyperblock

- 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

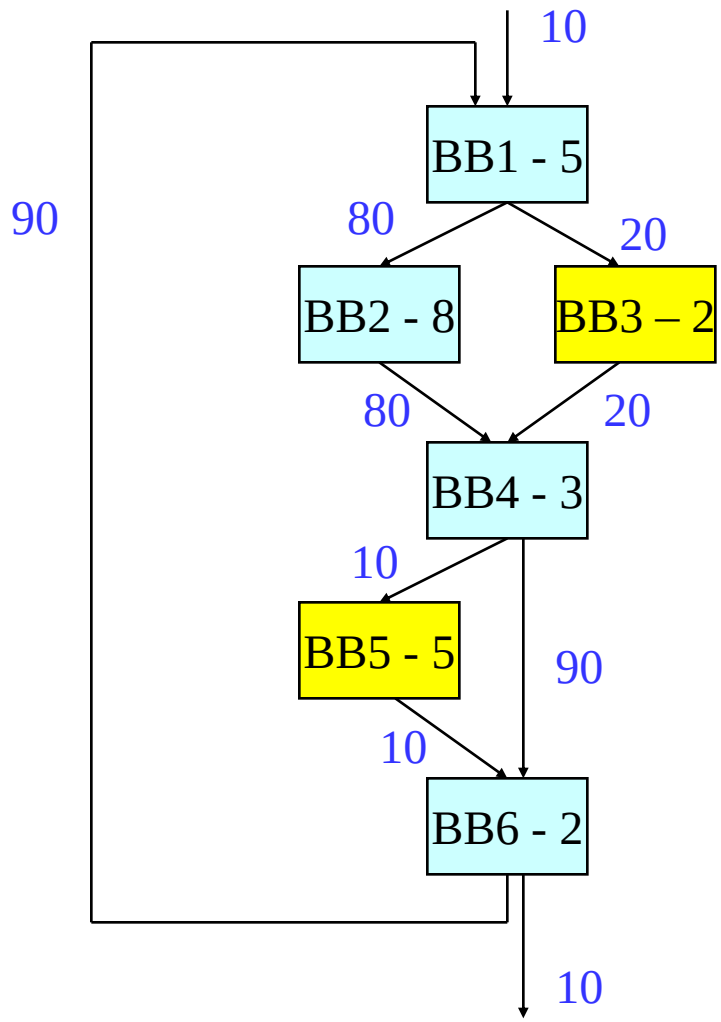


- Block selection
  - » Select subset of BBs for inclusion in HB
  - » Difficult problem
  - » Weighted cost/benefit function
    - Height overhead
    - Resource overhead
    - Hazard overhead
    - Branch elimination benefit
    - Weighted by frequency

Block Selection

- Create a trace →"main path"
  - Use a heuristic function to select other blocks that are "compatible" with the main path
  - Consider each BB by itself for simplicity
    - Compute priority for other BB's
    - Normalize against main path.
- BSVi = (K x (weight_bbi / size_bbi) x (size_main_path / weight_main_path) x bb_chari)
  - weight = execution frequency
  - size = number of operations
  - bb_char = characteristic value of each BB
    - Max value = 1, Hazardous instructions reduce this to 0.5, 0.25, ...
  - K = constant to represent processor issue rate
- Include BB when BSVi > Threshold

Example - Step 1 - Block Selection



main path = 1,2,4,6

num_ops = 5 + 8 + 3 + 2 = 18

weight = 80

Calculate the BSVs for BB3, BB5 assuming no hazards, K = 4

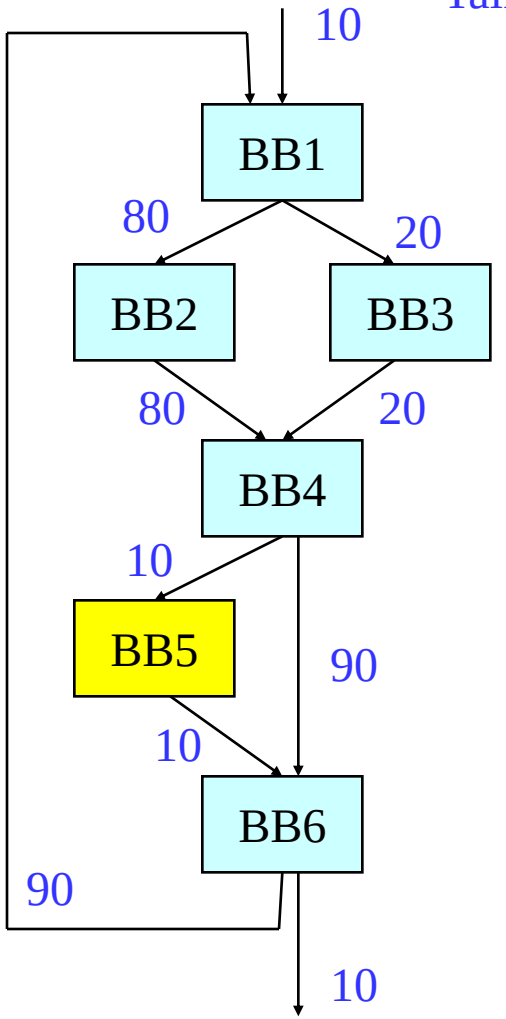
BSV3 = 4 x (20 / 2) x (18 / 80) = 9

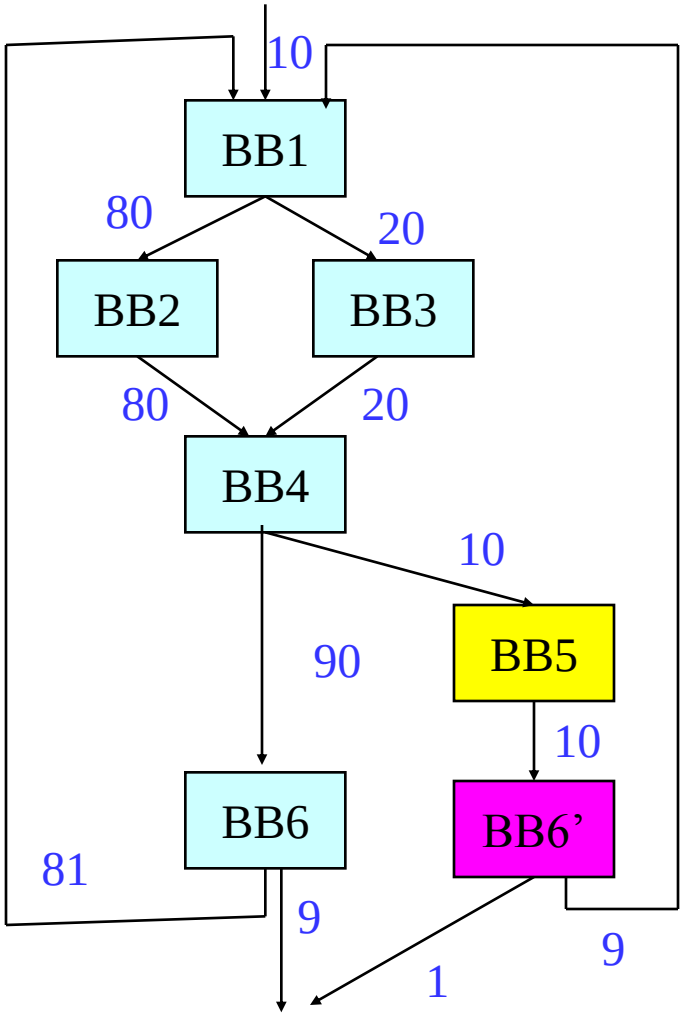
BSV5 = 4 x (10 / 5) x (18 / 80) = 1.8

If Threshold = 2.0, select BB3 along with main path

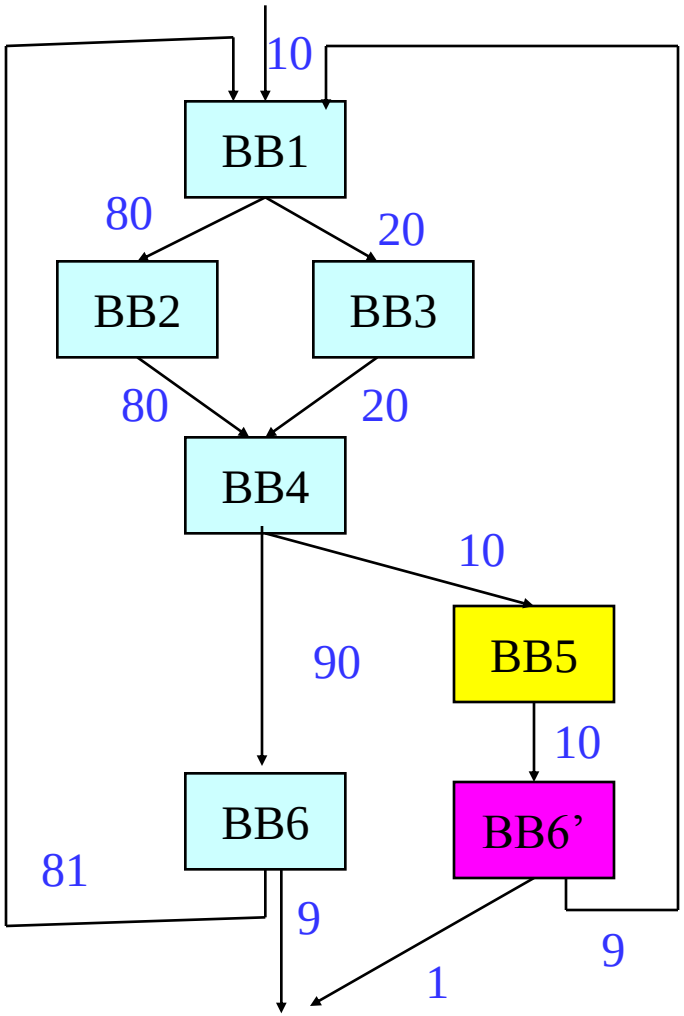
Example - Step 2 - Tail Duplication

Tail duplication same as with Superblock formation

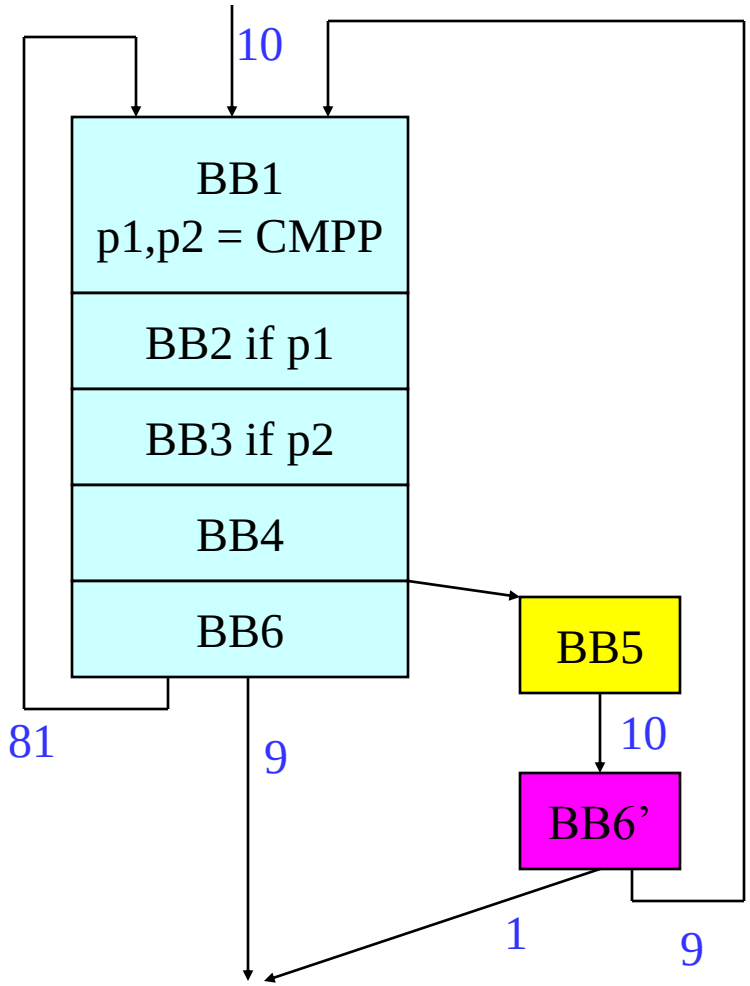




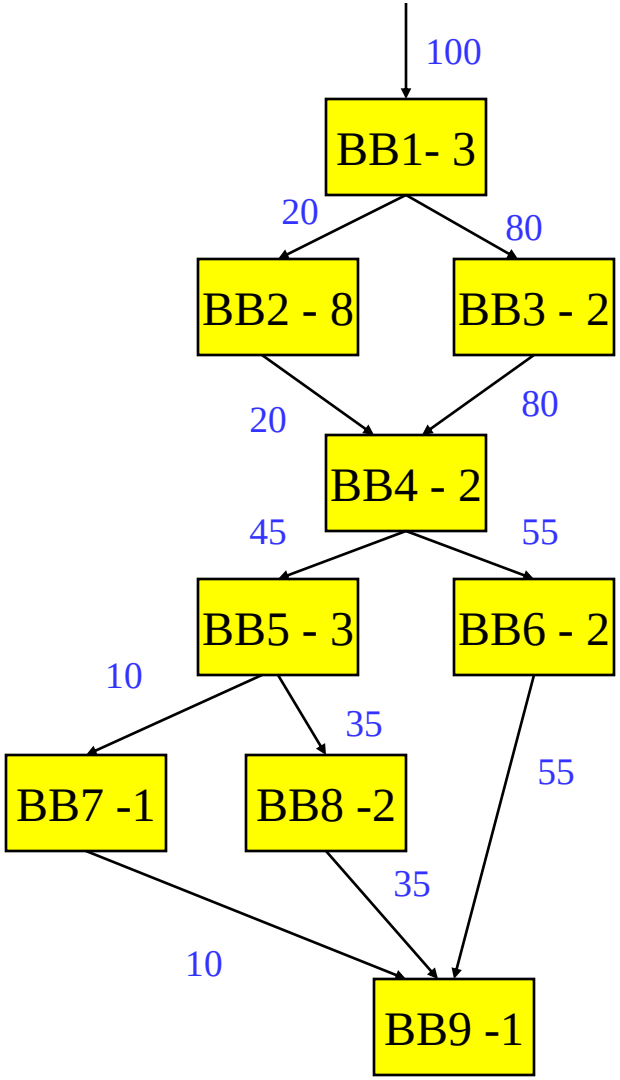
Example - Step 3 – If-conversion



If-convert intra-HB branches only!!



Class Problem



Form the HB for this subgraph
Assume K = 4, BSV Threshold = 2

Control CPR: A Branch Height Reduction Optimization for EPIC Architectures – PLDI 99

- Dependences limit performance
  - Data
  - Control
  - Long dependence chains
  - Sequential code
- Problem worse wide-issue processors
  - High degree hardware parallelism
  - Low degree of program parallelism
  - Resources idle most of the time
- Height reduction optimizations
  - Traditional compilers focus on reducing operation count
  - VLIW compilers need on increasing program parallelism

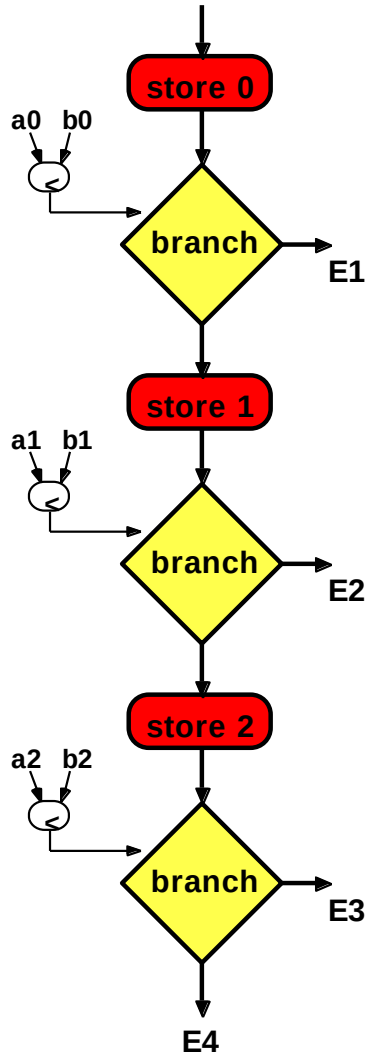
```
Loop:
        t1 = *a++;
        *b++ = t1;
        if (*a == 0) break;
        t2 = *a++;
        *b++ = t2;
        if (*a == 0) break;
        t3 = *a++;
        *b++ = t3;
        if (*a != 0) goto Loop;
```

Our Approach to Control Height Reduction

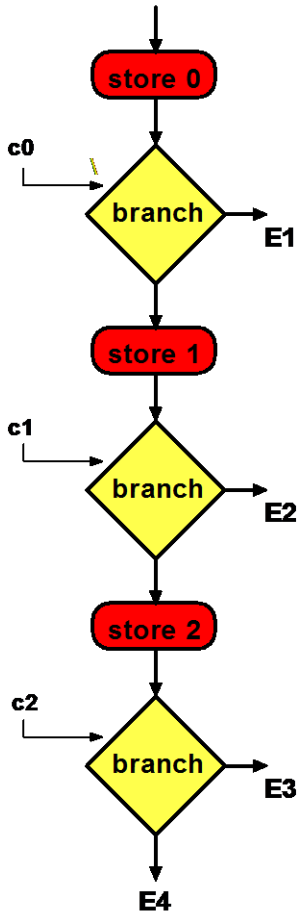
- Goals
  - Reduce dependence height through a network of branches
  - Reduce number of executed branches
  - Applicable to a large fraction of the program
  - Fit into our existing compiler infrastructure
- Difficulty
  - Reducing height while
  - Not increasing operation count
- Irredundant Consecutive Branch Method (ICBM)
  - Use branch profile information
  - Optimize likely the important control flow paths
  - Possibly penalize less important paths

Definitions

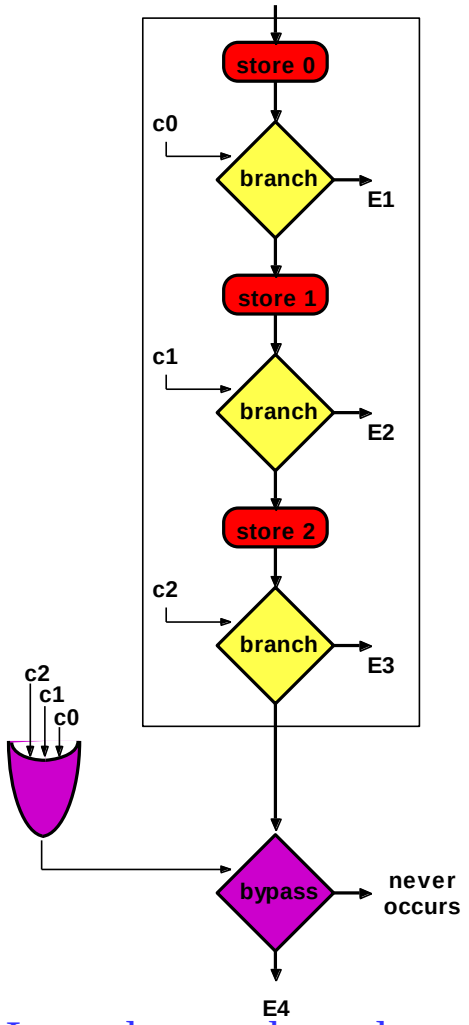
- Superblock
  - single-entry linear sequence of operations containing 1 or more branches
  - Our basic compilation unit
  - Non-speculative operations
- Exit branch
  - branch to allow early transfer out of the superblock
  - compare condition (ai < bi)
- On-trace
  - preferred execution path (E4)
  - identified by profiling
- Off-trace
  - non-preferred paths (E1, E2, E3)
  - taking an exit branch



ICBM for a Simple RISC Processor - Step 1

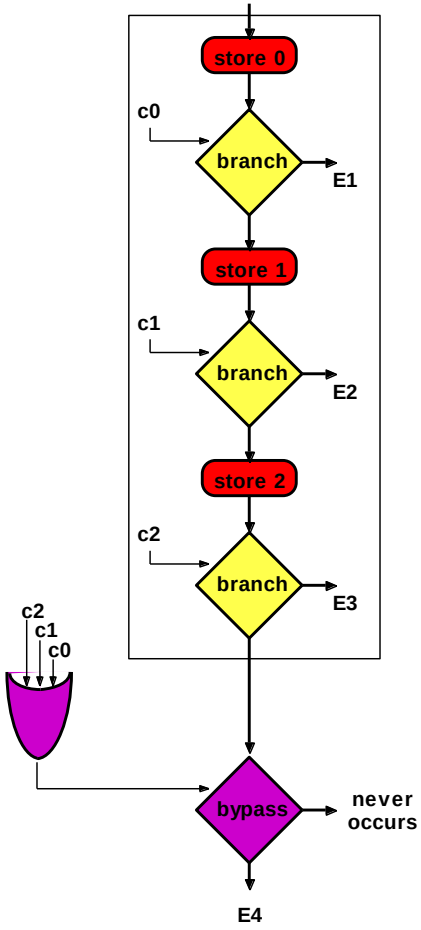


Input superblock

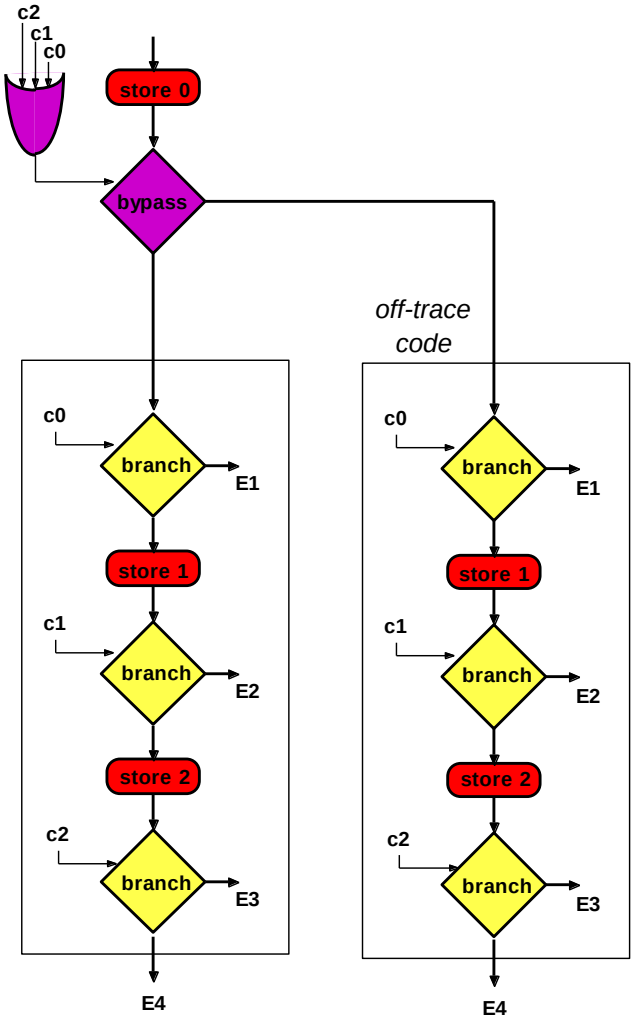


Insert bypass branch

ICBM for a Simple RISC Processor - Step 2

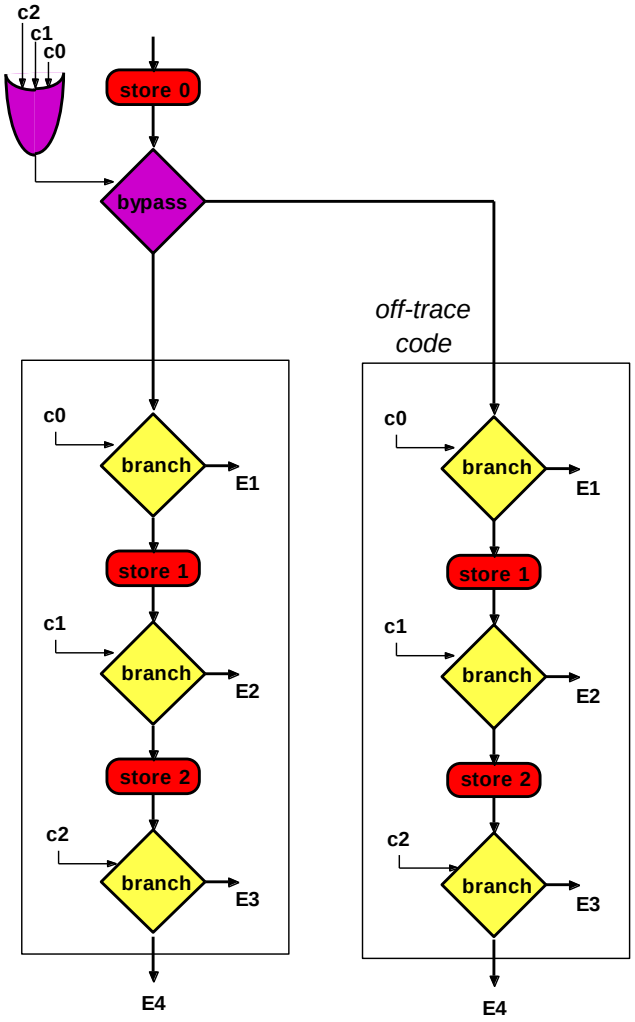


Superblock with bypass branch

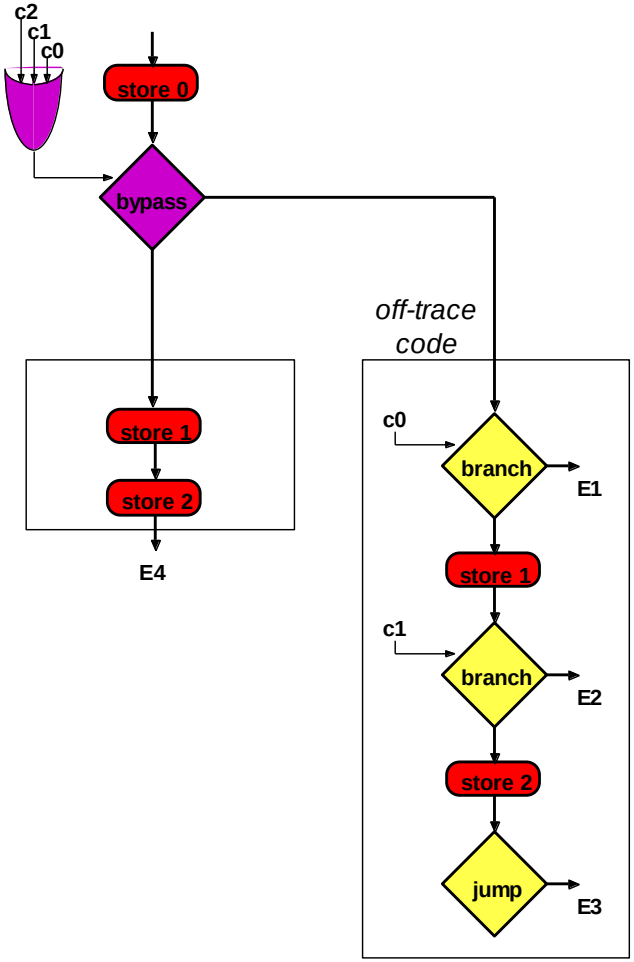


Move code down through bypass branch

ICBM for a Simple RISC Processor - Step 3

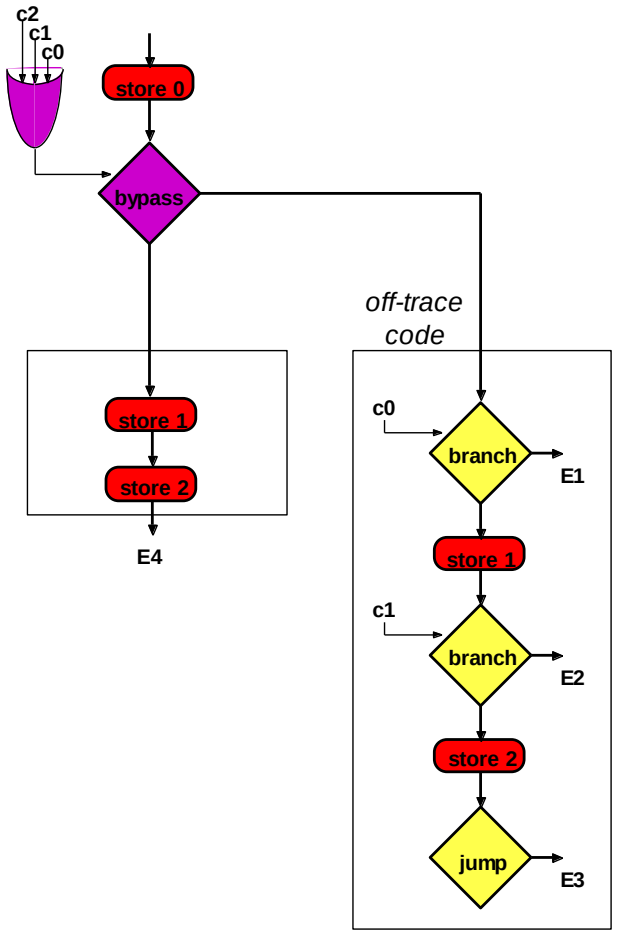


Code after downward motion

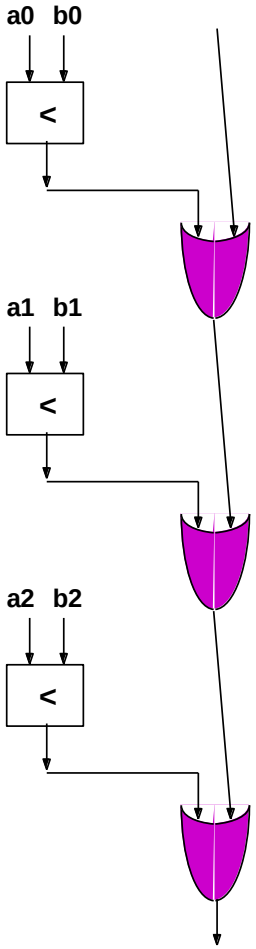


Simplify resultant code

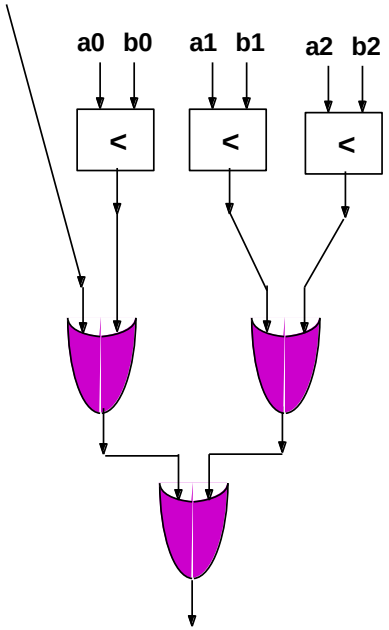
ICBM for a Simple RISC Processor - Step 4



Code after simplification



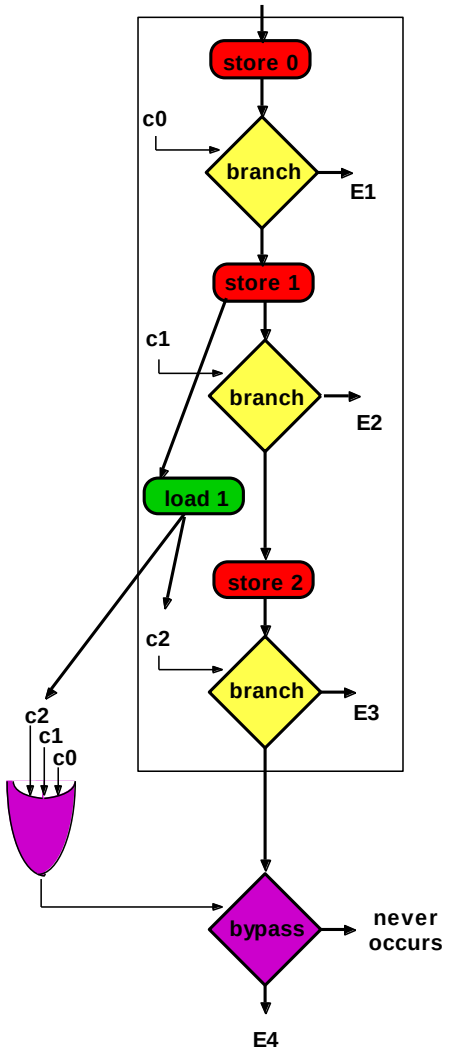
Sequential boolean expression



Height reduced expression

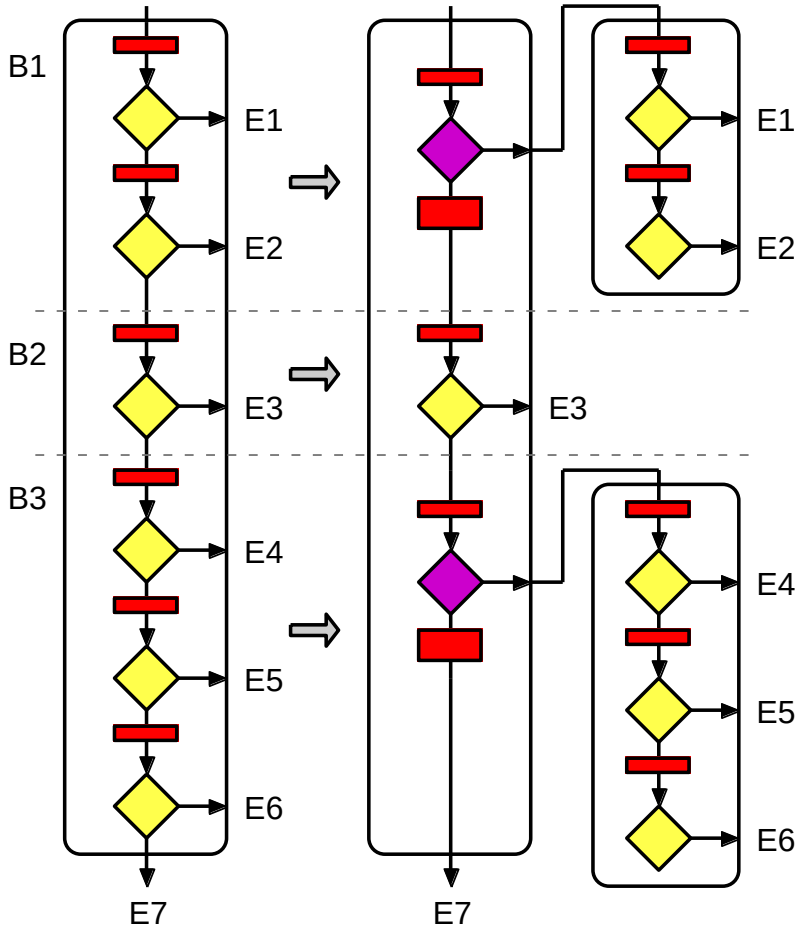
Is the ICBM Transformation Always Correct?

- Answer is no
- Problem with downward motion
  - » S1: ops to compute c0, c1, c2
  - » S2: ops dependent on branches
  - » S1 ops must remain on-trace
  - » S2 ops must move downward
  - » No dependences permitted between S1 and S2
- Separability violation
  - » Experiments - 6% branches failed
  - » Memory dependences



Blocking

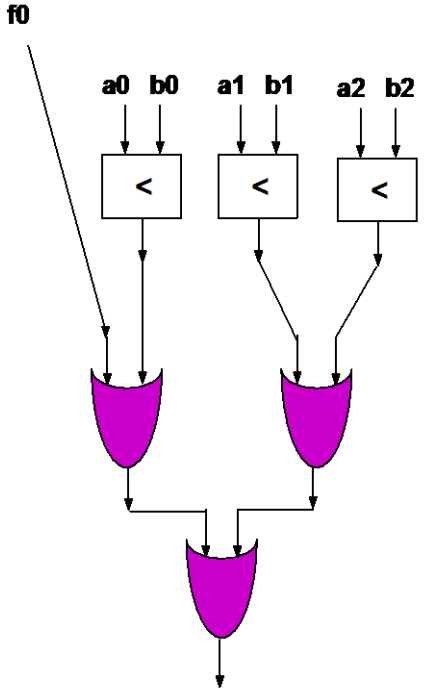
- Transforming an entire superblock
  - May not be possible
  - May not be profitable
- Solution - CPR blocks
  - Block into smaller subregions
  - Linear sequences of basic blocks
  - Apply CPR to each subregion
- Grow CPR block incrementally
- Terminate CPR block when
  - Correctness violation
  - Performance heuristic

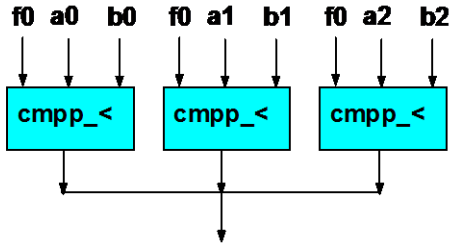


a) blocked input b) transformed result

ICBM for an EPIC Processor (HPL-PlayDoh)

- Predicated execution
  - » Boolean guard for all operations
  - » a = b + c if p
- Increases complexity of ICBM
  - » Generalize the schema
  - » Analyze and transform complex predicated code
  - » Suitability pattern match
  - » Proof of correct code generation
- Increases efficiency of ICBM
  - » Wired-AND/wired-OR compares
  - » Accumulate disjunction of conditions into a predicate
  - » Compare network reduced to 1 level





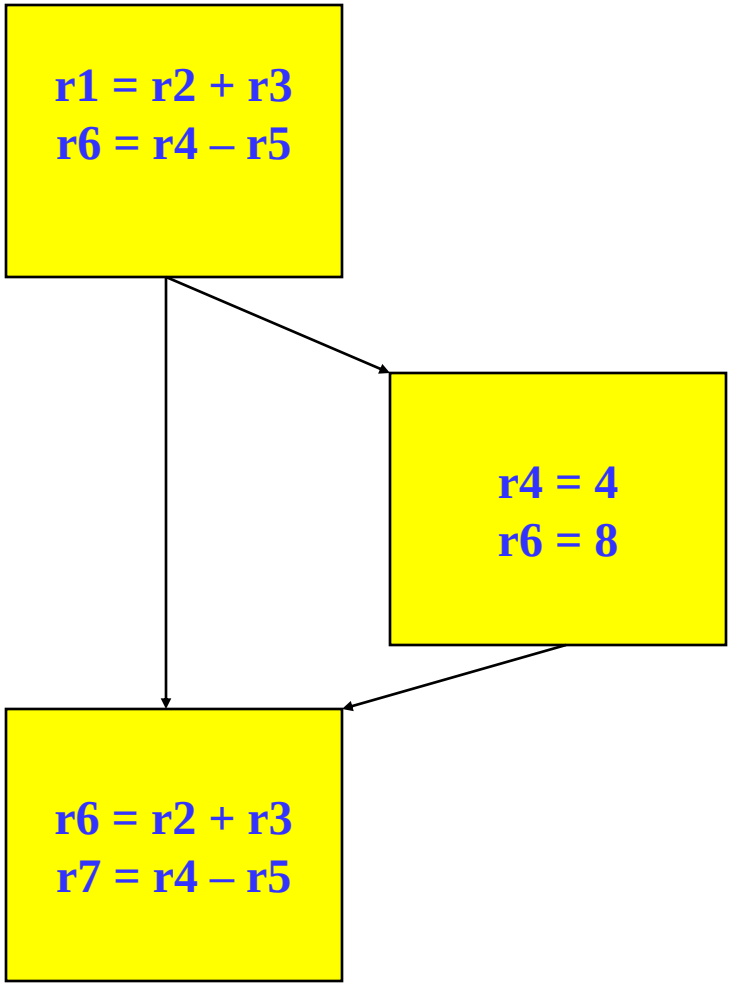
Taste of the Results

Operation Dilation				
	S total	S branch	D total	D branch
099.go	1.08	1.04	1.04	0.86
cmp	1.08	1.01	0.71	0.13
085.cc1	1.05	1.02	0.97	0.63
Gmean-all	1.08	1.03	0.93	0.42

Speedup					
	Sequential	Narrow	Medium	Wide	Infinite
099.go	0.96	1.01	1.02	1.02	1.02
cmp	1.53	1.25	1.79	2.87	3.6
085.cc1	1.13	1.06	1.12	1.15	1.18
Gmean-all	1.13	1.05	1.18	1.33	1.41

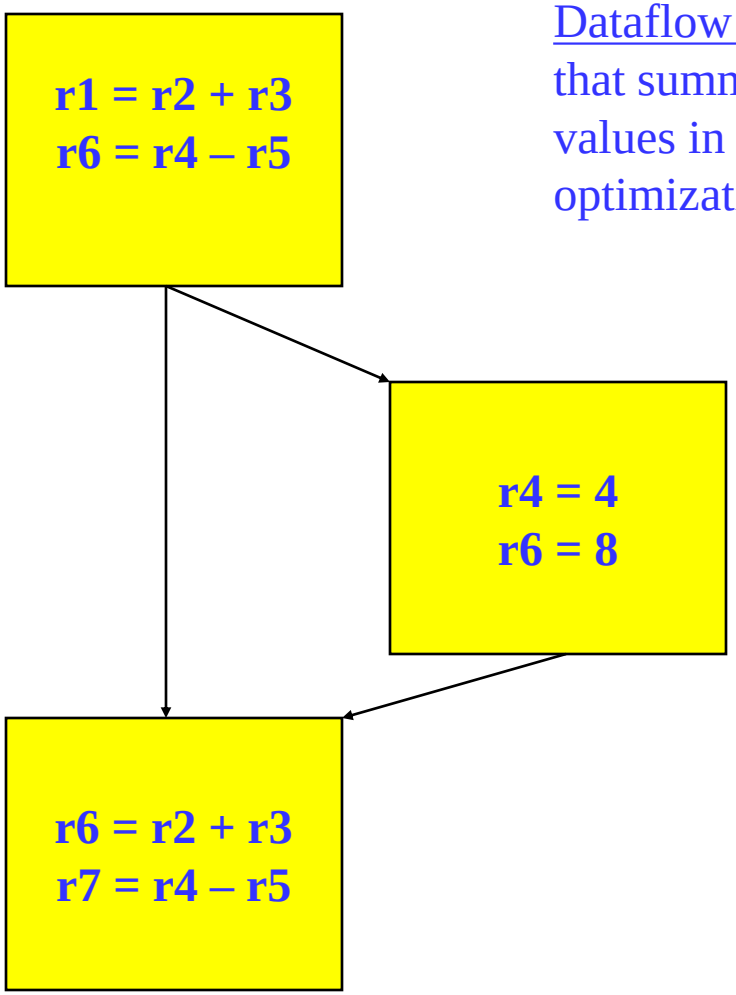
Next Topic: Dataflow Analysis + Optimization

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
```

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)