HY425 Lecture 05: Branch Prediction

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

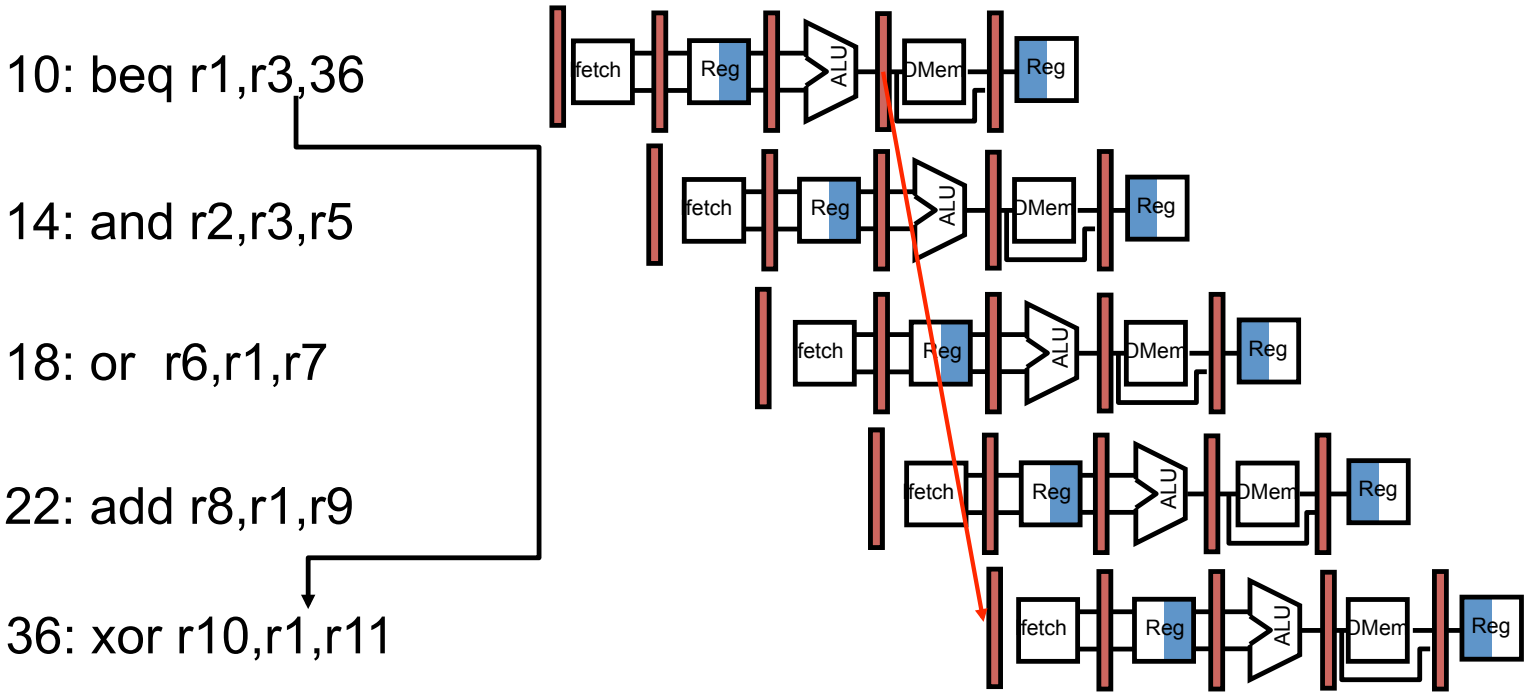
October 19, 2011

Exploiting ILP in hardware

Dynamic scheduling

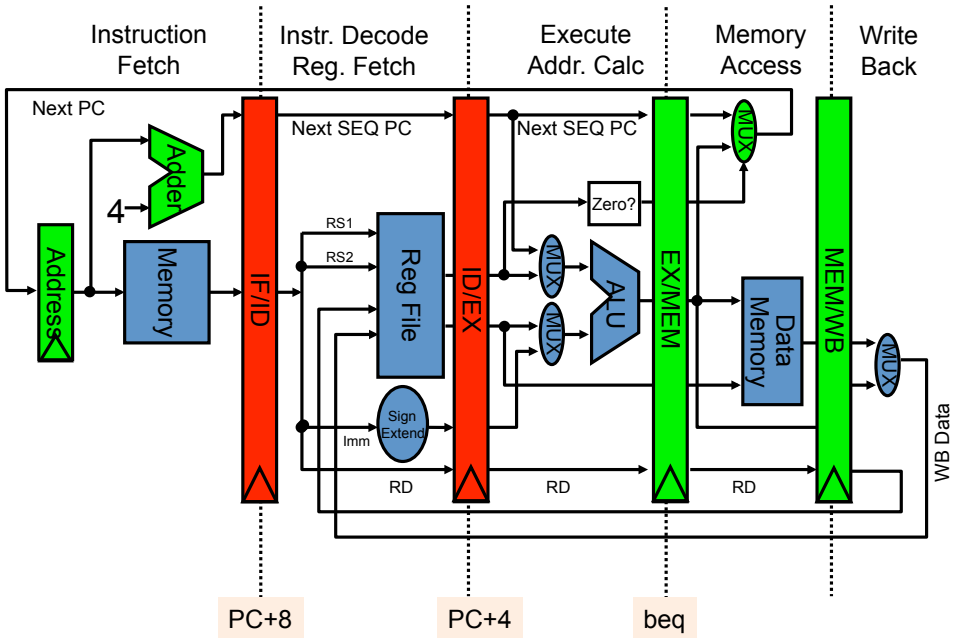
- Instruction scheduling critical for
  - Overlapping stalls due to hazards
  - Resolving hazards
- Two hardware dynamic scheduling schemes
  - Scoreboard: out-of-order execution, out-of-order completion, dependence tracking through register file, stall upon detection of hazards (RAW, WAW, WAR)
  - Tomasulo: out-of-order execution, out-of-order completion, renaming through FUs to resolve WAW, WAR hazards

Control dependences in simple pipeline



Understanding control hazards

MIPS datapath

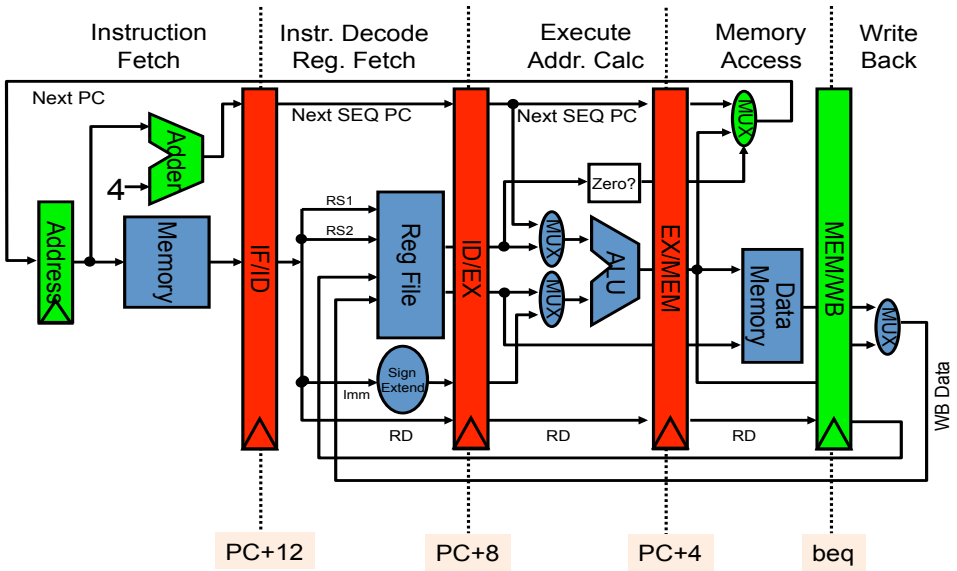


Branch execution

- Comparison with zero and target address calculation at EX stage
- 2 stall cycles

Understanding control hazards

MIPS datapath



Optimized branch

- Branch taken decision plus potential branch target out of EX stage
- Next PC forwarded from MEM stage through multiplexer
- 1 more stall cycle for a total of 3

Reducing branch stall impact

Impact of branches on performance

- Branch frequency (conditional, unconditional)
  - ca. 20% for integer programs
  - ca. 10% for floating point programs

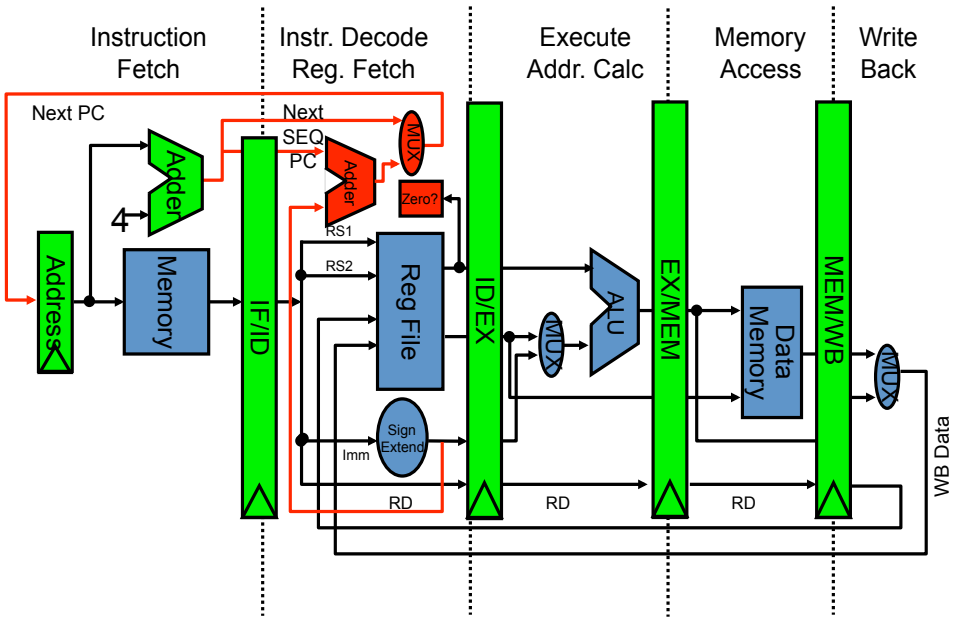
$$\textit{Stall CPI from branches} = \textit{branch frequency} \times \textit{branch penalty}$$

$$\textit{Speedup}_{\textit{pipeline}} = \frac{\textit{Pipeline depth}}{1 + \textit{branch frequency} \times \textit{branch penalty}}$$

- Max speedup drops from 5.0 to 3.1 (int) 3.8 (fp)

Reducing branch stall impact

HW solution



Explanation

- Comparison with zero happens at EX stage
- Move comparison to ED stage
- May increase cycle time!

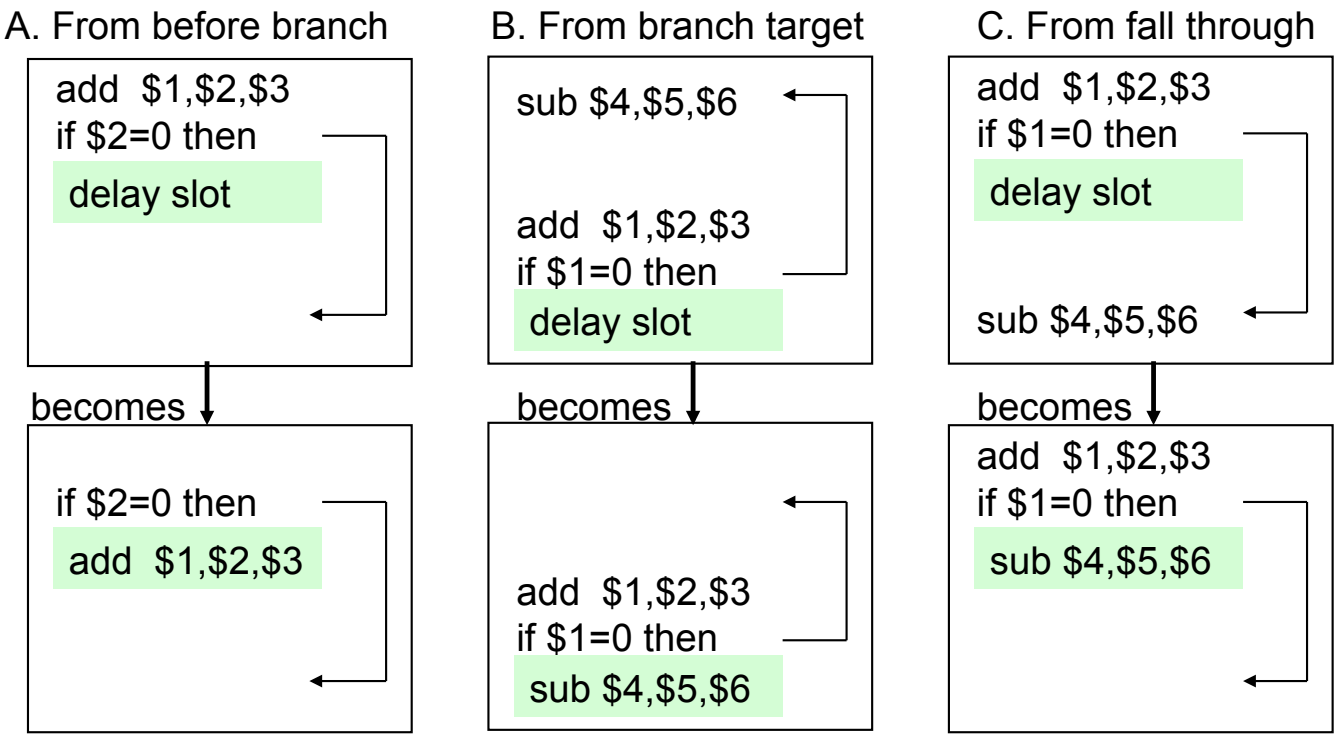
Reducing branch stall impact

HW/SW solution

- Delayed branches always execute the instruction in the slot following the branch (PC+4)
- Instruction two slots down (PC+8) affected by the branch
- Software (compiler) tasked with filling delay slots
- Choices are from before the branch, from the target (branch taken), or from the fall through path (branch not taken)

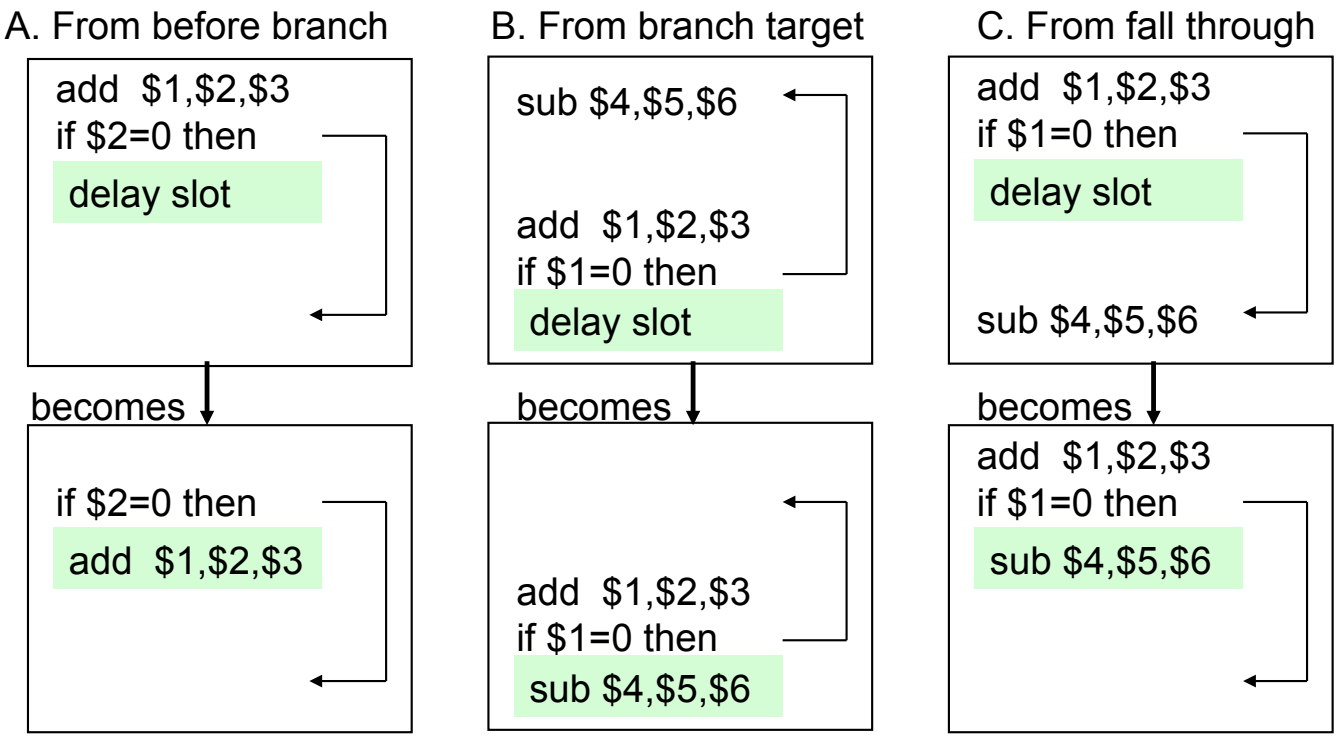
Options for filling delay slot

Three paths to look for instructions



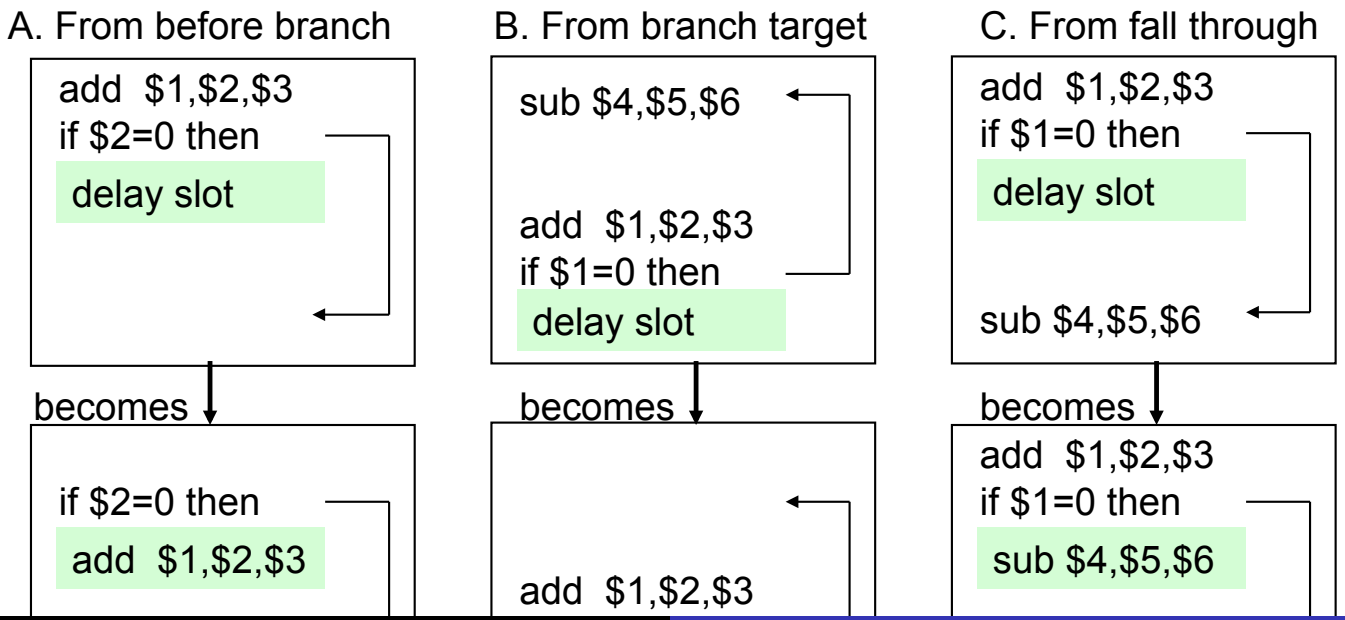
Options for filling delay slot

A: reduces instruction and improves performance



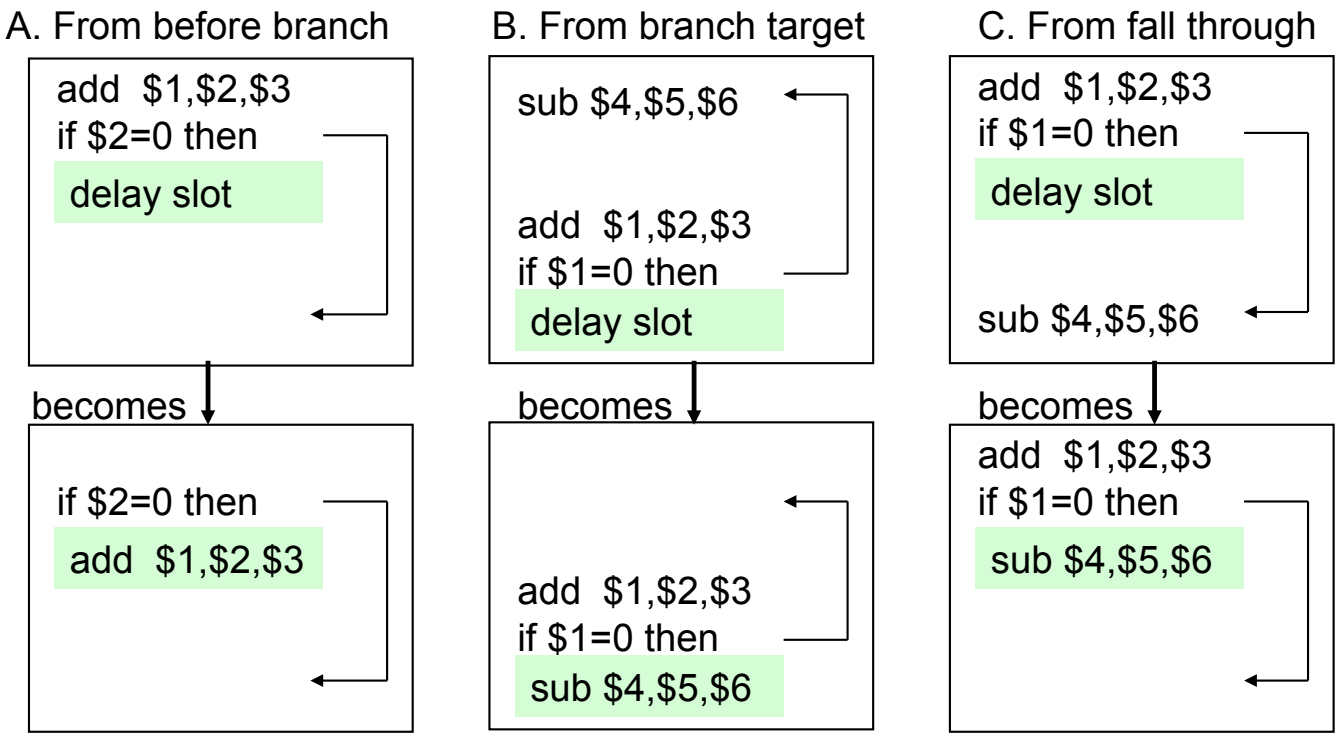
Options for filling delay slot

B: may require copying of the instruction because instruction may be reached from another path and must check if it is OK to execute instruction in delay slot if branch is not taken



Options for filling delay slot

C: may execute conditionally dependent instruction, must check if OK



Static branch prediction

Delay slots

- Effectiveness of delayed branch depends on compiler's ability to schedule instructions in delay slots
- Branch delays grow in deeper pipelines – need more effective instruction scheduling to fill more delay slots
- Compiler can find more instructions with speculation (prediction) on branch outcomes
- Static branch prediction – compiler or other software predicts outcome of all branches before program execution

Example

```
        LW   R1, 0(R2)
        SUB  R1,R1,R3
        BEQZ R1, L
        OR   R4,R5,R6
        ...
L:      ADD    R7, R8, R9
```

- Load-use stall between LW and SUB, BEQZ
- Assume
  - BEQZ almost always taken
  - fall-through path does not depend on R7
- Speculatively move ADD after LW to remove stall

Example

```
        LW   R1, 0(R2)
        SUB  R1,R1,R3
        BEQZ R1, L
        OR   R4,R5,R6
        ...
L:      ADD    R7, R8, R9
```

- Load-use stall between LW and SUB, BEQZ
- Assume
  - BEQZ almost never taken
  - taken path does not depend on R4
- Speculatively move OR after LW to remove stall

Static branch prediction

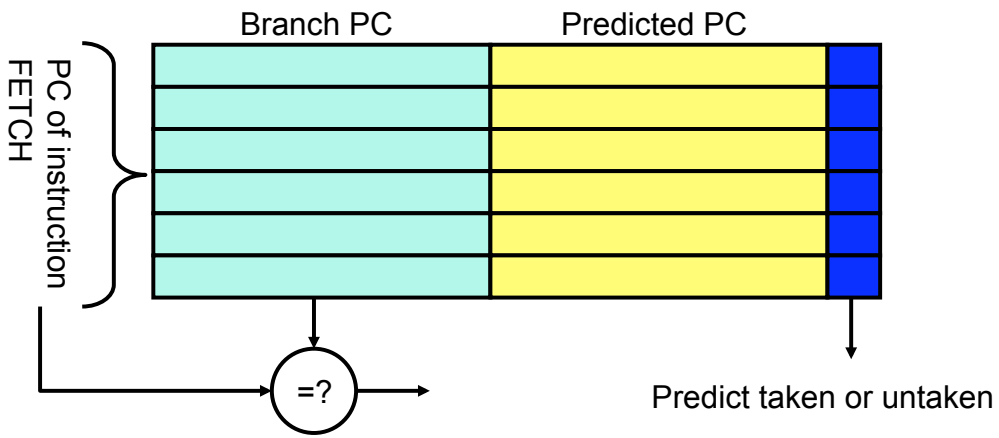
- Examine program behavior statically
- Alternative, use profiling information
- Predict always taken
  - Most forward and backward branches taken in programs
  - Inaccuracy up to ca. 40%
- Predict according to direction of branch
  - Backward taken (loops), forward not-taken
  - Inaccuracy ca. 30%–40%
- Profile-based prediction
  - FP programs misprediction rate 9%, stdev 4%
  - INT programs misprediction rate 15%, stdev 5%
- Pros: simple, no additional hardware
- Cons: Behavior varies widely with programs

Key ideas of dynamic branch prediction

Branch prediction elements

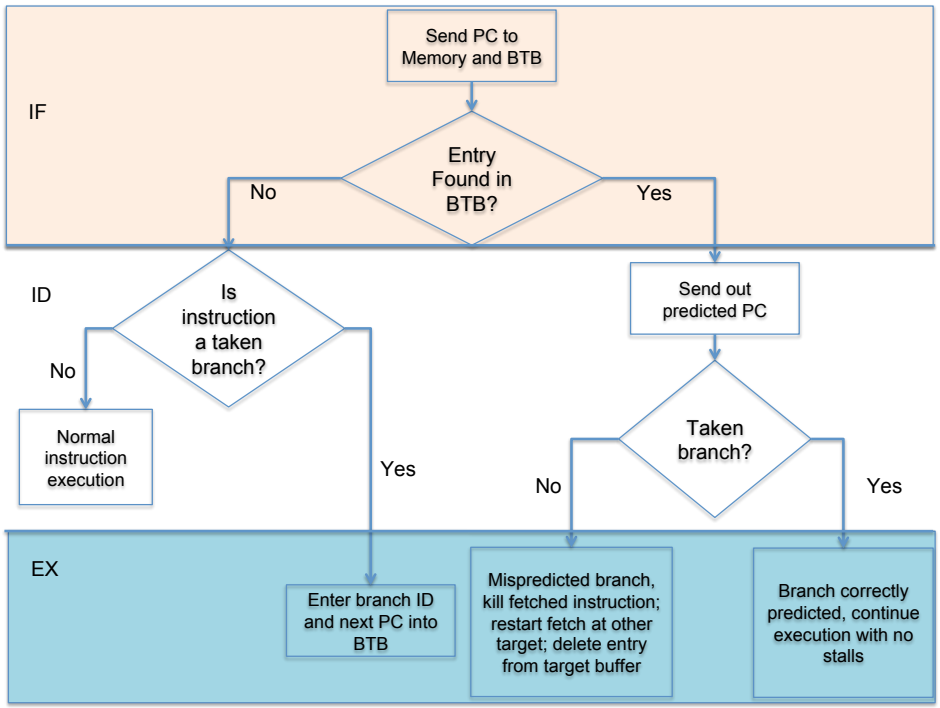
- Use some of the bits of the PC to index a branch prediction table
- Predict direction of branch
- Predict target PC of branch, if taken

Branch history table (BHT)



Dynamic branch prediction

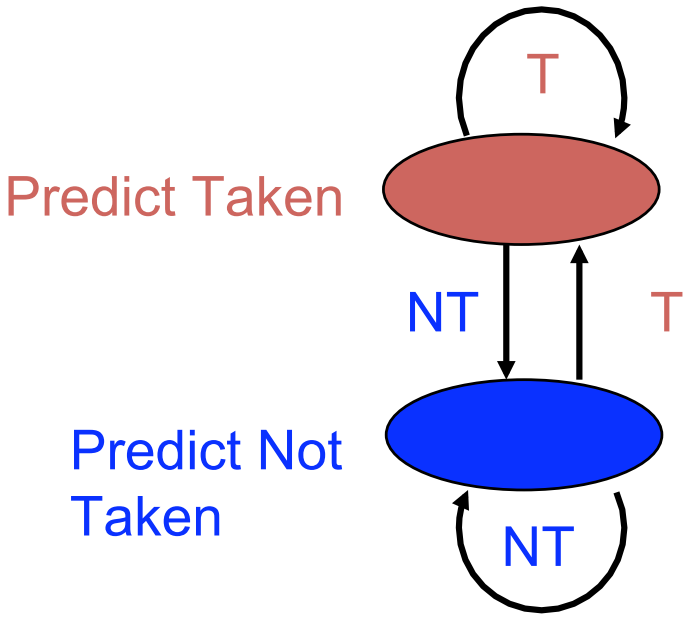
Branch predictor datapath



Details

- Instruction memory accessed in parallel with BTB
- Branch prediction tables updated at EX stage

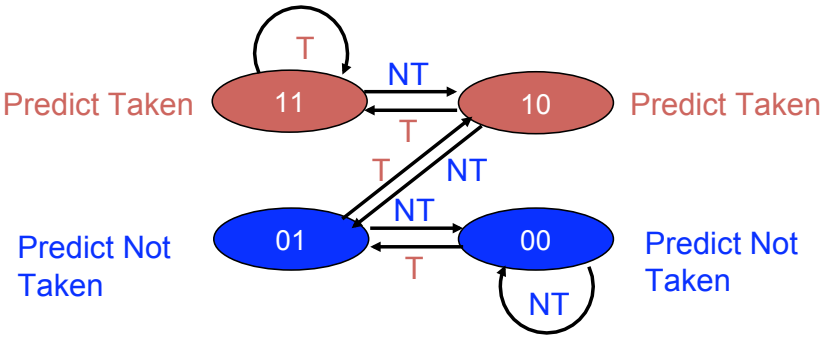
1-bit branch history (last branch outcome)



Explanation

- 0 not-taken, 1 taken
- Misses twice in a loop
  - Last loop's last branch not taken (prediction bit=0)
  - First iteration taken (miss, prediction bit=1)
  - Last iteration not-taken (miss, prediction bit=0)

2-bit branch history (saturating counter)



Explanation

- Prediction misses twice before reversing
- Generalization: n-bit saturating counter
  - Predict taken if counter $>= 2^{n-1}$, non-taken otherwise
- 1 miss in loop

Saturating counter predictor

Implementation

- BHT implemented as small cache accessed during IF stage
  - If branch and predicted taken fetch from taken path
- 5-stage pipeline performs branch comparison in ID stage
- Effective for 5-stage pipeline if we know branch target at IF

Saturating counter predictor

Performance

- Accuracy of branch direction prediction
- Accuracy of branch target PC prediction
- Frequency of branches in program
- Typically higher accuracy in FP programs (loop-dominated code)

Correlating branch predictors

Example in C

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa != bb) {...}
```

Example in assembly

```
        DSUBUI      R3,R1,#2
        BNEZ        R3,L1       ;branch b1 (aa!=2)
        DADD        R1,R0,R0    ;aa=0
L1:     DSUBUI      R3,R2,#2
        BNEZ        R3,L2       ;branch b2 (bb!=2)
        DADD        R2,R0,R0    ;bb=0
L2:     DSUBU       R3,R1,R2    ;R3=aa-bb
        BEQZ        R3,L3       ;branch b3 (aa==bb)
```

Correlating branch predictors

Correlation of branches

- Behavior of b3 is correlated with outcome of b1 and b2
- Saturating predictors predict the history of a branch based on the recent history of the same branch
- Correlating predictors use the history of the currently examined branch and prior branches

```
        DSUBUI      R3,R1,#2
        BNEZ        R3,L1          ;branch b1 (aa!=2)
        DADD        R1,R0,R0       ;aa=0
L1:     DSUBUI      R3,R2,#2
        BNEZ        R3,L2          ;branch b2 (bb!=2)
        DADD        R2,R0,R0       ;bb=0
L2:     DSUBU       R3,R1,R2       ;R3=aa-bb
        BEQZ        R3,L3          ;branch b3 (aa==bb)
```

Correlating branch predictors

Example in C

```
if (d==0)
        d=1;
if (d==1) {...}
```

Example in asembly

```
        BNEZ        R1,L1       ;branch b1 (d!=0)
        DADDIU      R1,R0,#1    ;d==0, so d=1
L1:     DADDIU      R3,R1,#-1
        BNEZ        R3,L2       ;branch b2 (d!=1)
...
L2:
```

Possible execution sequences

Initial value of d	d==0?	b1	Value of d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

Correlating branch predictors

Performance of simple 1-bit predictor

- Assume predictor initialized to not-taken
- Assume d alternates between 0 and 2

d==?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

- all branches mispredicted

Correlating branch predictors

Performance of correlating predictor

- Use 1 bit of correlation
- Predict separately for last branch taken, last branch not taken
- Last branch can be the same (loops) or different PC

Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	NT	NT
NT/T	NT	T
T/NT	T	NT
T/T	T	T

- Two prediction bits (last branch NT/last branch T)

Correlating branch predictors

Correlating predictor behavior, 1 bit correlation

Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	NT	NT
NT/T	NT	T
T/NT	T	NT
T/T	T	T

d==?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	NT/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T
2	T/NT	T	T/NT	NT/T	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T

- prediction, hit, miss

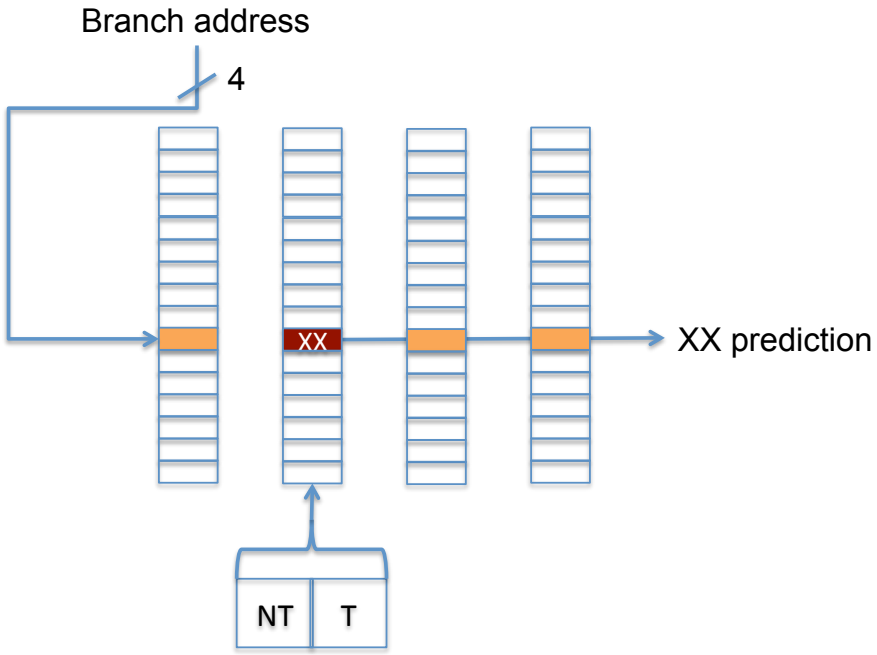
Generalizing correlating predictors

(m,n) notation

- (m, n) – use behavior of last m branches, predict using an n-bit predictor
- Example: (2,2) predictor:
  - Use history of last 2 branches (NT/NT, NT/T, T/NT, T/T)
  - History of last 2 branches selects a 2-bit saturating counter predictor
  - History of last m branches recorded with m-bit shift register
  - BHT indexed with branch PC (row) and history register (column)

Generalizing correlating predictors

(2,2) predictor



Global history of last two branches

Explanation

- m=2, 2-bit history of last two branches executed in the program
- n=2, 2-bit saturating counter predictor of next branch, selected using branch PC (row) and history (column)

Multilevel branch prediction

Two-level correlating (adaptive) predictors

- Two levels of branch prediction
- First level indexed using history of last k executed branches
  - Global: last k executed branches of the program
  - Per-address: last k executions of the examined branch
- Second level records pattern for the given PC-history combination

Multilevel branch prediction

Tournament predictor

- First level selects among different adaptive predictors
- 2-bit saturating counter per branch to select between predictors
- Second and third-levels, implement (m, n) predictor

Yeh and Patt's classification (ISCA'93)

Taxonomy of two-level branch predictors

- XAy predictor
- X=G, global history register
- X=P, per-branch history register
- X=S, per-set-of-branches history register
- y=g, global branch history table
- y=p, per-branch history table
- y=s, set-associative per branch history table
  - Set-associative mapping of branch PCs reduces conflicts (aliases)

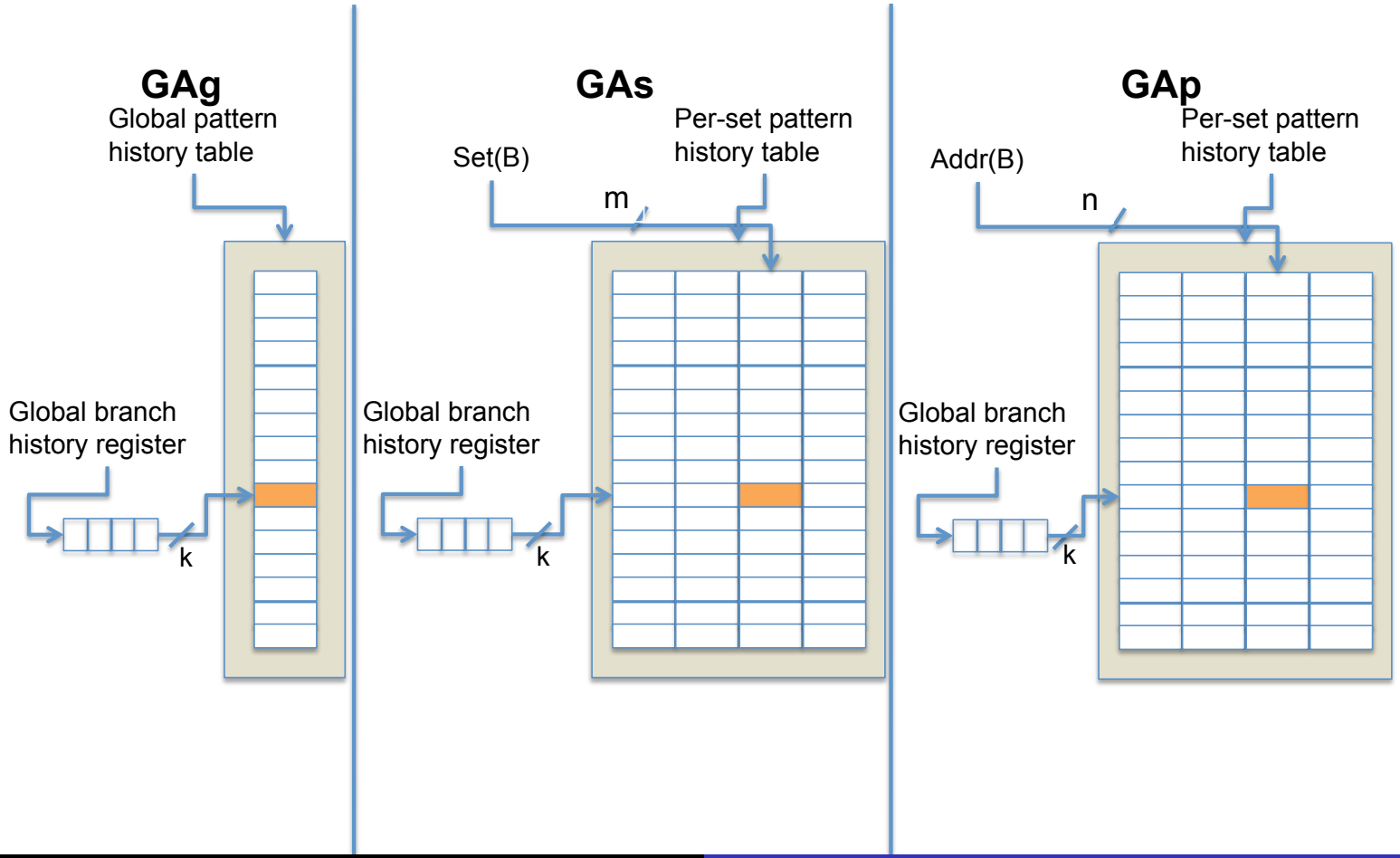
Yeh and Patt's classification (cont.)

Variations of two-level branch predictors

Variation	Description
GAg	Global Adaptive Branch Prediction using one global pattern history table
GAs	Global Adaptive Branch Prediction using per-set (of branch PCs) pattern history tables
GAp	Global Adaptive Branch Prediction using per-address (of branch PC) pattern history tables
PAg	Per-address Adaptive Branch Prediction using global pattern history table
SAg	Per-Set Adaptive Branch Prediction using global pattern history table
SAs	Per-Set Adaptive Branch Prediction using per-set pattern history tables
SAp	Per-Set Adaptive Branch Prediction using per-address pattern history tables

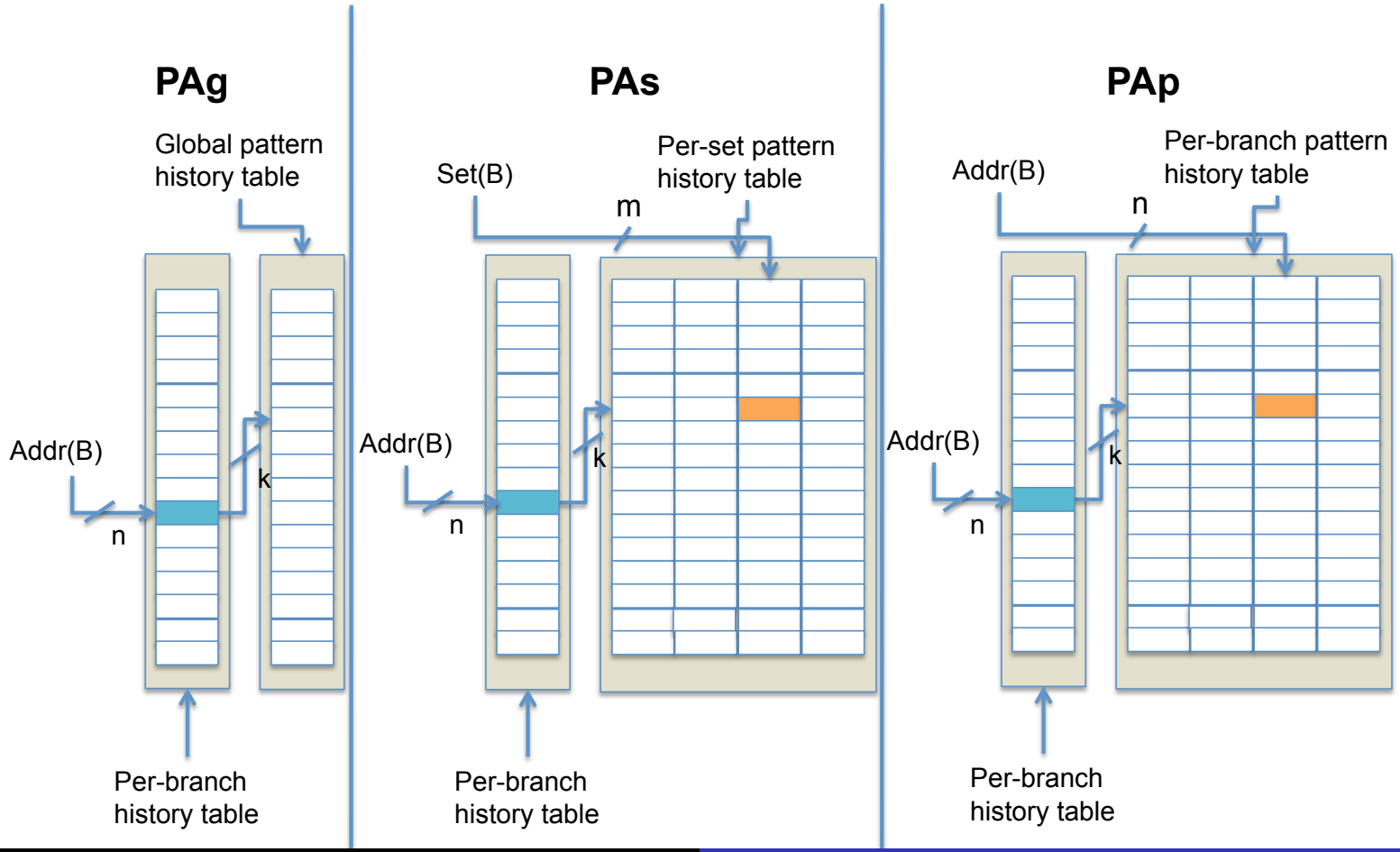
Yeh and Patt's classification (cont.)

Global history predictors



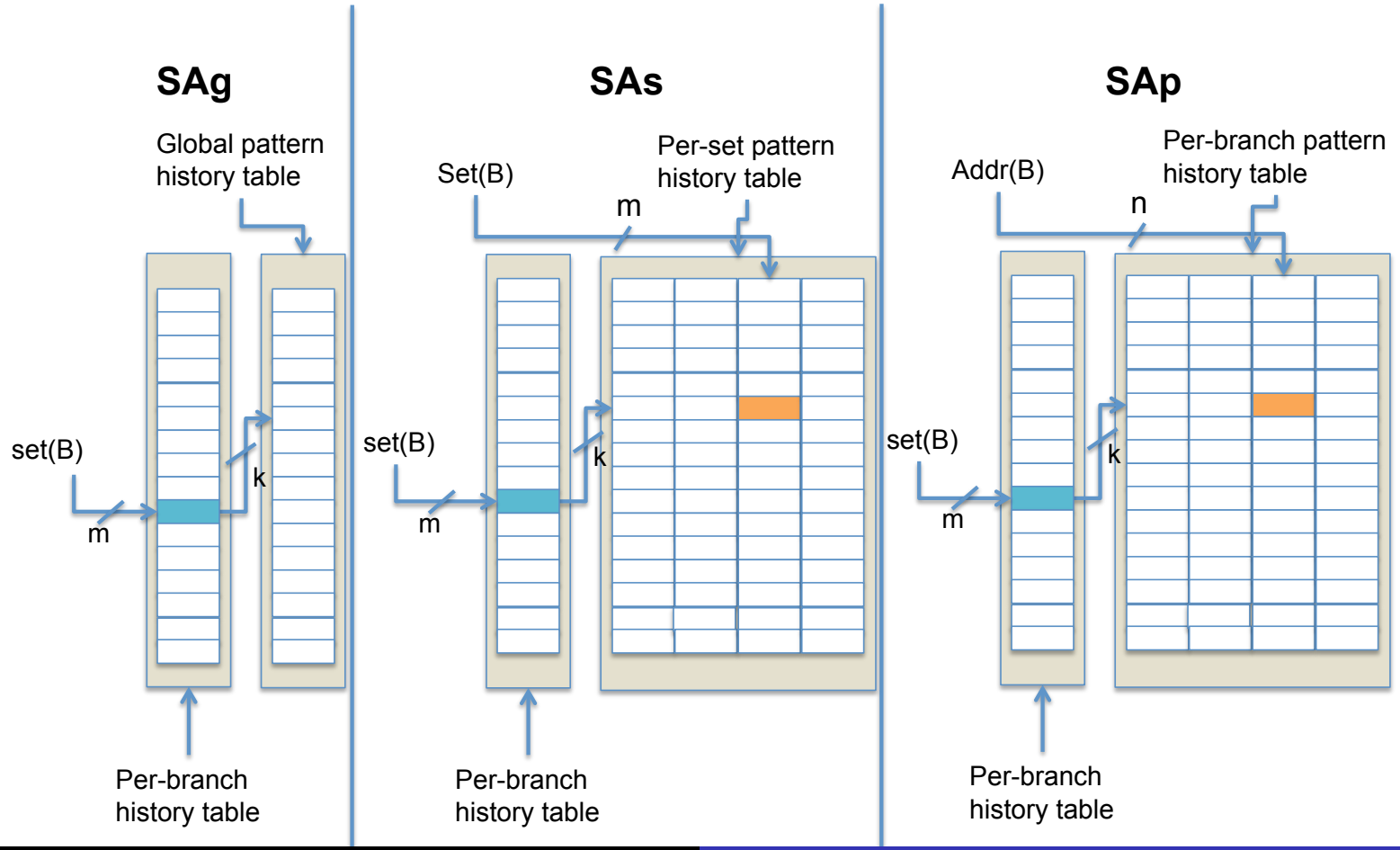
Yeh and Patt's classification (cont.)

Per-branch history predictors



Yeh and Patt's classification (cont.)

Per-branch history predictors



Return address prediction

Indirect jumps

- Usually originating in procedure returns
- Also in indirect function calls, case statements, goto's
- Can be predicted to avoid stall for reading registers
- Prediction using a small stack
  - Push return address on call
  - Pop return address upon return (prediction)
- If stack has sufficient size then perfect prediction

Summary

- Branch prediction essential to reduce stalls due to branches
- Two elements to any predictor
  - Predict direction of the branch
  - Predict target of the branch (sensitive to conflicts in history tables)
- History and correlation help improve branch prediction accuracy
- Overall performance depends on:
  - Branch prediction accuracy
  - Penalty of misprediction
    - Fetch from both paths

First programming assignment information