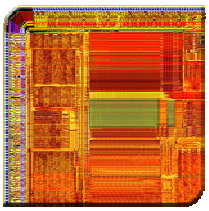
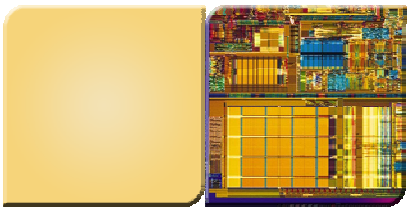
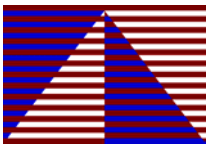
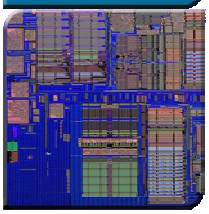







ECE 4100/6100
Advanced Computer Architecture

Lecture 5 Branch Prediction

Prof. Hsien-Hsin Sean Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology

Georgia Institute of Technology

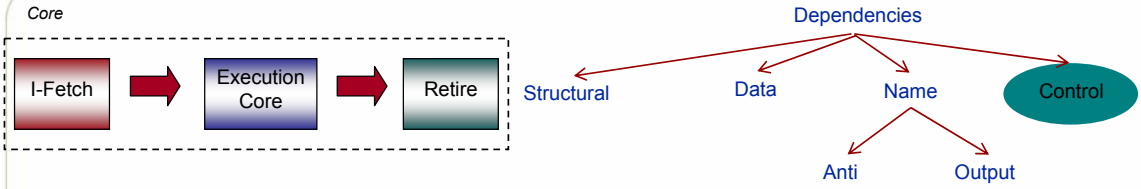


Reading for this Module

- Branch Prediction
  - Appendix A.2 (pg. A-21 – A-26), Section 2.3
- Branch Target Buffers and Return Address Predictors
  - Section 2.9
- Reading assignments
  - Papers on class website

Georgia Institute of Technology

Control Dependencies



- Control dependencies determine execution order of instructions
  - Instructions may be control dependent on a branch

DADD R5, R6, R7

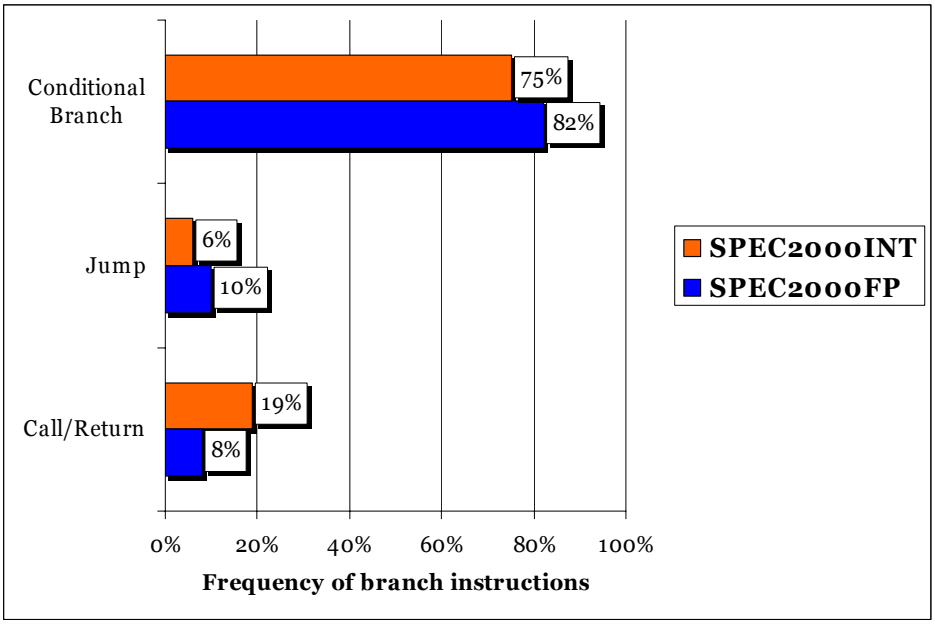
BNE R4, R2, CONTINUE

DMUL R4, R2, R5 DSUB R4, R9, R5

Predict What?

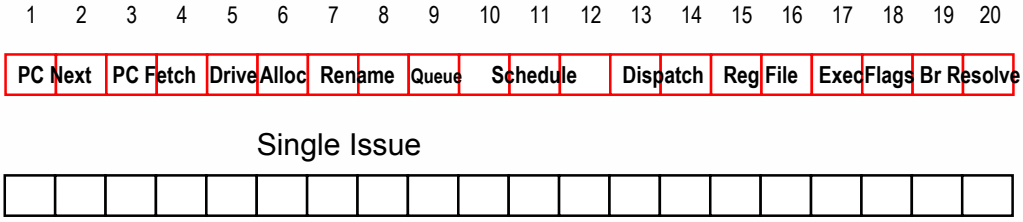
- Direction (1-bit)
  - Single direction for unconditional jumps and calls/returns
  - Binary for conditional branches
- Target (32-bit or 64-bit addresses)
  - Some are easy
    - One: Uni-directional jumps
    - Two: Fall through (Not Taken) vs. Taken
  - Many: Function Pointer or Indirect Jump (e.g. jr r31)

Categorizing Branches

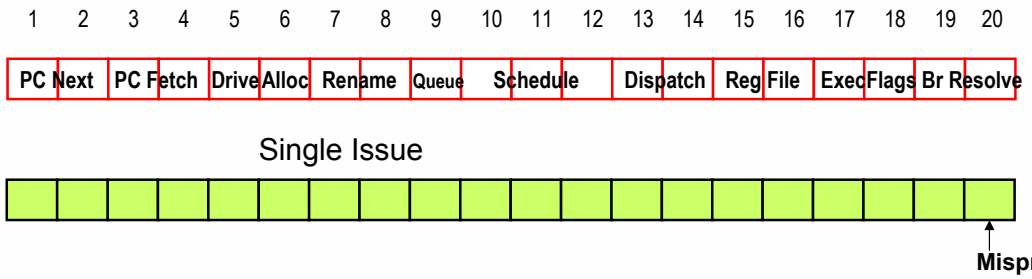


Source: H&P using Alpha

Branch Misprediction



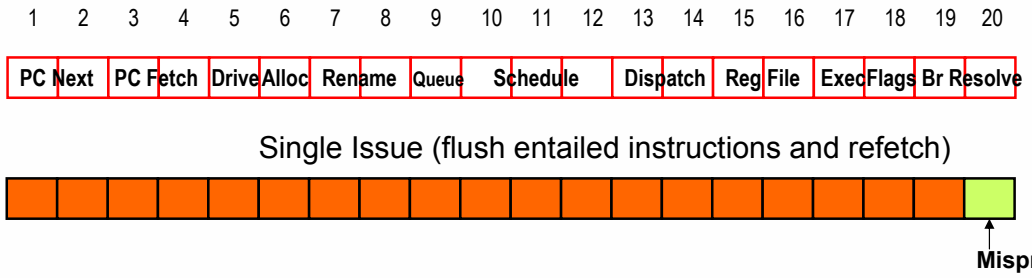
Branch Misprediction



Single Issue

Mispredict

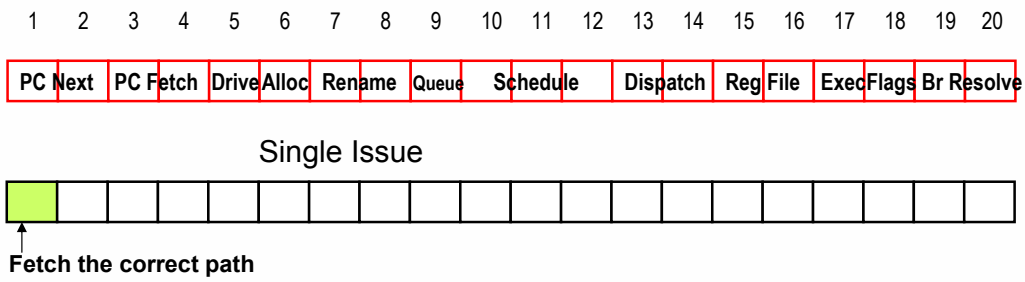
Branch Misprediction



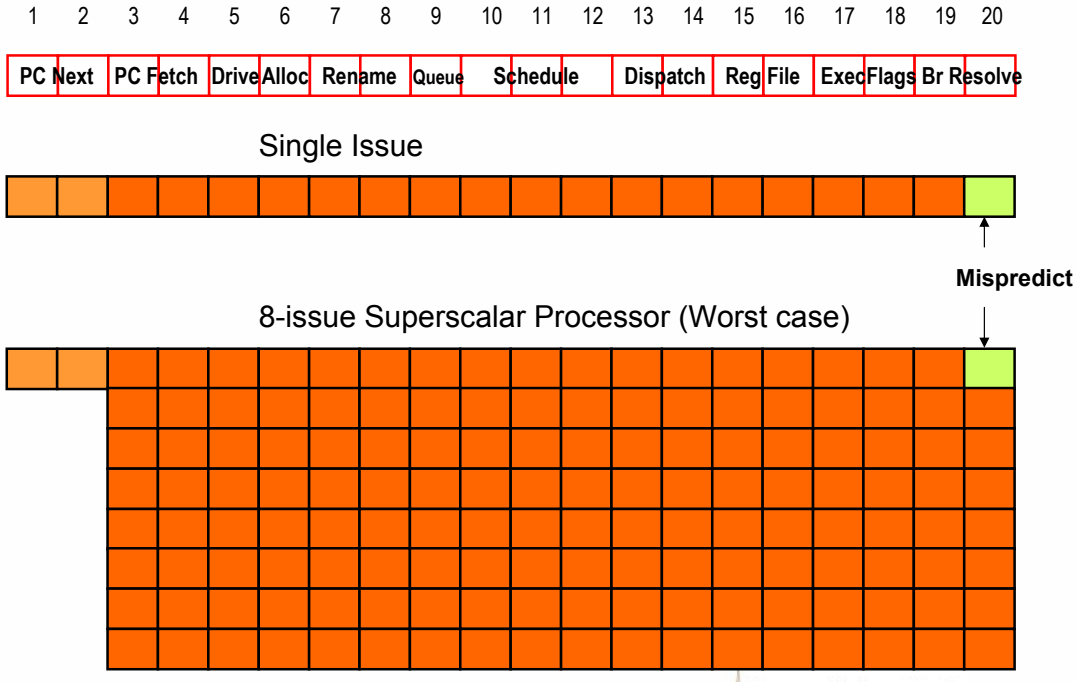
Single Issue (flush entailed instructions and refetch)

Mispredict

Branch Misprediction



Branch Misprediction



Why Branch is Predictable?

```
for (i=0; i<100; i++) {

        ….
}
```

```
addi   r10, r0, 100
addi   r1,   r0, r0

L1:
… …
… …
addi   r1, r1,   1
bne    r1, r10, L1
… …
```

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

```
addi  r2,   r0, 2
bne   r10, r2, L_bb
xor   r10, r10, r10
j     L_exit
L_bb:
bne   r11, r2, L_xx
xor   r11, r11, r11
j     L_exit
L_xx:
beq   r10, r11, L_exit
…
Lexit:
```

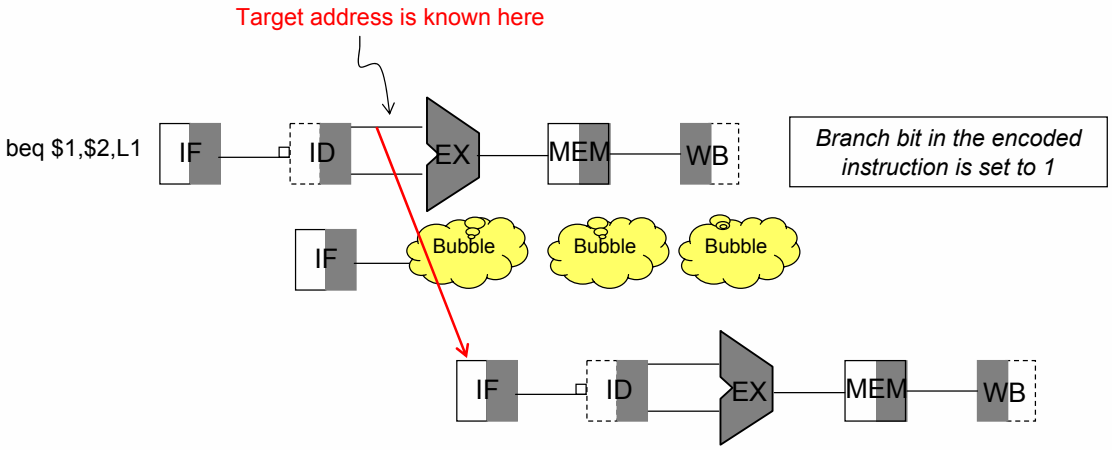

Control Speculation

- Execute instruction beyond a branch before the branch is resolved → Performance
- Speculative execution
  - Difference between speculation and prediction?
- What if mis-speculated? need
  - Recovery mechanism
  - Squash instructions on the incorrect path
- Branch prediction: Dynamic vs. Static
- What to predict?

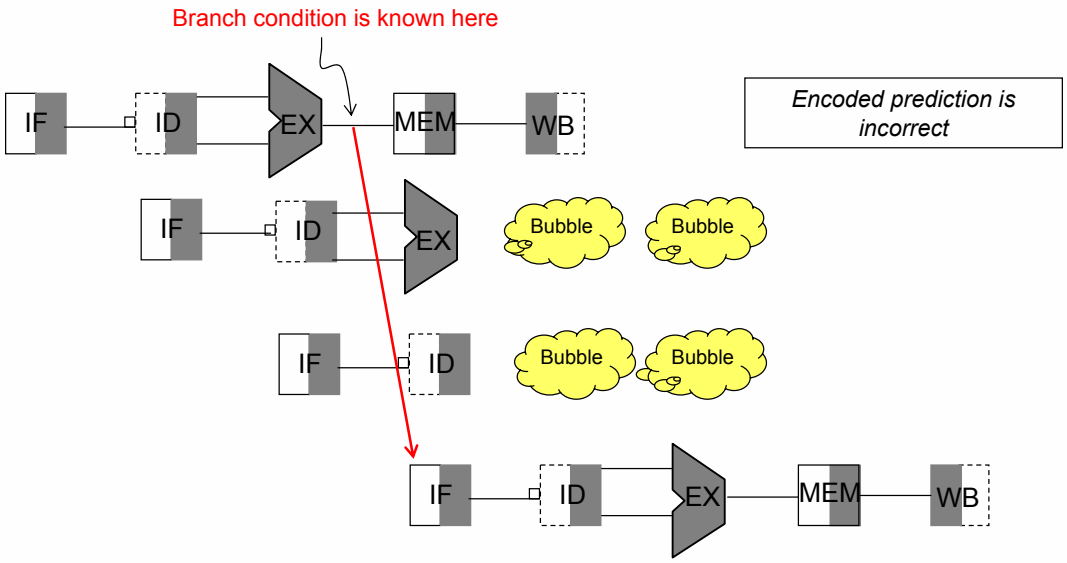
Static prediction

- Static prediction is used to guide code scheduling strategies
  - Simple strategy for all branches in the code
  - Based on opcode or direction of branches
  - Profile based →
    - individual branches tend to be strongly bimodal (set a bit in the opcode)
- Provide mechanisms for compilers or programmers to provide hints
  - Bit in the instruction encoding
  - I-fetch is steered accordingly

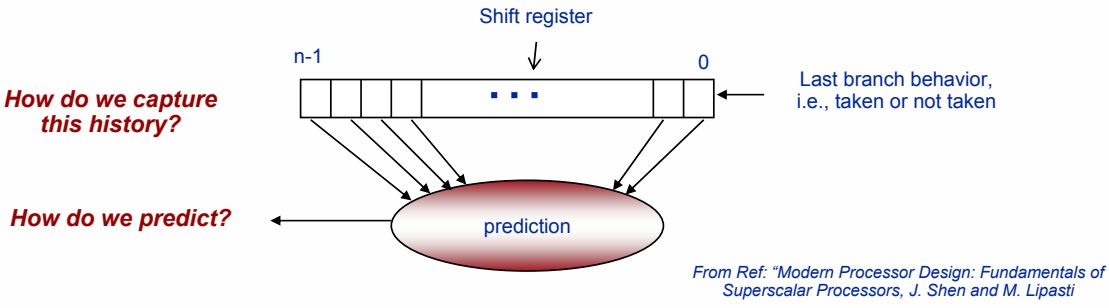
Profile Guided Static Prediction



Profile Guided Static Prediction



Dynamic Branch Prediction Strategies



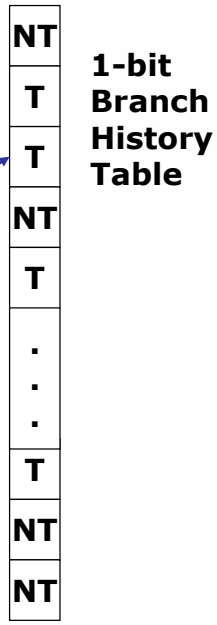
From Ref: "Modern Processor Design: Fundamentals of Superscalar Processors, J. Shen and M. Lipasti

- Use past behavior to predict the future
- Local vs. global behaviors
  - Branches show surprisingly good correlation with one another and their history
    - They are not totally random events

Simplest Dynamic Branch Predictor

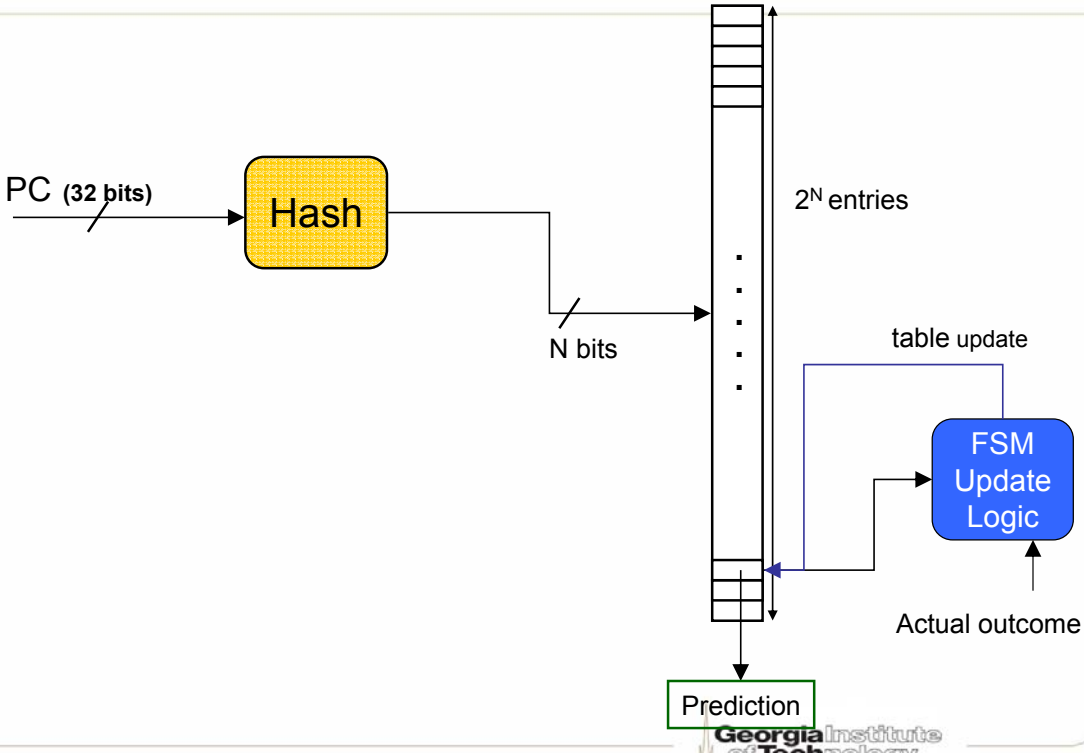
- Prediction based on latest outcome
- Index by some bits in the branch PC
  - Aliasing

```
                for (i=0; i<100; i++) {
                        ....
                }

0x40010100   addi   r10, r0, 100
0x40010104   addi   r1,   r1, r0

0x40010108   L1:
              ... ...
...           ... ...
0x40010A04   addi   r1, r1,   1
0x40010A08   bne    r1, r10, L1
              ... ...
```



How accurate?

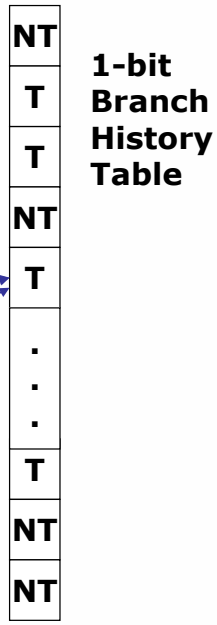
Typical Table Organization



Simplest Dynamic Branch Predictor

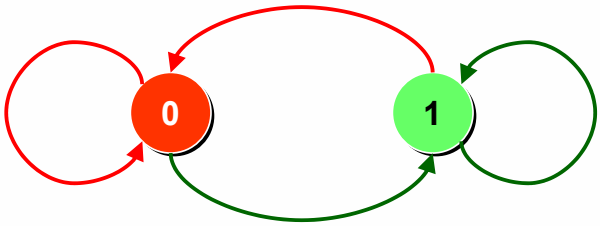
```
for (i=0; i<100; i++) {
  if (a[i] == 0) {
    …
  }
  …
}
```

```
0x40010100   addi  r10, r0, 100
0x40010104   addi  r1,  r1, r0
             L1:
0x40010108   add   r21, r20, r1
0x4001010c   lw    r2, (r21)
0x40010110   beq   r2, r0, L2
             … …
0x40010210   j     L3
             L2:
             … … …
             L3:
0x40010B0c   addi  r1, r1,  1
0x40010B10   bne   r1, r10, L1
```



FSM of the Simplest Predictor

- A 2-state machine
- Change mind fast



If branch taken

If branch not taken

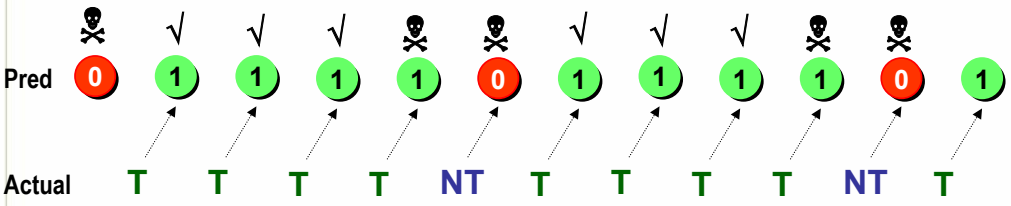
0 Predict not taken

1 Predict taken

Example using 1-bit branch history table

```
for (i=0; i<4; i++) {
        ….
}
```

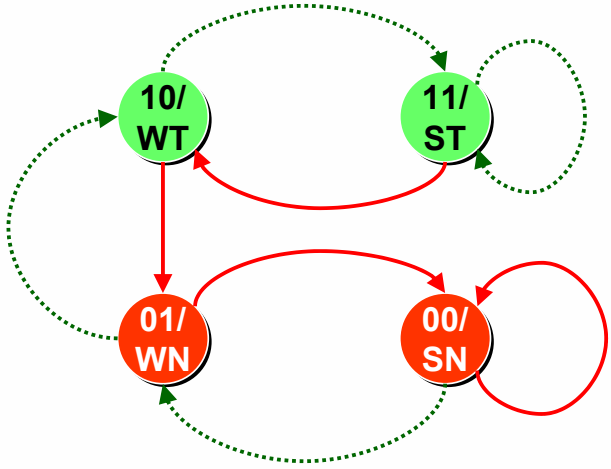
```
addi   r10, r0, 4
addi   r1,   r1, r0
L1:
… …
addi   r1, r1,   1
bne    r1, r10, L1
```



60% accuracy

2-bit Saturating Up/Down Counter Predictor

MSB: Direction bit
LSB: Hysteresis bit



Taken

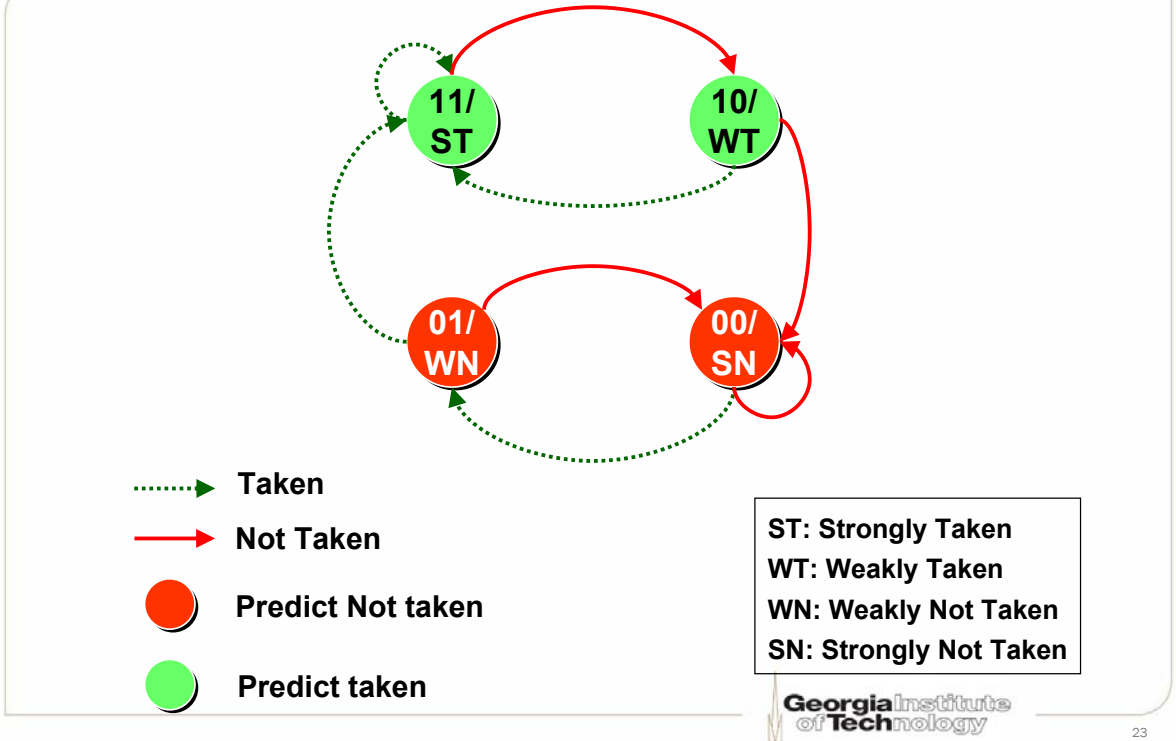
Not Taken

Predict Not taken

Predict taken

ST: Strongly Taken
WT: Weakly Taken
WN: Weakly Not Taken
SN: Strongly Not Taken

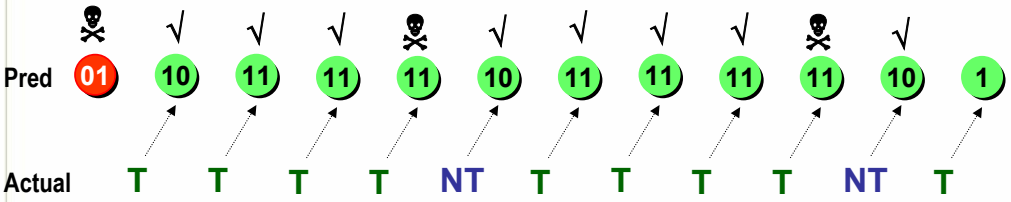
2-bit Counter Predictor (Another Scheme)



Example using 2-bit up/down counter

```
for (i=0; i<4; i++) {
        ....
}
```

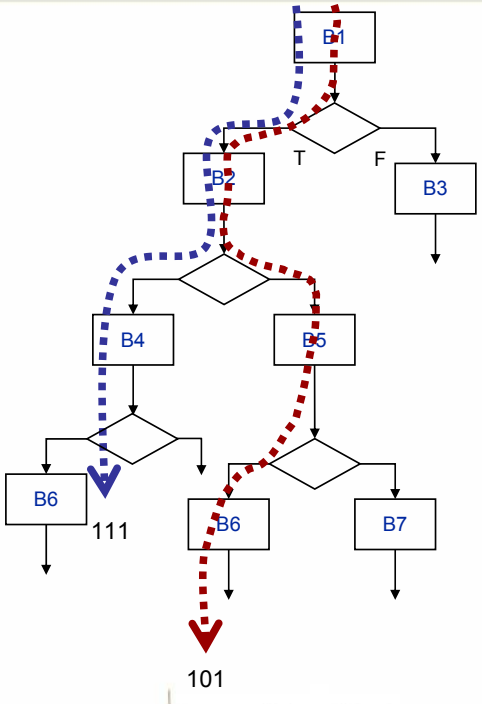
```
addi   r10, r0, 4
addi   r1,   r1, r0
L1:
... ...
addi   r1, r1,   1
bne    r1, r10, L1
```



80% accuracy

Capturing Global Behavior

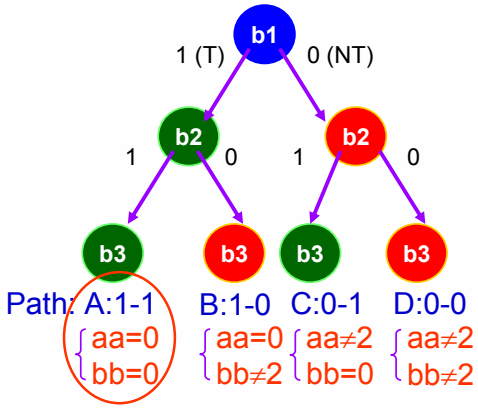
- A shift register captures the local path through the program
- For each unique path a predictor is maintained
- Prediction is based on the behavior history of each local path
- Shift register length determines program region size



Branch Correlation

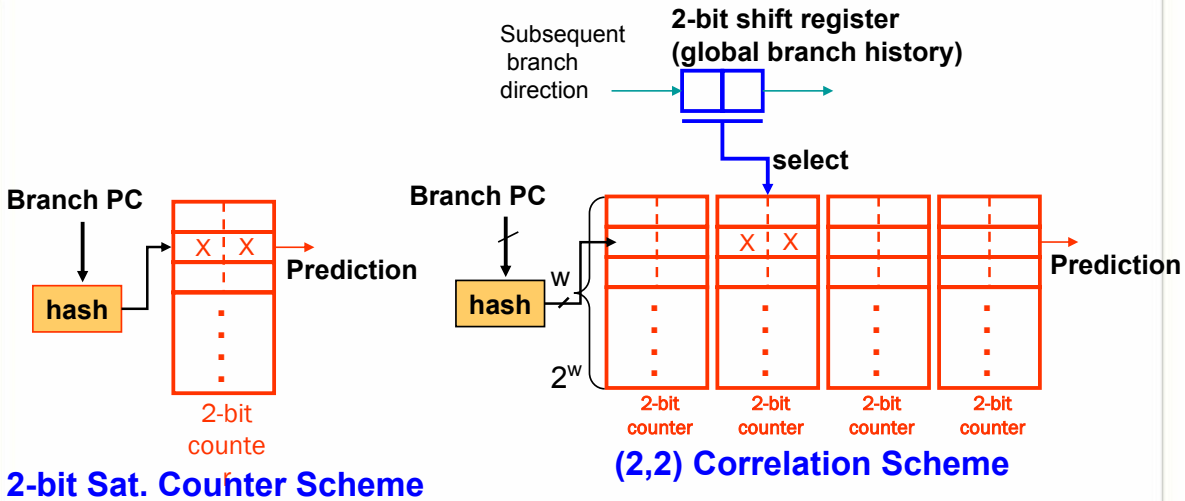
Code Snippet

```
if (aa==2)       // b1
       aa = 0;
if (bb==2)       // b2
       bb = 0;
if (aa!=bb) {    // b3
       ……
}
```



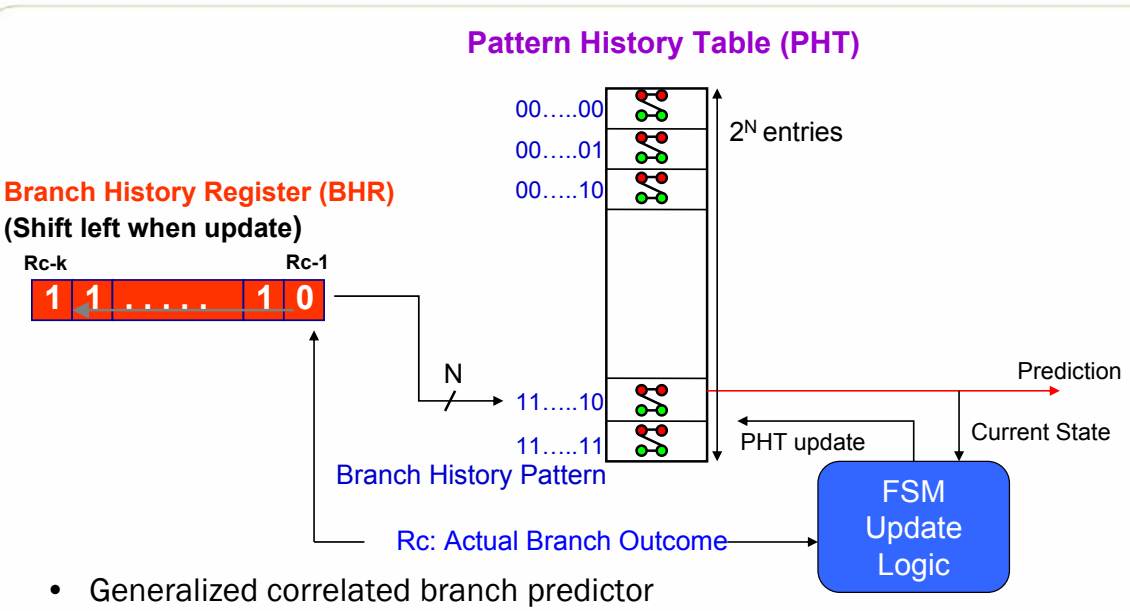
- Branch direction
  - Not independent
  - Correlated to the path taken
- Example: Path 1-1 of b3 can be surely known beforehand
- Track path using a 2-bit register

Correlated Branch Predictor [PanSoRahmeh'92]



- (M,N) correlation scheme
  - M: shift register size (# bits)
  - N: N-bit counter

Two-Level Branch Predictor [YehPatt91,92,93]



- Generalized correlated branch predictor
- 1st level keeps branch history in Branch History Register (BHR)
- 2nd level segregates pattern history in Pattern History Table (PHT)

Branch History Register

- An N-bit Shift Register = 2^N patterns in PHT
- Shift-in branch outcomes
  - 1 ⇒ taken
  - 0 ⇒ not taken
- First-in First-Out
- BHR can be
  - Global
  - Per-set
  - Local (Per-address)

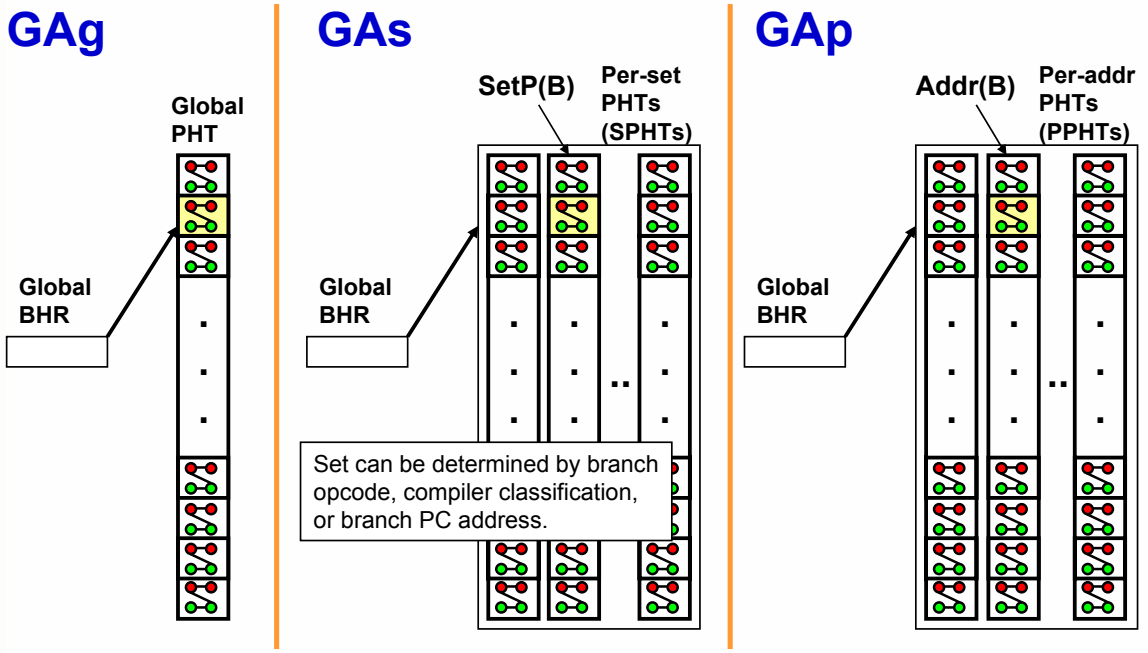
Pattern History Table

- 2^N entries addressed by N-bit BHR
- Each entry keeps a counter (2-bit or more) for prediction
  - Counter update: the same as 2-bit counter
  - Can be initialized in alternate patterns (01, 10, 01, 10, ..)
- Alias (or interference) problem

Key Idea

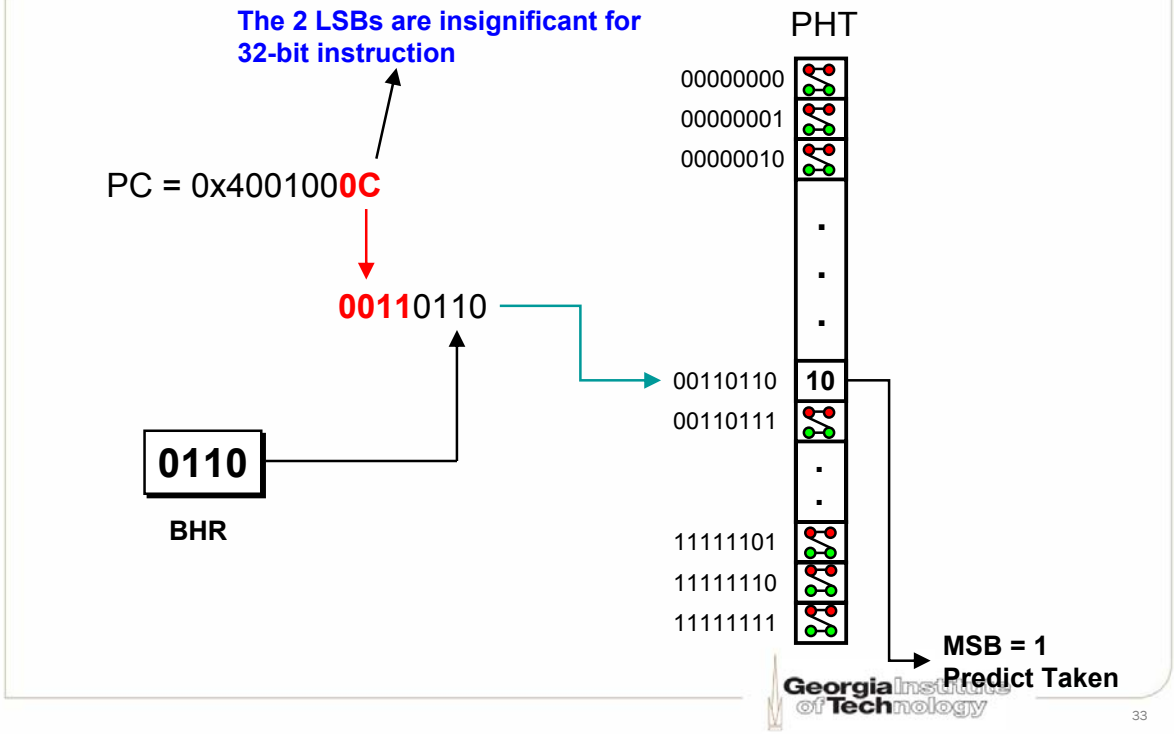
- Separate all of the histories of a branch → *sub-histories*
- For each sub-history employ a separate predictor
  - Each history maps to a FSM

Global History Schemes

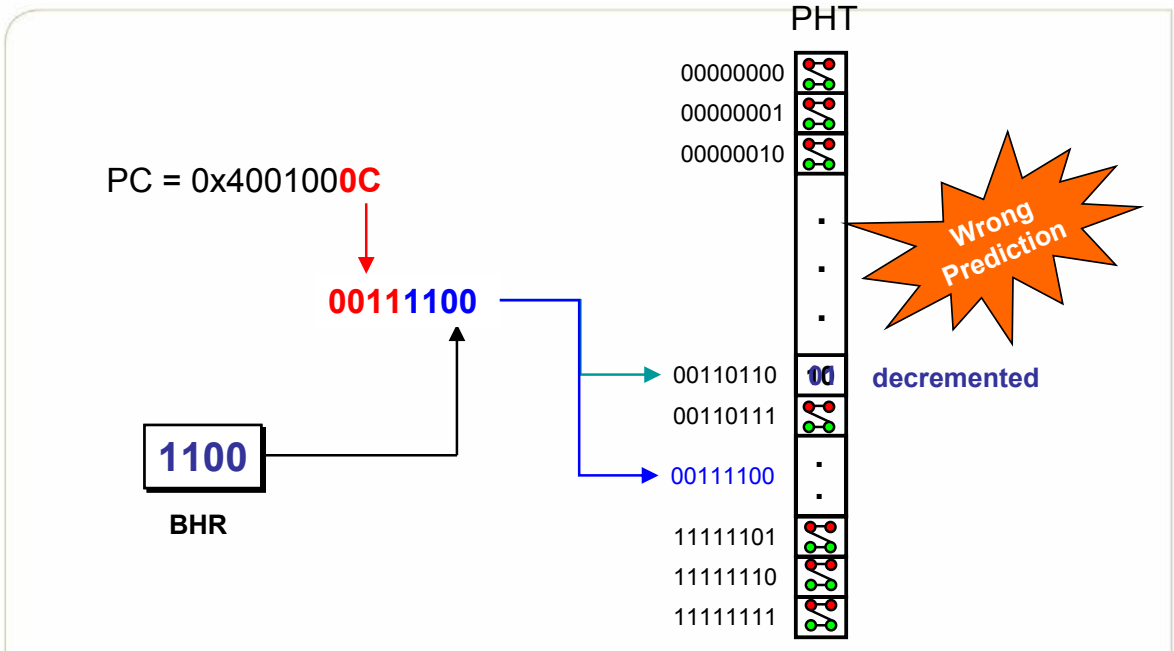


* [PanSoRahmeh'92] similar to **GAp**

GAs Two-Level Branch Prediction

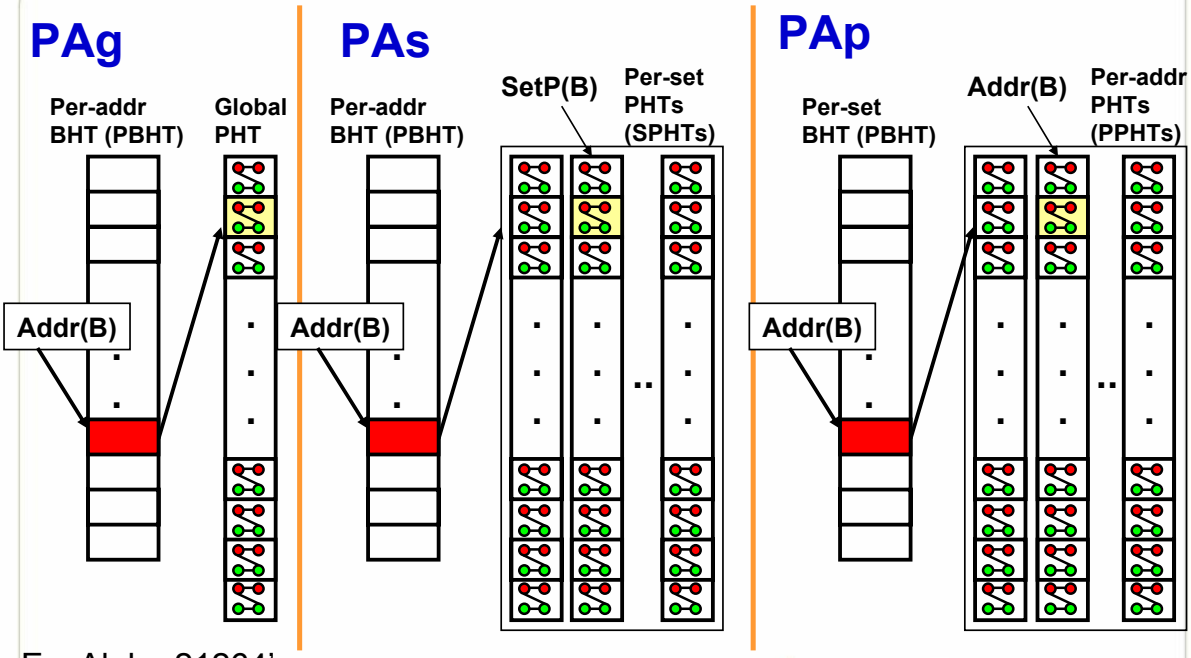


Predictor Update (Actually, Not Taken)



- Update Predictor after branch is resolved

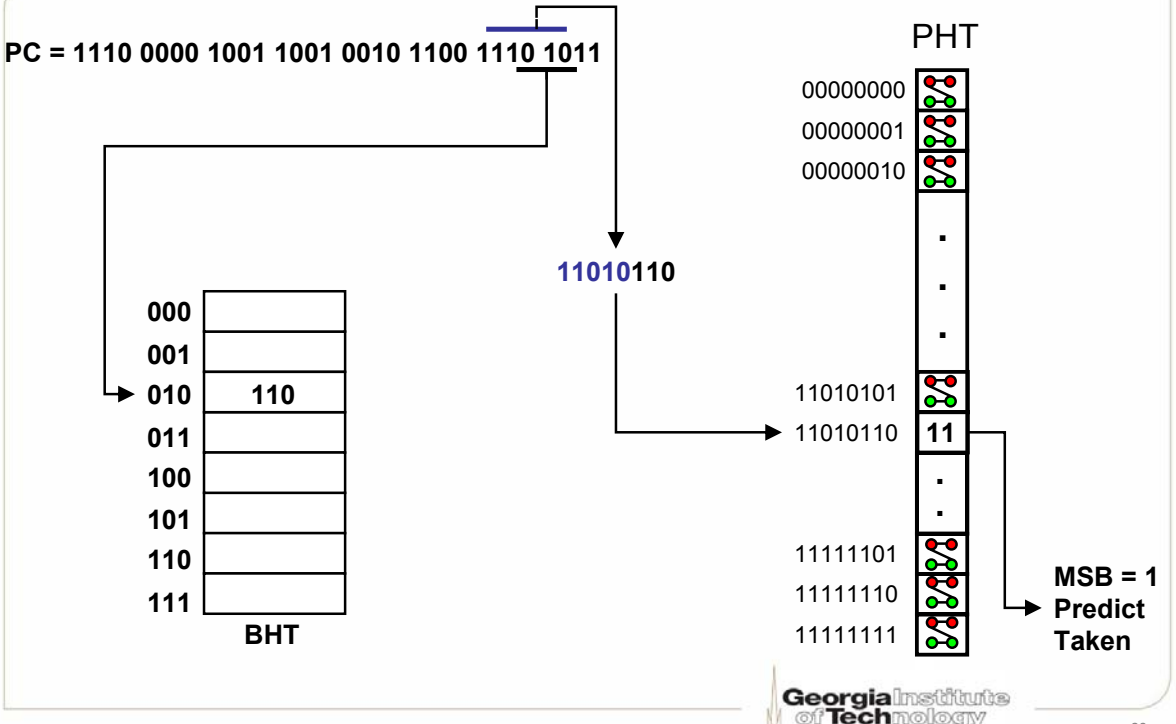
Per-Address History Schemes



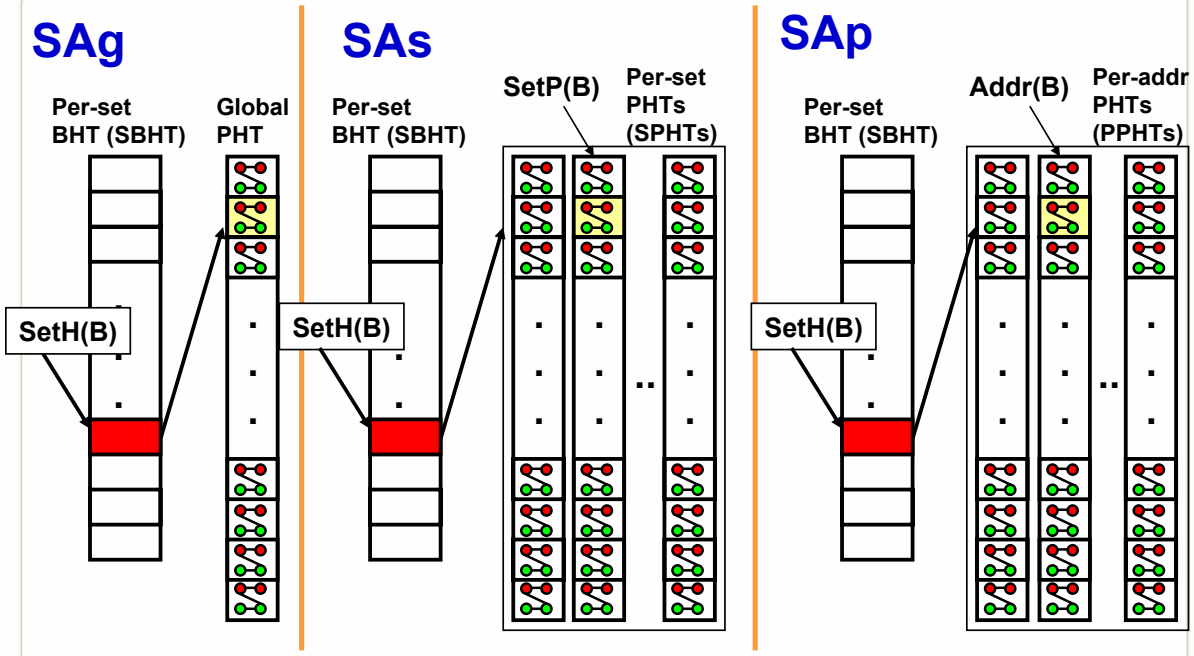
•Ex: Alpha 21264's local predictor

•Ex: P6, Itanium

PAs Two-Level Branch Predictor



Per-Set History Schemes



PHT Indexing

Branch addr	Global history	Gselect 4/4	
00000000	00000001	00000001	
00000000	00000000	00000000	
11111111	00000000	11110000	
11111111	10000000	11110000	

Insufficient History

- Tradeoff between more history bits and address bits
- Too many bits needed in Gselect ⇒ sparse table entries

Gshare Branch Predictor [McFarling93]

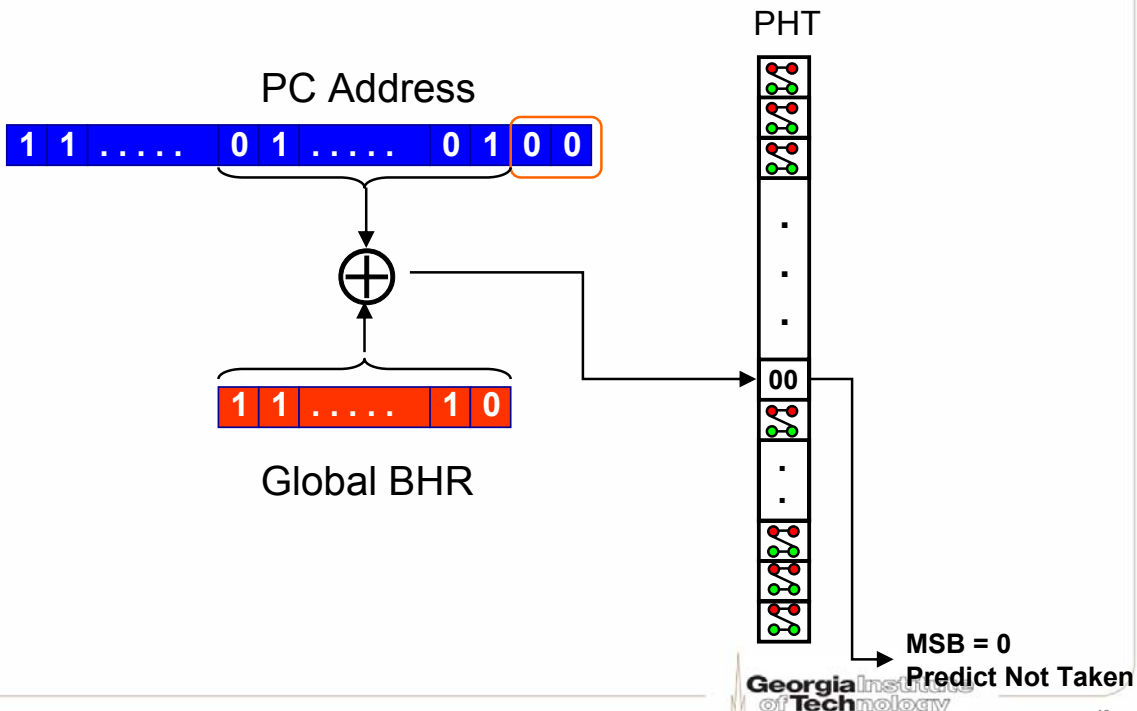
Branch addr	Global history	Gselect 4/4	Gshare 8/8
00000000	00000001	00000001	00000001
00000000	00000000	00000000	00000000
11111111	00000000	11110000	11111111
11111111	10000000	11110000	01111111

Gselect 4/4: Index PHT by concatenate low order 4 bits

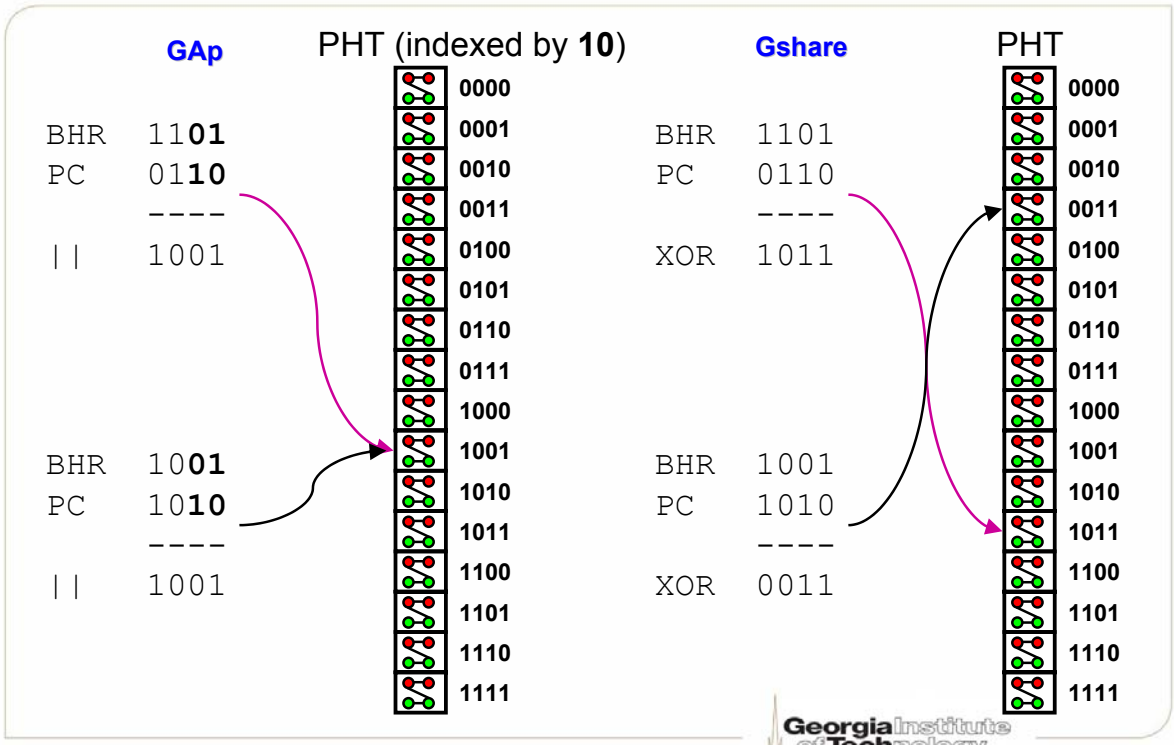
Gshare 8/8: Index PHT by {Branch address ⊕ Global history}

- Tradeoff between more history bits and address bits
- Too many bits needed in Gselect ⇒ sparse table entries
- Gshare ⇒ Not to lose global history bits
- Ex: AMD Athlon, MIPS R12000, Sun MAJC, Broadcom SiByte's SB-1

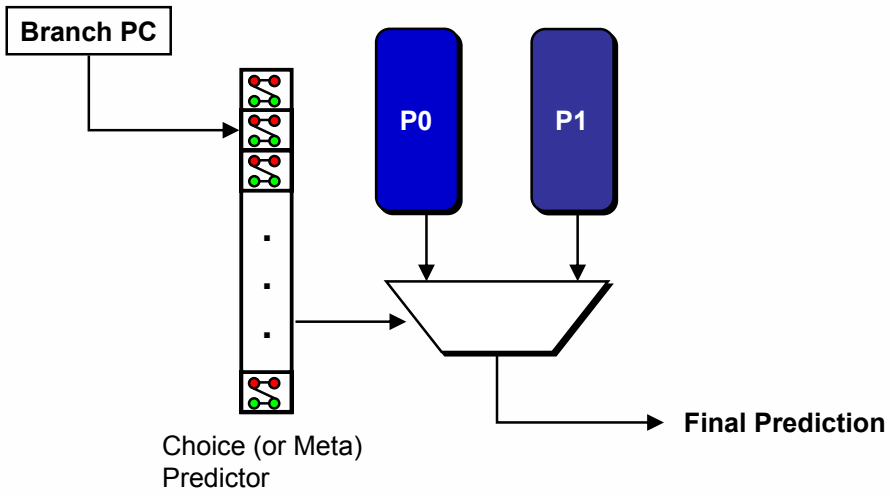
Gshare Branch Predictor



Aliasing Example



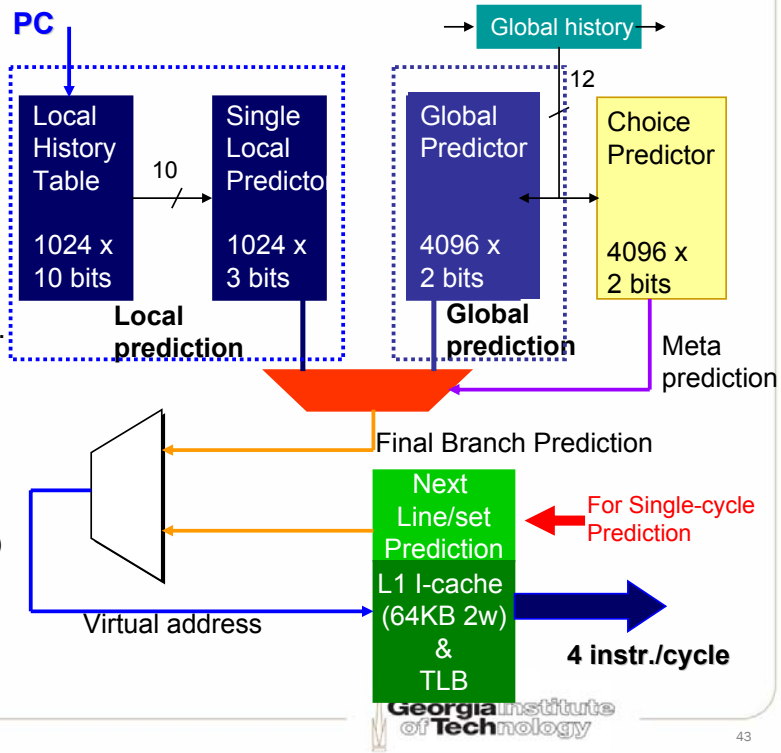
Hybrid Branch Predictor [McFarling93]



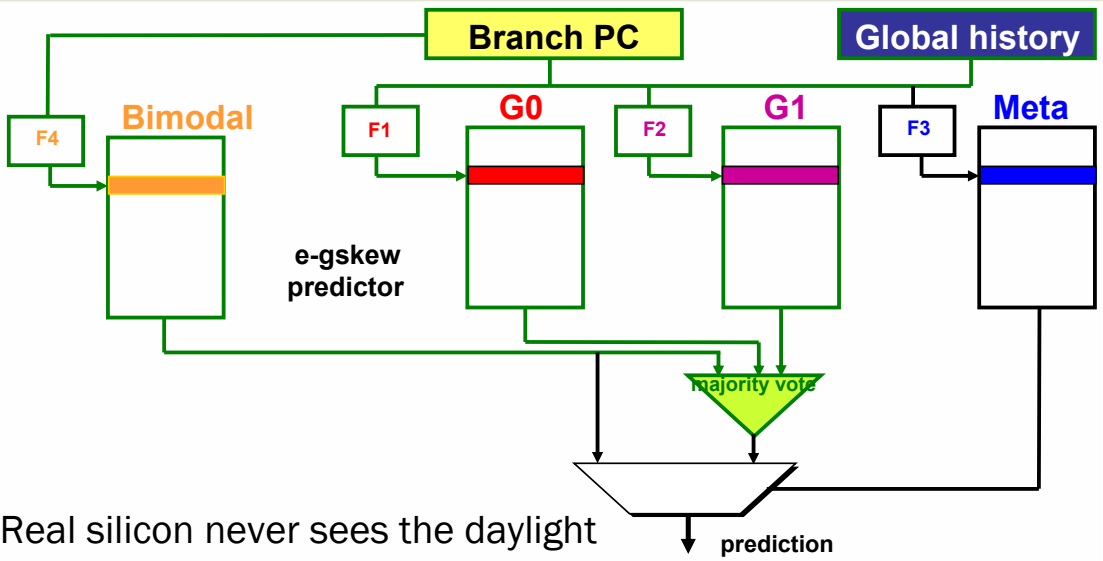
- Some branches correlated to global history, some correlated to local history
- Only update the meta-predictor when 2 predictors disagree

Alpha 21264 (EV6) Hybrid Predictor

- A "**tournament branch predictor**"
- Multi-predictor scheme w/
  - **Local predictor** (~PAg)
    - Self-correlation
  - **Global predictor**
    - Inter-correlation
  - **Choice predictor** as the **decision maker**: a 2-bit sat. counter to credit either local or global predictors.
- Die size impact
  - History info tables ~2%
  - BTB ~ 2.7% (associated with I-$ on a per-line basis)
- 2 cycle latency, we will discuss more later



Alpha EV8 Branch Predictor



- Real silicon never sees the daylight
- Use a 2Bc-gskew predictor (one form of enhanced gskew)
  - Bimodal predictor used as (1) static biased predictor and (2) part of e-gskew predictor
  - Global predictors G0 and G1 are part of e-gskew predictor
  - Table sizes: 352Kbits in total (208Kbits for prediction table; 144Kbits for hysteresis table.)

Branch Target Prediction

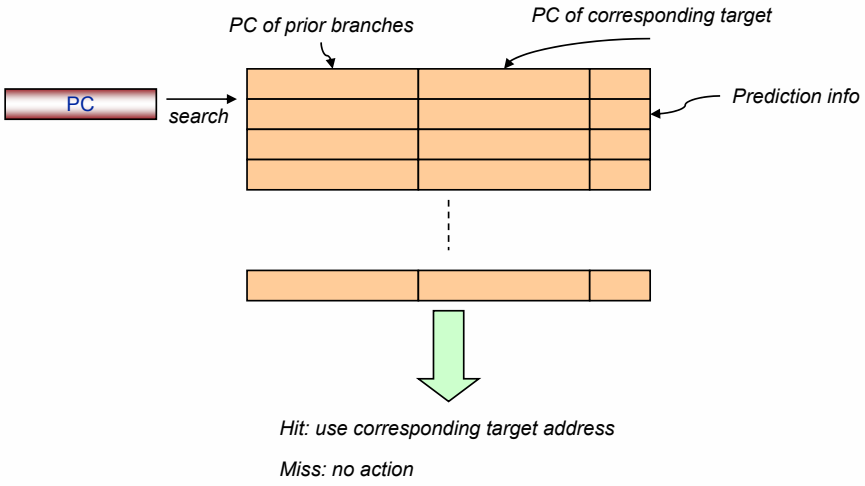
- Try the easy ones first
  - Direct jumps
  - Call/Return
  - Conditional branch (bi-directional)
- Branch Target Buffer (BTB)
- Return Address Stack (RAS)

Branch Target Buffer

A cache that contains three pieces of information:

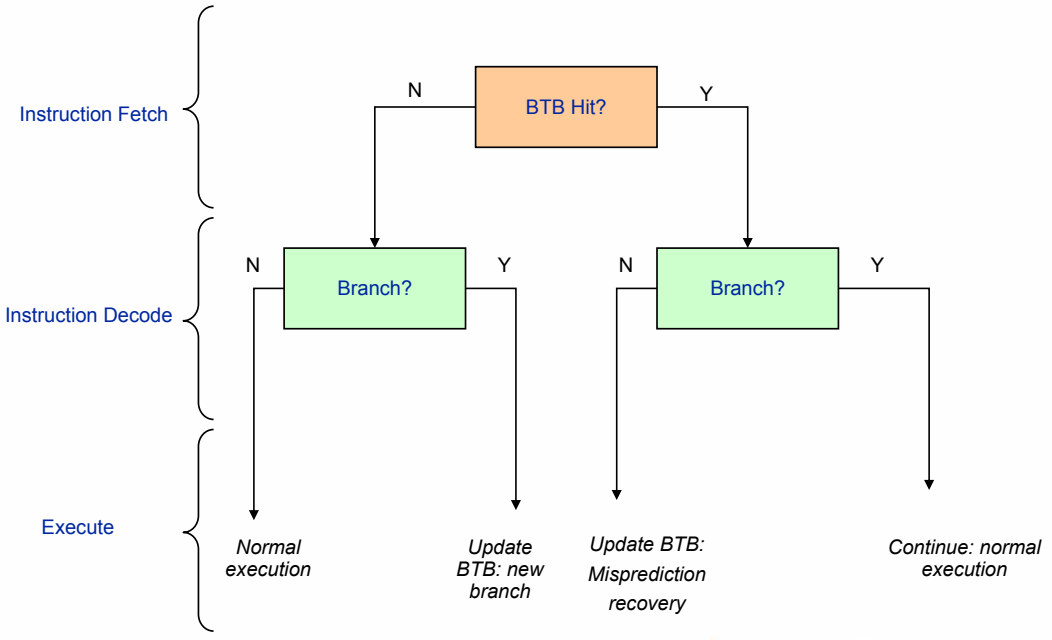
- The address of branch instructions
  - The BTB is managed like a cache and the addresses of branch instructions are kept for lookup purpose
- Branch target address
  - To avoid re-computation of branch target address where possible
- Prediction statistics
  - Different strategies are possible to maintain this portion of the BTB

Branch Target Buffers



- Access in parallel with instruction cache
  - Hit produces the branch target address

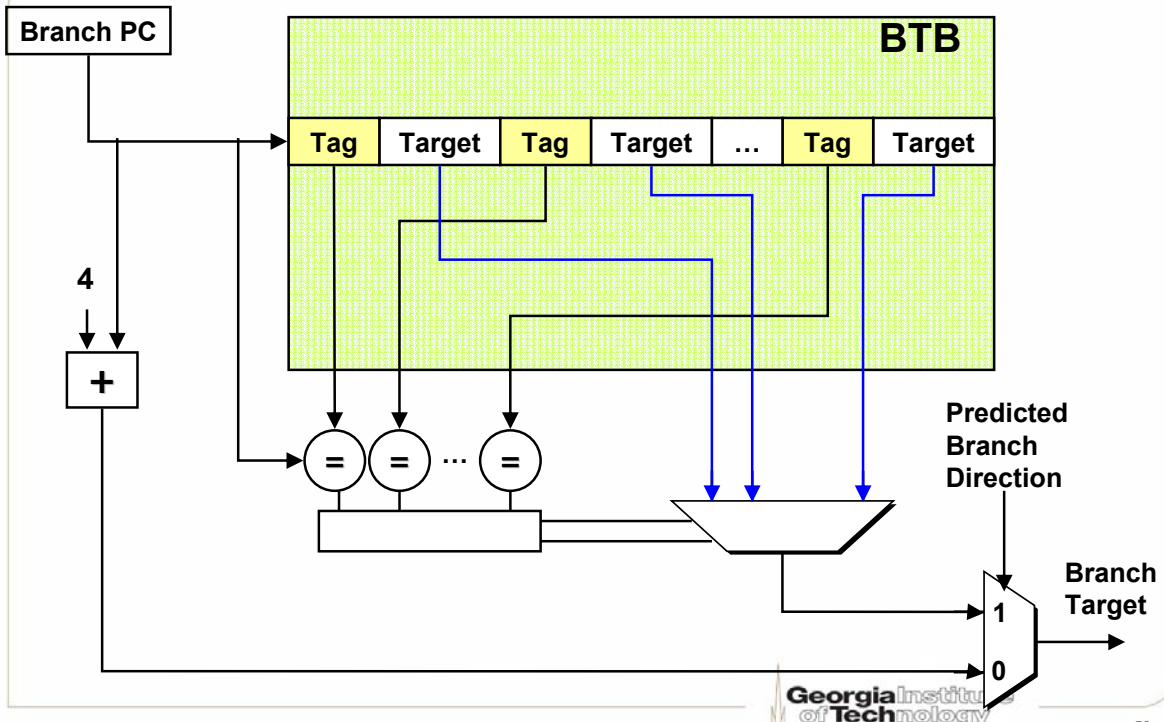
Branch Target Buffer Operation



Branch Target Buffers: Operation

- Couple speculative generation of the branch target address with branch prediction
  - Continue to fetch and resolve branch condition
    - Take appropriate action if wrong
- Any of the preceding history based techniques can be used for branch condition speculation
- Store prediction information, e.g., *n*-bit predictors, along with BTB entry
- Branch folding optimization: store target instruction rather than target address

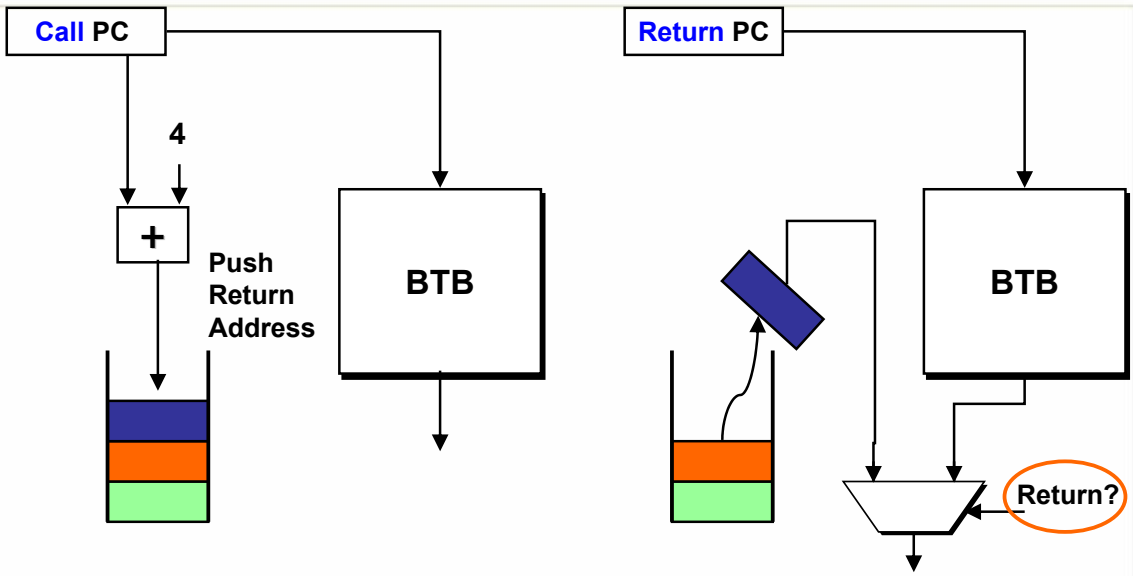
Branch Target Buffer (BTB)



Return Address Stack (RAS)

- Different call sites make return address hard to predict
  - Printf() being called by many callers
  - The target of “return” instruction in printf() is a moving target
- A hardware stack (LIFO)
  - Call will push return address on the stack
  - Return uses the prediction off of TOS

Return Address Stack

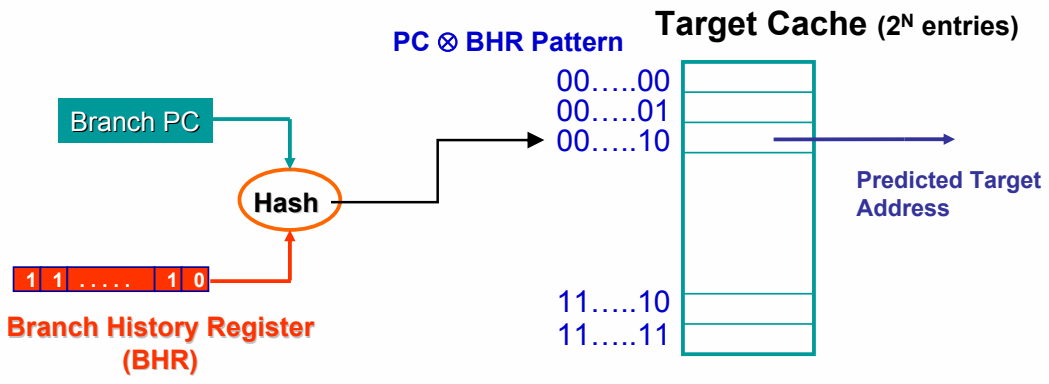


- Does it always work?
  - Call depth
  - Setjmp/Longjmp
  - Speculative call?
- May not know it is a return instruction prior to decoding
  - Rely on BTB for speculation
  - Fix once recognize Return

Indirect Jump

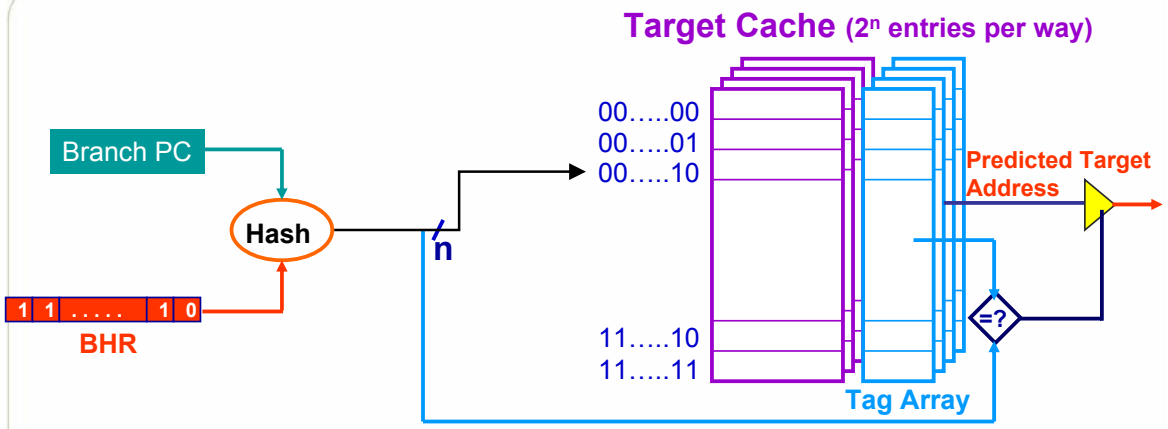
- Need Target Prediction
  - Many (potentially 2^{30} for 32-bit machine)
  - In reality, not so many
  - Similar to predicting values
  - Think about case statements
    - Is the target influenced by history?
- Tagless Target Prediction
- Tagged Target Prediction

Tagless Target Prediction [ChangHaoPatt'97]



- Modify the PHT to be a "Target Cache"
  - (indirect jump) ? (from target cache) : (from BTB)
- Alias?
  - Multiple targets for the same jump?
  - How does this improve accuracy over the BTB?

Tagged Target Prediction [ChangHaoPatt'97]



- To reduce aliasing with set-associative target cache
- Use branch PC and/or history for tags

Multiple Branch Prediction

- For a really wide machine
  - Across several basic blocks
  - Need to predict multiple branches per cycle
- How to fetch non-contiguous instructions in one cycle

Study Guide: Glossary

- Bimodal predictor
- Branch correlation and correlated predictors
- Branch direction
- Branch history table
- Branch history register
- Branch misprediction
- Branch prediction
- Branch target
- Branch target buffer
- Control dependency
- Dynamic branch prediction
- Global vs. local predictors
- Global history schemes: GAg, GAs, GAp
- gshare and gselect predictors
- Histories of a branch
- Hybrid branch predictors
- Multiple branch prediction
- Multi-level predictor
- N-bit counter predictors
- Pattern history table
- Per Address History Schemes: PAg, PAs, PAp
- Per Set History Schemes: SAg, SAs, SAp
- Profile guided branch prediction
- Return address stack
- Static branch prediction
- Tagless and tagged target prediction
- Two level branch predictor