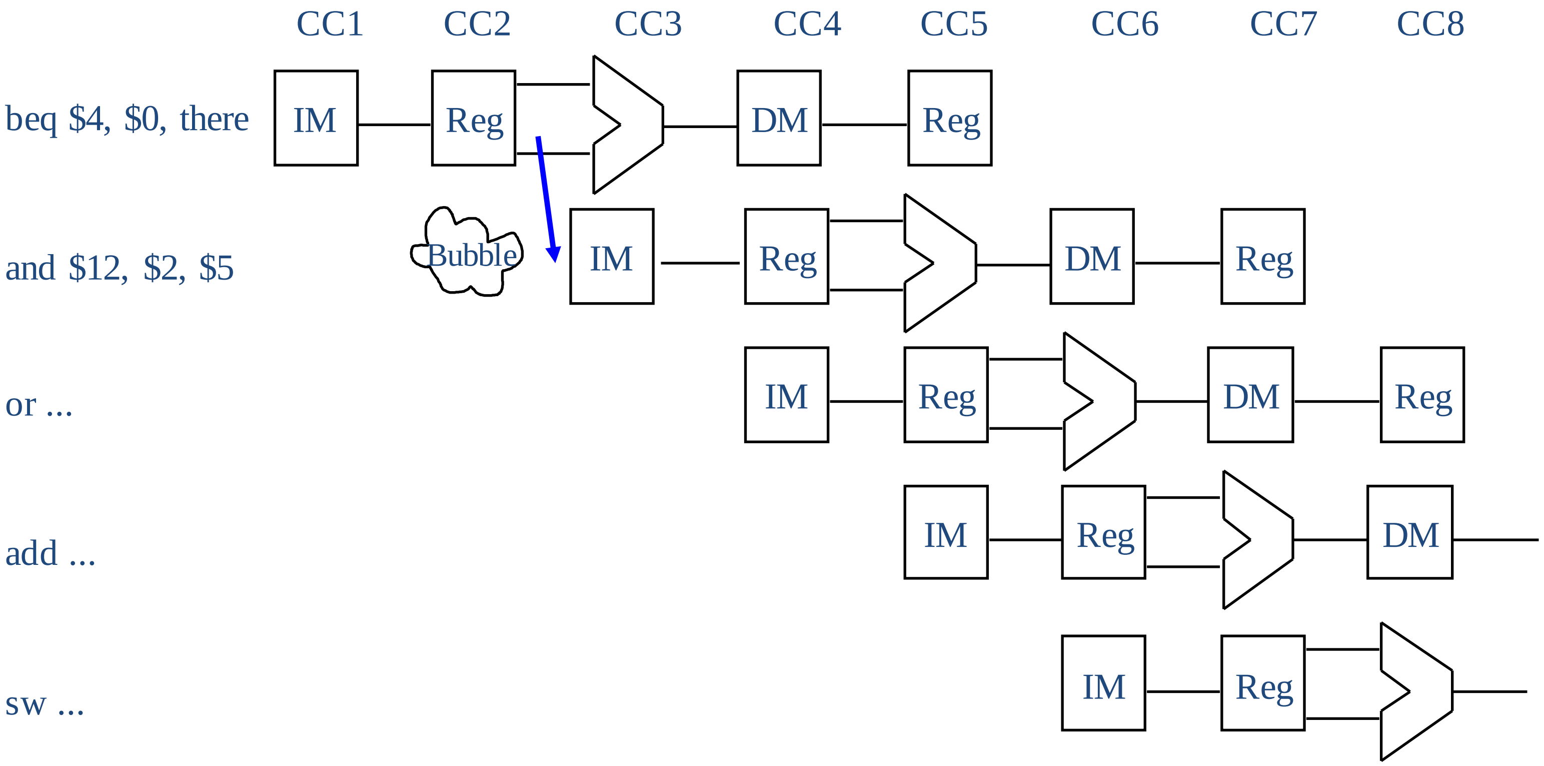
# Pipelining

# Stalling for Branch Hazards



CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8

beq $4, $0, there

and $12, $2, $5

or ...

add ...

sw ...

# Data Hazards for Branches

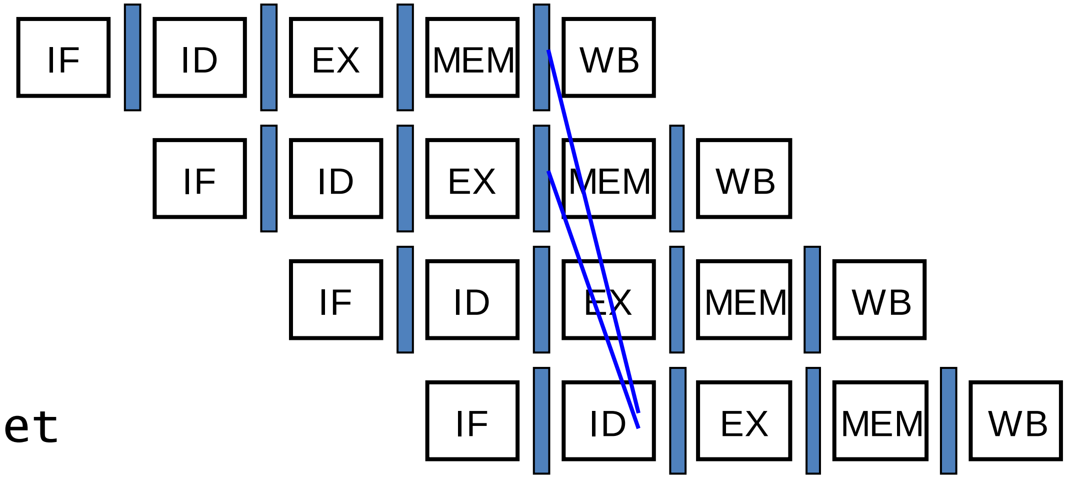
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add $1, $2, $3

add $4, $5, $6

…

beq $1, $4, target
```

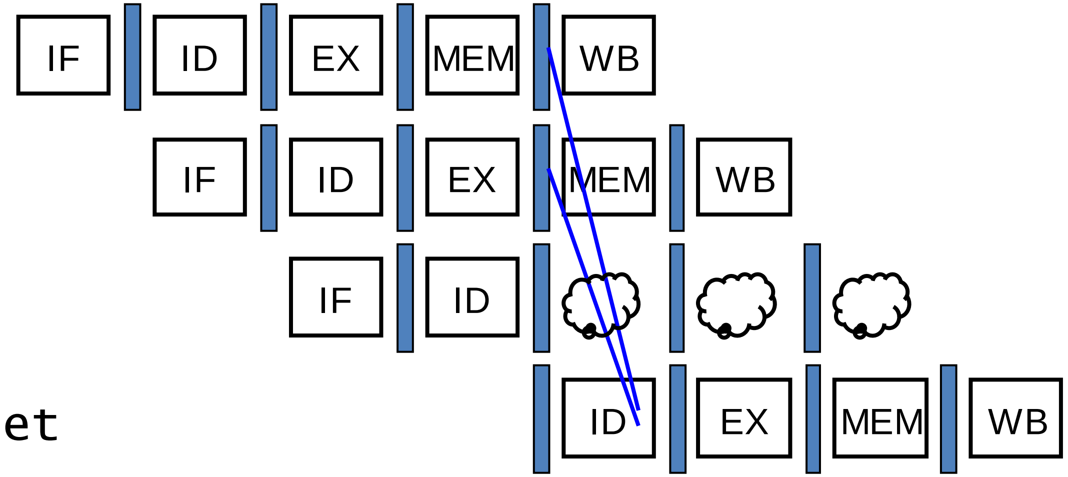


- Can resolve using forwarding

# Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

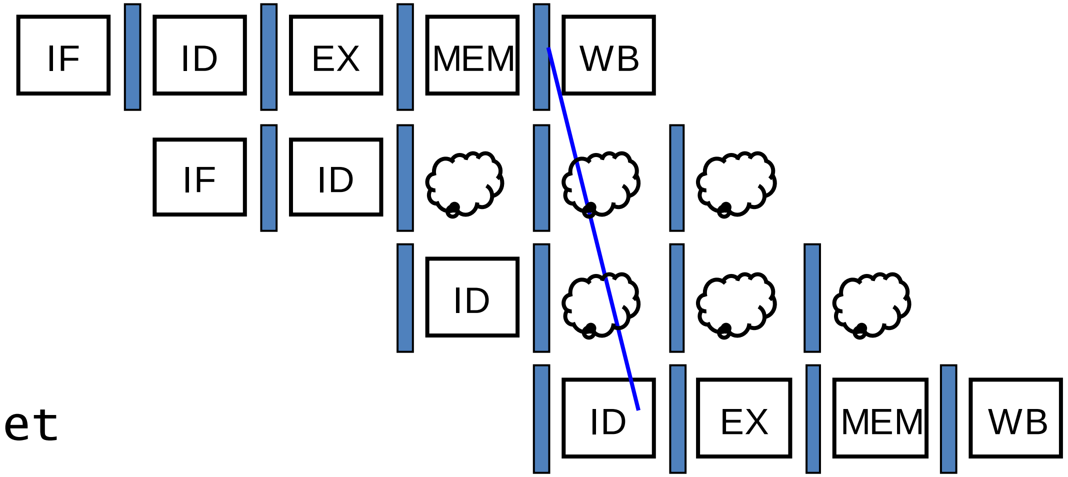
```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



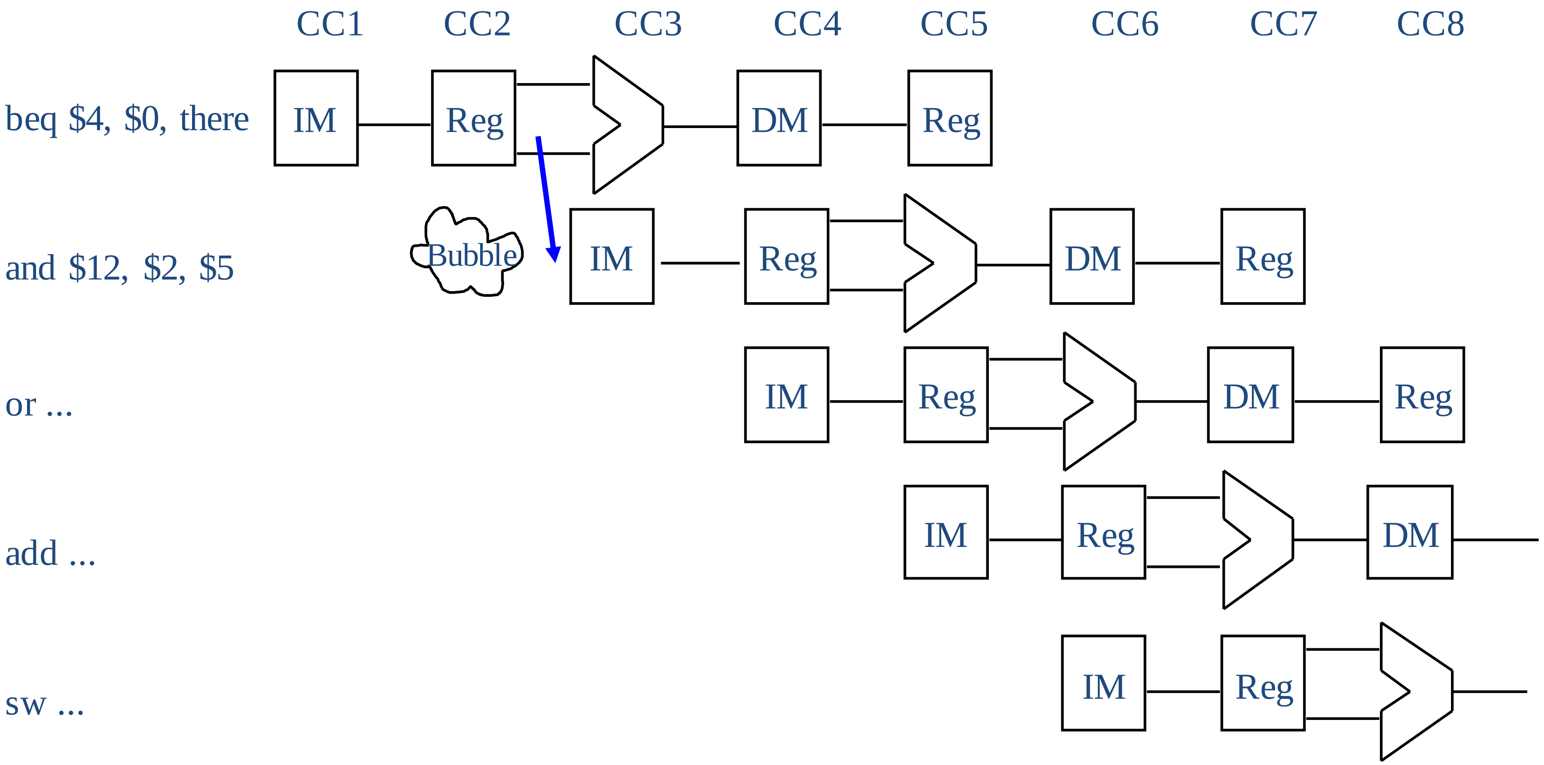
# Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



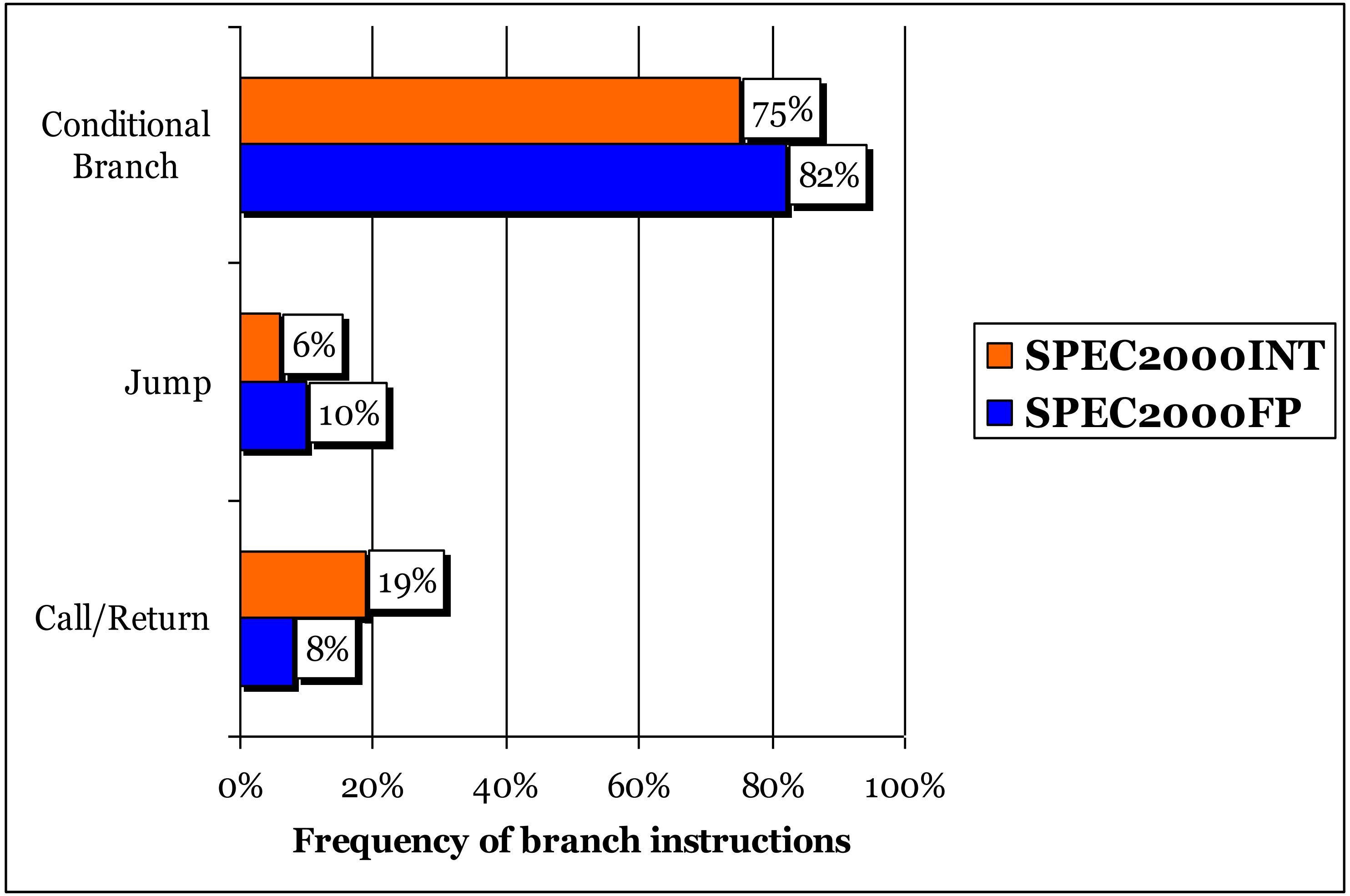
# Stalling for Branch Hazards



# Types of Branches

| | Conditional | Unconditional |
|---|---|---|
| Direct | if - then- else<br>for loops<br>(bez, bnez, etc) | procedure calls (jal)<br>goto (j) |
| Indirect | | return (jr)<br>virtual function lookup<br>function pointers (jalr) |

# Categorizing Branches



Source: H&P using Alpha

# Branch Hazard Resolutions

#1: stall until branch direction is clear (☹)

#2: static branch prediction

- predict branch Not Taken (fall through, as shown in previous slide)
    - execute successor instructions in sequence
    - “squash” instructions in pipeline if branch actually taken
    - PC+4 already calculated, so use it to get next instruction
- predict branch Taken
    - but haven’t calculated branch target address
    - might incur branch penalty

#3: dynamic branch prediction

- will talk about it later today

# What happens when a branch is predicted?

- On mispredict:
  - No speculative state may commit
    - Squash instructions in the pipeline
    - Must not allow stores in the pipeline to occur
      - Cannot allow stores which would not have happened to commit
    - Need to handle exceptions appropriately

# Alternative Branch Hazard Resolutions

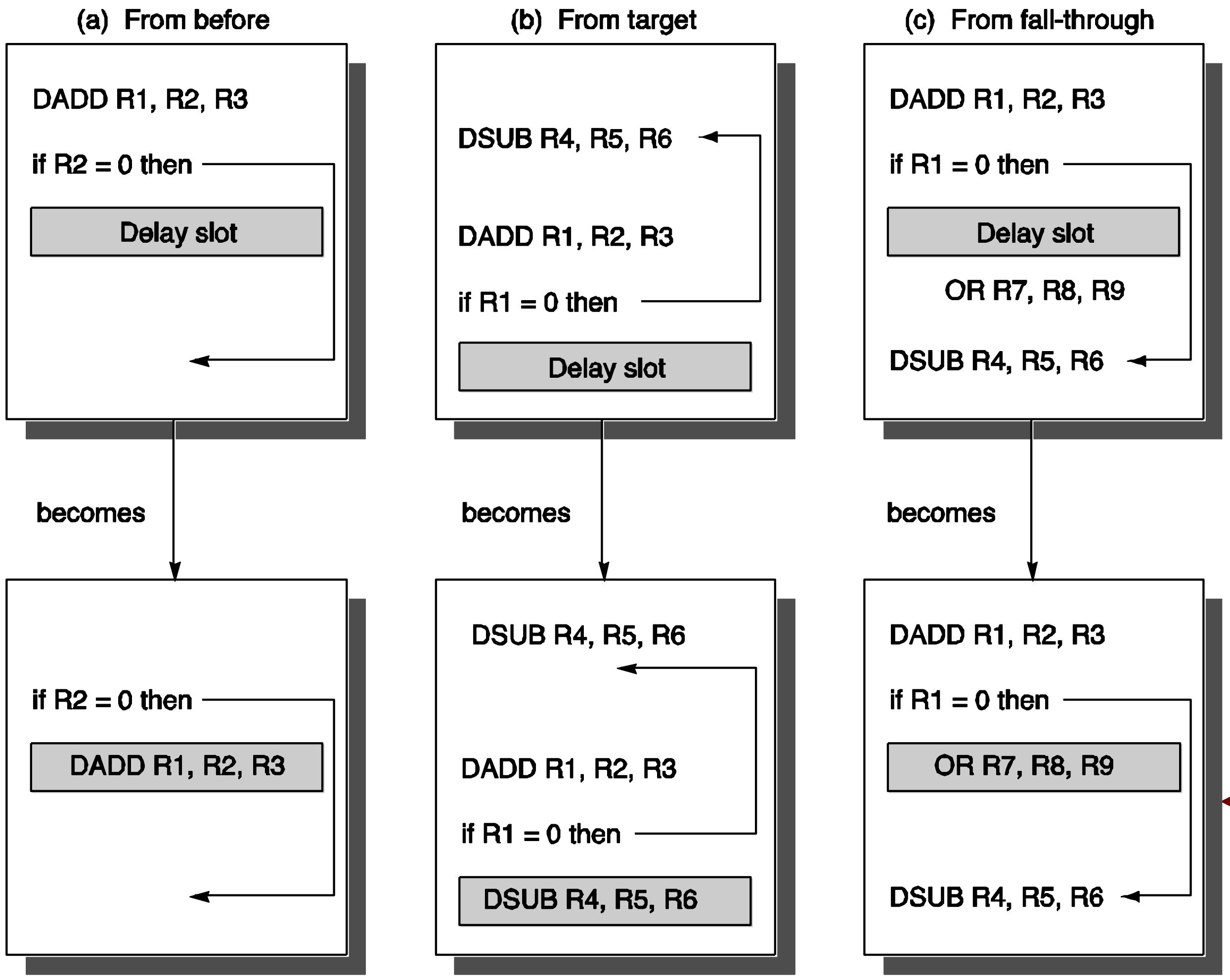
#4 delayed branch

- delay branch to take place after a following instruction

  branch instruction<br>
  sequential successor$_1$<br>
  sequential successor$_2$<br>
  ........<br>
  sequential successor$_n$<br>
  branch target if taken

- 1 slot delay allows proper decision and will get branch target address (next page)

# Filling Branch Delay Slot



© 2003 Elsevier Science (USA). All rights reserved.

- This is a compiler optimization.
- The instruction in the delay slot will always be executed.

- From before: the instruction will be executed anyway
- From target: will get benefit if the branch is taken
- From fall-through: will get benefit if the branch is not taken

make sure R7 will not be used in taken path before redefined

- If compiler cannot find a useful instruction, there is nothing we can do, we can just insert a nop instruction

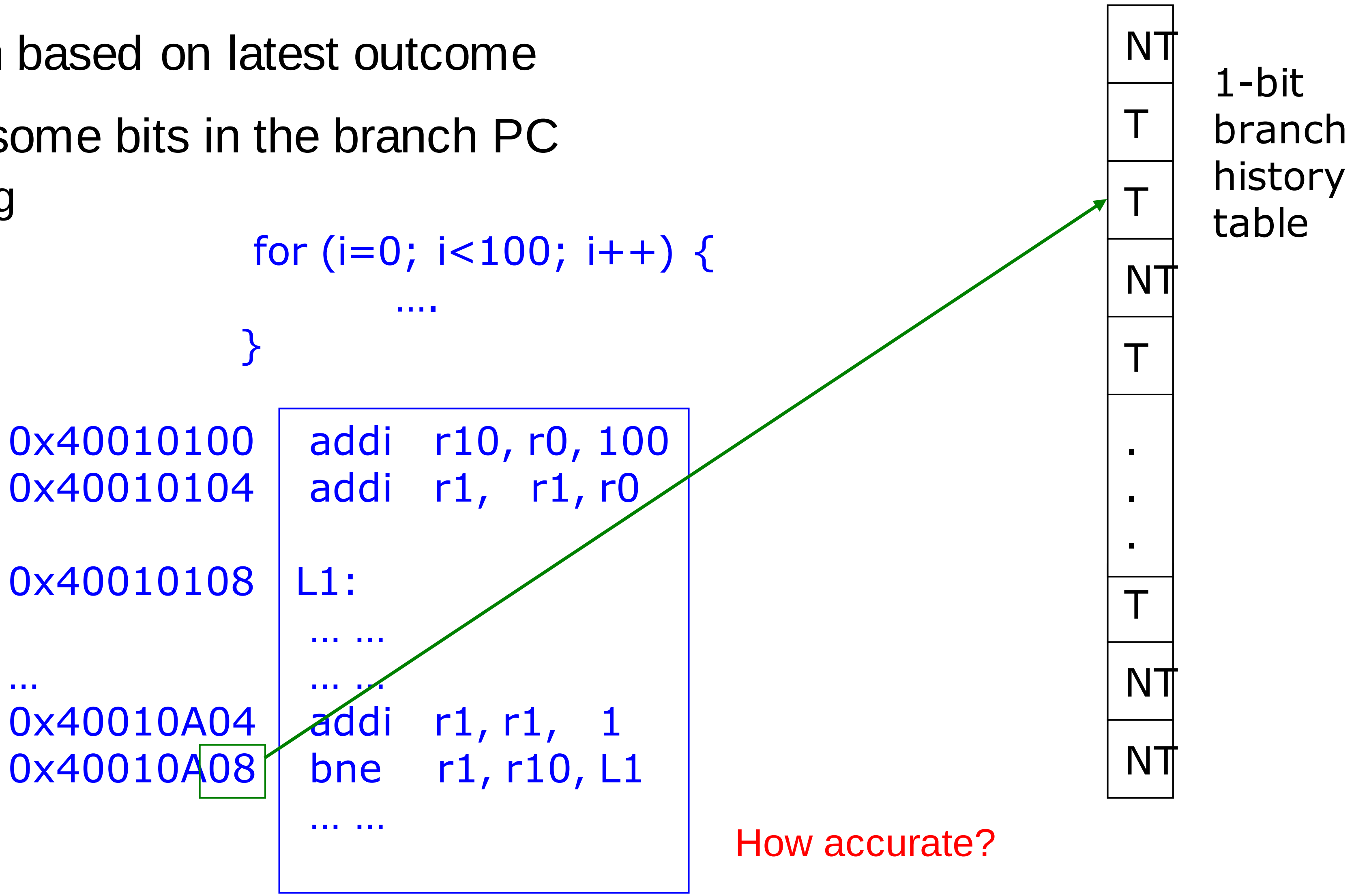
# Will Prediction Work and Predict What?

- Direction (1-bit)
  - single direction for unconditional jumps and calls/returns
  - binary for conditional branches
- Target (32-bit or 64-bit addresses)
  - one
    - Uni-directional jumps
  - two
    - fall through (not Taken) vs. taken
  - many:
    - function pointer or indirect jump (e.g., jr r31)

# Simplest Dynamic Branch Predictor

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

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

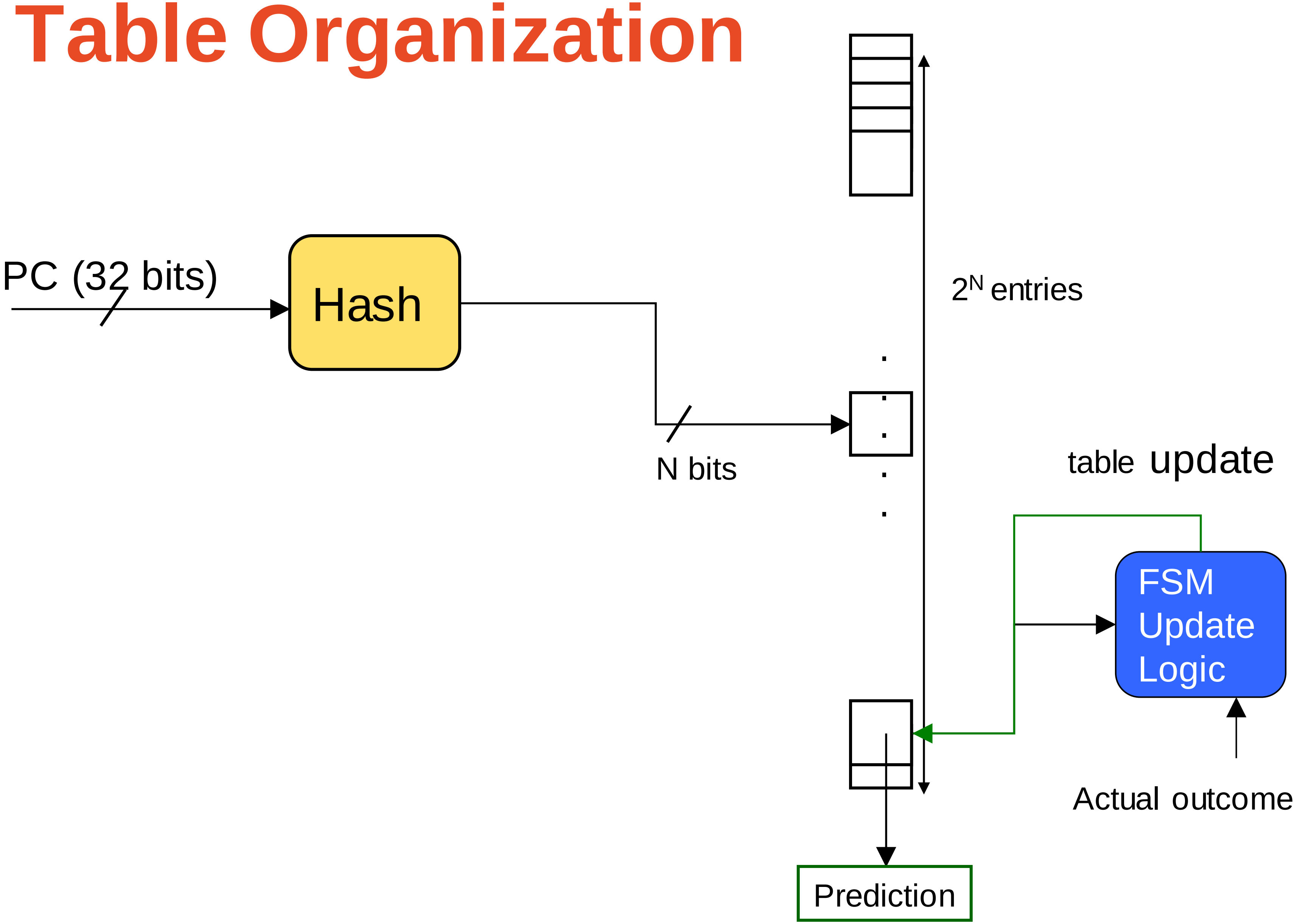


```
0x40010100   addi  r10, r0, 100
0x40010104   addi  r1,  r1, r0

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

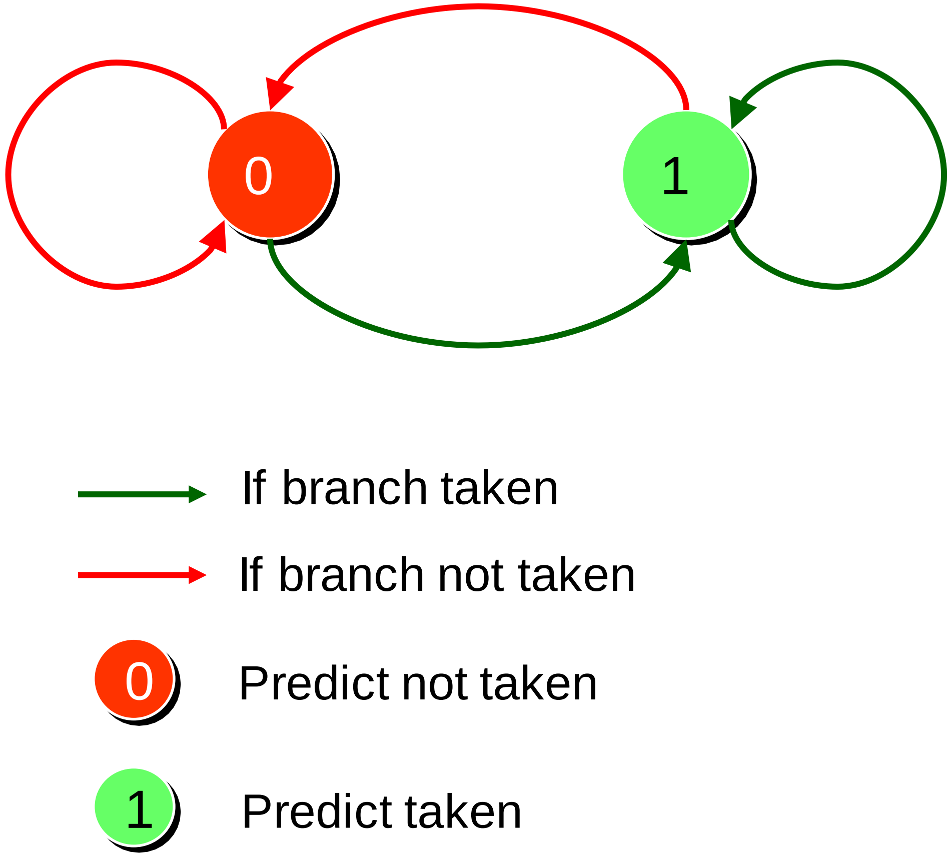
How accurate?

# Typical Table Organization



# FSM of the Simplest Predictor

- A 2-state machine
- Change mind fast



→ If branch taken

→ If branch not taken

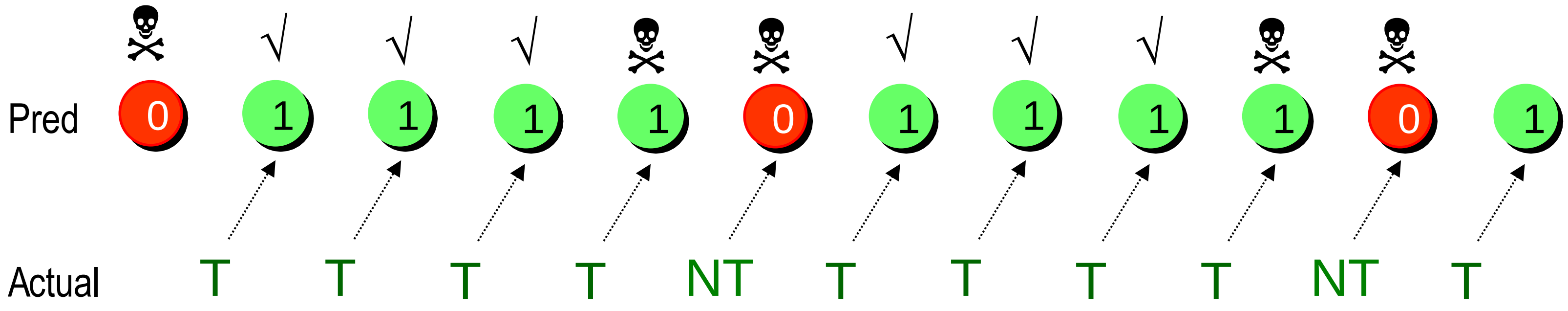
0 Predict not taken

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
```

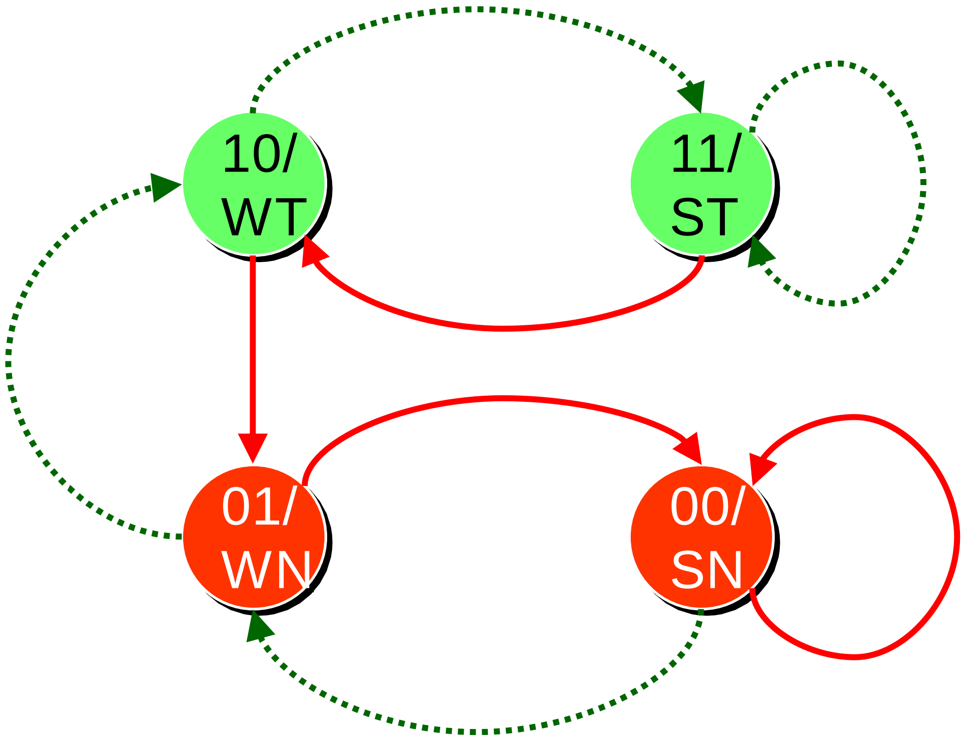


| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ☠ | √ | √ | √ | ☠ | ☠ | √ | √ | √ | ☠ | ☠ | |
| Pred | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| Actual | T | T | T | T | NT | T | T | T | T | NT | T | |

3/5 = 60% accuracy

# 2-bit Sat. Up/Down Counter Predictor

MSB: Direction bit
LSB: Hysteresis bit



⋯▶ Taken

→ Not Taken

🔴 Predict Not taken

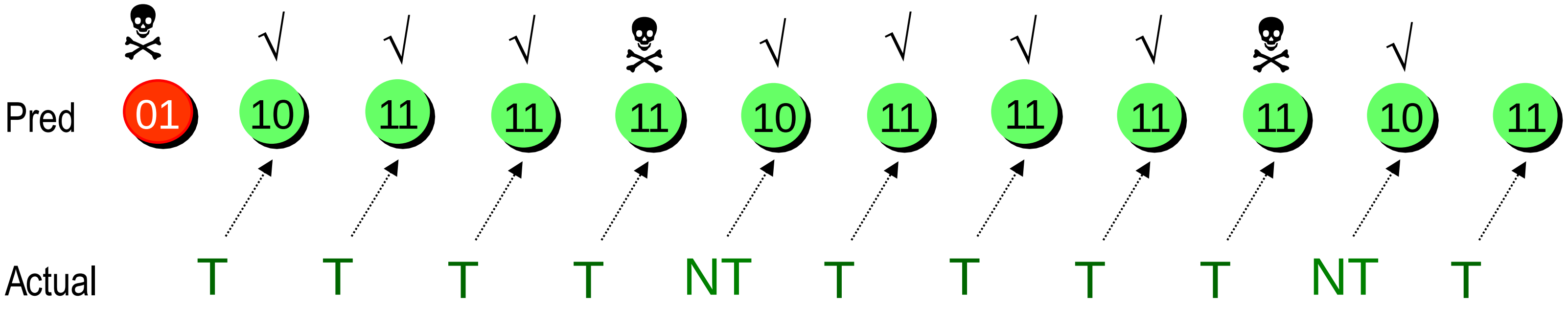
🟢 Predict taken

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

# 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
```



☠ √ √ √ ☠ √ √ √ √ ☠ √

Pred 01 10 11 11 11 10 11 11 11 11 10 11

Actual T T T T NT T T T T NT T

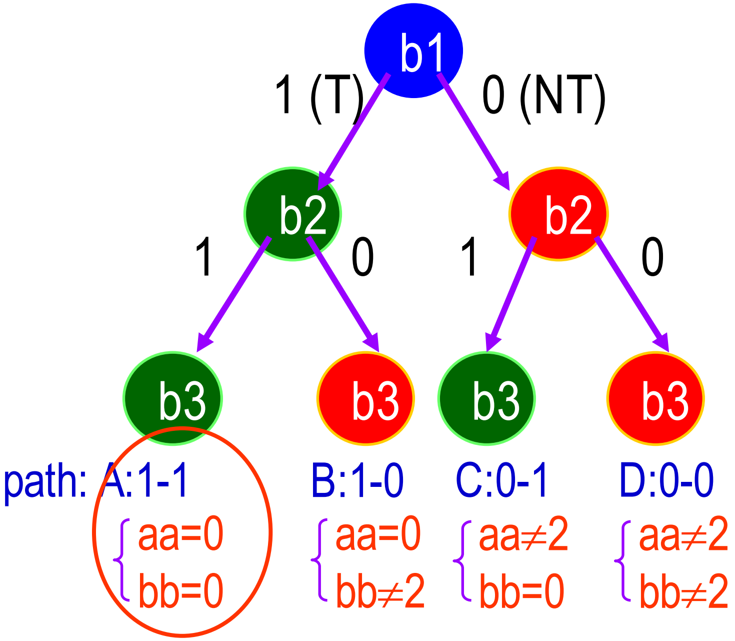
4/5 = 80% accuracy

# Branch Correlation

- 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

code snippet

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

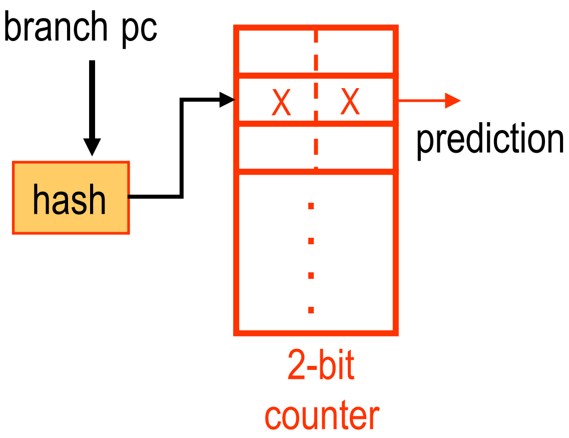


Optional slide

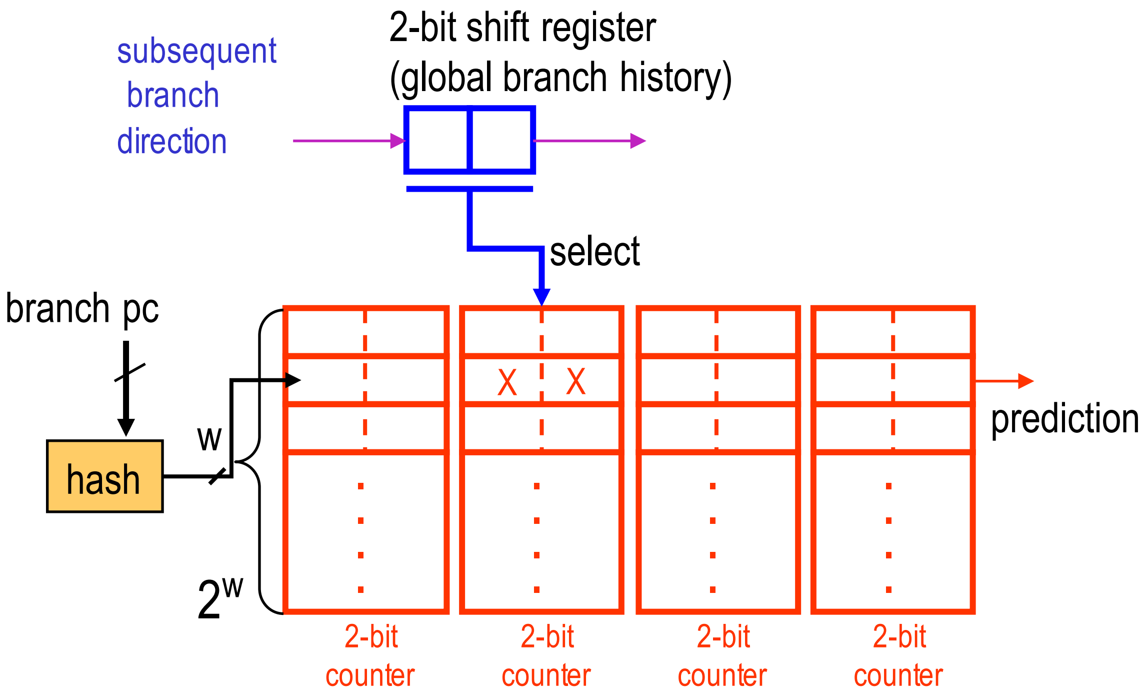
# Correlated Branch Predictor

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

[Branch Correlation, ASPLOS'92]



2-bit sat. counter scheme



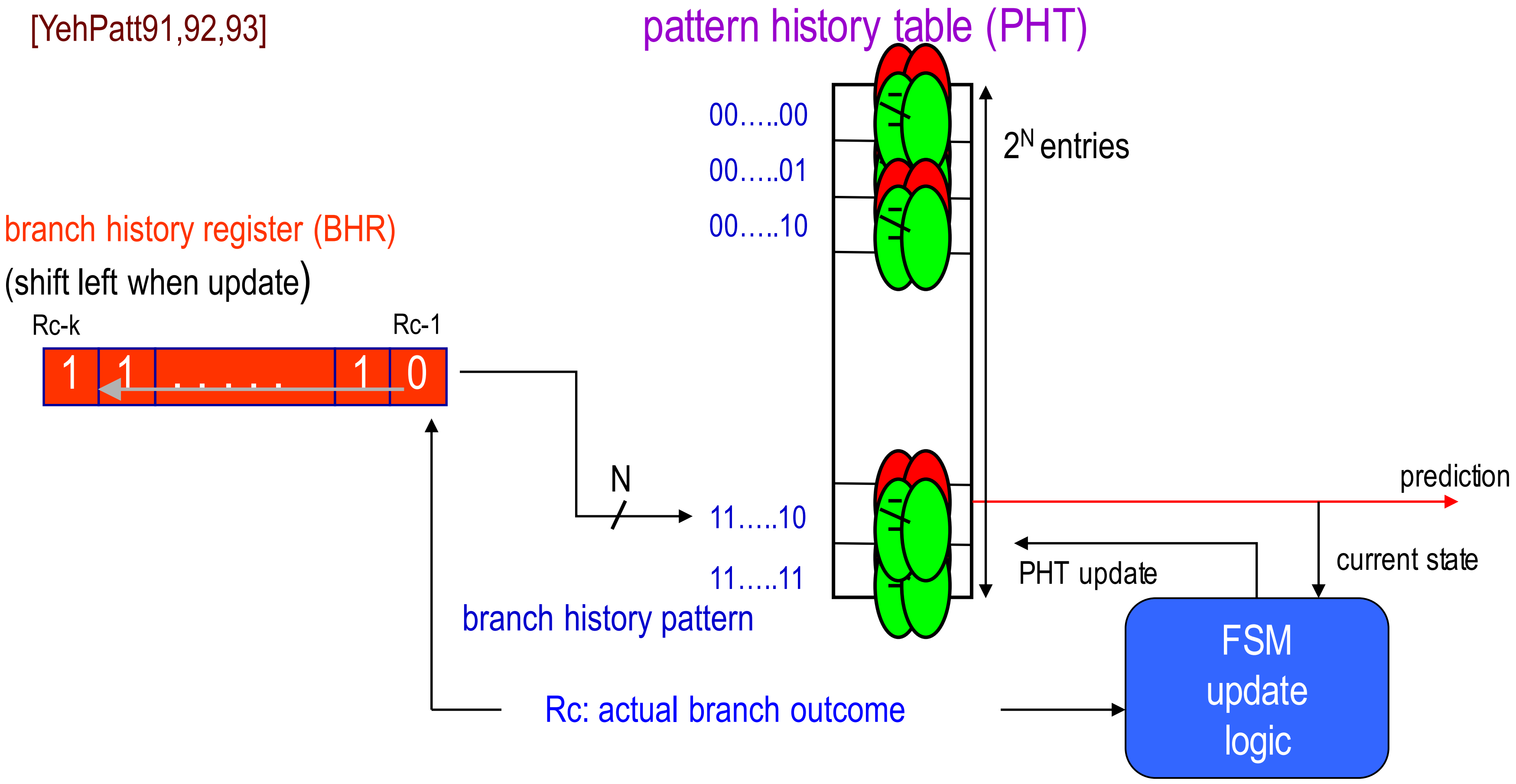
(2,2) correlation scheme

Optional slide

# Two-Level Branch Predictor

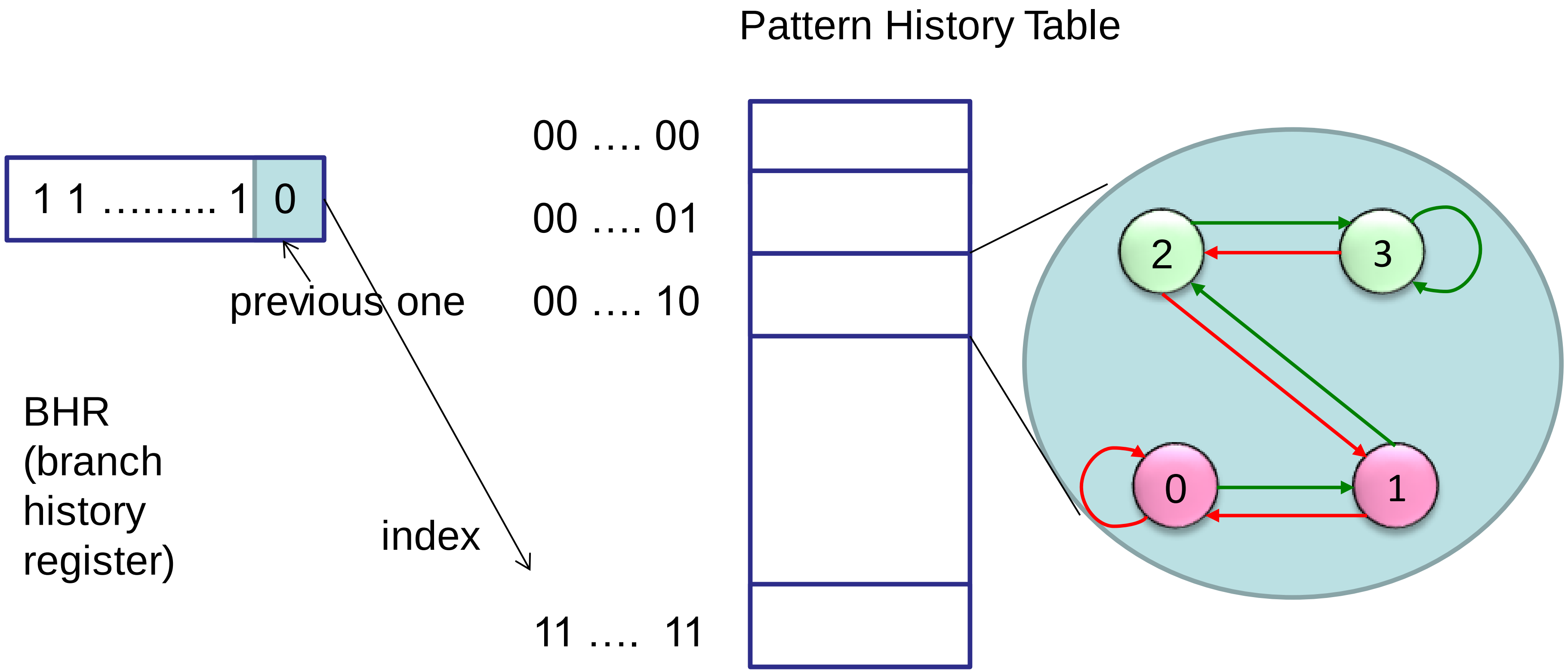
- Generalized correlated branch predictor
  - 1st level keeps branch history in branch hist reg (BHR)
  - 2nd level keeps pattern history in pattern hist. tab. (PHT)

[YehPatt91,92,93]



Optional slide

# Two-Level Branch Predictor



Pattern History Table

1 1 ……… 1 0

previous one

BHR (branch history register)

index

00 …. 00

00 …. 01

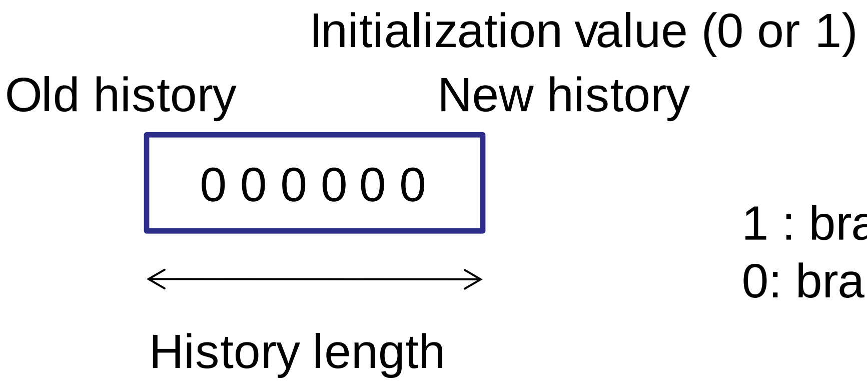
00 …. 10

11 …. 11

Yeh&patt’92

Optional slide

# Branch History Register



Initialization value (0 or 1)

Old history

New history

0 0 0 0 0 0

History length

1 : branch is taken
0: branch is not-taken

New BHR = old BHR<<1 | (br_dir)

Example

BHR: 00000

Br1 : taken → BHR 00001
Br 2: not-taken → BHR 00010
Br 3: taken → BHR 00101

Optional slide

# Why Does Global Predictor Work?

- Branches are correlated

Branch X: if (cond1)

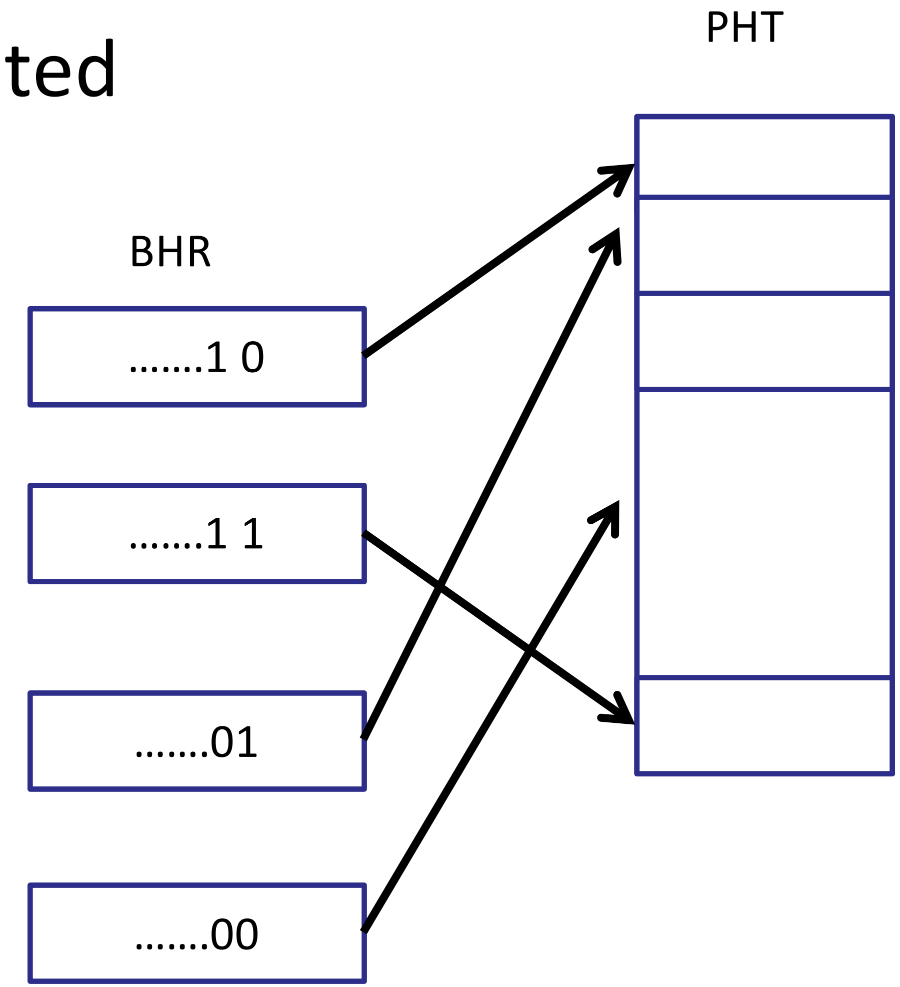
….

Branch Y: if (cond 2)

….

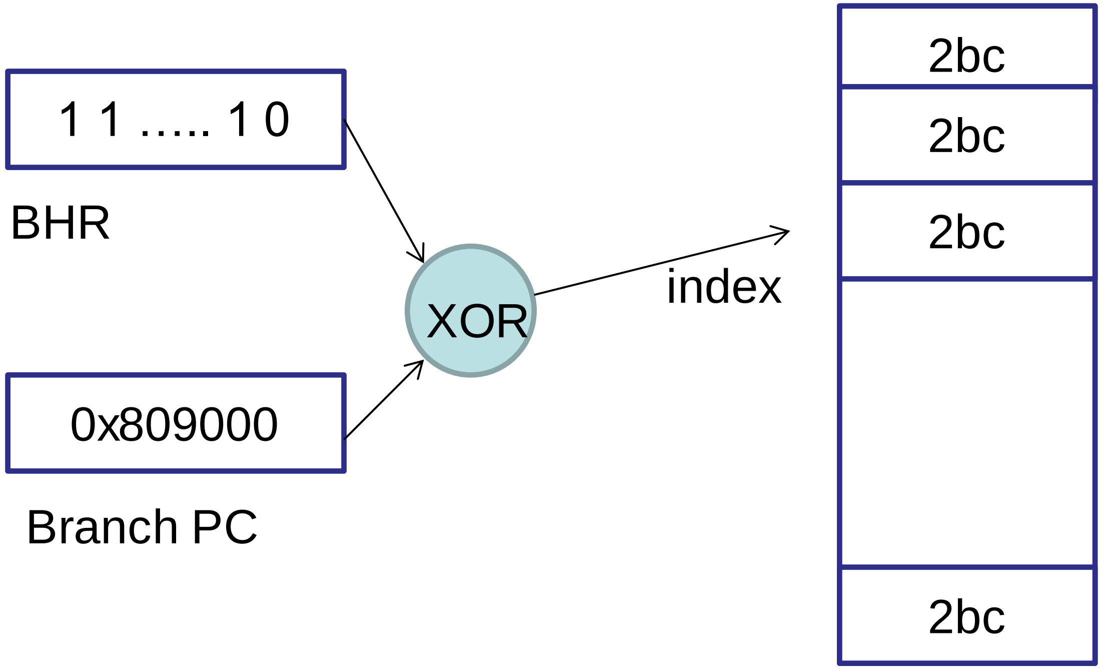
Branch Z : if (cond 1 and cond 2)

| Branch X | Branch Y | Branch Z |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



Optional slide

# Gshare Branch Predictor



McFarling’93

Predictor size: 2^(history length)*2bit

Optional slide