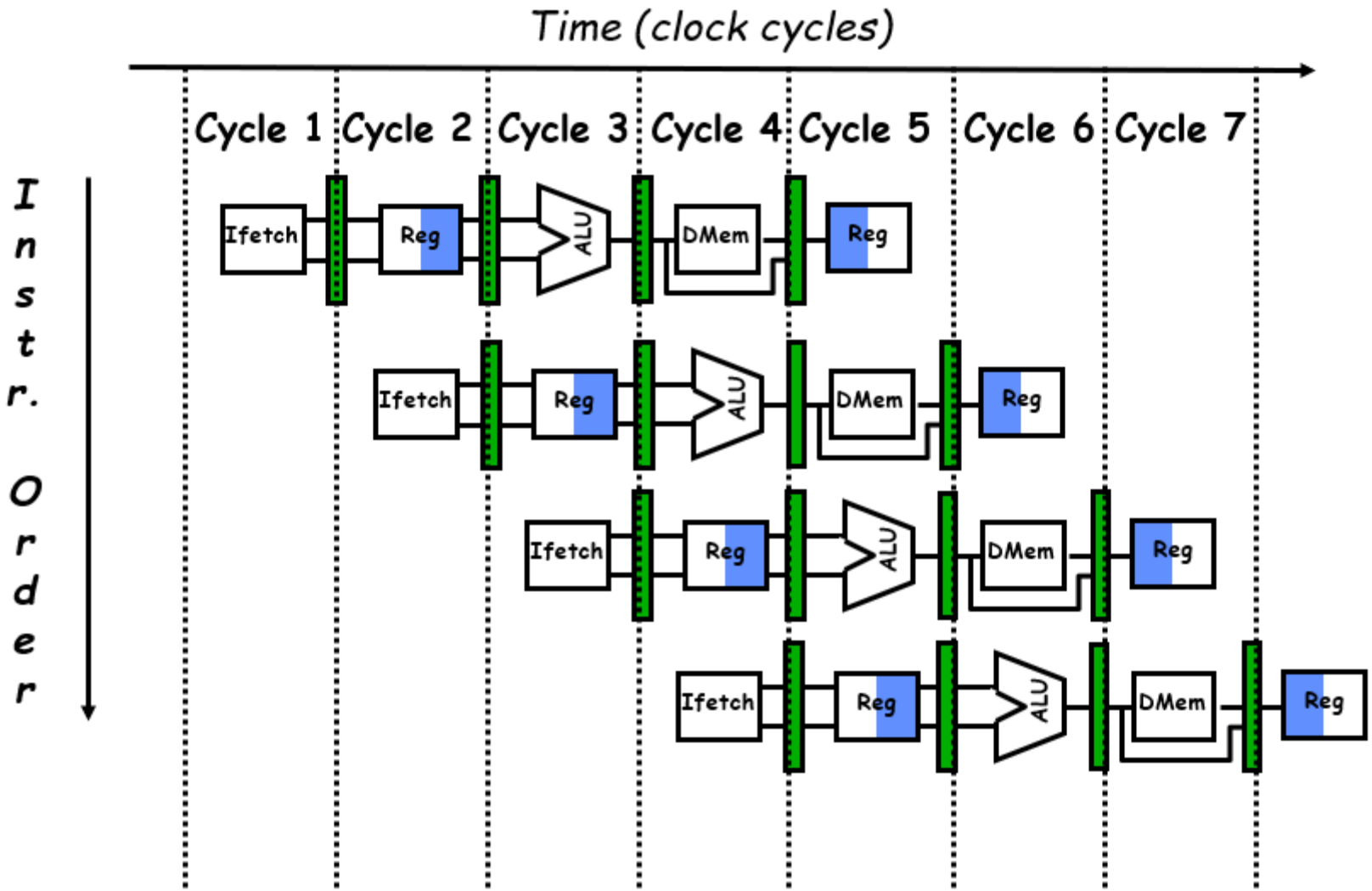
Branch Prediction

Adapted from Prof. David A. Patterson slides

Bilal Habib

DCSE, UET Peshawar

Visualizing Pipelining

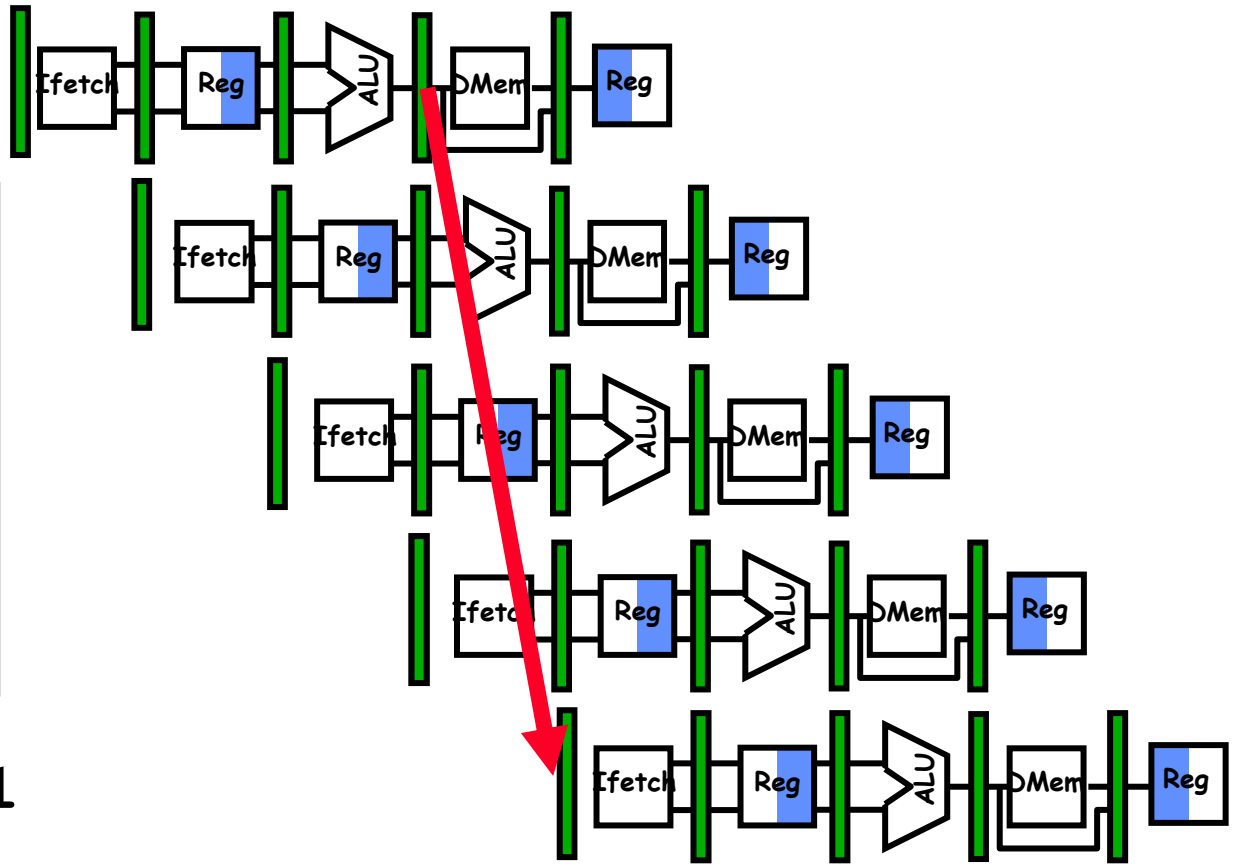


Its Not That Easy for Computers

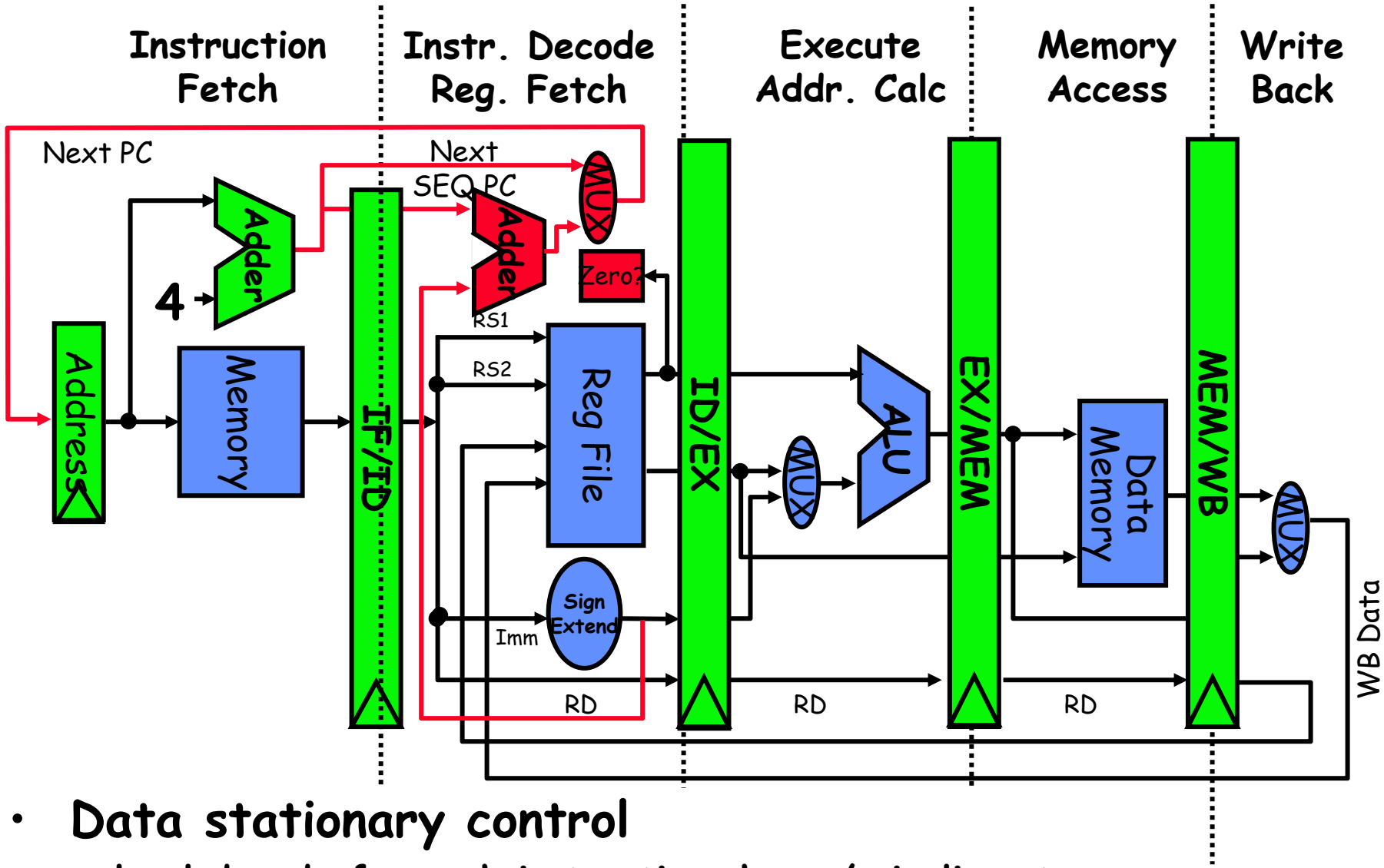
- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

Control Hazard on Branches
Three Stage Stall

```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



Pipelined MIPS Datapath



- Data stationary control
  - local decode for each instruction phase / pipeline stage

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken

- 53% MIPS branches taken on average
- But haven't calculated branch target address in MIPS
  - » MIPS still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

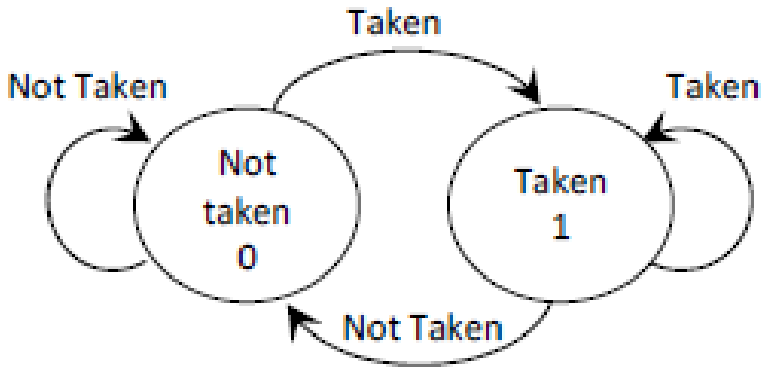
Branch Prediction

- **Static Branch Prediction**: During compile time
  - Backward always taken due to loops (for, while)
- **Dynamic Branch Prediction**: During run-time
  - More accurate than Static predictors
  - Needs more hardware to design
  - Examples: 1-bit and 2-bit predictors

Dynamic Branch Prediciton

- **Hardware-based schemes that utilize run-time behavior of branches to make dynamic predictions:** Information about outcomes of previous occurrences of branches are used to dynamically predict the outcome of the current branch.
- **Make more accurate predictions than possible using static prediction**

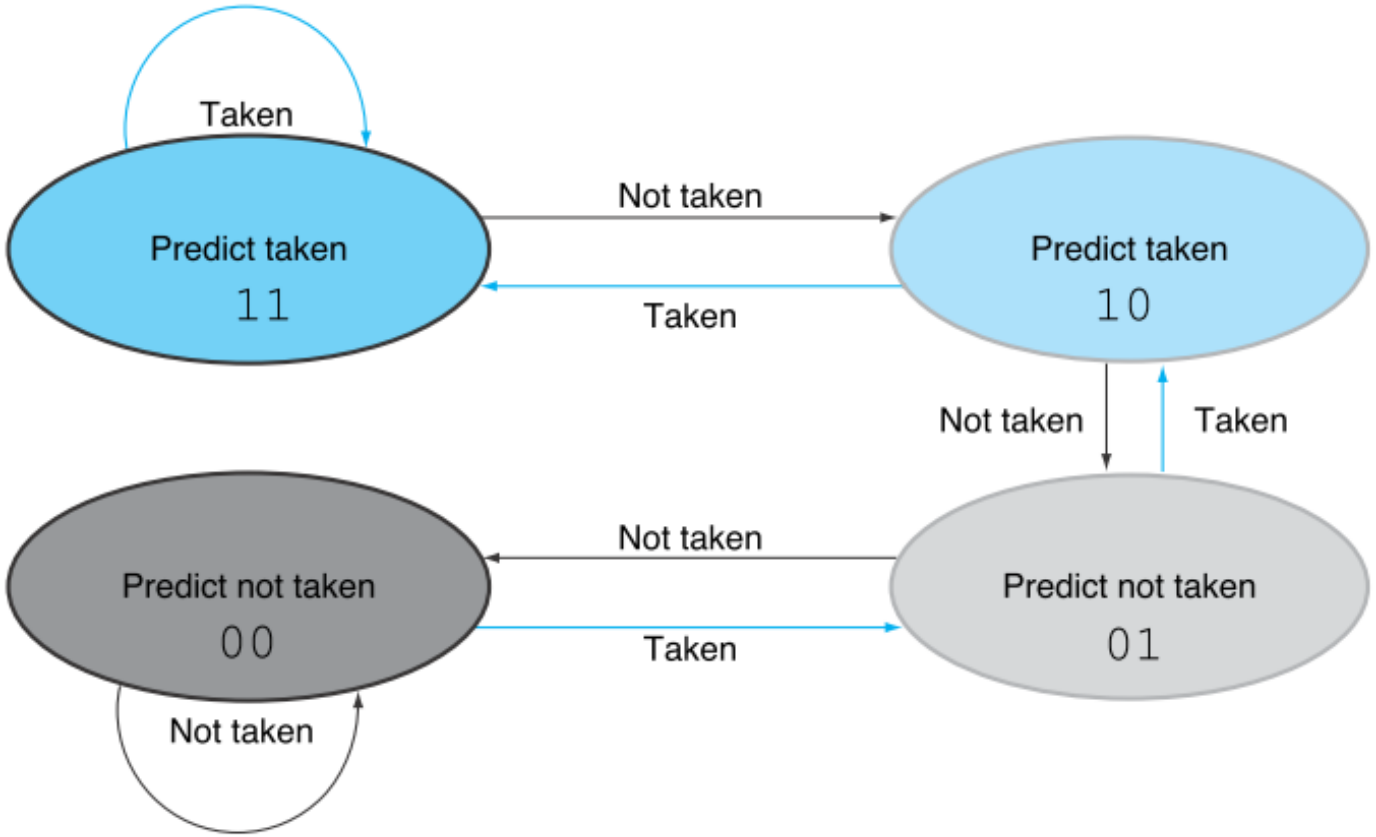
1-bit Branch Prediction




```
main:
        li $t2, 10          # loop variable
        li $t1,0            # sum = 0
loop:   add $t1, $t2, $t1   # Add t1 = t1 + t2
        subi $t2, $t2, 1    # t2 = t2 - 1
        bne $t2,0,loop      # check
```

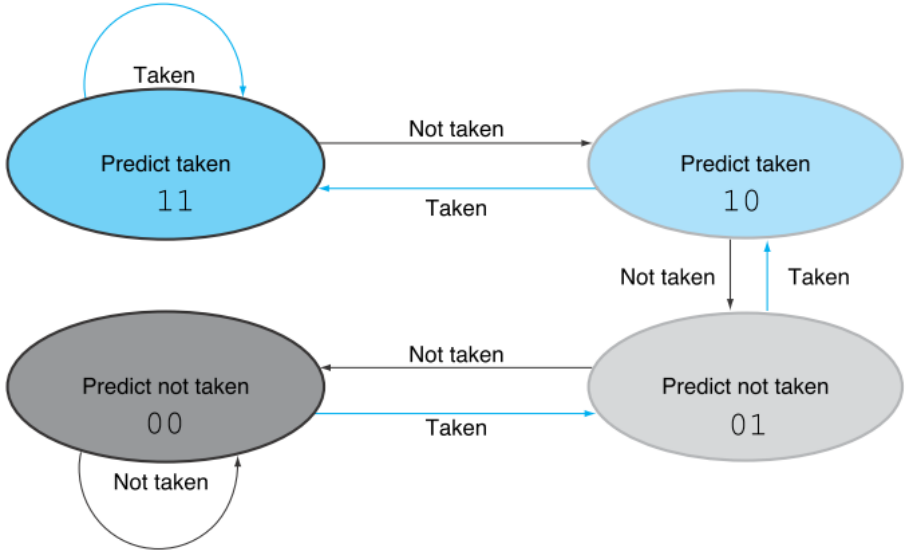
Problem:
Mispredict twice

2-bit Branch Prediction



2-bit Branch Predictor

- Nested loop:
- 1st outer loop execution: -
  - 00 → predict not taken; actually taken → update to 01 (misprediction)
  - 01 → predict not taken; actually taken → update to 10 (misprediction)
  - 10 → predict taken; actually taken → update to 11
  - 11 → predict taken; actually taken
  - …
  - 11 → predict taken; actually not taken → update to 10 (misprediction)




```
Loop1: …
       …
Loop2: …
       bne r1,r0,loop2
       …
       bne r2,r0,loop1
```

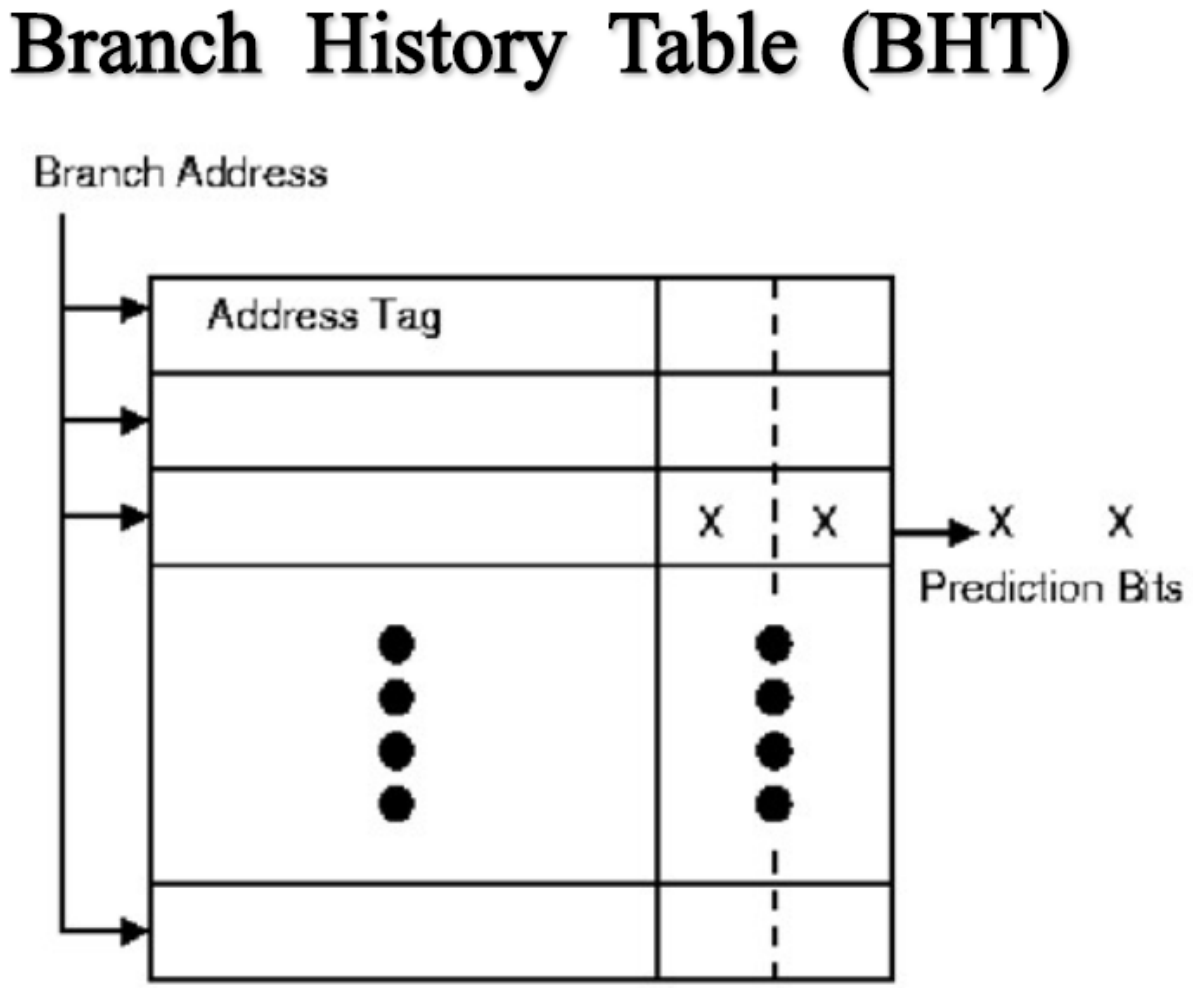
2-bit Branch Predictor

- 2nd loop execution onwards:
  - 10 → predict taken; actually taken → update to 11
  - 11 → predict taken; actually taken
  - …
  - 11 → predict taken; actually not taken → update to 10 (misprediction)

```
Loop1: …
       …
Loop2: …
       bne r1,r0,loop2
       …
       bne r2,r0,loop1
```

2-bit Branch Predictor

- BHT keep tracks of history of previous branches: Taken or Not-Taken
- 2-bit branch predictors work very well for loop-intensive applications



2-Bit Branch Predictor

- Assume branches resolve in stage 3
- Reasonable for a modern high-frequency processor
- 20% of instructions are branches
- Correctly-predicted branches have a 0-cycle penalty (CPI=1)
- 2-bit predictor: 92% accuracy
- 2-bit predictor:
- CPI = 0.8(1) + 0.2 * (3*0.08 + 1*0.92) = __
- Speed-up over no predictor?

Calculate accuracy of 1-b and 2-b Predictors

```
main:
   li $t2, 4       # i = 4
   li $t3, 0       # k = 0
outer_Loop:
 li $t1,5        # j = 5
        inner_Loop:
          add $t3, $t3, 1     # k = k + 1
          subi $t1, $t1,1     # j = j-1
                bne $t1,0, inner_Loop
   subi $t2, $t2,1     # i = i-1
   bne $t2,0, outer_Loop
```