Chapter 4

The Processor

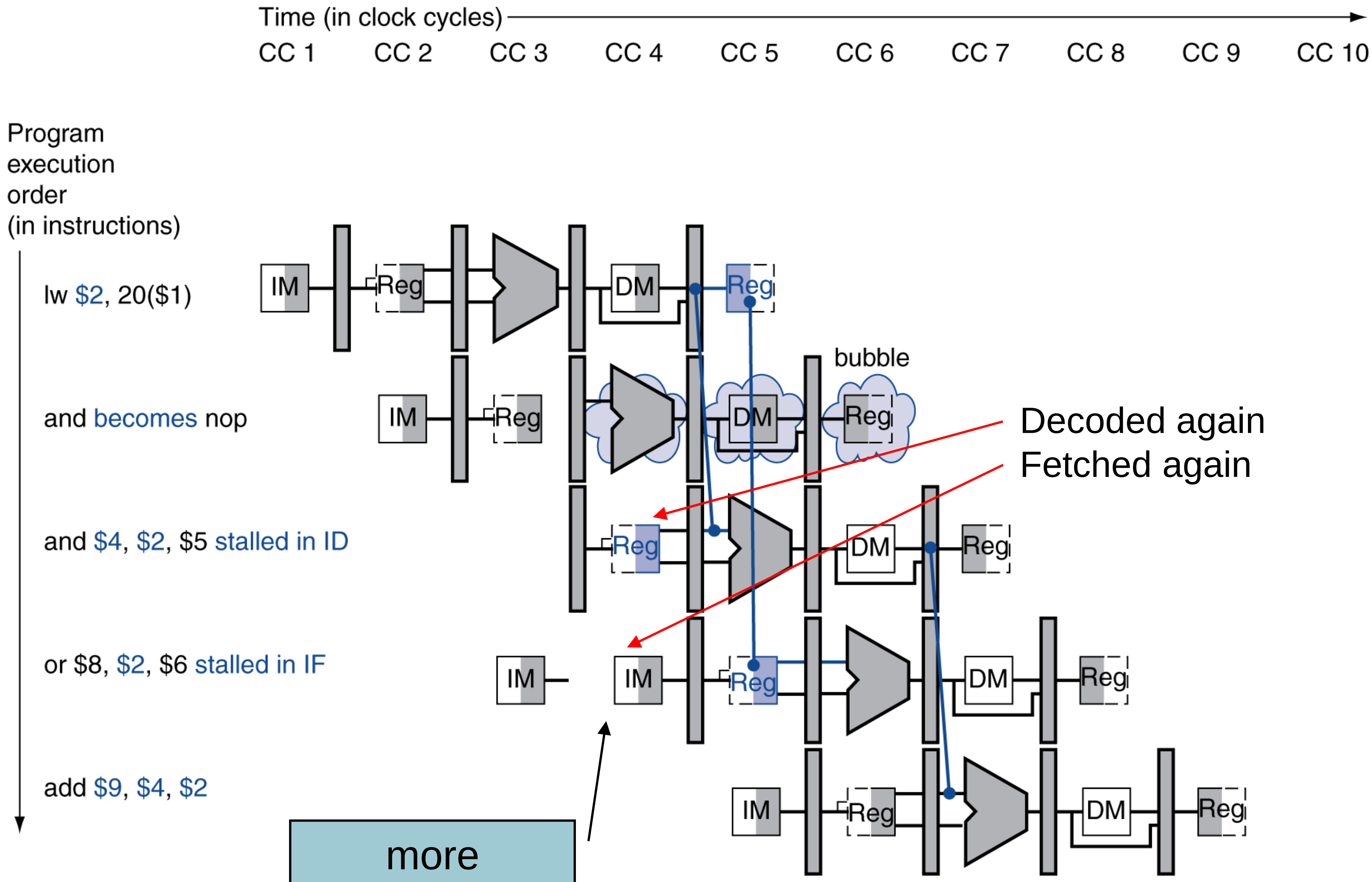
Fall 2018

Soontae Kim
School of Computing, KAIST

Announcement

- Project #2
  - Due on Nov. 9
  - Studying pipelining with simulator
- Midterm exam results
  - Avg 68.19
  - Max 100
  - Final exam will be more difficult

Stall/Bubble in the Pipeline

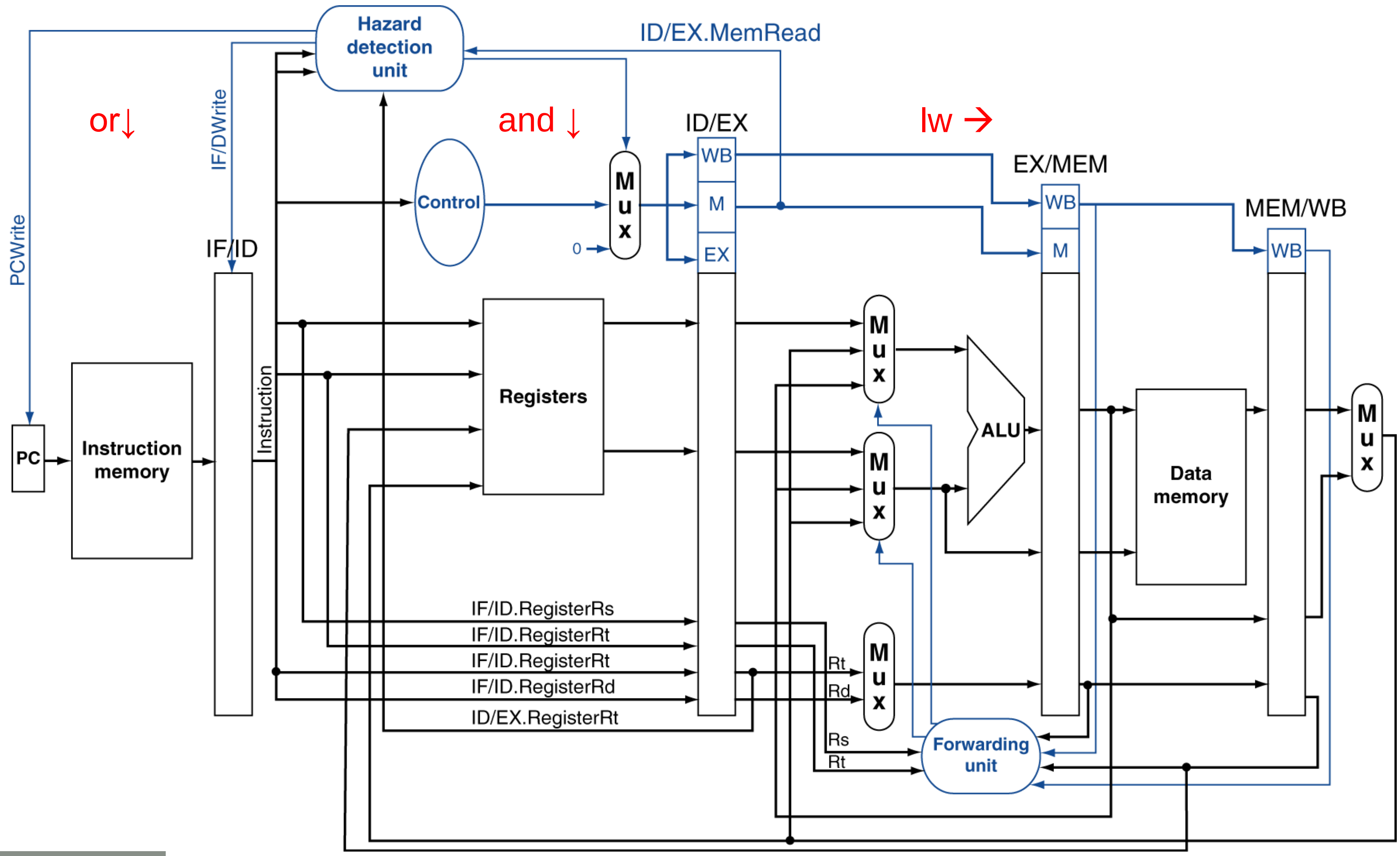


Decoded again

Fetched again

more accurately…

Datapath with Hazard Detection

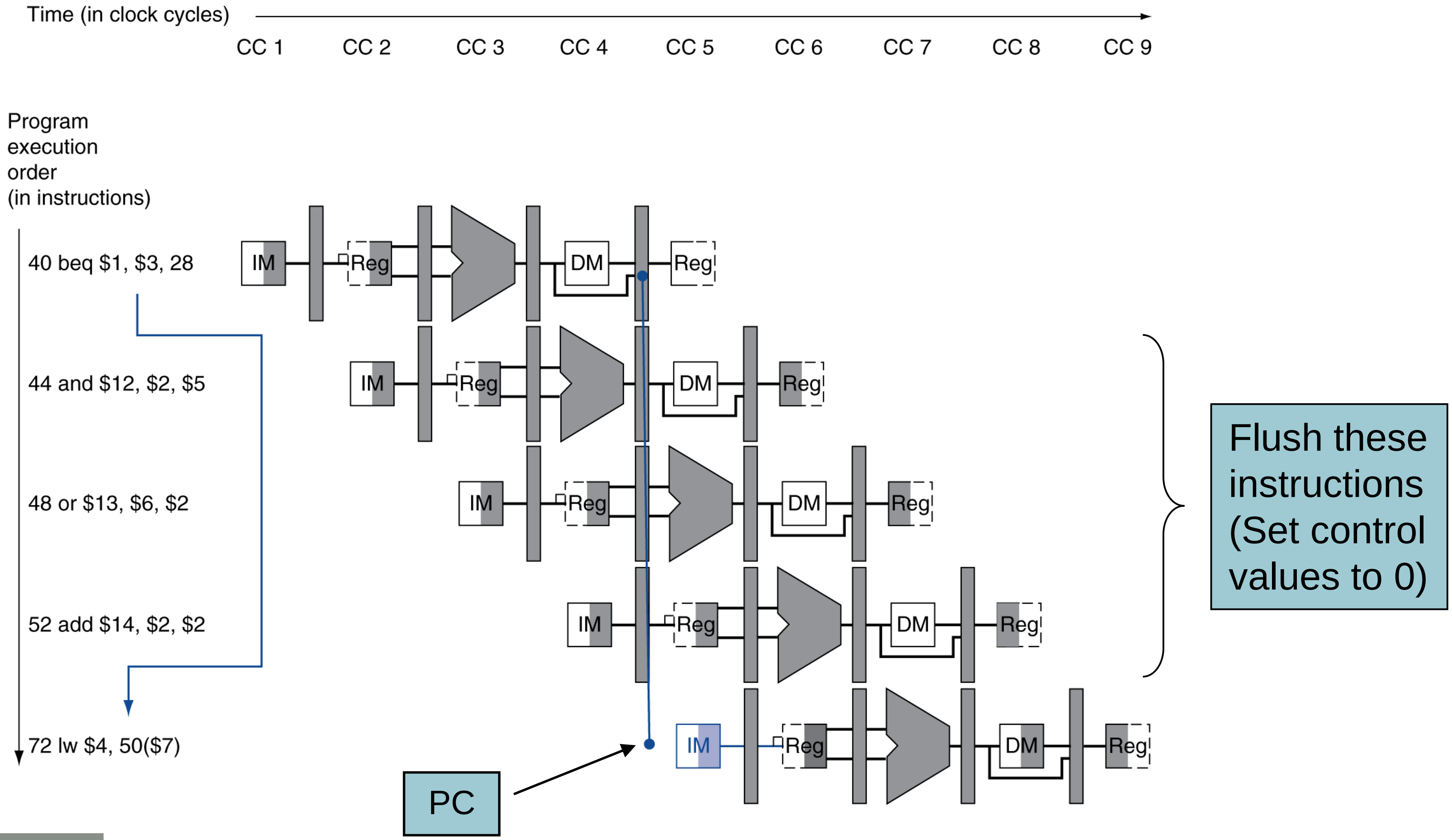


How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `lw`
    - Can subsequently forward to EX stage
- lw proceeds while and and or insts. Stay in the same stages

Branch Hazards

- If branch outcome determined in MEM



Flush these instructions (Set control values to 0)

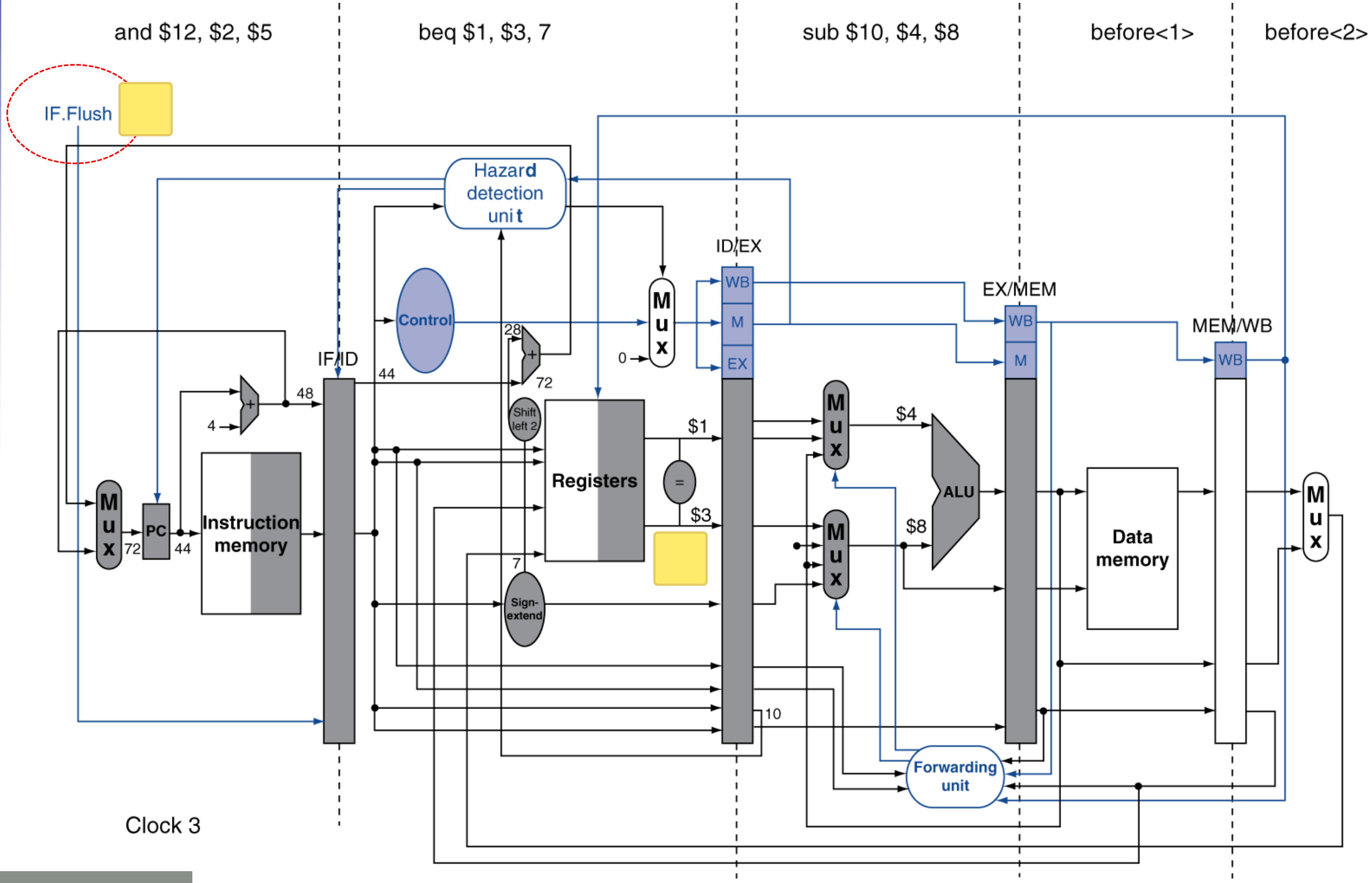
PC

Reducing Branch Delay

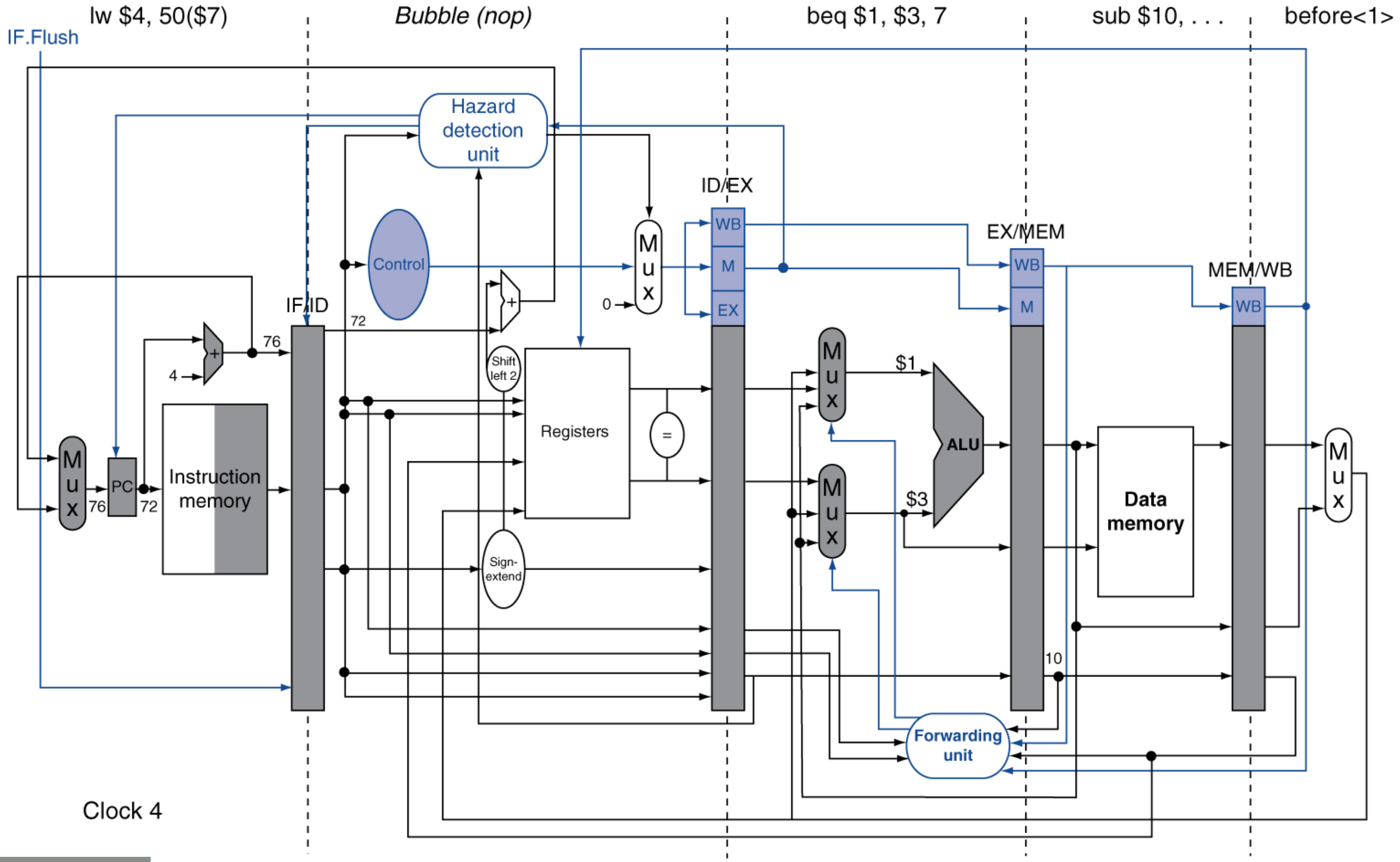
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

Example: Branch Taken



Example: Branch Taken



Data Hazards for Branches

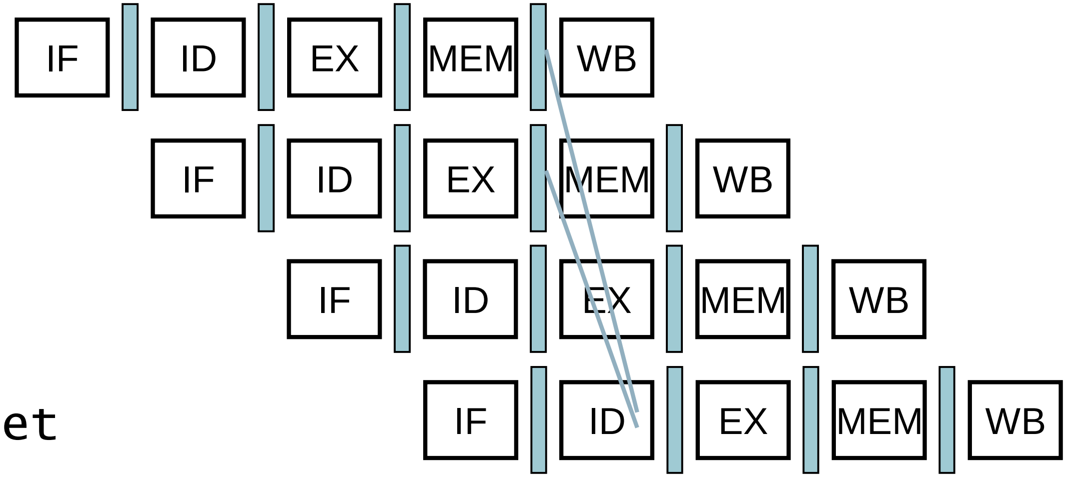
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

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

add $4, $5, $6

…

beq $1, $4, target
```

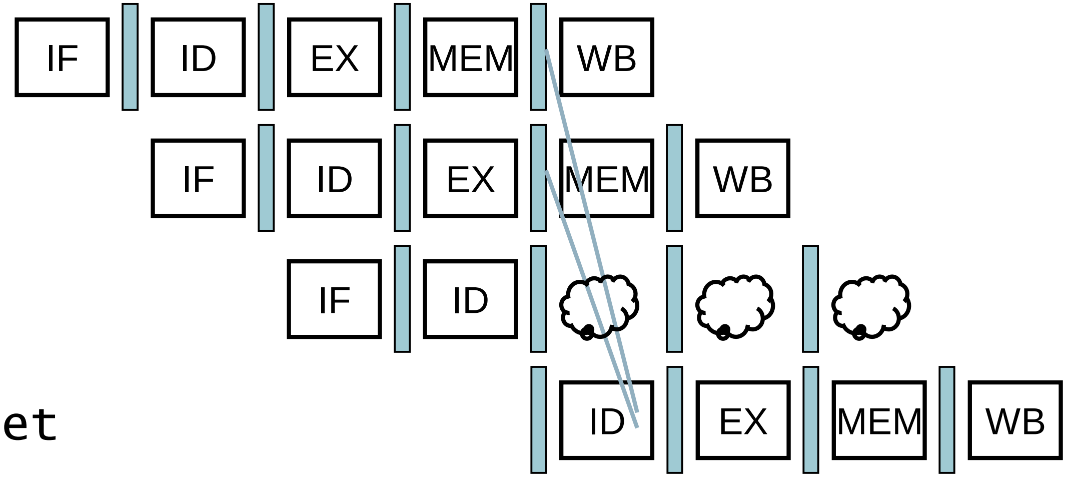


- Can resolve using forwarding

Data Hazards for Branches

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

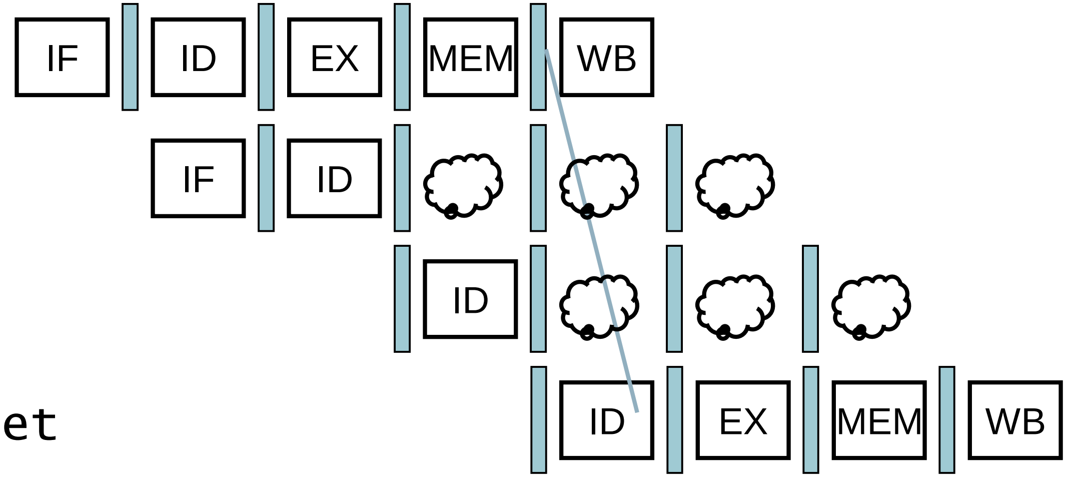
```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



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
```

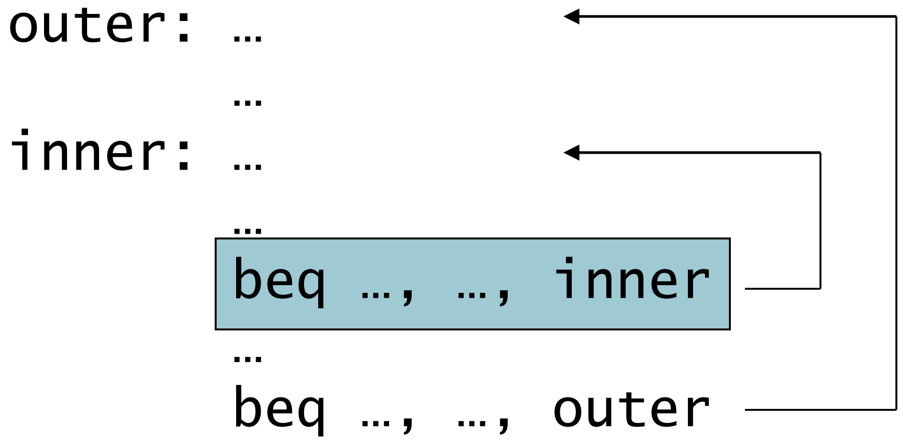


Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!



```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

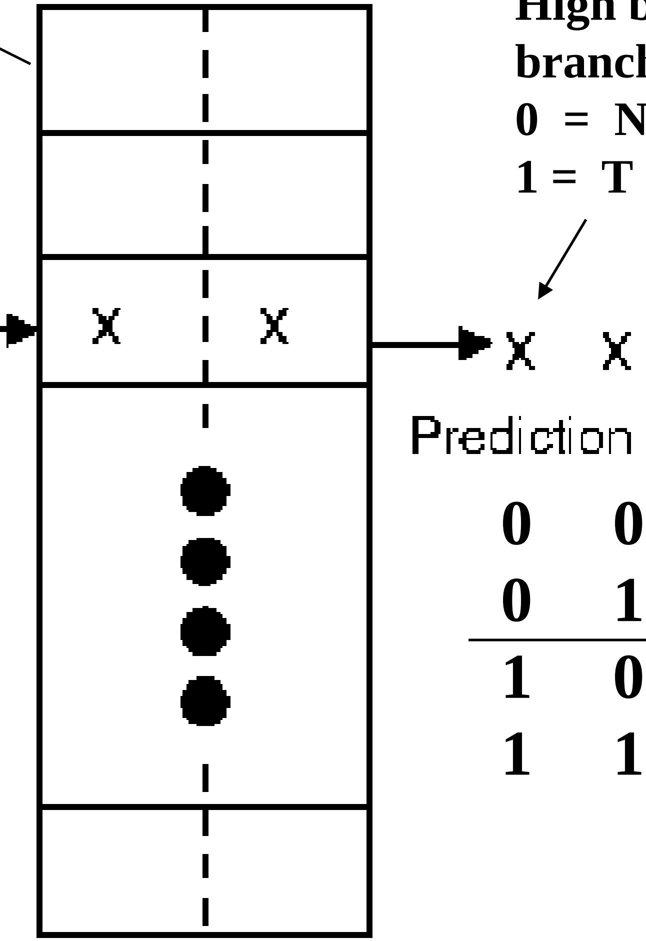
  - Mispredict as taken on last iteration of inner loop
  - Then mispredict as not taken on first iteration of inner loop next time

One-Level Bimodal Branch Predictors

2-bit saturating counters

High bit determines branch prediction
0 = NT = Not Taken
1 = T = Taken

N Low Bits of Branch Address



Prediction Bits

0	0	Not Taken (NT)
0	1	
1	0	Taken (T)
1	1	

Table has 2^N entries.

Example:

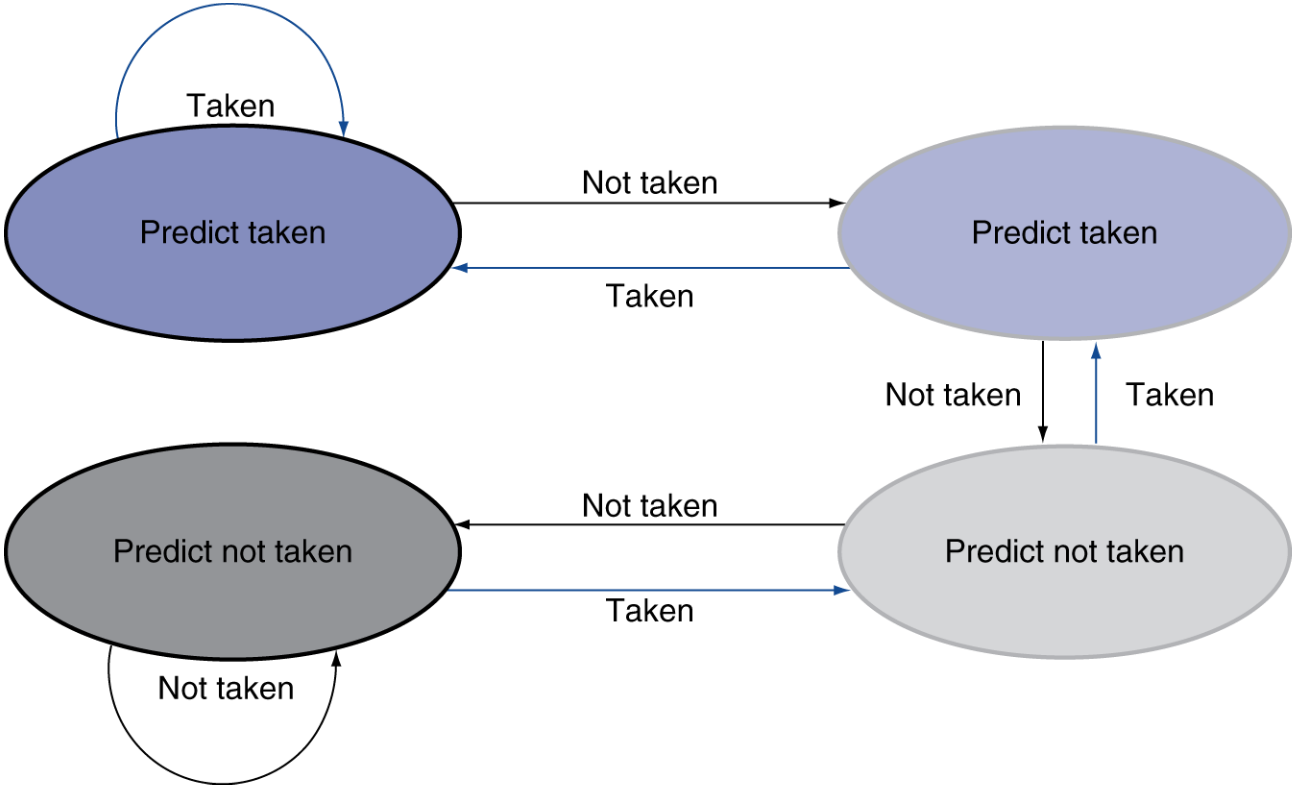
For N =12
Table has $2^N = 2^{12}$ entries
= 4096 = 4k entries

Number of bits needed = 2 x 4k = 8k bits

Common one-level implementation

2-Bit Predictor

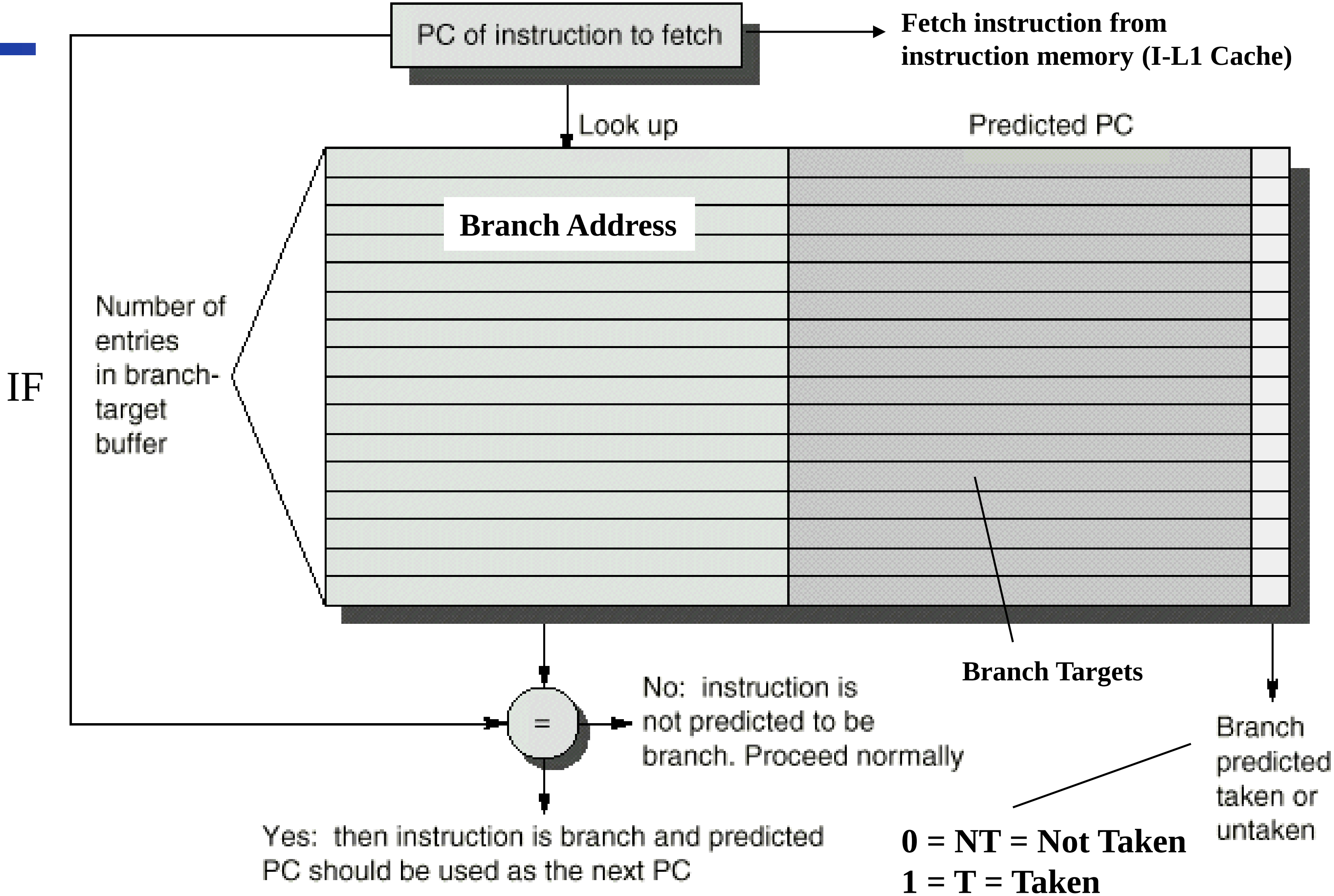
- Only change prediction on two successive mispredictions



Calculating the Branch Target

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- Branch target buffer (BTB)
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately

Basic Branch Target Buffer (BTB)



A branch-target buffer.