Introduction to Computer Architecture
Chapter 4 - 6

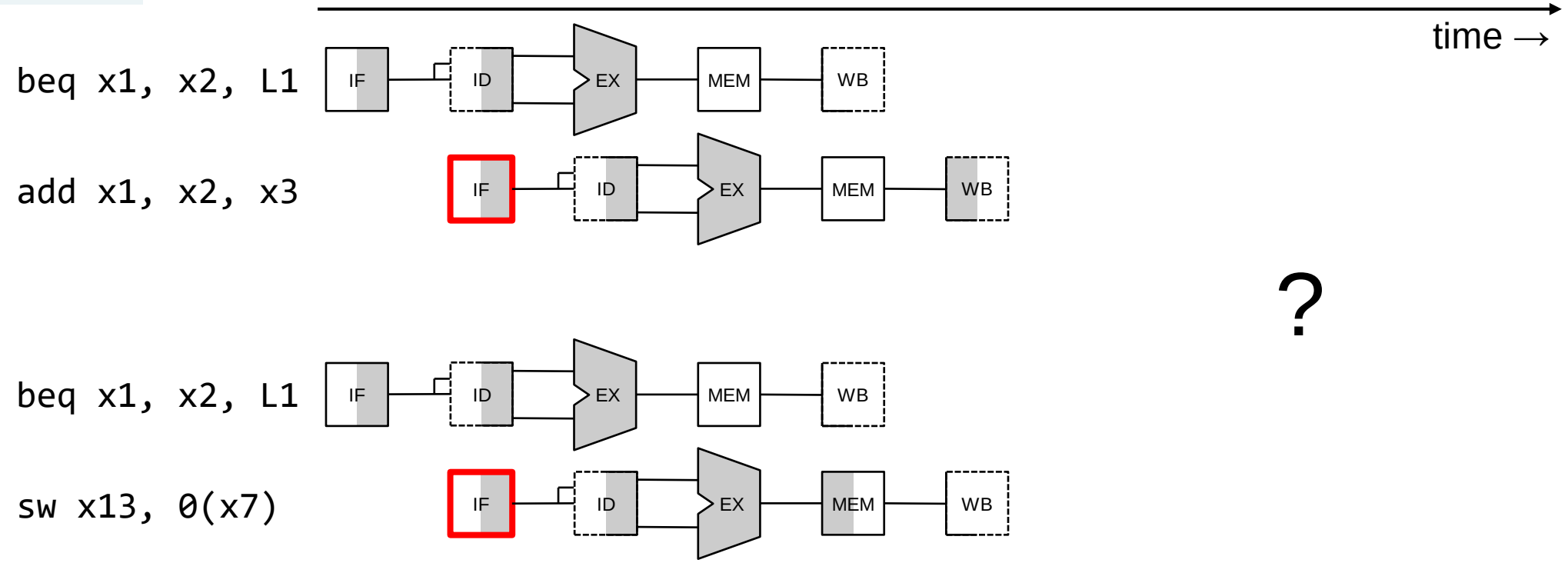
Pipeline Hazards 2

Hyungmin Cho

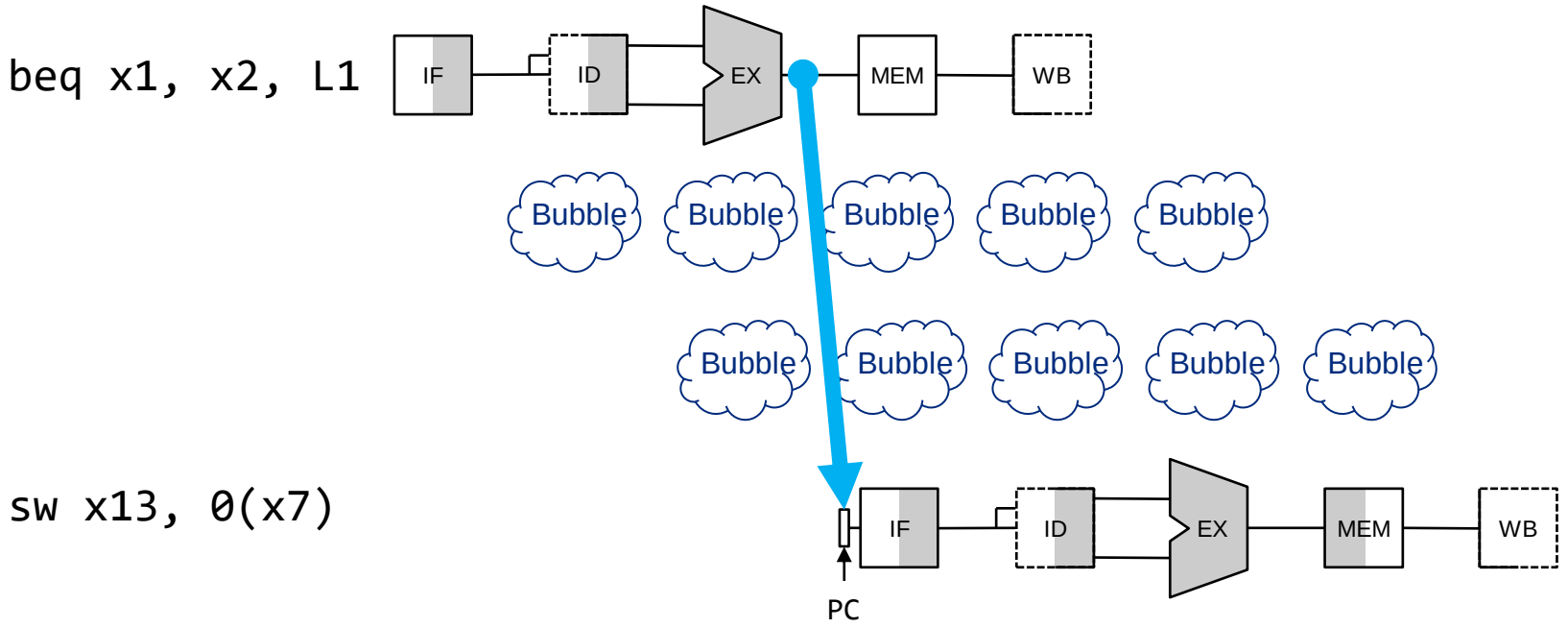
Department of Computer Science and Engineering
Sungkyunkwan University

Branch Hazards

```
     beq x1, x2, L1
     add x1, x2, x3
     …
     …
L1:  sw x13, 0(x7)
```



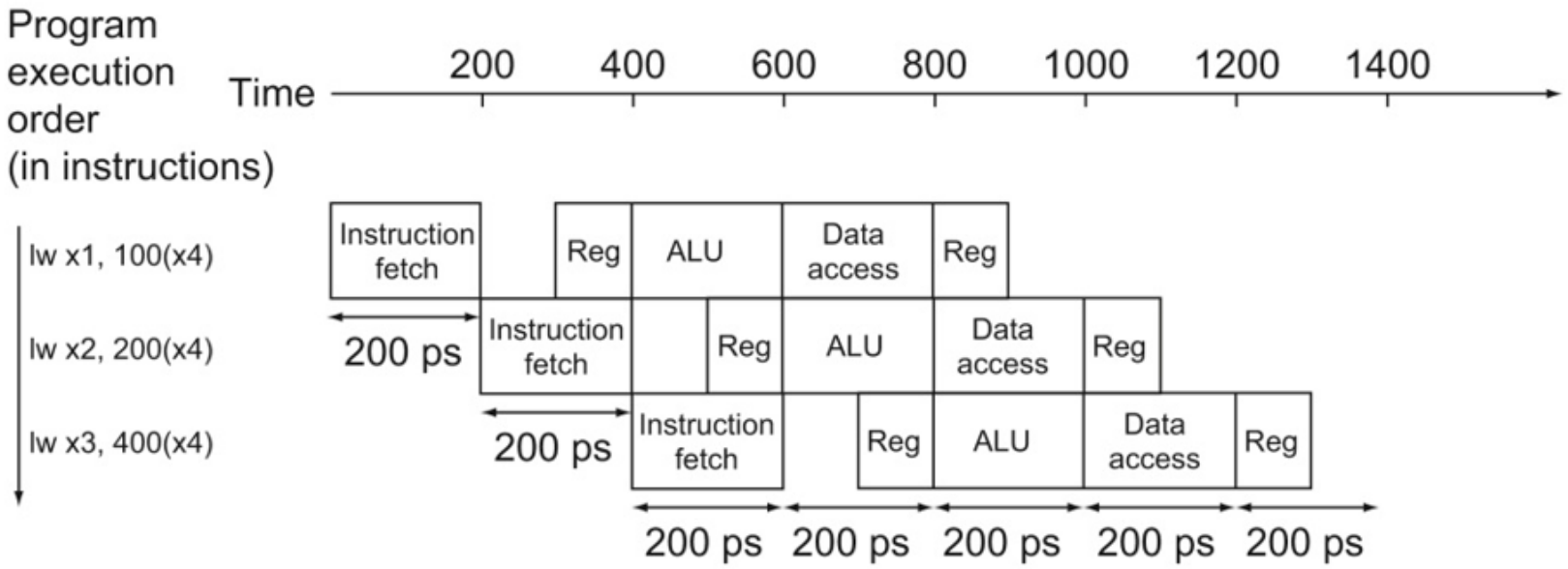
Branch Hazards



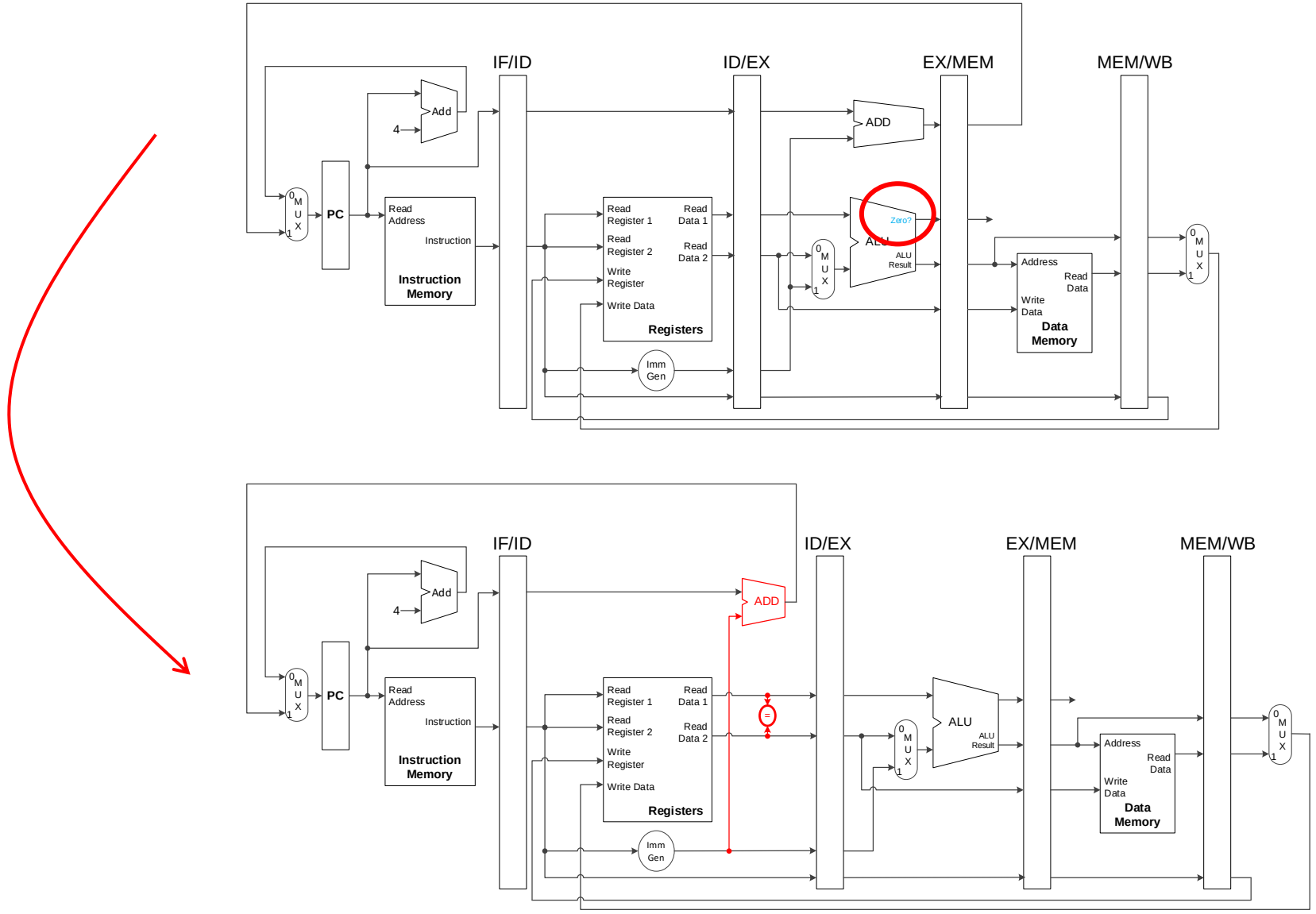
Reducing Branch Delay

- Move register comparison to the ID stage

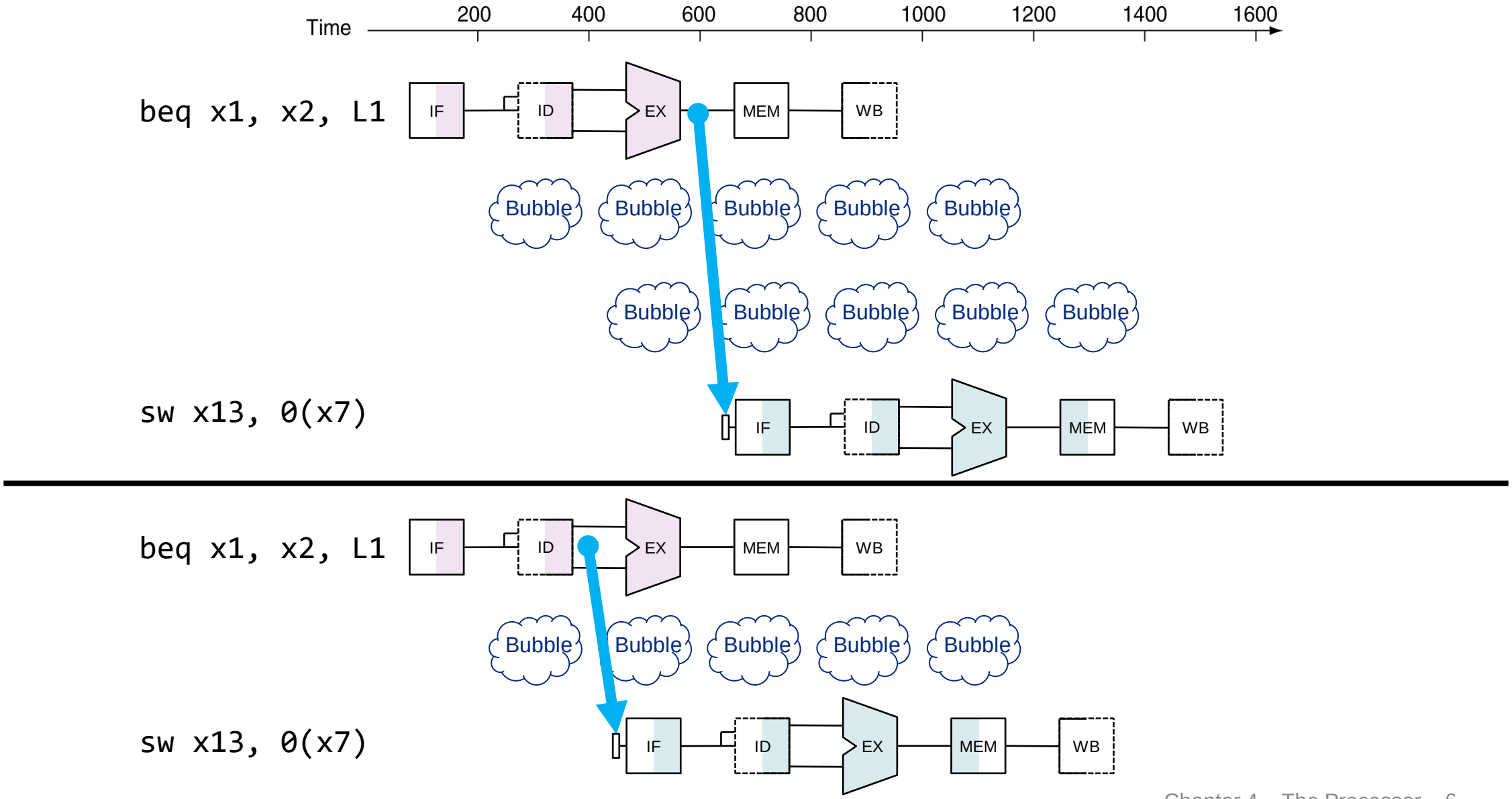
- Move target address calculation to the ID stage



Reducing Branch Delay



Reducing Branch Delay



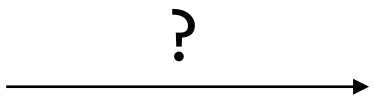
Data Hazards for Branches

```
lw  x8, …

add x6, x5, x6

beq x1, x4, target

bubble

bubble

PC+4 or Target
```

```
lw  x8, …

add x6, x5, x6

beq x1, x4, target

bubble

PC+4 or Target
```

Data Hazards for Branches

```
lw  x1, …

add x4, x5, x6

beq x1, x4, target

bubble

bubble

PC+4 or Target
```

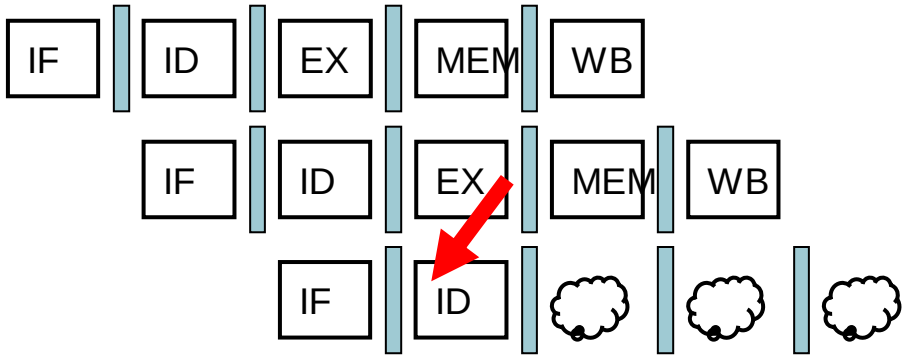



```
lw  x1, …

add x4, x5, x6

beq x1, x4, target

bubble

PC+4 or Target
```

Data Hazards for Branches

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

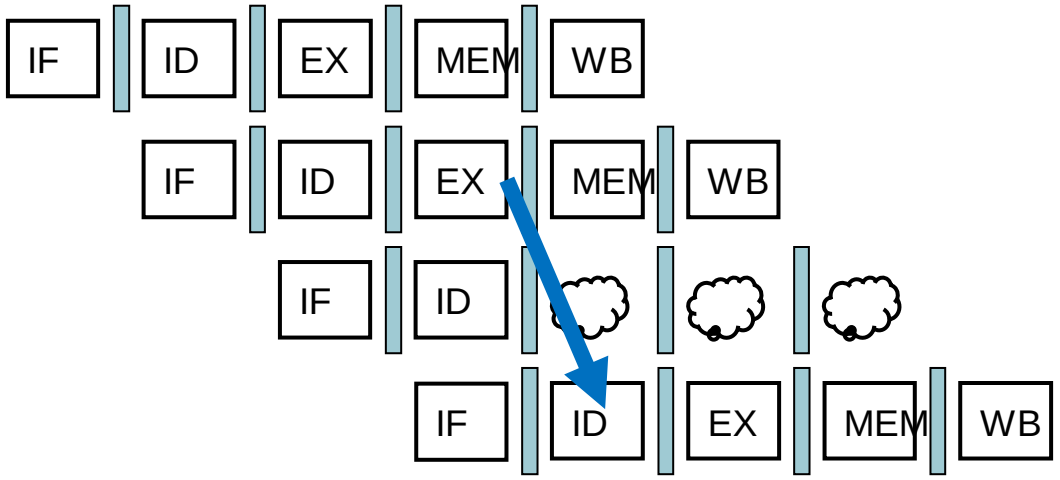
```
add …
add x4, x5, x6
beq x1, x4, target
bubble
PC+4 or Target
```



Data Hazards for Branches

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

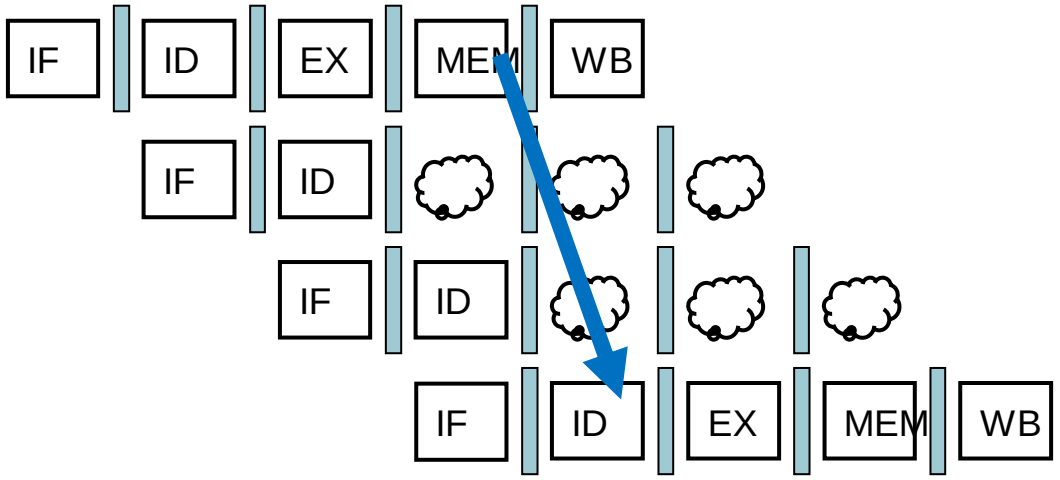
```
add …

add x4, x5, x6

~~beq~~ bubble

beq x1, x4, target

bubble

PC+4 or Target
```



Data Hazards for Branches

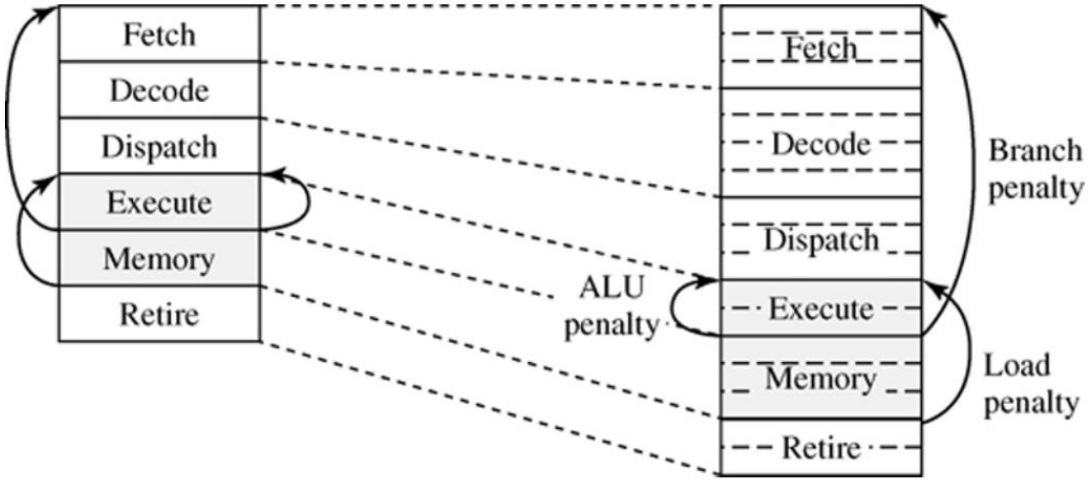
- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles BEFORE beq

```
lw  x1, …

~~beq~~ bubble

~~beq~~ bubble

beq x1, x0, target

bubble

PC+4 or Target
```



Branch Prediction

- Longer pipelines can't determine branch outcome early
  - Intel i7: 14 pipeline stages
  - Stall penalty becomes unacceptable



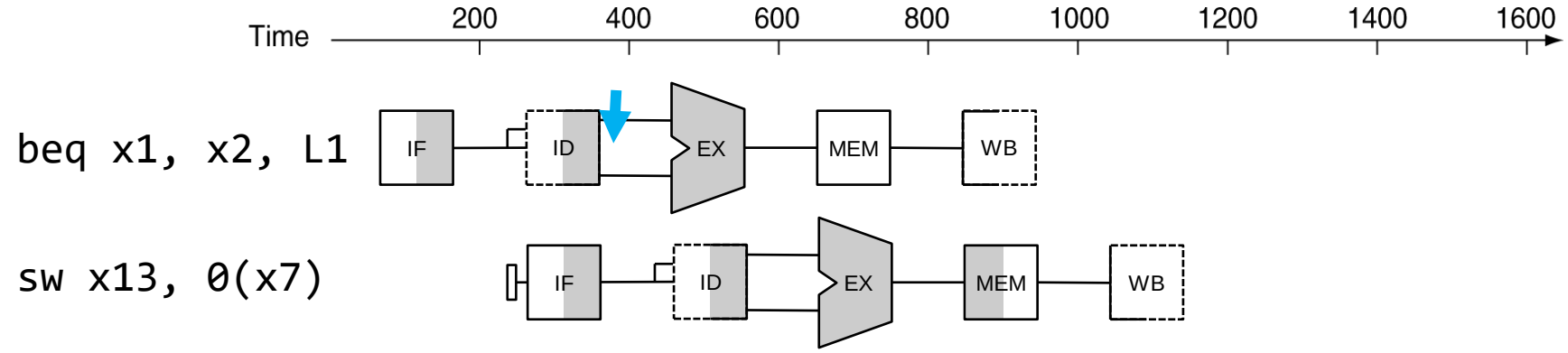
- Predict the outcome of the branch instructions
  - Stall only when the prediction was incorrect

Branch Prediction

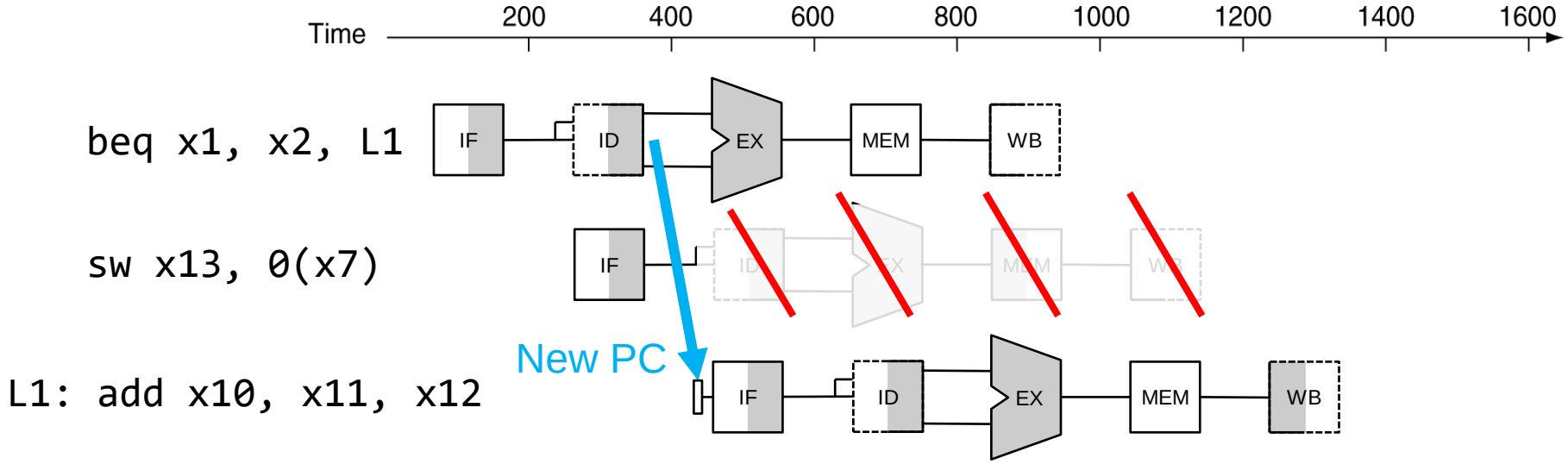
- An easy solution
  - **Always predict branches as not taken**
    - Proceed to PC+4
  - If the prediction was incorrect,
    - Cancel the predicted instructions in the pipeline.
    - Reset the PC value with the true target of the branch

Predict Not Taken

Prediction was correct

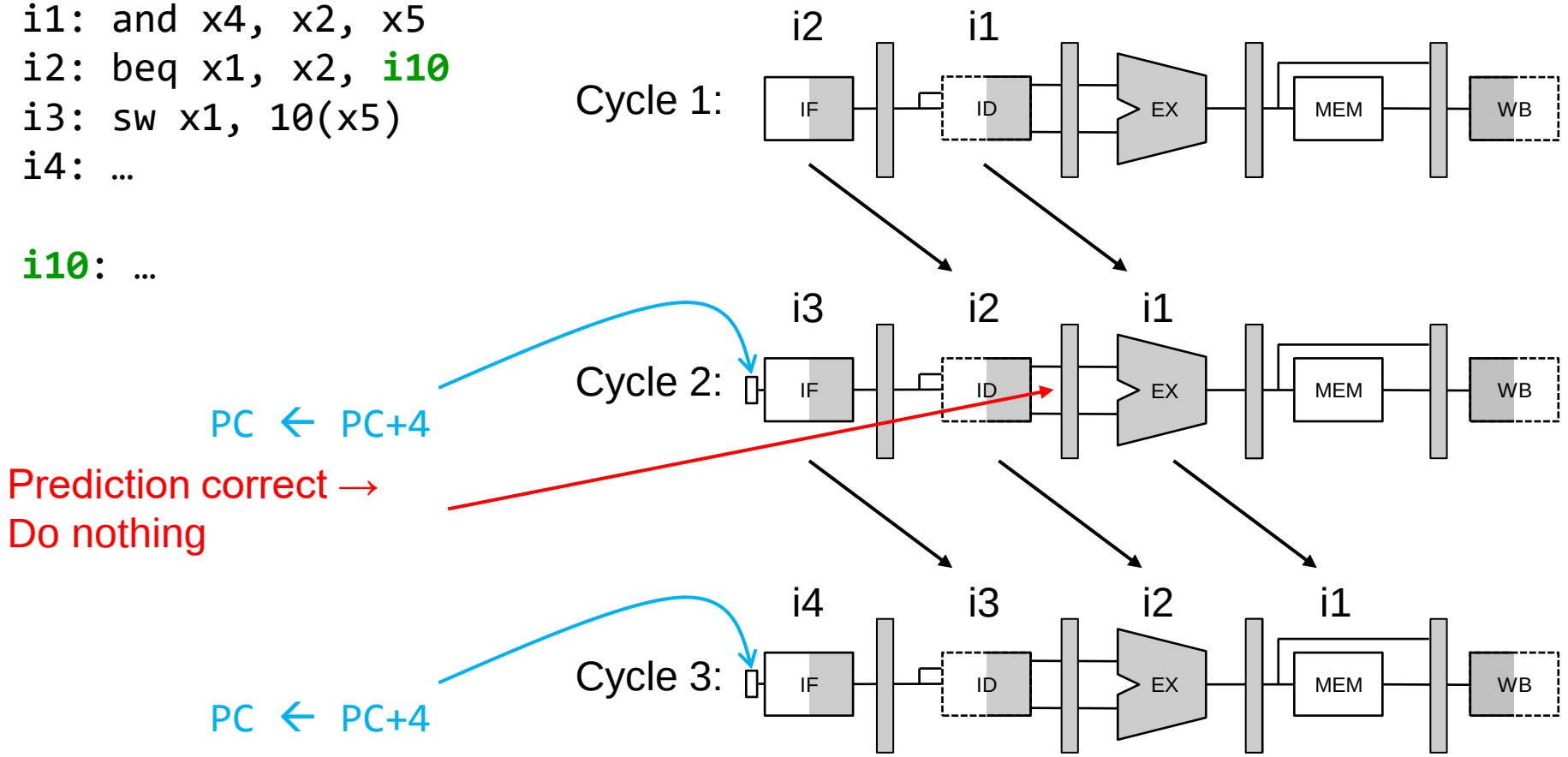


Prediction was incorrect



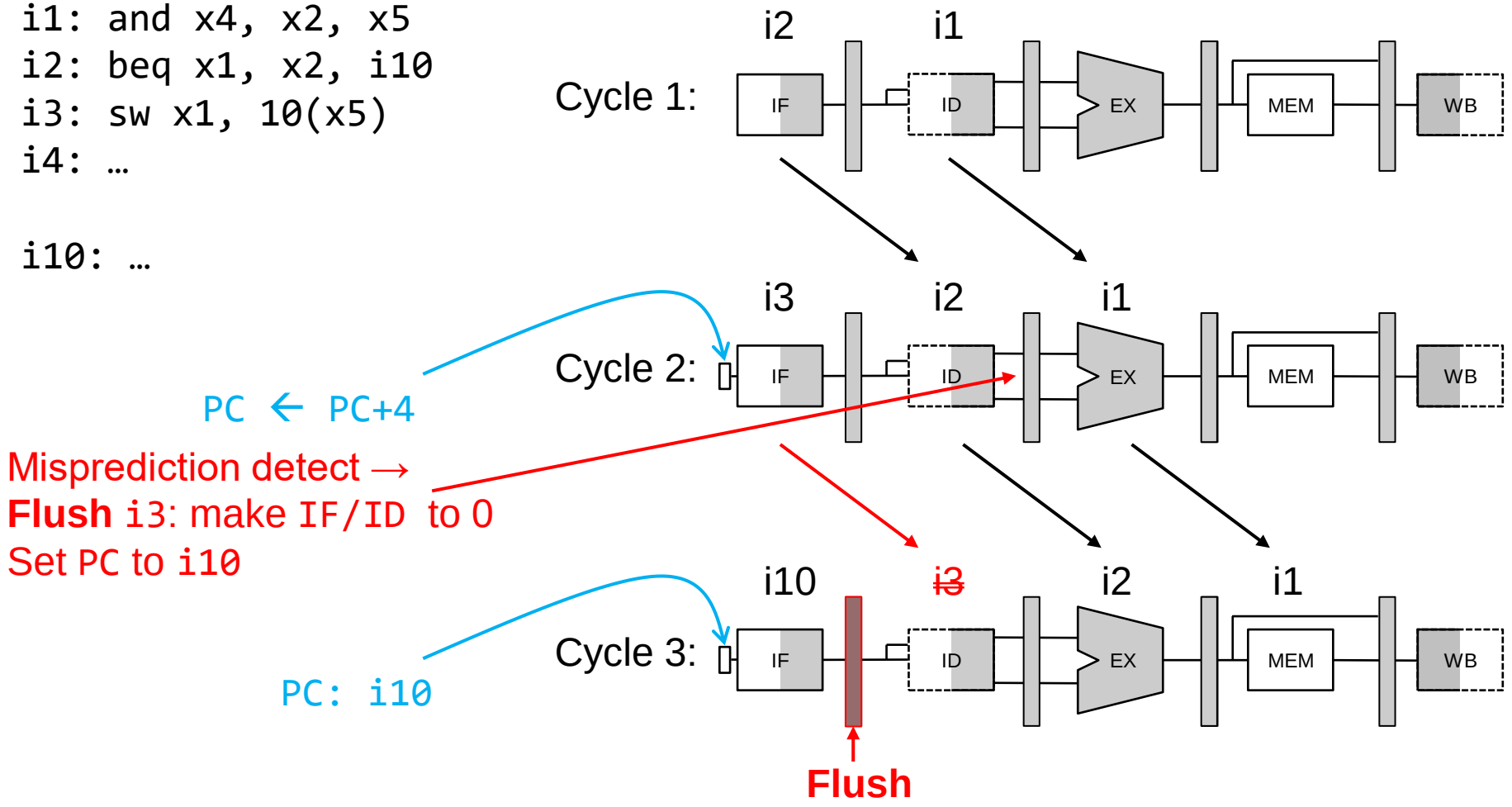
Branch Prediction: Always Not Taken

```
i1: and x4, x2, x5
i2: beq x1, x2, i10
i3: sw x1, 10(x5)
i4: …

i10: …
```

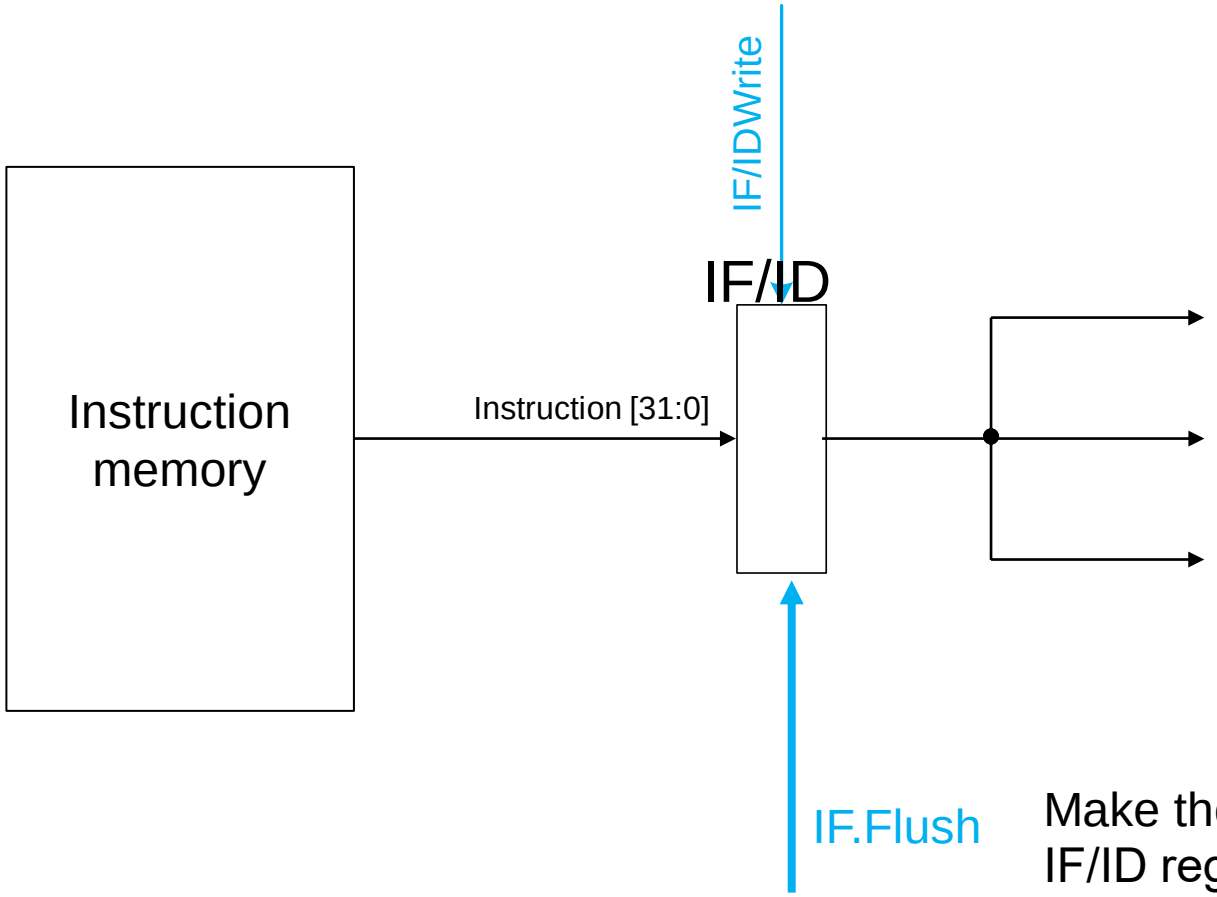


Flushing Mispredicted Branch Result

```
i1: and x4, x2, x5
i2: beq x1, x2, i10
i3: sw x1, 10(x5)
i4: …

i10: …
```



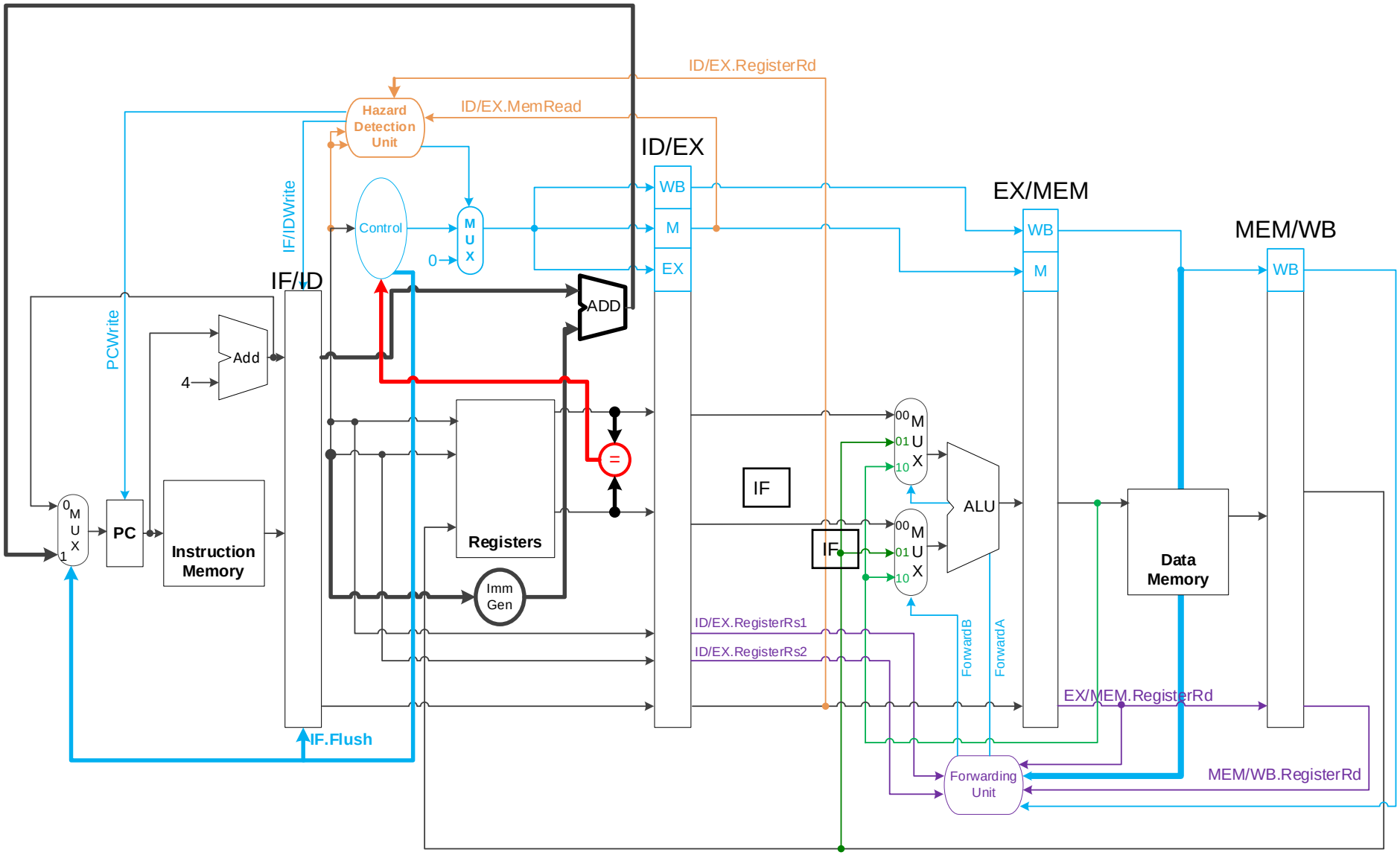
Adding a Bubble While Canceling the Inst.



Make the instruction stored in IF/ID register to “0x00000013”

```
0x00000013 = addi x0, x0, 0 = nop
```

Pipeline with Mispredicted Branch Flushing



More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Backward branches → Predict to be **taken**
    - Forward branches → Predict to be **not taken**

- Dynamic branch prediction
  - Assume the future will be similar to the current trend
  - Hardware monitors the current branch behaviors
    - e.g., record the recent history of branch outcomes of each branch instructions
  - Predict the branch outcome based on the recorded information

Dynamic Branch Prediction

- **Branch Prediction Buffer** (a.k.a. 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 the fall-through or target
  - If wrong, flush pipeline and flip prediction

Branch Prediction Buffer

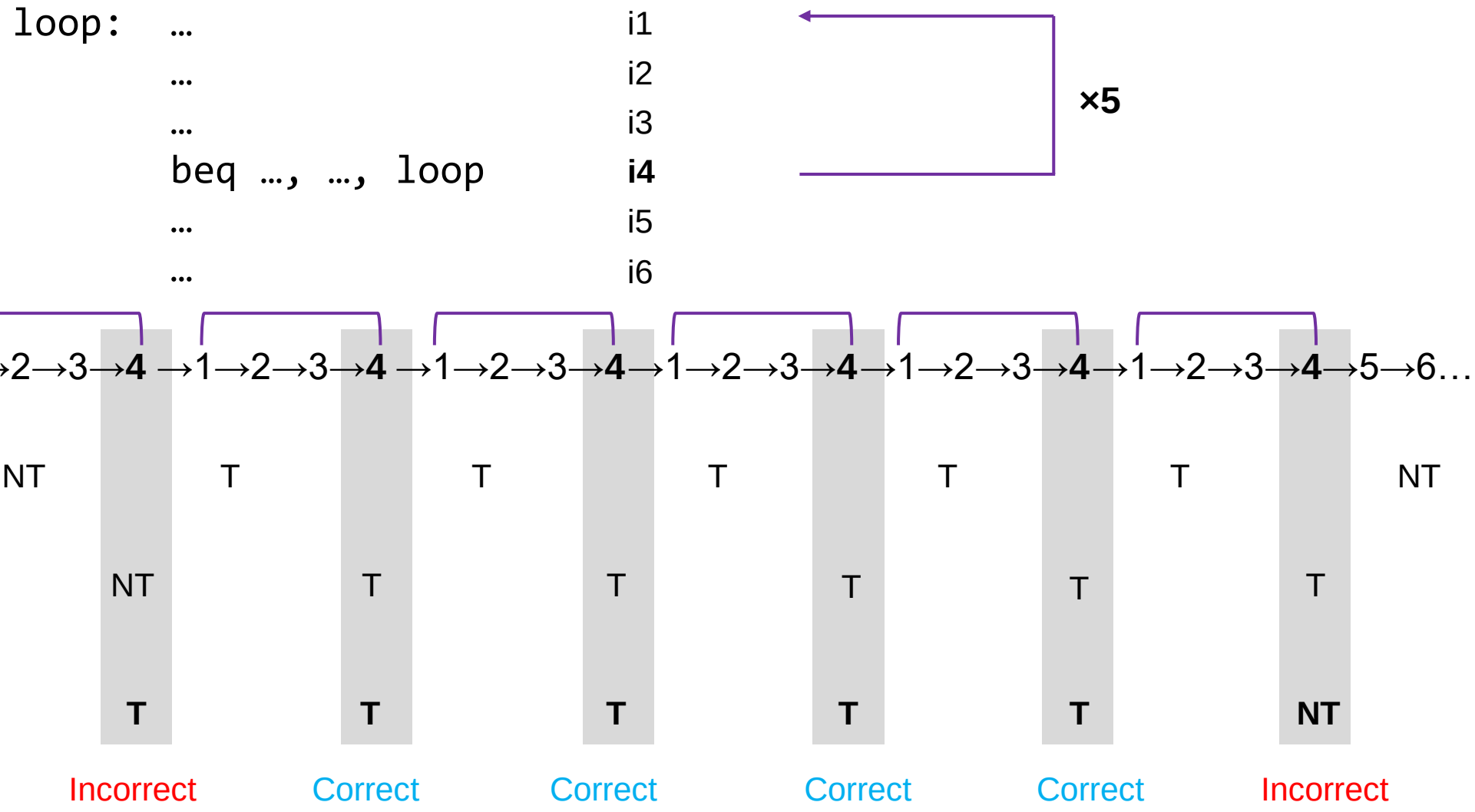
Instruction Memory

Address	Instruction
…	…
0x1000	**beq**
0x1004	lw
0x1008	**beq**
0x100c	**beq**
0x1010	add
0x1014	**beq**
0x1018	nop
0x101c	**beq**
…	…
…	…

Branch Prediction Buffer

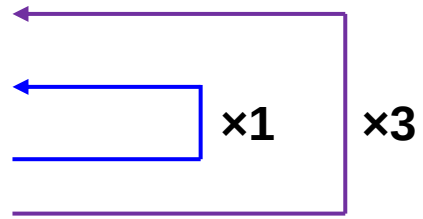
Last branch result
…
0 (not taken)
0
1 (taken)
0
0
1
0
0
…
…

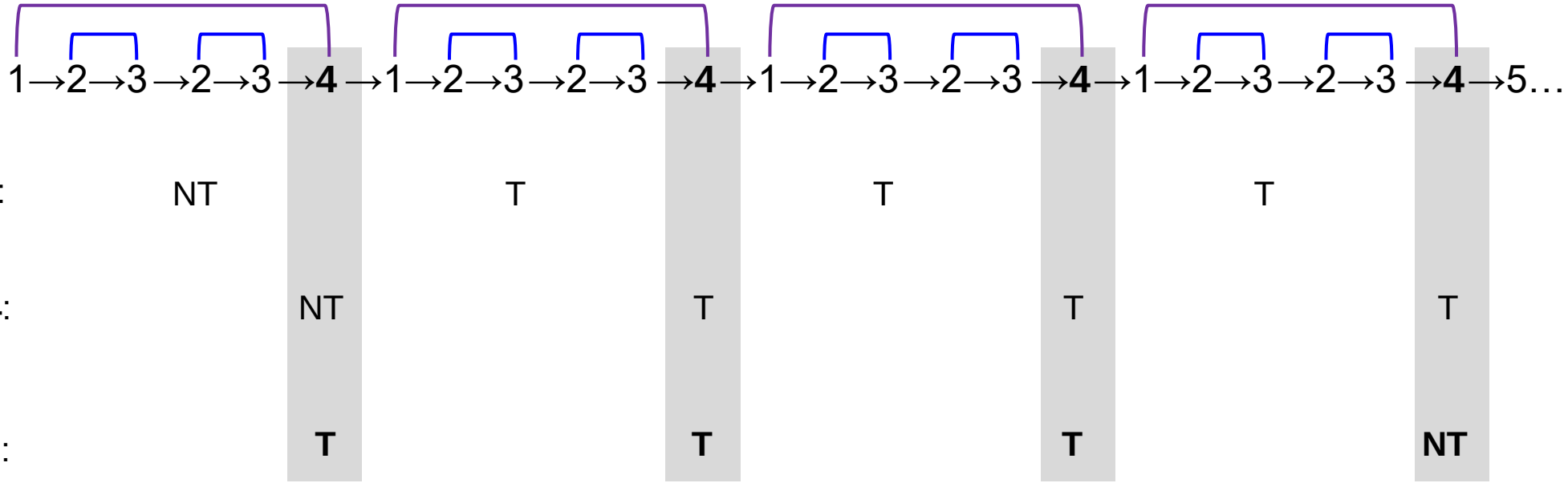
1-Bit Predictor



1-Bit Predictor: Limitations

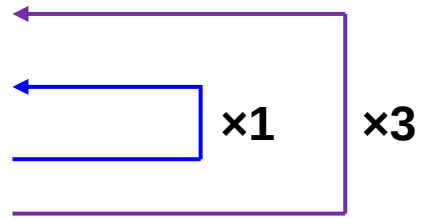
```
outer: …                        i1
inner: …                        i2
       beq …, …, inner          i3
       beq …, …, outer          i4
       …                        i5
```

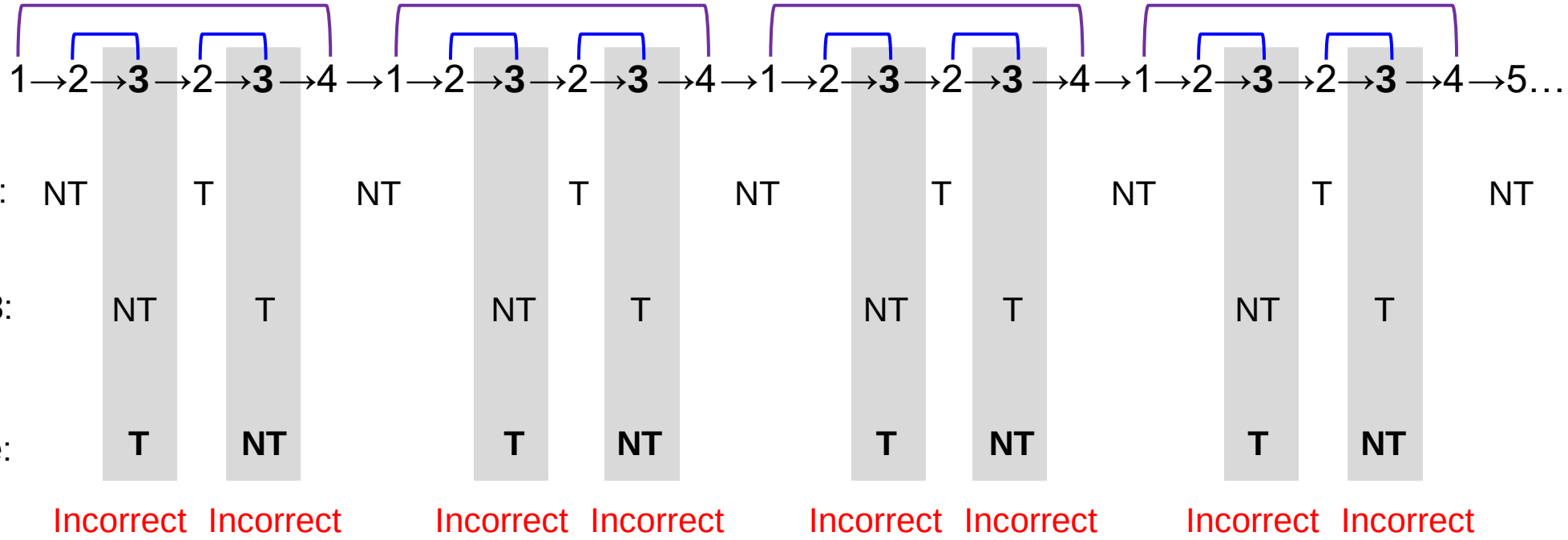




1-Bit Predictor: Limitations

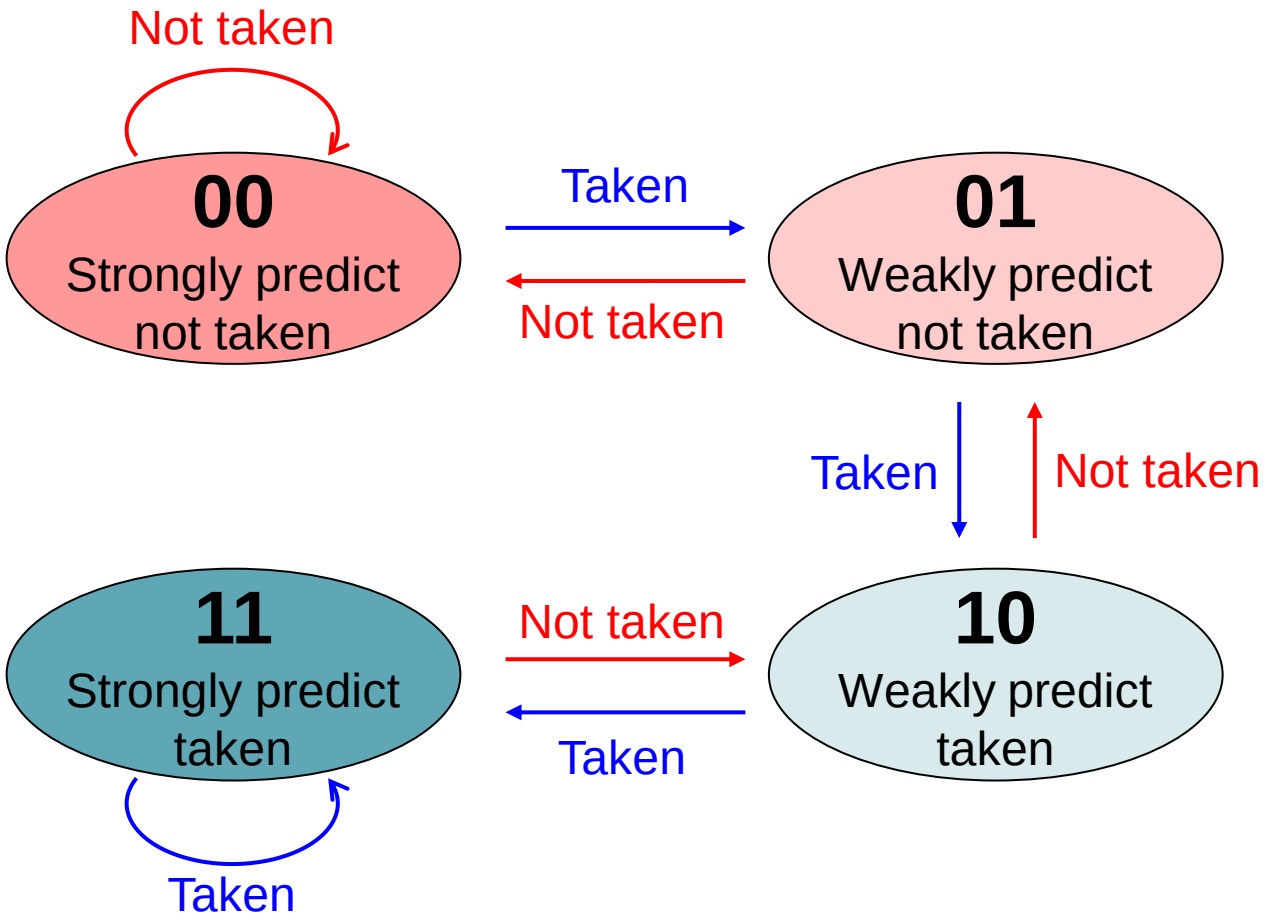
```
outer: …                          i1
inner: …                          i2
       beq …, …, inner            i3
       beq …, …, outer            i4
       …                          i5
```





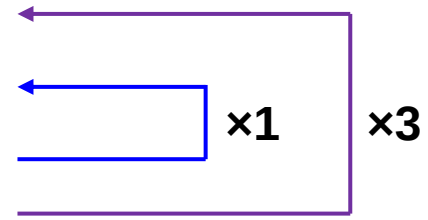
2-Bit Predictor

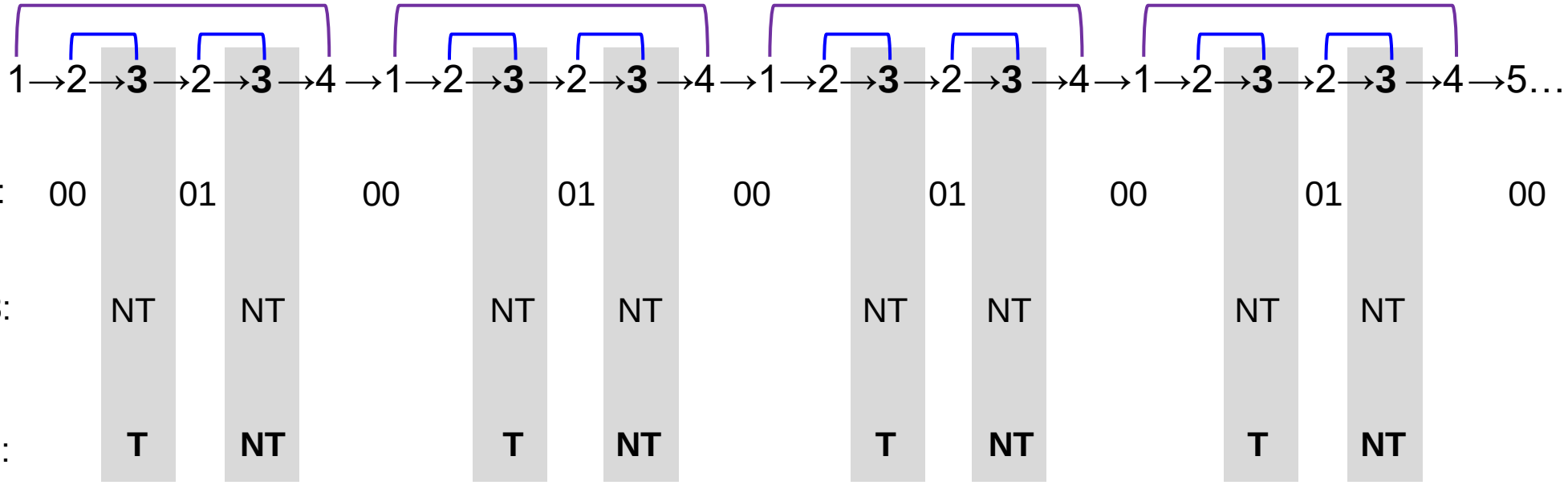
- Only change prediction on two successive mispredictions



2-Bit Predictor

```
outer: …                        i1
inner: …                        i2
       beq …, …, inner          i3
       beq …, …, outer          i4
       …                        i5
```



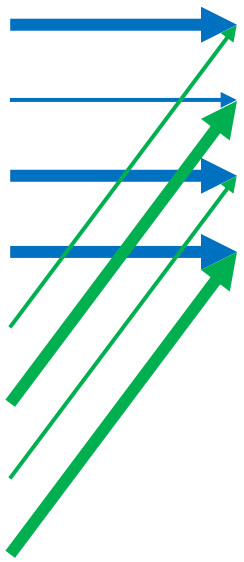


Branch Prediction Buffer

- `sizeof(instruction memory) >> sizeof(prediction buffer)`
  - Multiple instructions are mapped to a single entry
  - Aliasing (i.e., collision) happens

Instruction Memory

Address	Instruction
…	…
0x1000	beq
0x1004	lw
0x1008	beq
0x100c	beq
0x1010	add
0x1014	beq
0x1018	nop
0x101c	beq
…	…

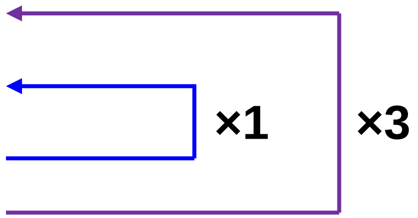


Branch Prediction Buffer

Last branch result	
0x0	00
0x4	11
0x8	10
0xc	01

2-Bit Predictor

```
outer: …                        i1
inner: …                        i2
       beq …, …, inner          i3
       beq …, …, outer          i4
       …                        i5
```

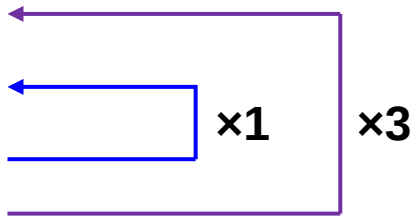


1→2→**3**→2→**3** → **4**→1→2→**3**→2→**3** → **4**→1→2→**3**→2→**3** → **4**→1→2→**3**→2→**3** → **4**→5…

BPB entry for i3&i4: (2-bit)	00	01	00	01	10	01	10	11	10	11	11	10	01
Prediction:	NT	NT	NT	NT	T	NT	T	T	T	T	T	T	
Branch outcome:	**T**	**NT**	**T**	**T**	**NT**	**T**	**T**	**NT**	**T**	**T**	**NT**	**NT**	
	I	C	I	I	I	I	C	I	C	C	I	I	

2-Bit Predictor

```
outer: …                          i1
inner: …                          i2
       beq …, …, inner            i3
       beq …, …, outer            i4
       …                          i5
```



	1	→	2	→	**3**	→	2	→	**3**	→	**4**	→	1	→	2	→	**3**	→	2	→	**3**	→	**4**	→	1	→	2	→	**3**	→	2	→	**3**	→	**4**	→	5…
BPB entry for i3&i4: (2-bit)	01				10				01				10				11				10				11				11				10				01
Prediction:			NT				T				NT				T				T				T				T				T				T	
Branch outcome:			**T**				**NT**				**T**				**T**				**NT**				**T**				**T**				**NT**				**NT**	
			I				I				I				C				I				C				C				I				C				C				I				I	

Calculating the Branch Target

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch

- **Branch Target Buffer**
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately