Computer Systems II

Li Lu

Room 605, CaoGuangbiao Building

li.lu@zju.edu.cn
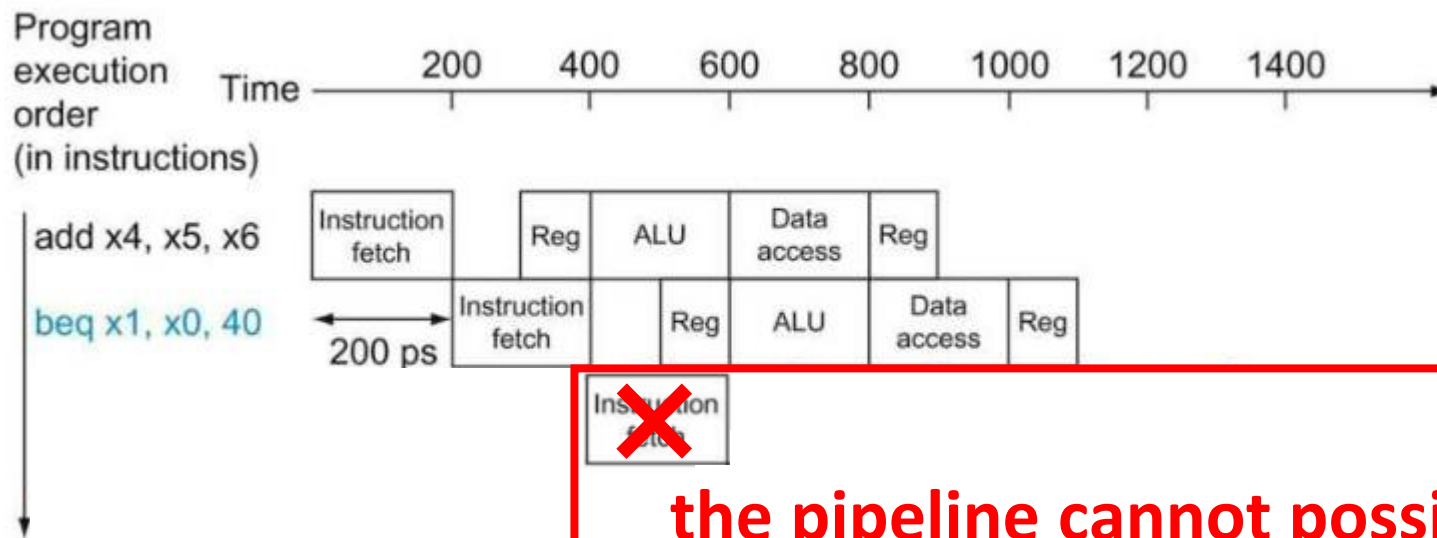
https://person.zju.edu.cn/lynnluli

Control Hazards

Control Hazard

Flow of execution depends on previous instruction
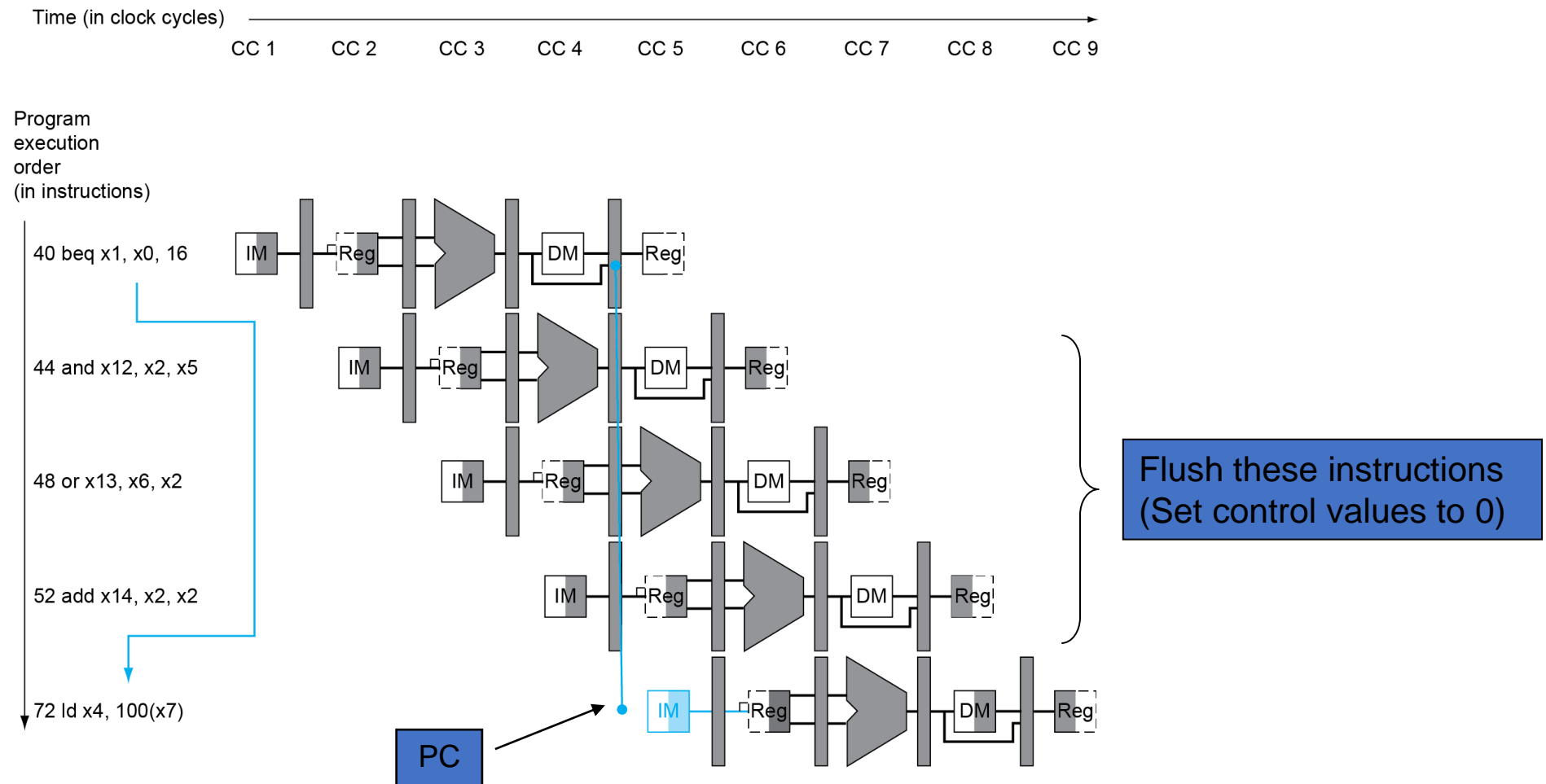
Problem: The conditional branch instruction

**the pipeline cannot possibly know what the next instruction should be**

Branch Hazards

- If branch outcome determined in MEM

Time (in clock cycles)

CC 1   CC 2   CC 3   CC 4   CC 5   CC 6   CC 7   CC 8   CC 9

Program
execution
order
(in instructions)

40 beq x1, x0, 16

44 and x12, x2, x5

48 or x13, x6, x2

52 add x14, x2, x2

72 ld x4, 100(x7)

Flush these instructions
(Set control values to 0)

PC

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipelining can't always fetch correct instruction
    - Still working on ID stage of branch
- Wait until branch outcome determined before fetching next instruction
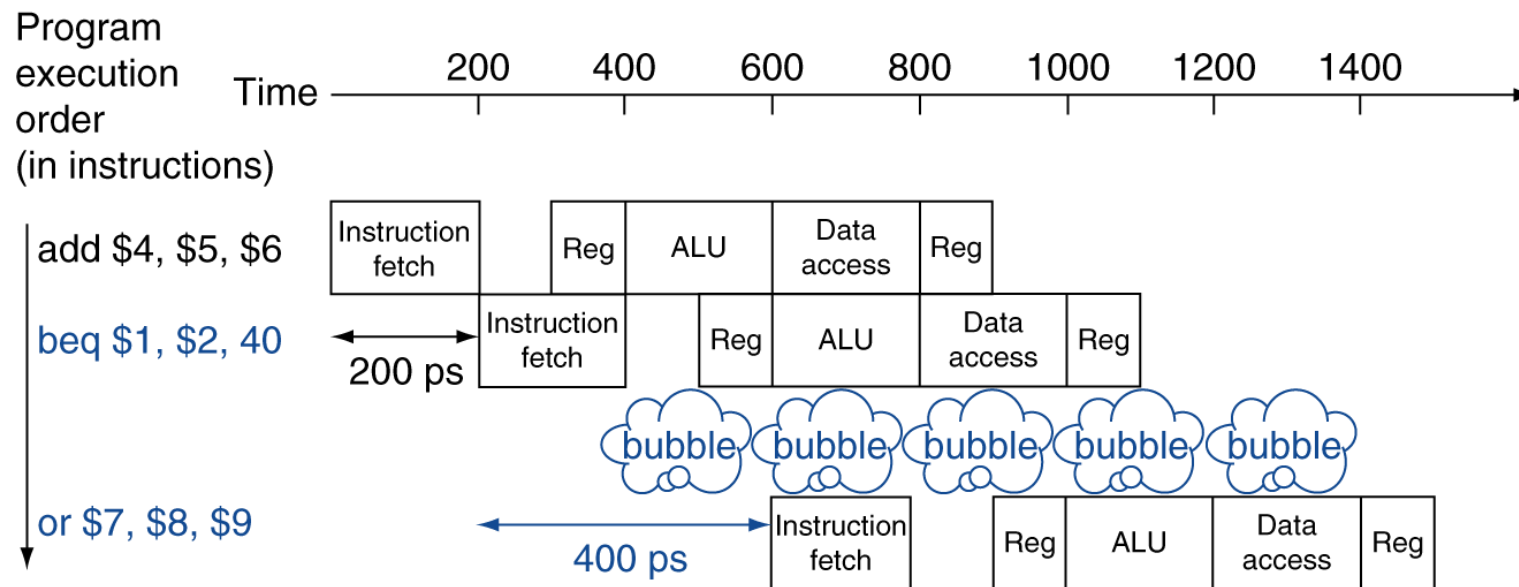
Stall on Branch

Branch taken

		IF	ID	EX	MEM	WB				
Branch		IF	ID	EX	MEM	WB				
Object			**IF**	**stall**	**stall**	IF	ID	EX	MEM	WB	
Object+1							IF	ID	EX	MEM	WB
Object+2								IF	ID	EX	MEM
Object+3									IF	ID	EX

Branch untaken

		IF	ID	EX	MEM	WB				
Branch		IF	ID	EX	MEM	WB				
subsequent			IF	ID	EX	MEM	WB			
subsequent+1				IF	ID	EX	MEM	WB		
subsequent+2					IF	ID	EX	MEM	WB	
subsequent+3						IF	ID	EX	MEM	WB
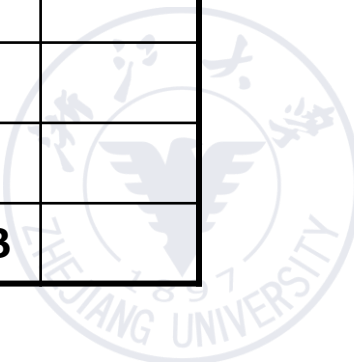
How to Reduce Stall

- In RISC-V pipelining
  - Need to **compare registers** and **compute target** early in the pipelining
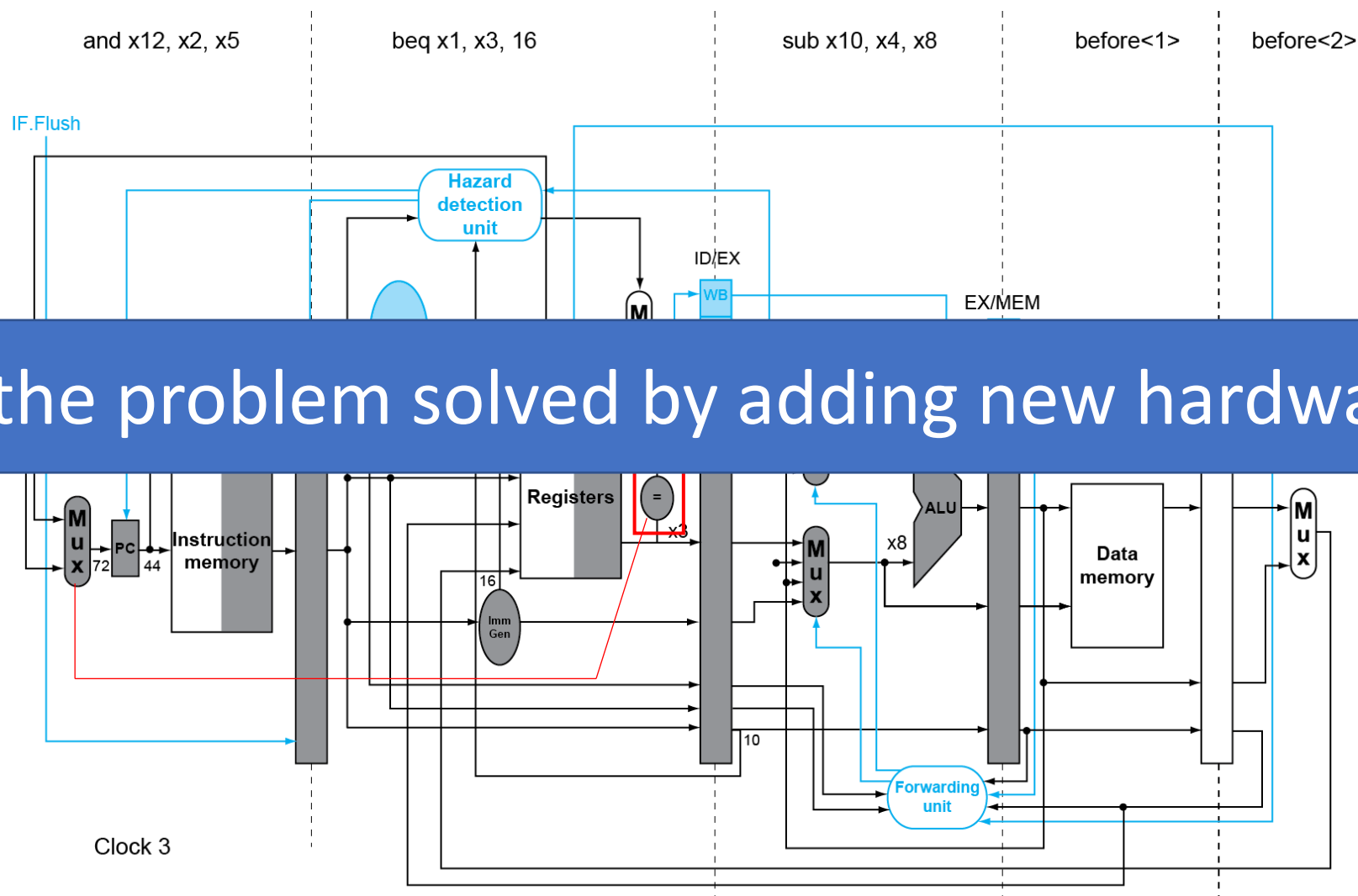 - Add hardware to do it in ID stage

What is the hardware?

- Key processes in branch instructions
  - Compute the branch target address
  - Judge if the branch success

Which stages do they happen?

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator

Forwarding Branch to Earlier Stage

Is the problem solved by adding new hardware?

Anything wrong?

Stall on Branch with Optimized Solution

No!

Branch still causes a stall

Branch		IF	ID	EX	MEM	WB				
subsequent			IF	IF	ID	EX	MEM	WB		
subsequent+1					IF	ID	EX	MEM	WB	
subsequent+2						IF	ID	EX	MEM	WB
subsequent+3							IF	ID	EX	MEM	WB

Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable

- Predict outcome of branch
  - Only stall if prediction is wrong

- In RISC-V pipeline
  - Can predict branches not taken
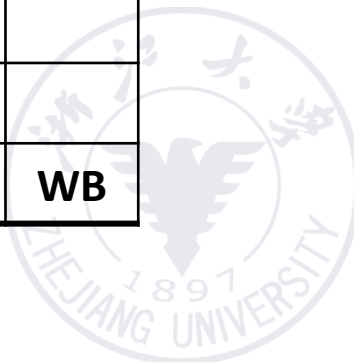 - Fetch instruction after branch, with no delay

Reducing Branch Delay

- Predict branch taken

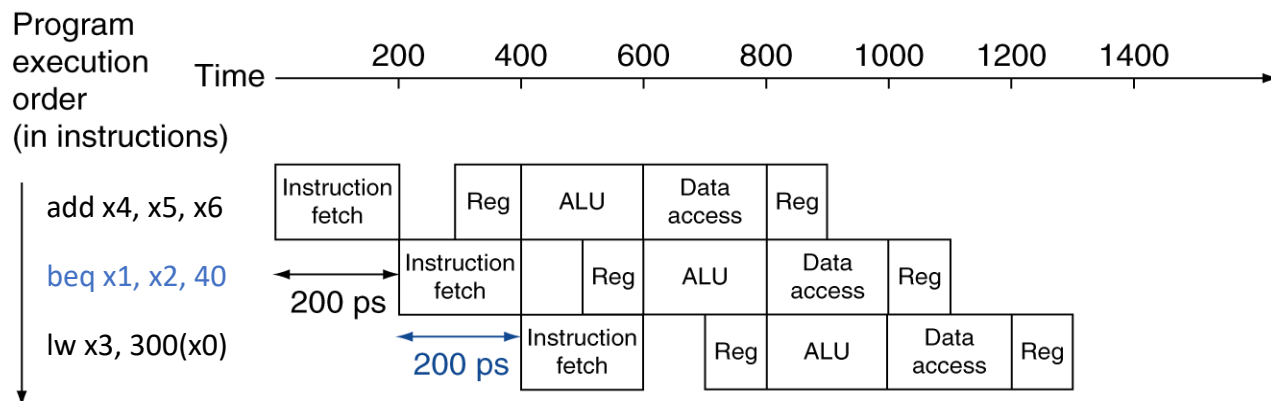- Predict branch untaken

- Delay Branch

Predict branch

Branch i (Taken)	IF	ID	EX	MEM	WB			
i+1		IF	stall	stall	stall	stall		
Object j			IF	ID	EX	MEM	WB	
Object j+1				IF	ID	EX	MEM	WB
Object j+2					IF	ID	EX	MEM	WB

Branch i (Untaken)	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM	WB

RISC-V with Predict Not Taken
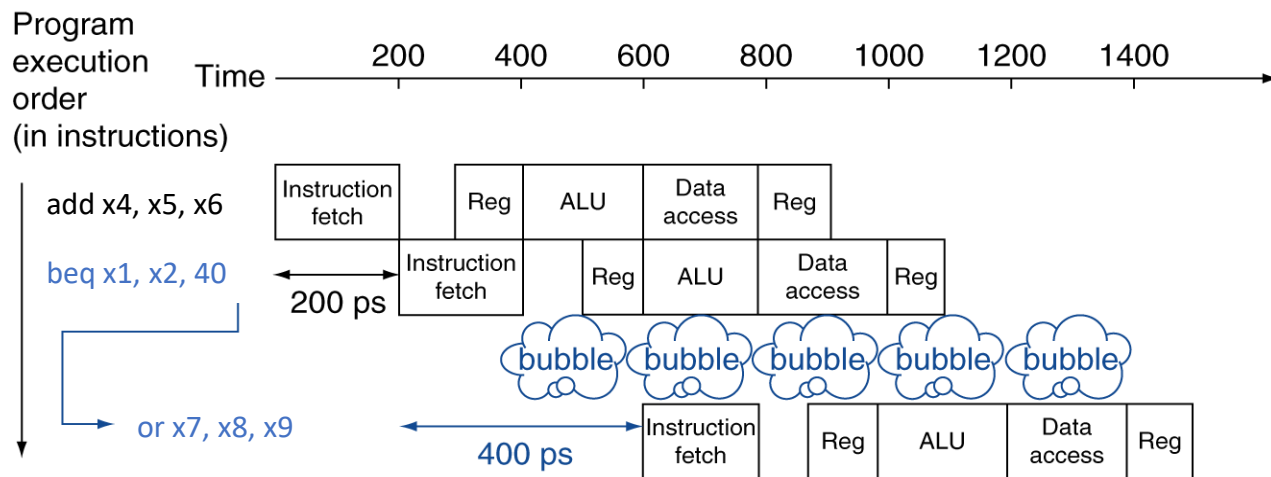
How to Reduce Branch Delay

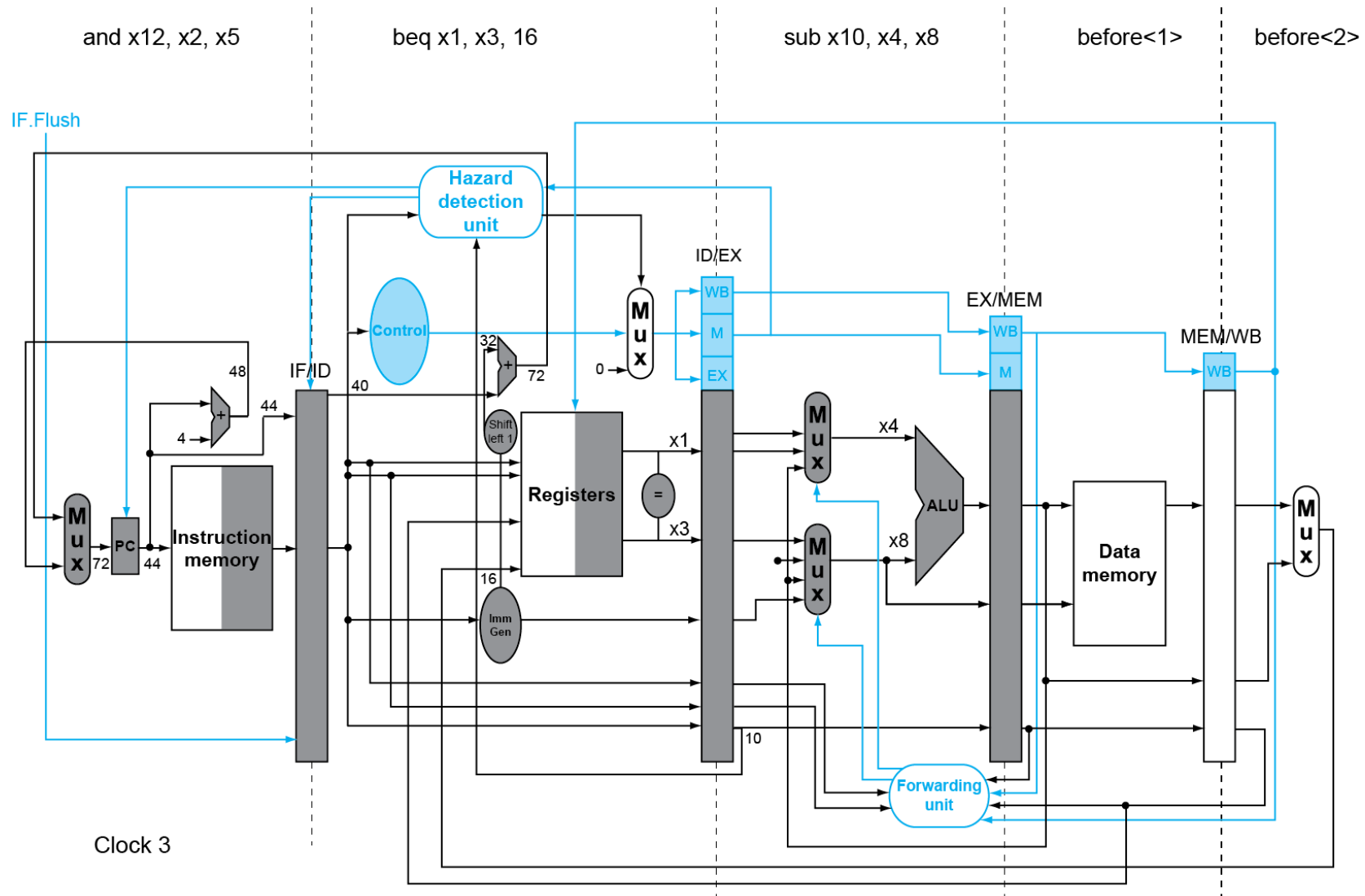- Example: branch taken

```
36:  sub  x10, x4, x8
40:  beq  x1,  x3, 16    // PC-relative branch
                         // to 40+16*2=72
44:  and  x12, x2, x5
48:  or   x13, x2, x6
52:  add  x14, x4, x2
56:  sub  x15, x6, x7
     ...
72:  ld   x4, 50(x7)
```
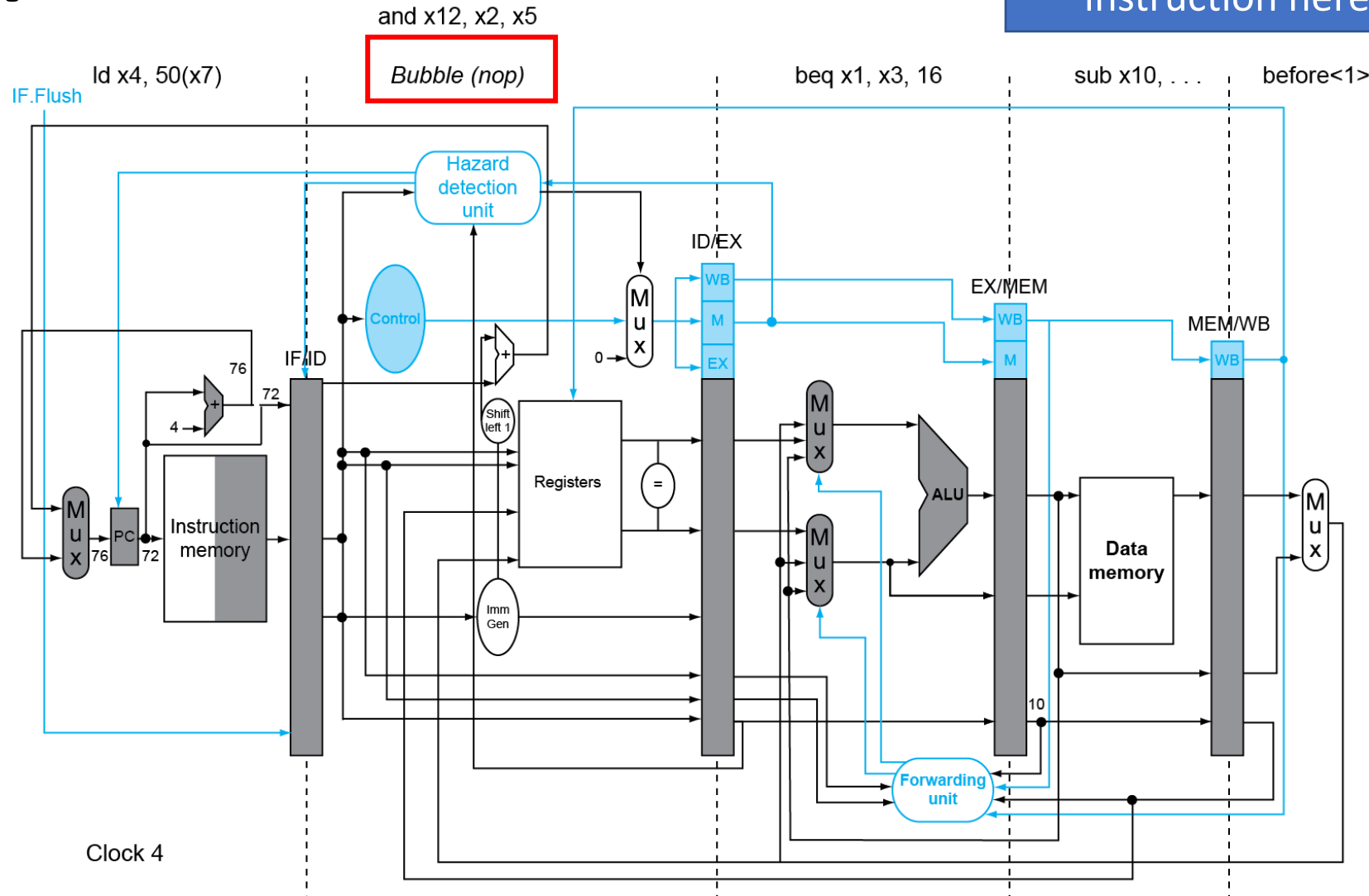
Example: Branch Taken

Example: Branch Taken

What is the original instruction here?

and x12, x2, x5

ld x4, 50(x7)   Bubble (nop)   beq x1, x3, 16   sub x10, . . .   before<1>
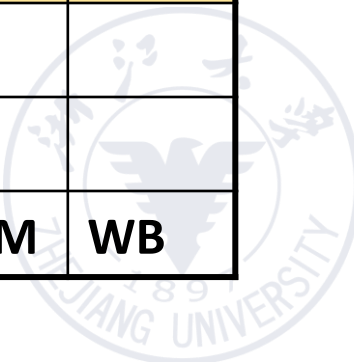

Clock 4

Reducing Branch Delay

- Predict branch success

- Predict branch failure

- Delay Branch

Pipelining with a branch delay slot

Branch Failure	Branch i	IF	ID	EX	MEM	WB				
	Delay slot i+1		IF	ID	EX	MEM	WB			
	i+2			IF	ID	EX	MEM	WB		
	i+3				IF	ID	EX	MEM	WB	
	i+4					IF	ID	EX	MEM	WB

Branch Success	Branch i	IF	ID	EX	MEM	WB				
	Delay slot i+1		IF	ID	EX	MEM	WB			
	Object j			IF	ID	EX	MEM	WB		
	Object j+1				IF	ID	EX	MEM	WB	
	Object j+2					IF	ID	EX	MEM	WB

Code Scheduling

(a) Scheduling from former   (b) Scheduling from object   (c) Scheduling from failure
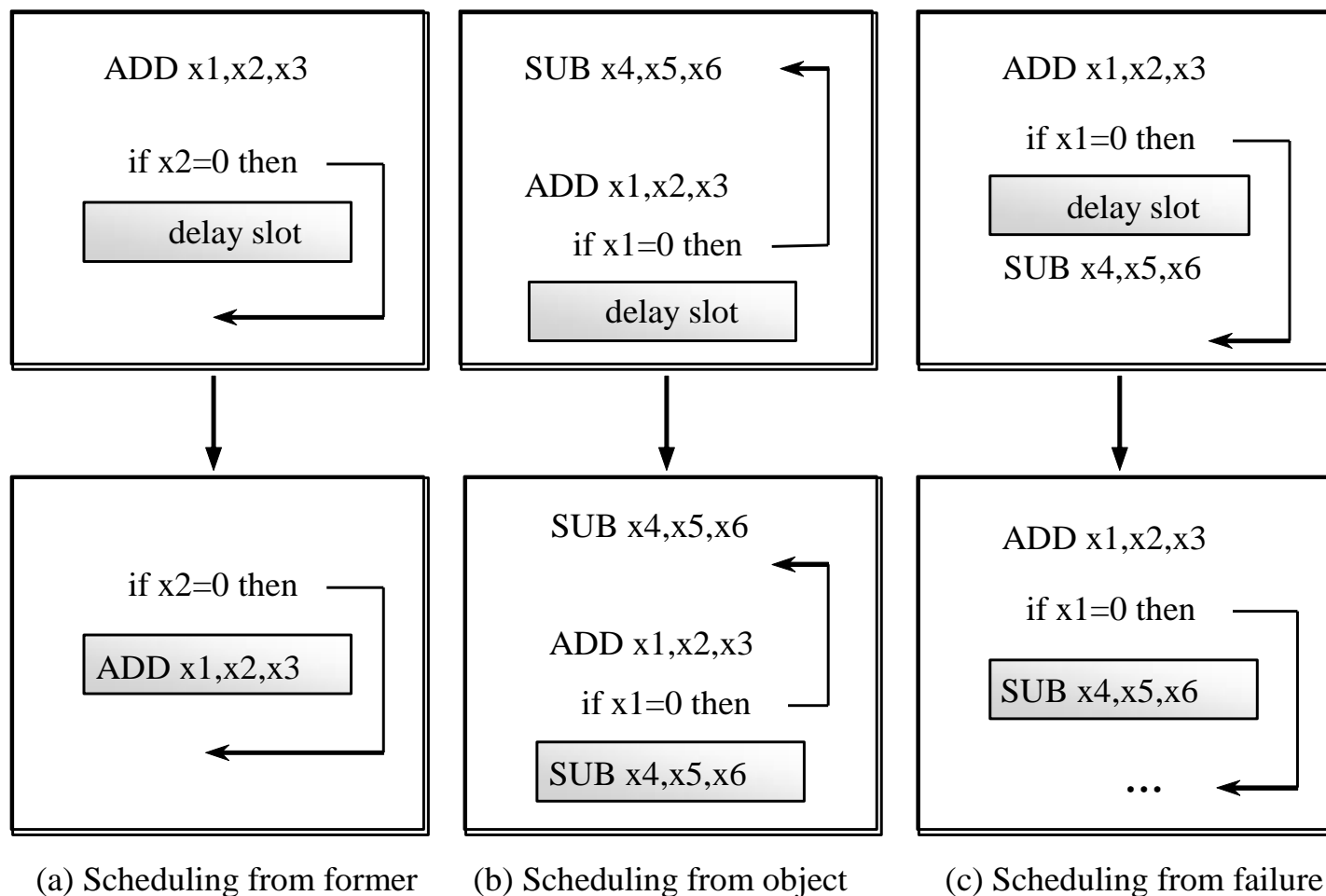
Code Scheduling

Branch Failure	Branch i	IF	ID	EX	MEM	WB				
	Delay slot i+1		IF	idle	idle	idle	idle			
	i+2			IF	ID	EX	MEM	WB		
	i+3				IF	ID	EX	MEM	WB	
	i+4					IF	ID	EX	MEM	WB

Branch Success	Branch i	IF	ID	EX	MEM	WB				
	Delay slot i+1		IF	ID	EX	MEM	WB			
	Object j			IF	ID	EX	MEM	WB		
	Object j+1				IF	ID	EX	MEM	WB	
	Object j+2					IF	ID	EX	MEM	WB

Data Hazards for Branches

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

add x1, x2, x3

add x4, x5, x6

…
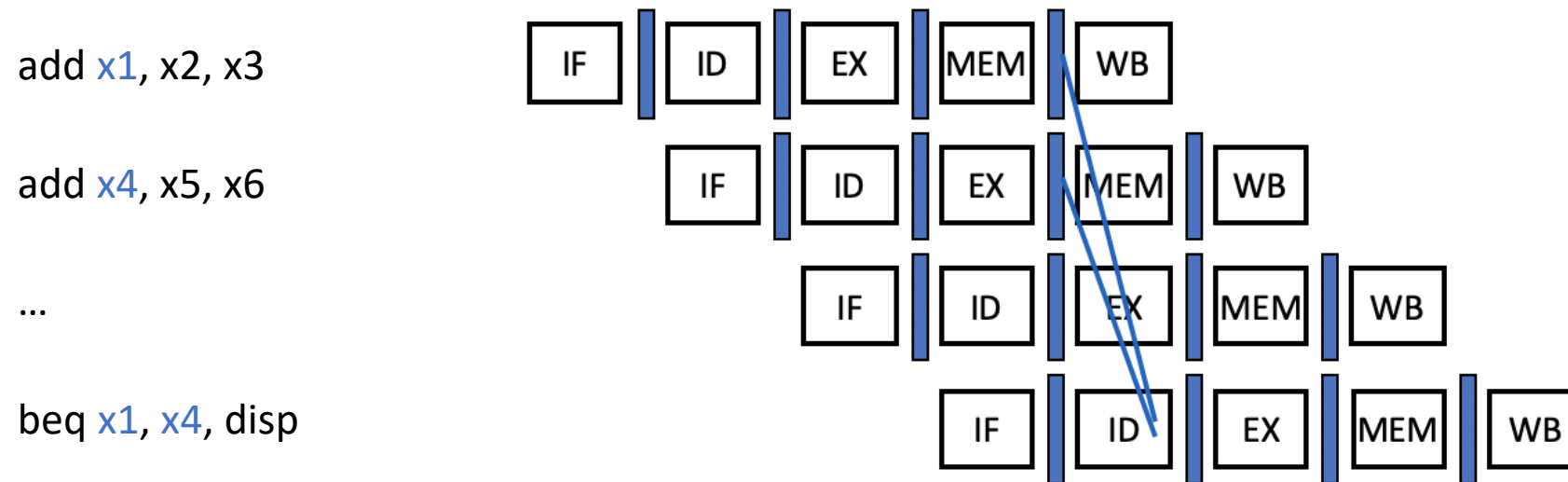
beq x1, x4, disp

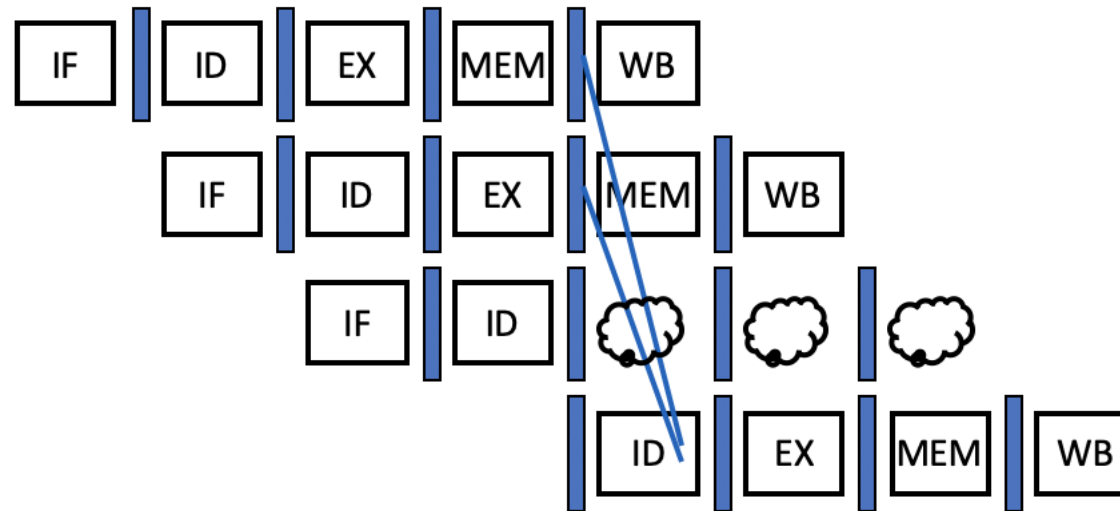- Can resolve using forwarding

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 x1, addr

add x4, x5, x6
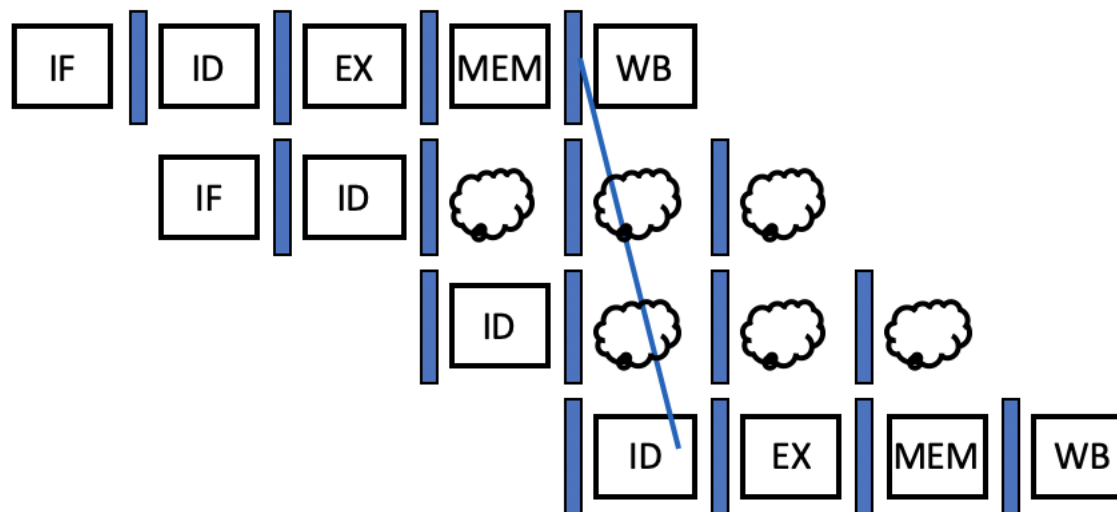
beq stalled

beq x1, x4, disp

Data Hazards for Branches

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

lw x1, addr

beq stalled

beq stalled

beq x1, x0, disp

Question: Is delay slot a really good design?

Review

- "A *RISC-V ISA* is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA.

- The base integer ISAs are very similar to that of the early RISC processors except *with no branch delay slots* and with support for optional variable-length instruction encodings. "

——The RISC-V Instruction Set Manual Volume I

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

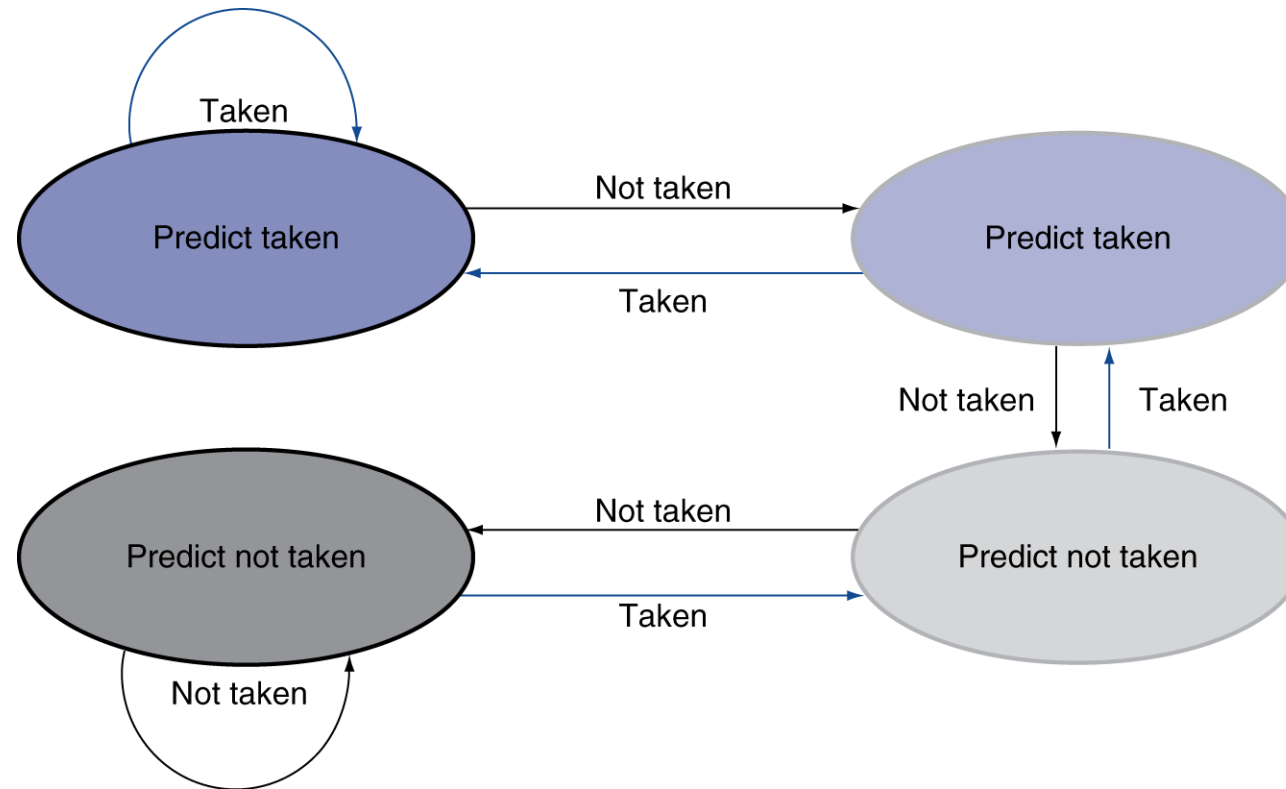# Branch History Table (BHT)
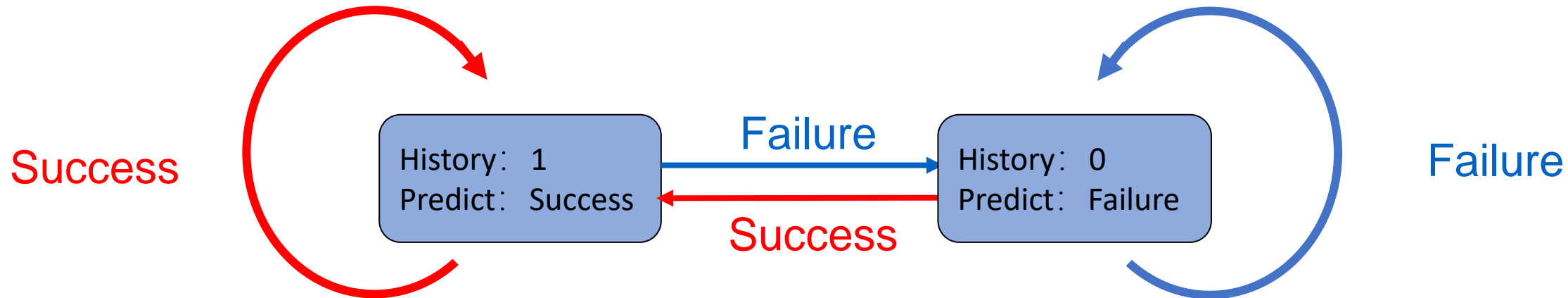
1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```
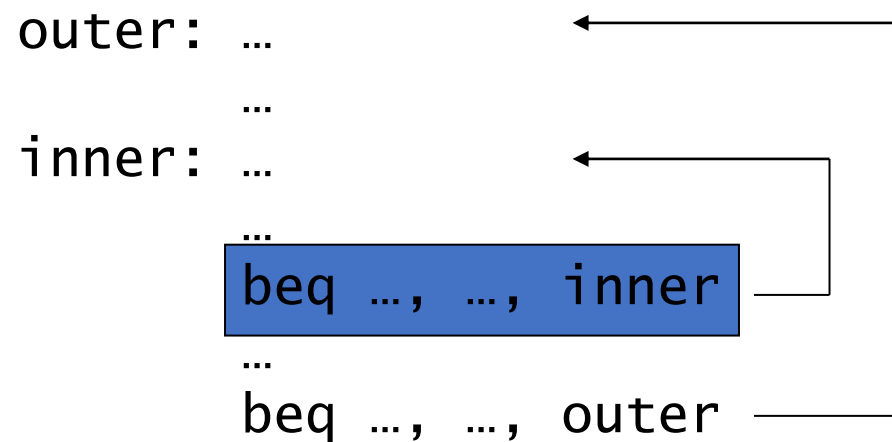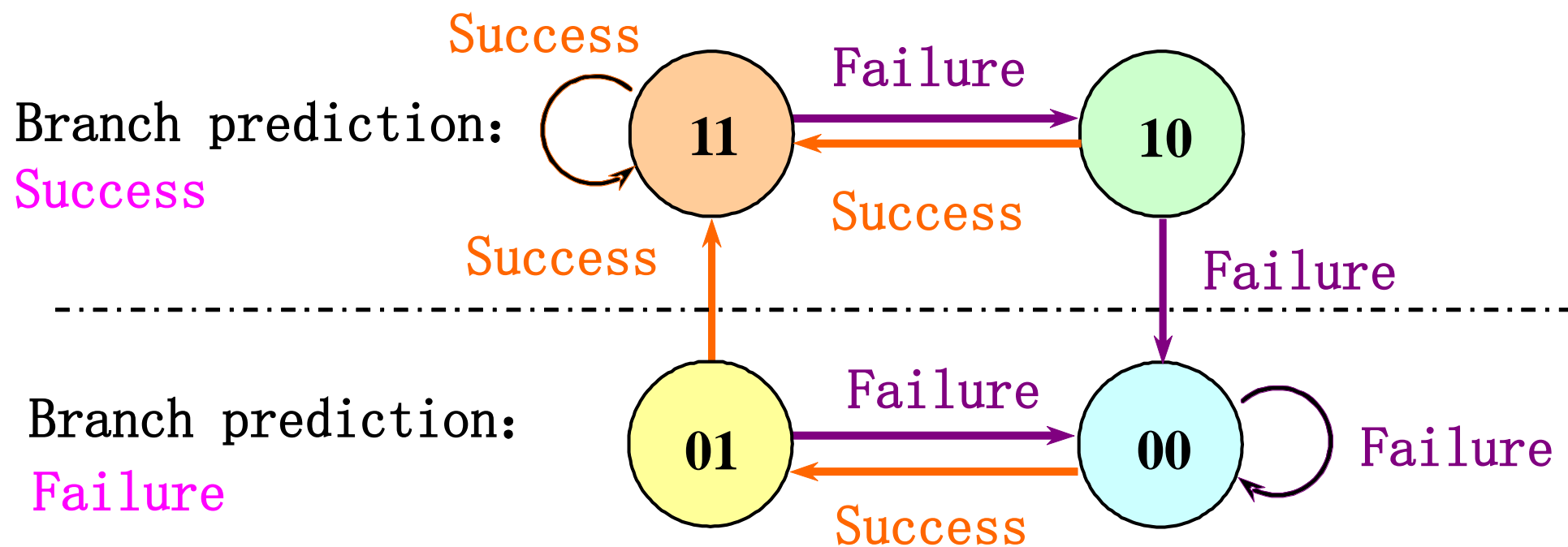
- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

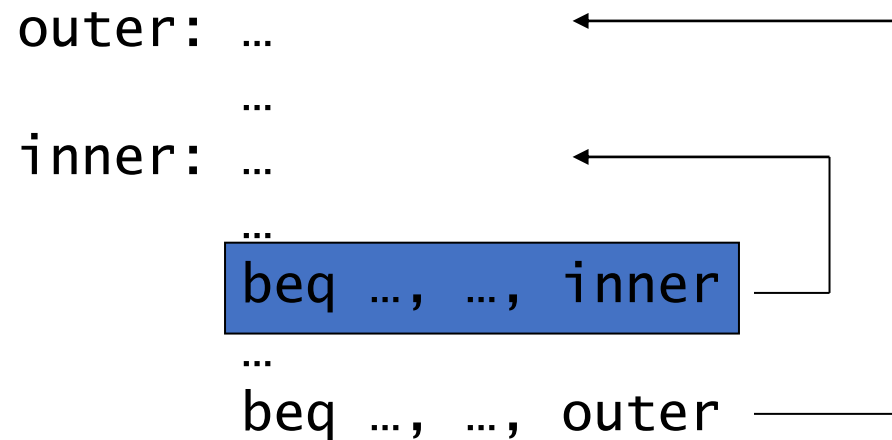- Only change prediction on two successive mispredictions

Branch History Table

2-Bit Predictor: Example

- Inner loop branches mispredicted only once!

```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```
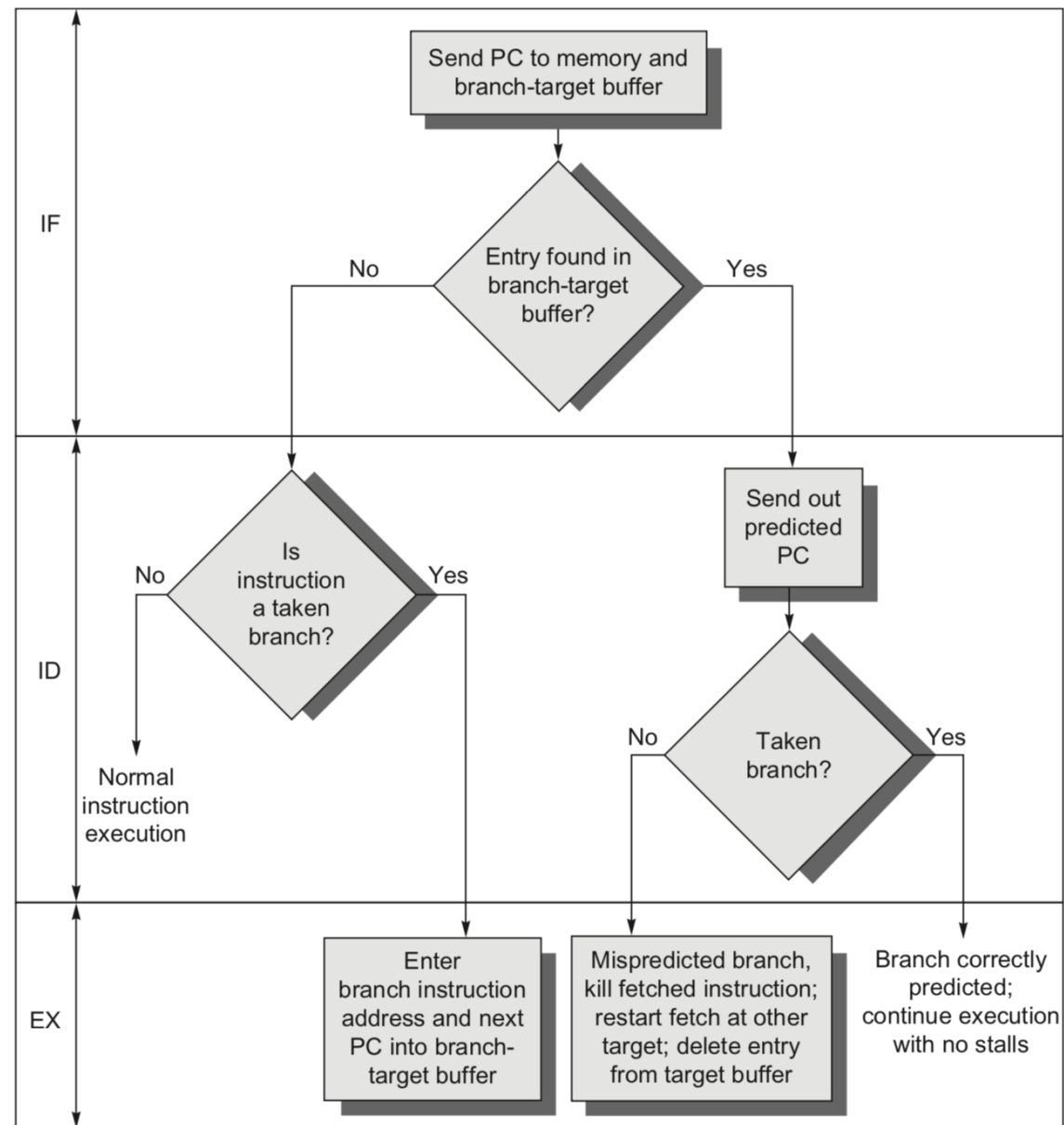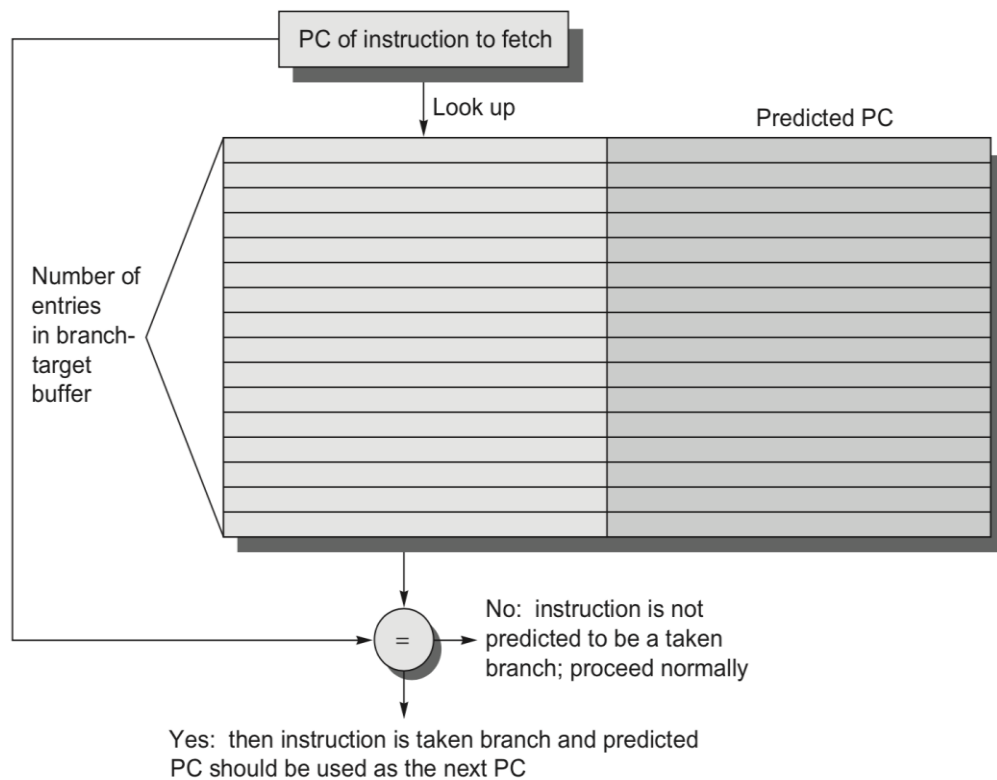
- Only mispredict as taken on last iteration of inner loop

Advanced Techniques for Instruction Delivery and Speculation

- Increasing Instruction Fetch Bandwidth

  - Branch-Target Buffers

- Specialized Branch Predictors: Predicting Procedure Returns, Indirect Jumps, and Loop Branches

  - Integrated Instruction Fetch Units
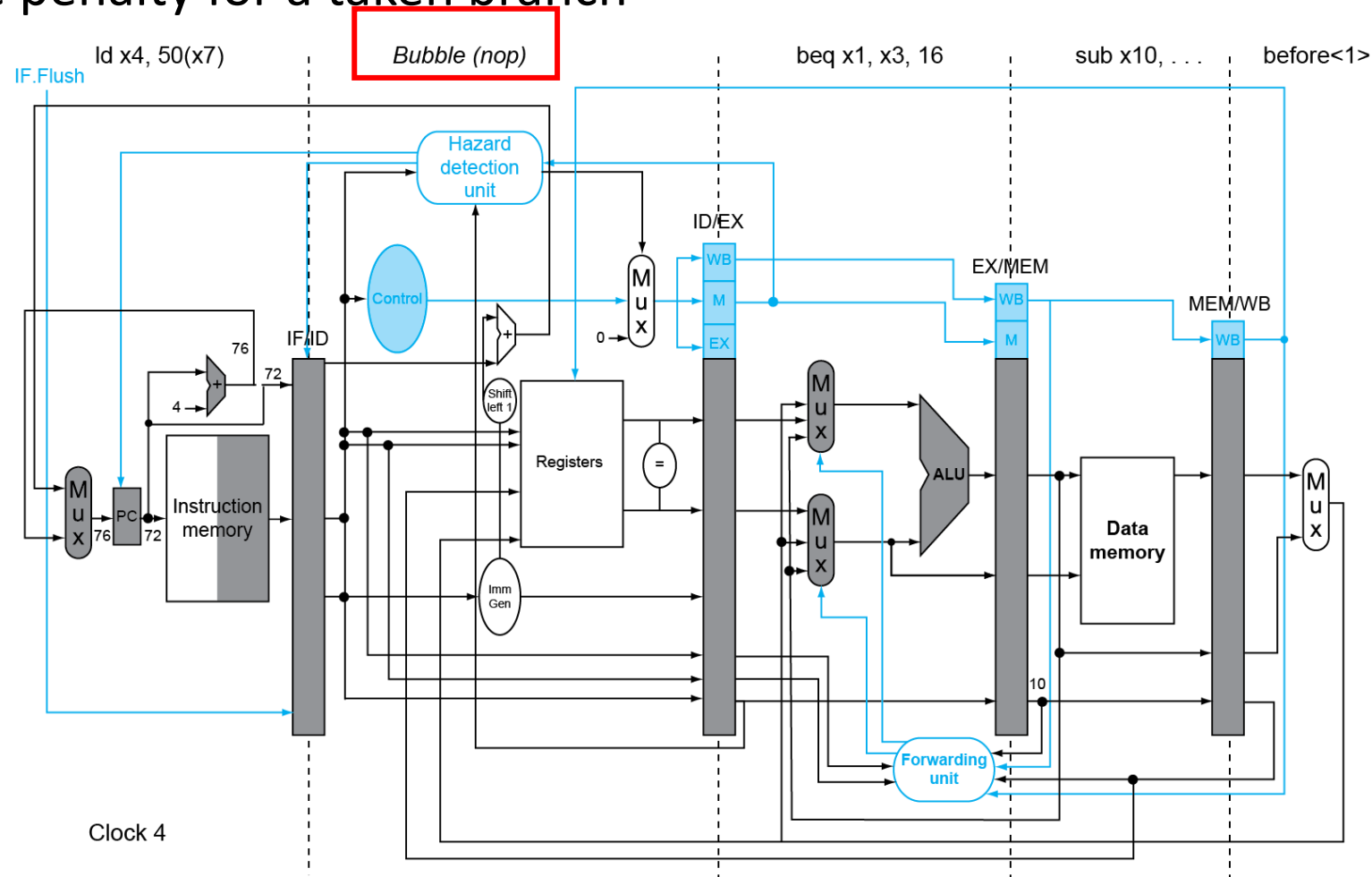
Branch-Target Buffers

Integrated Instruction Fetch Units

- An integrated instruction fetch unit that integrates several functions:

  - Integrated branch prediction
  - Instruction prefetch
  - Instruction memory access and buffering

- Instruction fetch as a simple single pipe stage given the complexities of multiple issue is no longer valid
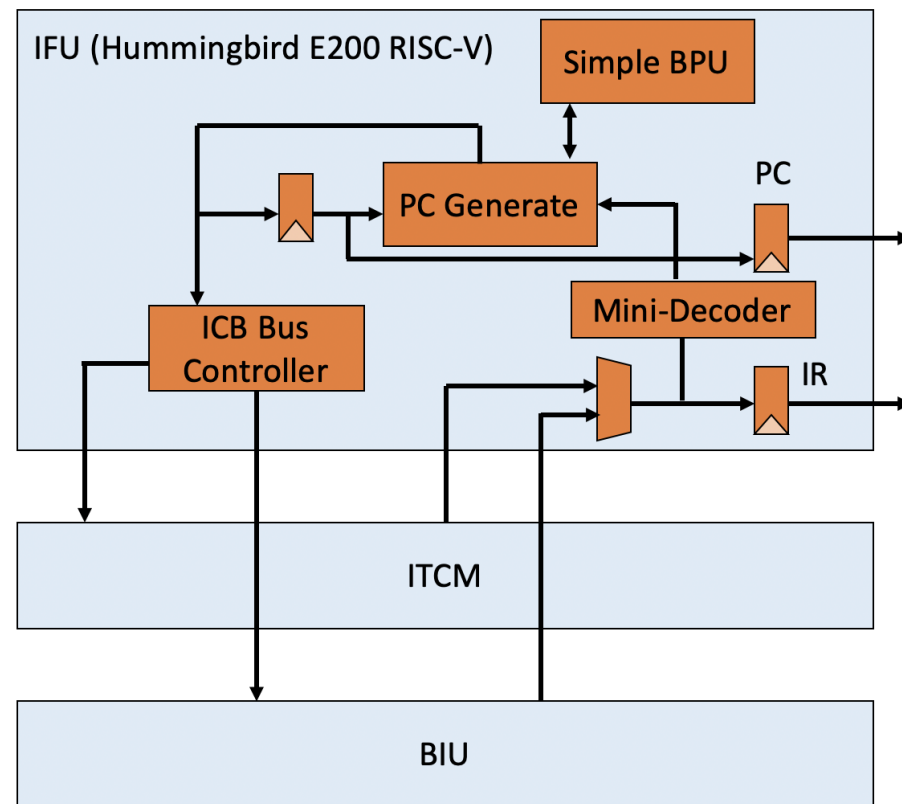
Calculating the Branch Target

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

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**

Branch-Target Buffer/Branch-Target Cache

Is instruction in BTB？	Predict	Reality	Delay cycle
Yes	Taken	Taken	0
Yes	Taken	Untaken	2
No		Taken	2
No		Untaken	0

Branch-Target Buffer/Branch-Target Cache

Benefit

- Get instructions at branch target faster
- It can provide multiple instructions at the branch target once, which is necessary for the multi processor
- branch folding
  - It is possible to achieve unconditional branching without delay, or sometimes conditional branching without delay.