Computer Systems II

Li Lu

Room 319, Yifu Business and Management Building

Yuquan Campus

li.lu@zju.edu.cn

https://person.zju.edu.cn/lynnluli

Pipeline Hazards

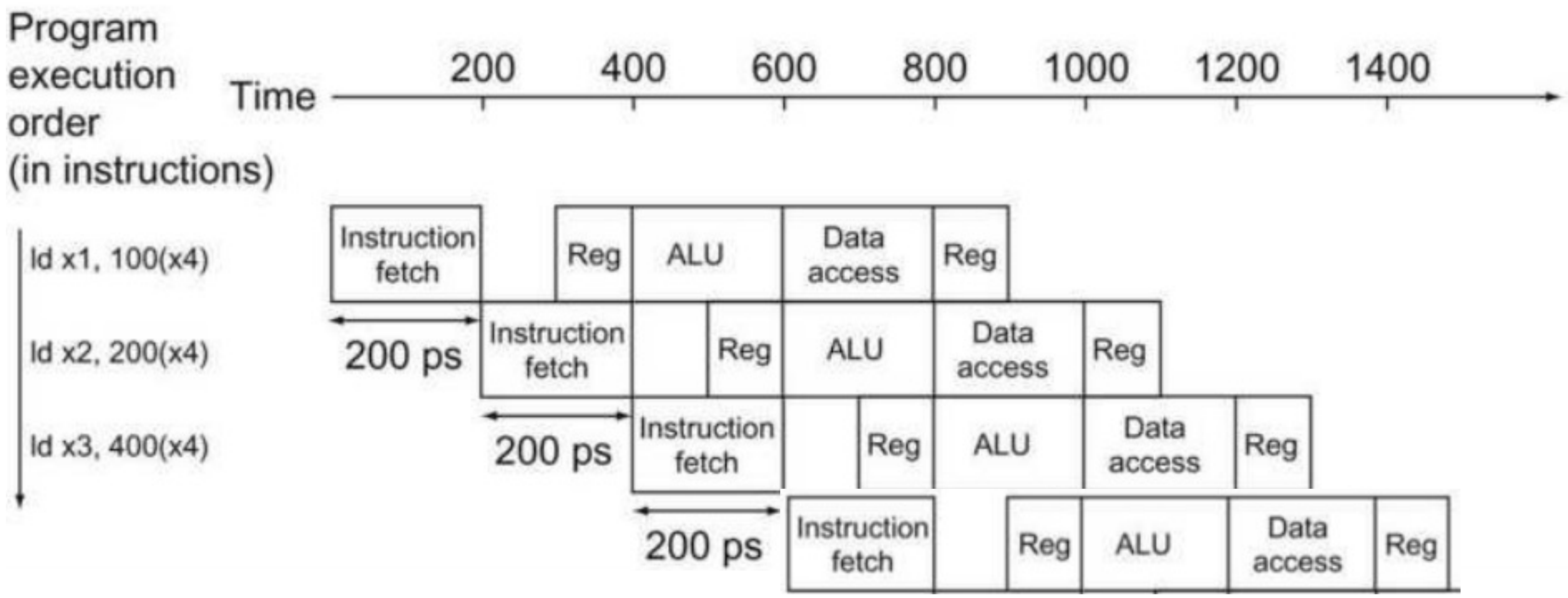
• Structural Hazard	A required resource is busy
• Data Hazards	• Data dependency between instructions • Need to wait for previous instruction to complete its data read/write
• Control Hazards	Flow of execution depends on previous instruction

Structural Hazard

Structural Hazard

A required resource is busy

Example: Consider the situation while the pipeline only has a single memory



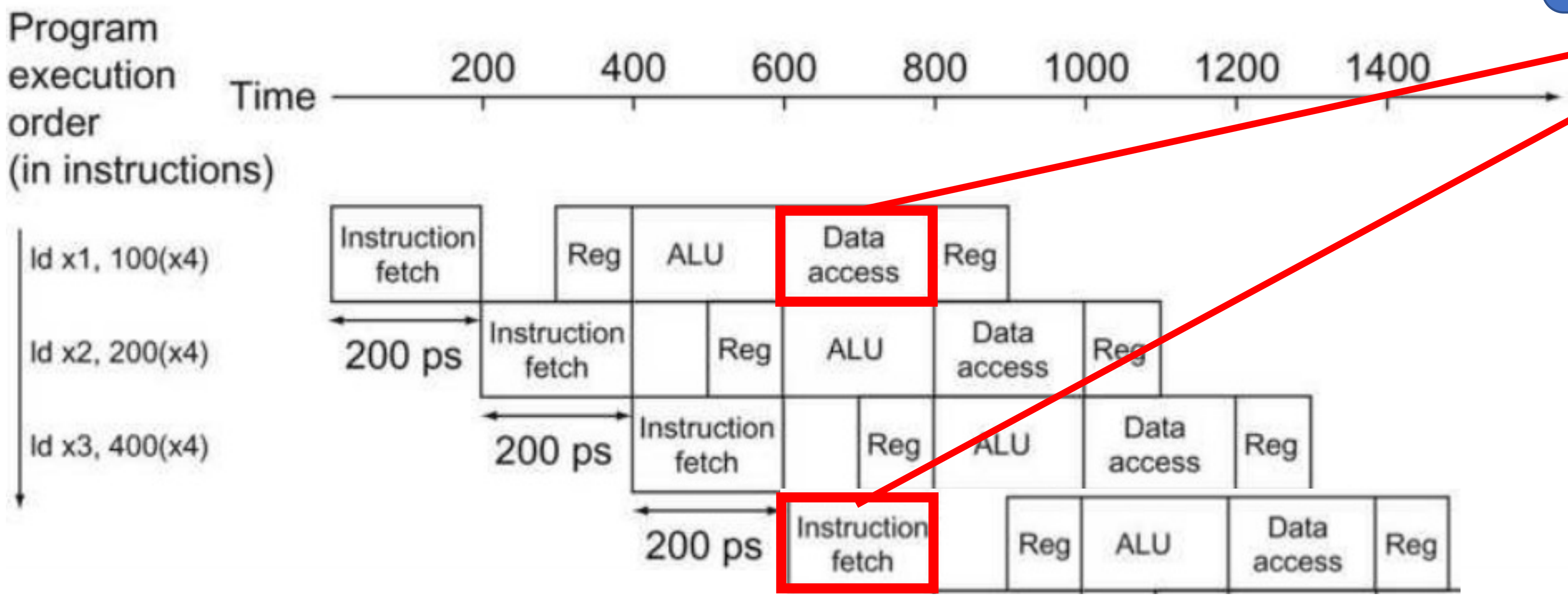
Question: Can the four instructions execute correctly?

Structural Hazard

A required resource is busy

Solution:
Use Instruction and data memory simultaneously.

Example: Consider the situation while the pipeline only has a single memory



How to Deal with Structural Hazard?

Problem: Two or more instructions in the pipeline compete for access to a single physical resource

- Solution 1: Instructions take it in turns to use resource, some instructions have to stall.

- Solution 2:Add more hardware to machine.

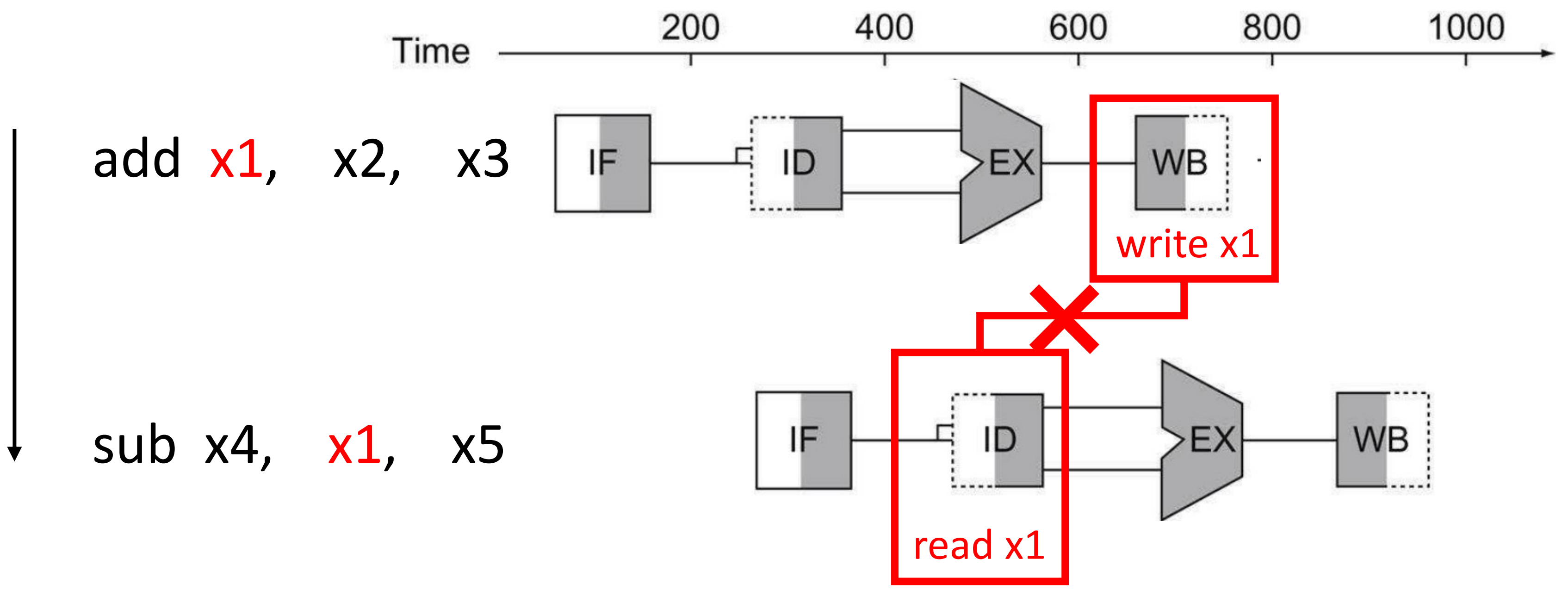
Can always solve a structural hazard by adding more hardware

Data Hazards

Data Hazard

- Data dependency between instructions
- Need to wait for previous instruction to complete its data read/write

Problem: Instruction depends on result from previous

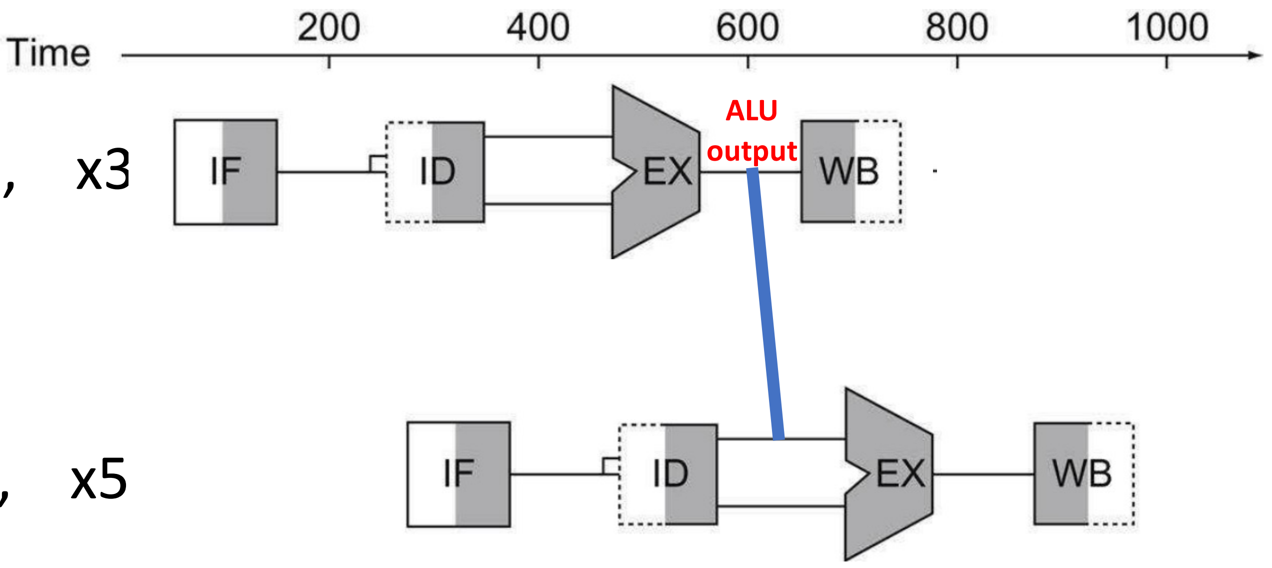


Data Hazard

Solution "forwarding": Adding extra hardware to retrieve the missing item early from the internal resources

add x1, x2, x3

sub x4, x1, x5



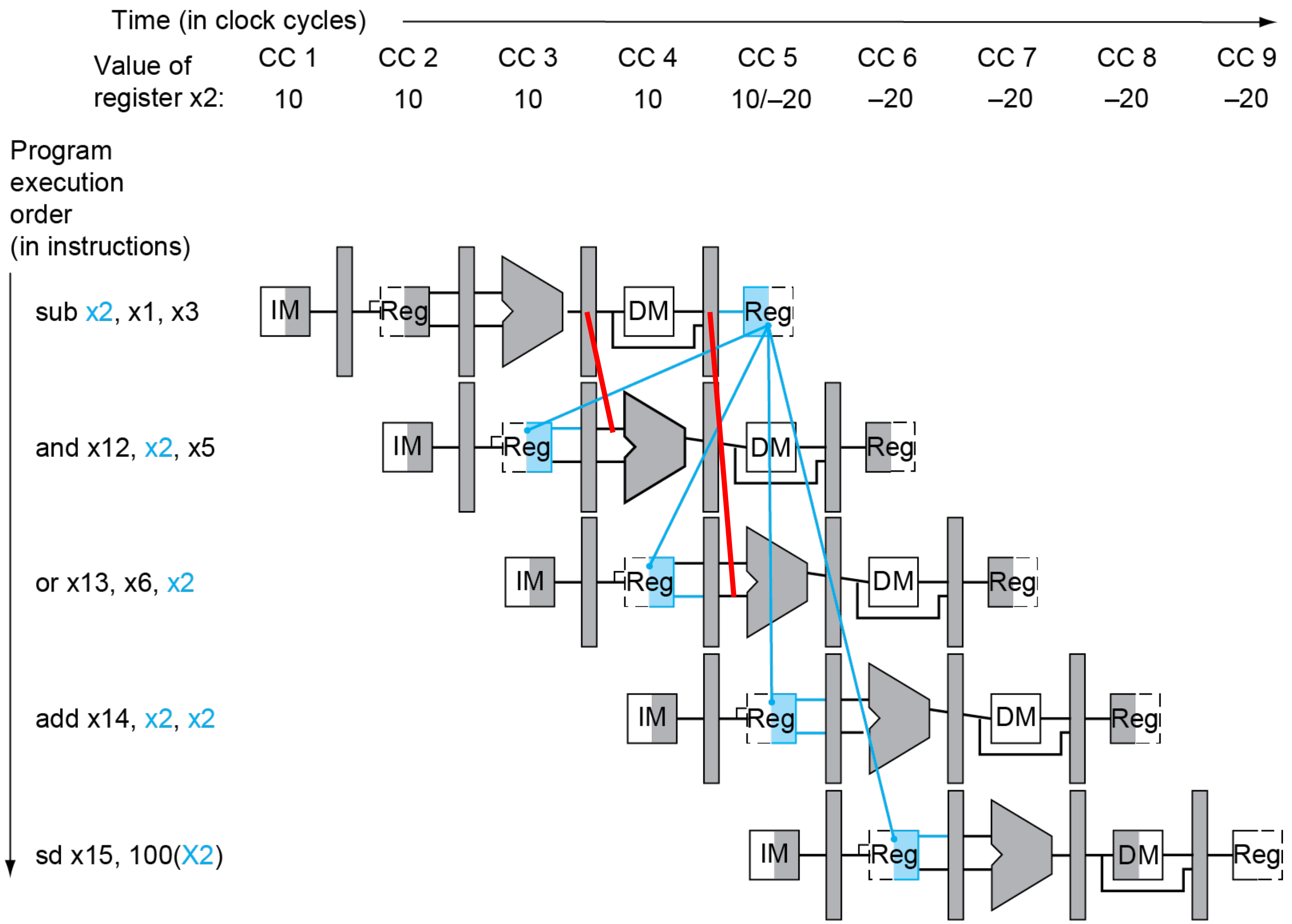
Data Hazard in ALU Instructions

- Consider this sequence:

```
sub   x2, x1,x3
and   x12,x2,x5
or    x13,x6,x2
add   x14,x2,x2
sd    x15,100(x2)
```

- We can resolve hazards with forwarding
  - How do we detect when to forward?

Dependencies & Forwarding



Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., ID/EX.Rs1 = register number for Rs1 sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.Rs1, ID/EX.Rs2
- Data hazards when

1a. EX/MEM. Rd = ID/EX. Rs1
1b. EX/MEM. Rd = ID/EX. Rs2
} Fwd from EX/MEM pipeline reg — R-type

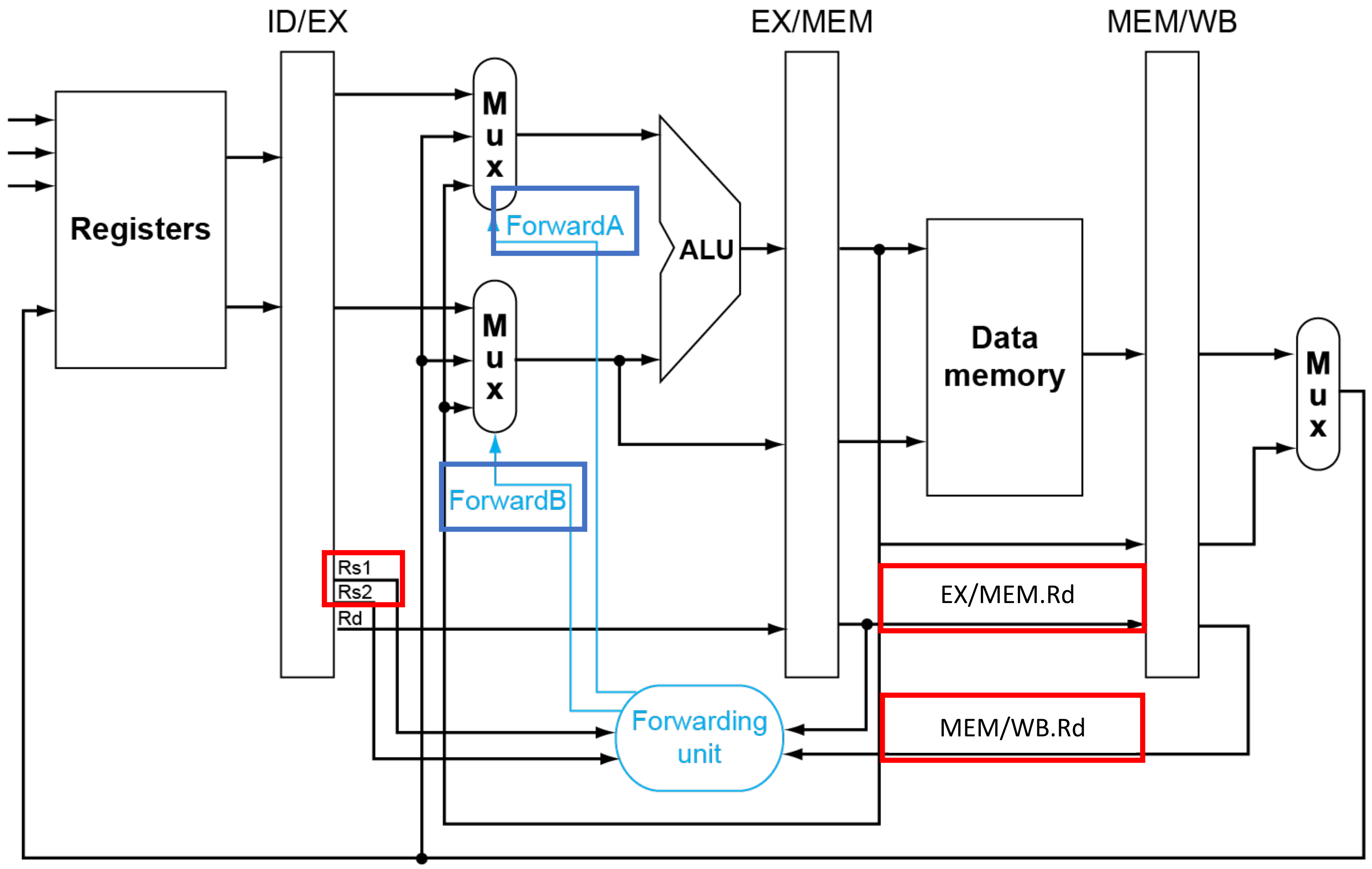
2a. MEM/WB. Rd = ID/EX. Rs1
2b. MEM/WB. Rd = ID/EX. Rs2
} Fwd from MEM/WB pipeline reg — Load

Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite

- And only if Rd for that instruction is not x0
  - EX/MEM. Rd ≠ 0,
    MEM/WB. Rd ≠ 0

Forwarding Paths

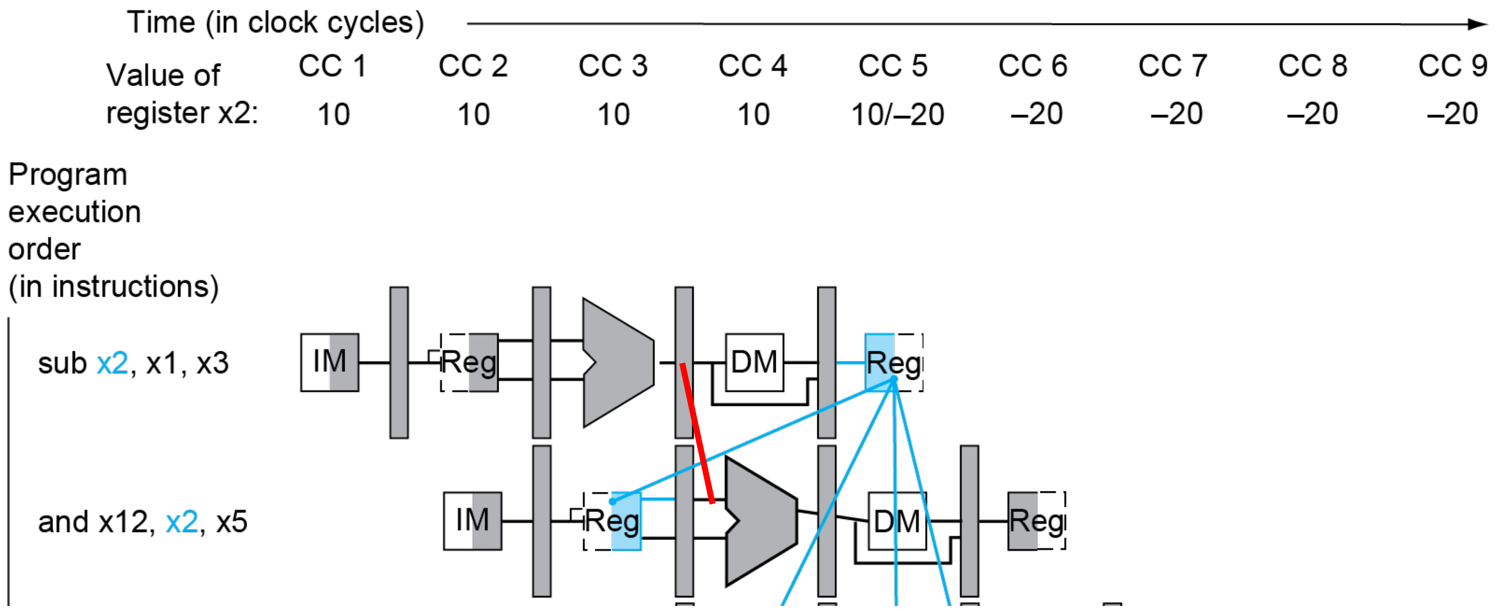


Forwarding Conditions

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Example 1

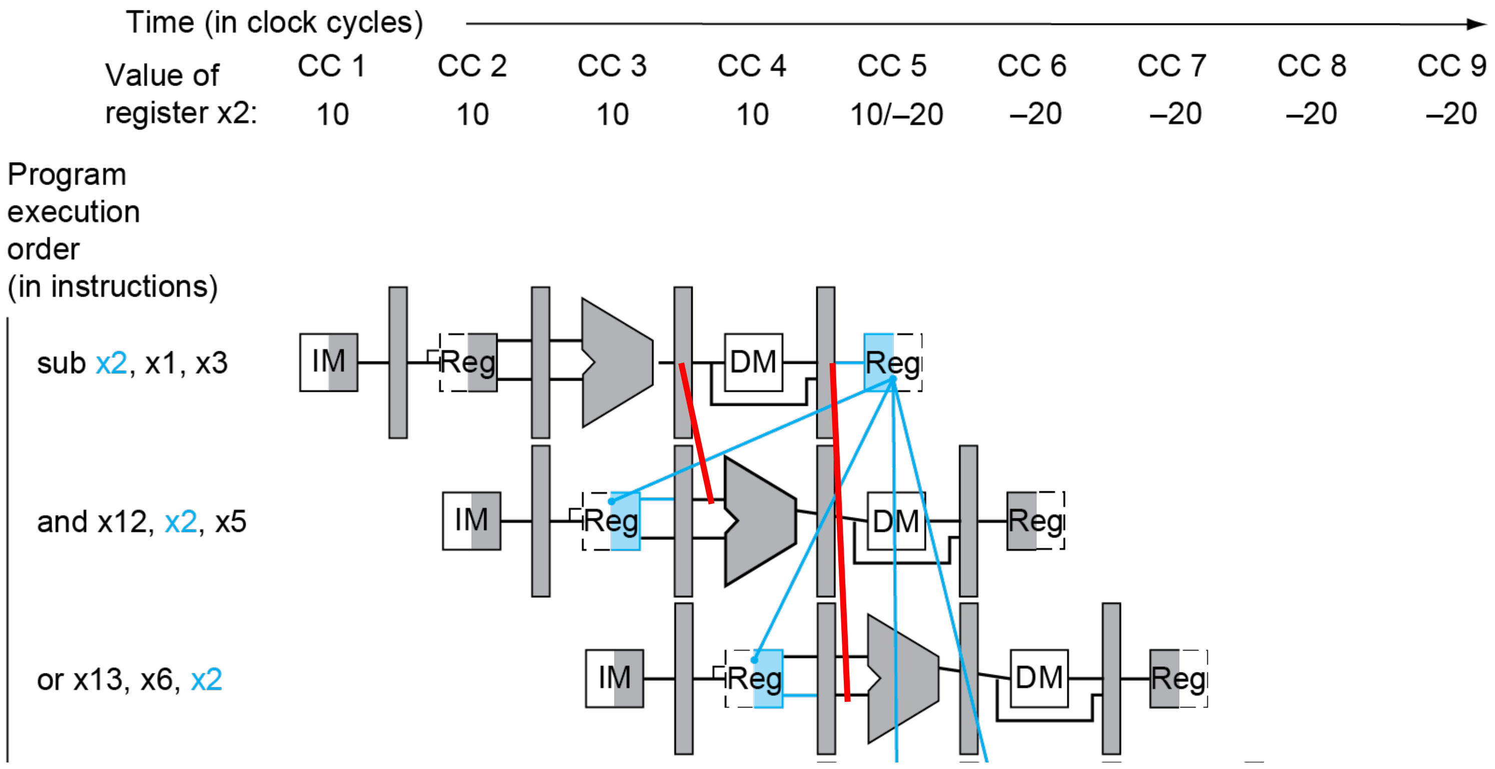
What is the Data hazards condition in the following case?



EX/MEM. Rd = ID/EX. Rs1

Example 2

What is the Data hazards condition in the following case?



MEM/WB. Rd = ID/EX. Rs2

Forwarding Conditions

Mux control	Source	Explanation
ForwardA/B = 00	ID/EX	No forwarding
ForwardA/B = 10	EX/MEM	Forwarding with data hazard in EX/MEM
ForwardA/B = 01	MEM/WB	Forwarding with data hazard in MEM/WB

Forwarding Conditions

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM. Rd ≠ 0)
      and (EX/MEM. Rd = ID/EX. Rs1))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM. Rd ≠ 0)
      and (EX/MEM. Rd = ID/EX. Rs2))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)
      and (MEM/WB. Rd = ID/EX. Rs1))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)
      and (MEM/WB. Rd = ID/EX. Rs2))
    ForwardB = 01

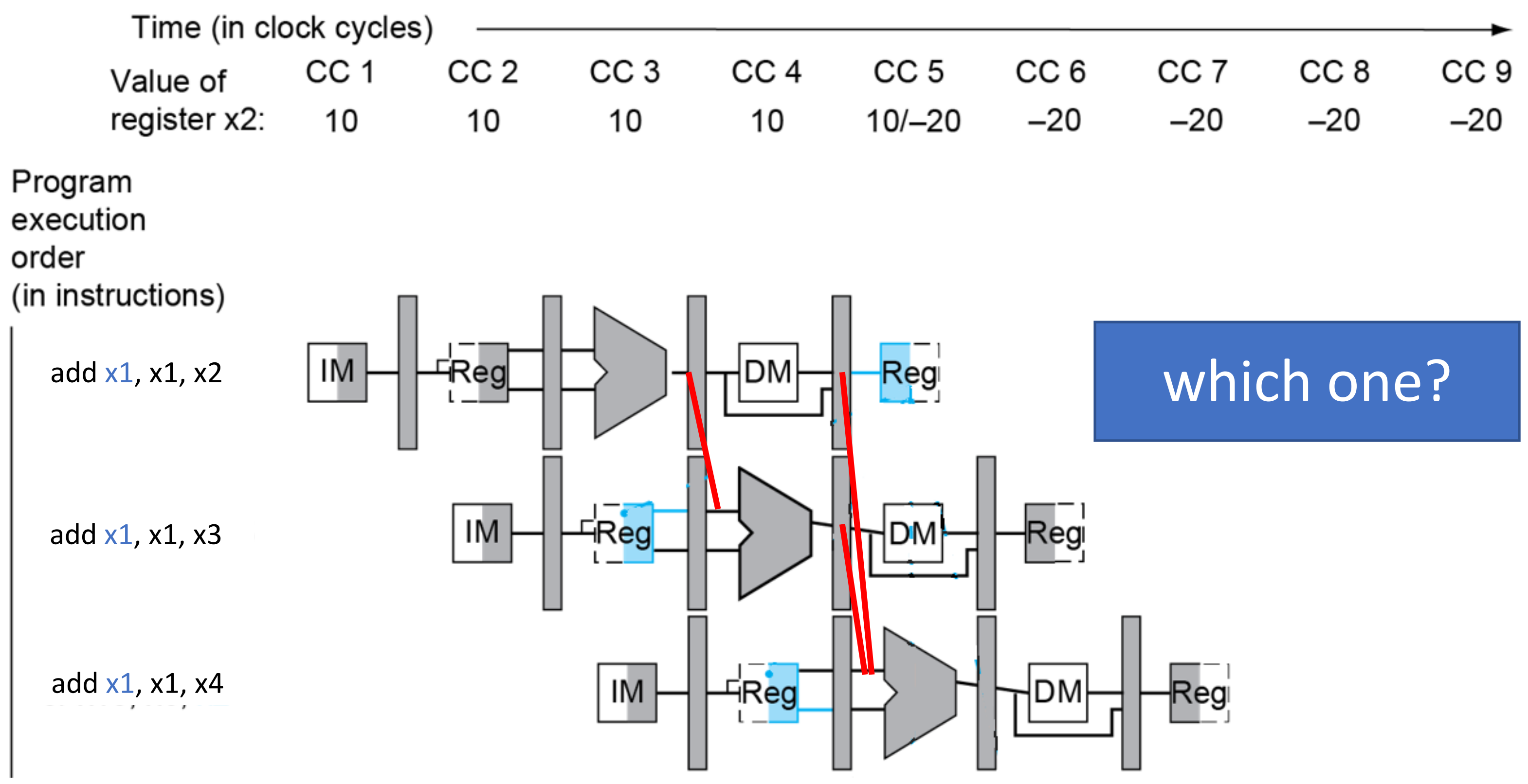
Double Data Hazard

- Consider the sequence:

```
add x1,x1,x2
add x1,x1,x3
add x1,x1,x4
```

- Both hazards occur
  - Want to use the most recent

Double Data Hazard



which one?

Such an exception should be added into MEM hazards!

Double Data Hazard

- Consider the sequence:

```
add x1,x1,x2
add x1,x1,x3
add x1,x1,x4
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn’t true

Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)

    and not(EX/MEM.RegWrite and (EX/MEM. Rd ≠ 0)

        and (EX/MEM. Rd = ID/EX. Rs1))

    and (MEM/WB. Rd = ID/EX. Rs1))

    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)

    and not(EX/MEM.RegWrite and (EX/MEM. Rd ≠ 0)

        and (EX/MEM. Rd = ID/EX. Rs2))

    and (MEM/WB. Rd = ID/EX. Rs2))

    ForwardB = 01

Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)

    and not(EX hazard)

    and (MEM/WB. Rd = ID/EX. Rs1))

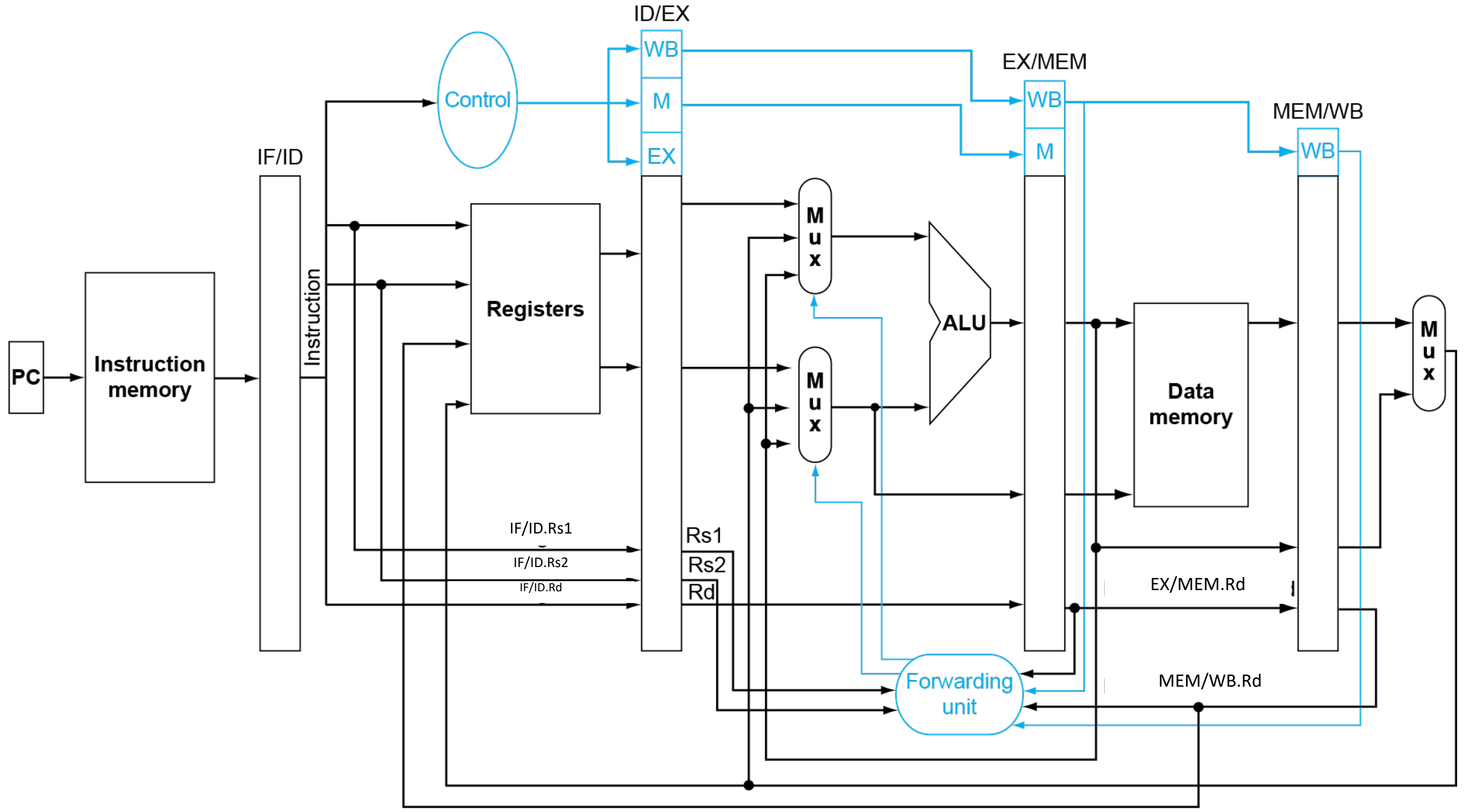
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)

    and not(EX hazard)

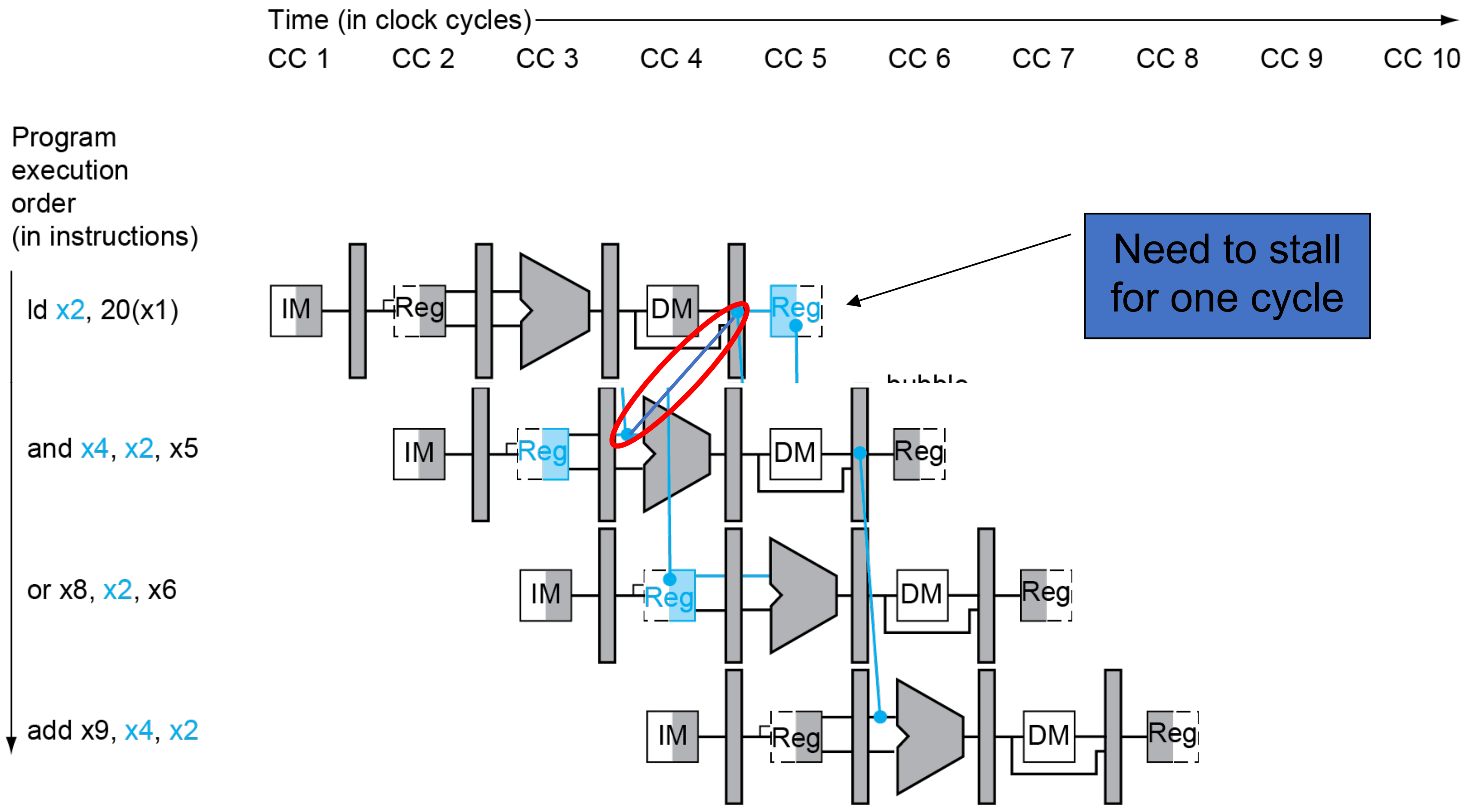
    and (MEM/WB. Rd = ID/EX. Rs2))

    ForwardB = 01

Datapath with Forwarding



Load-Use Data Hazard



Need to stall for one cycle

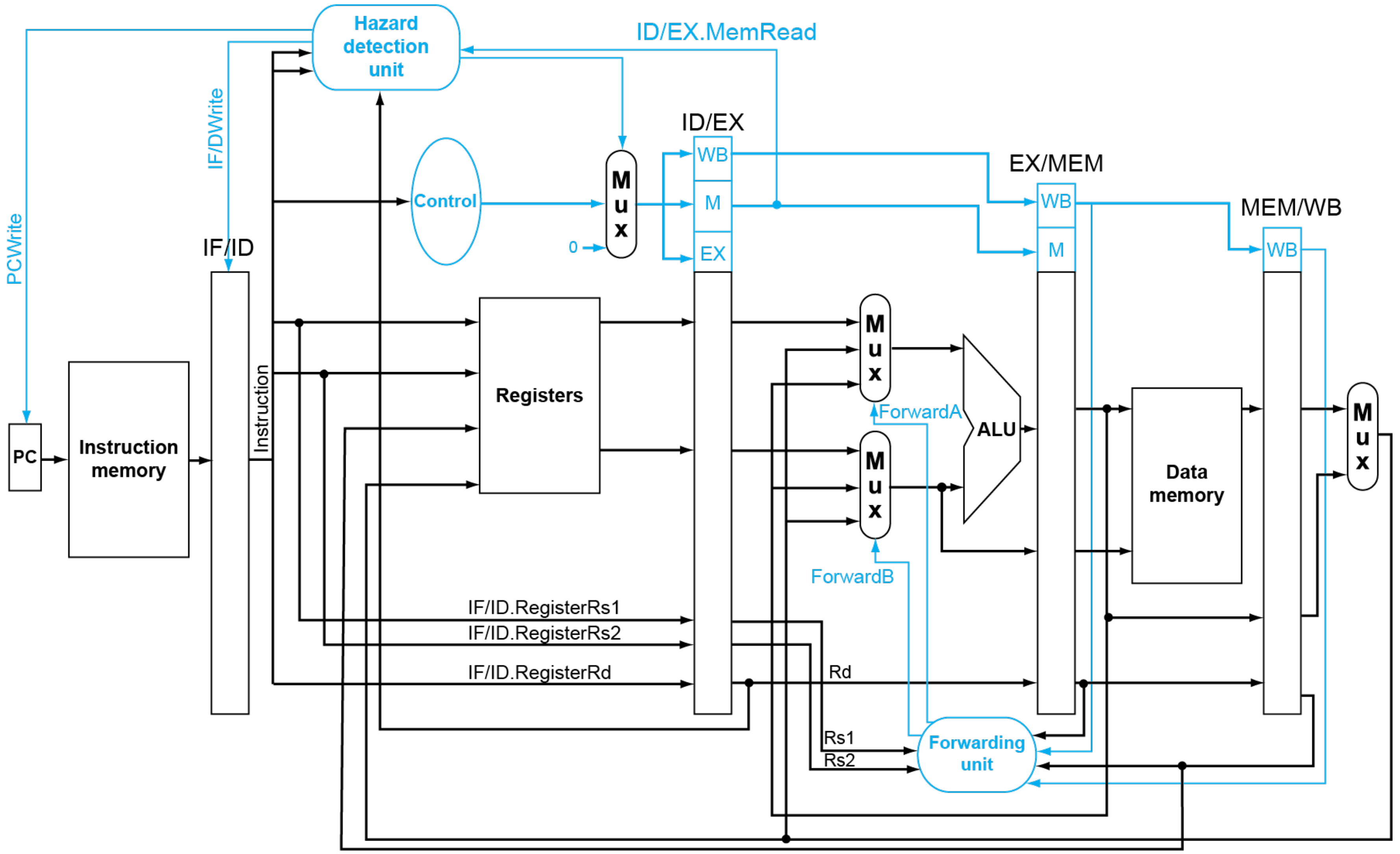
Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID. Rs1, IF/ID. Rs2
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX. Rd = IF/ID. Rs1) or
    (ID/EX. Rd = IF/ID. Rs2))
- If detected, stall and insert bubble

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 `ld`
    - Can subsequently forward to EX stage

Datapath with Hazard Detection



Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

Summary of Data Hazards

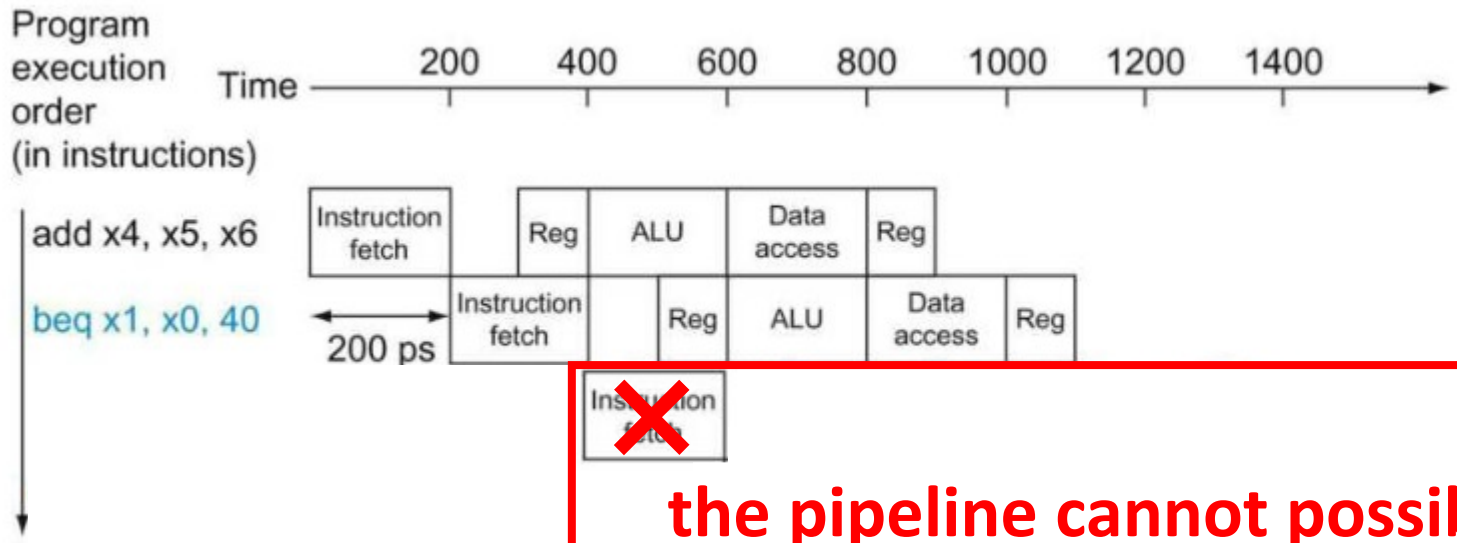
- EX hazard & MEM hazard
  - forwarding
- Double hazard
  - Revise the forwarding condition
- Load-use hazard
  - One stall is needed, except for forwarding

Control Hazards

Control Hazard

Flow of execution depends on previous instruction

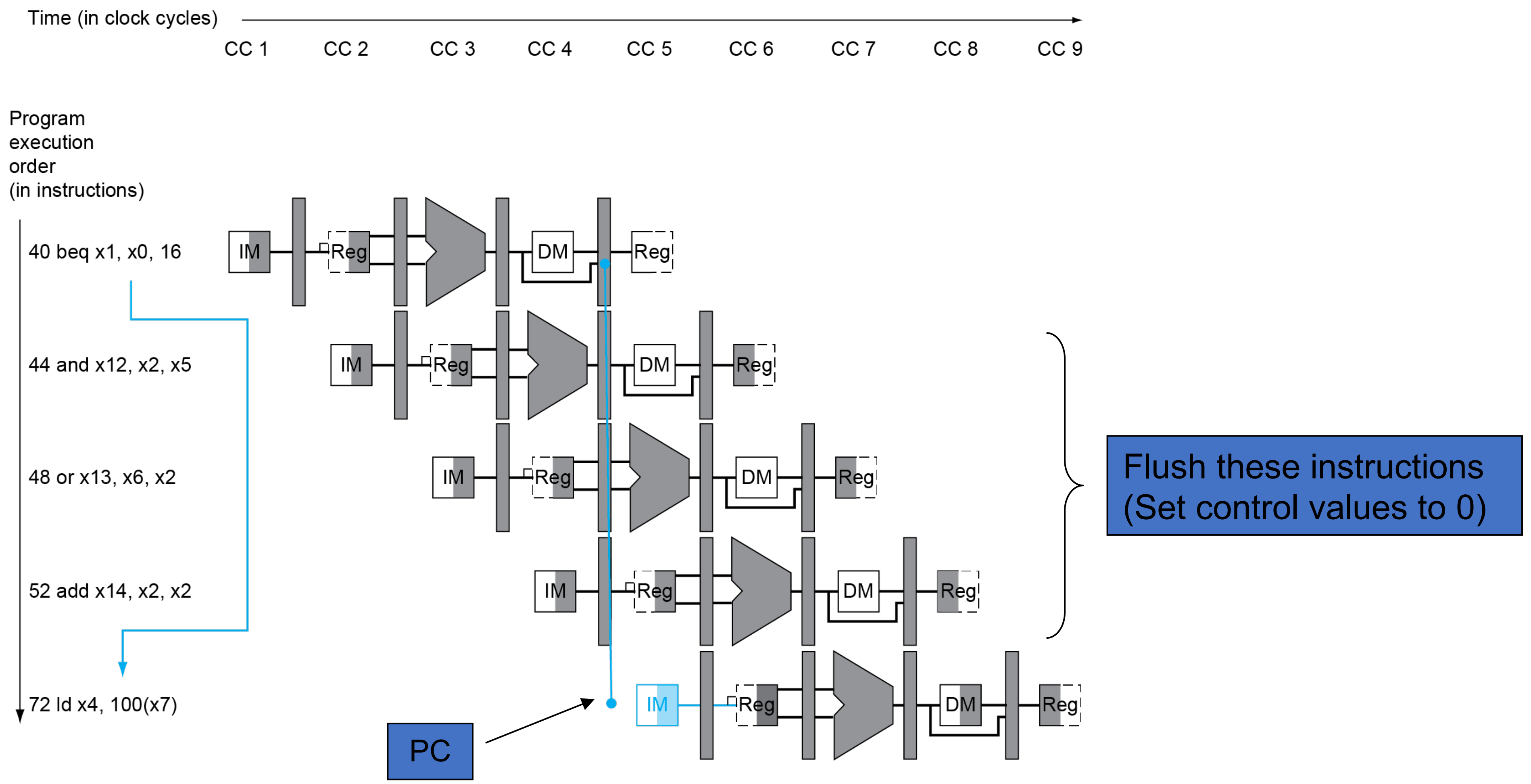
Problem: The conditional branch instruction



the pipeline cannot possibly know what the next instruction should be

Branch Hazards

- If branch outcome determined in MEM



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
  - The penalty is significant
  - How many clock cycles does the stall waste?

3 clock cycles! Almost downgrading to single-cycle CPU!

Stall on Branch

Branch taken

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

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

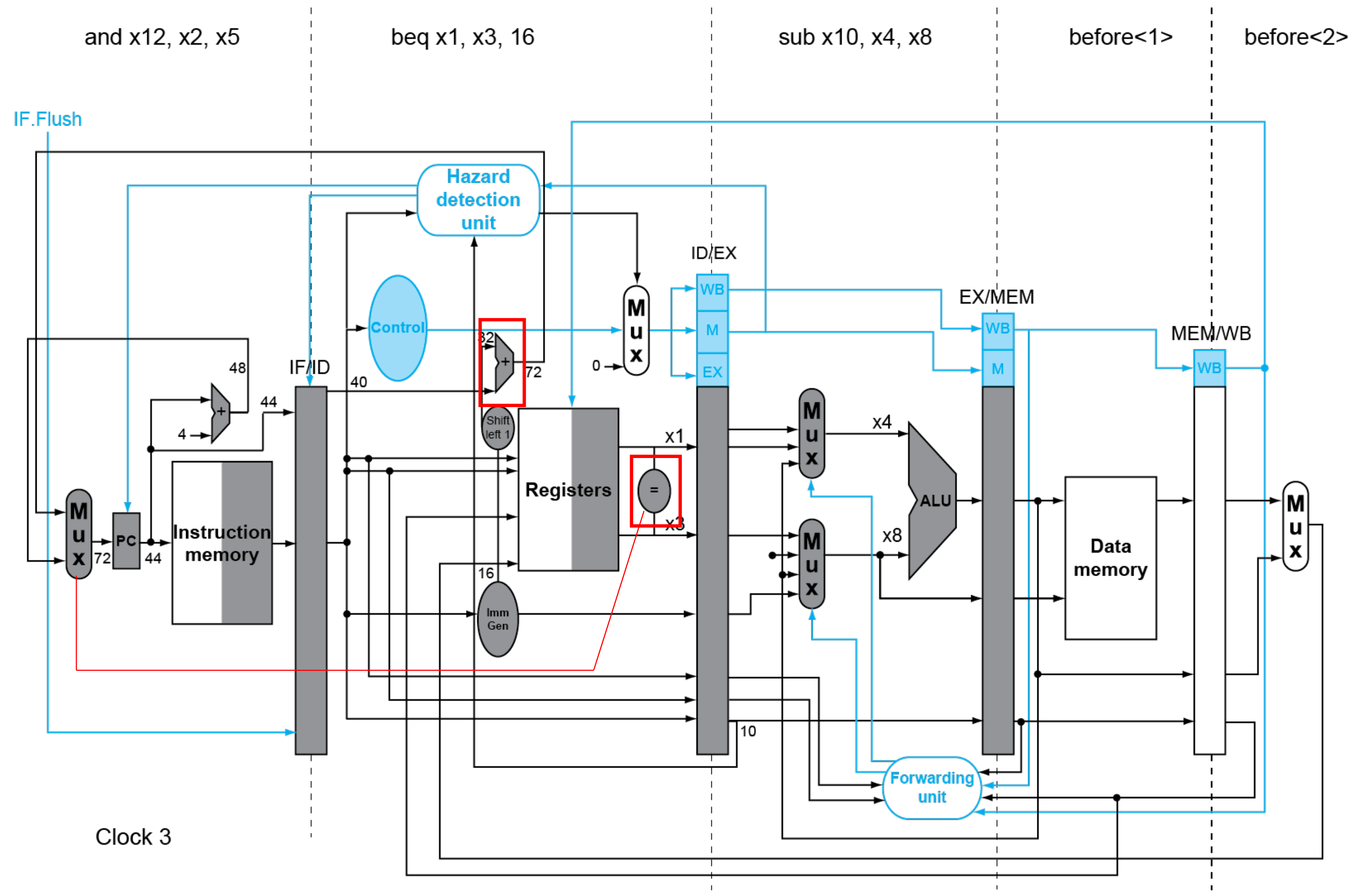
What is the hardware?

- 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



Anything wrong?

Data Hazards for Branches

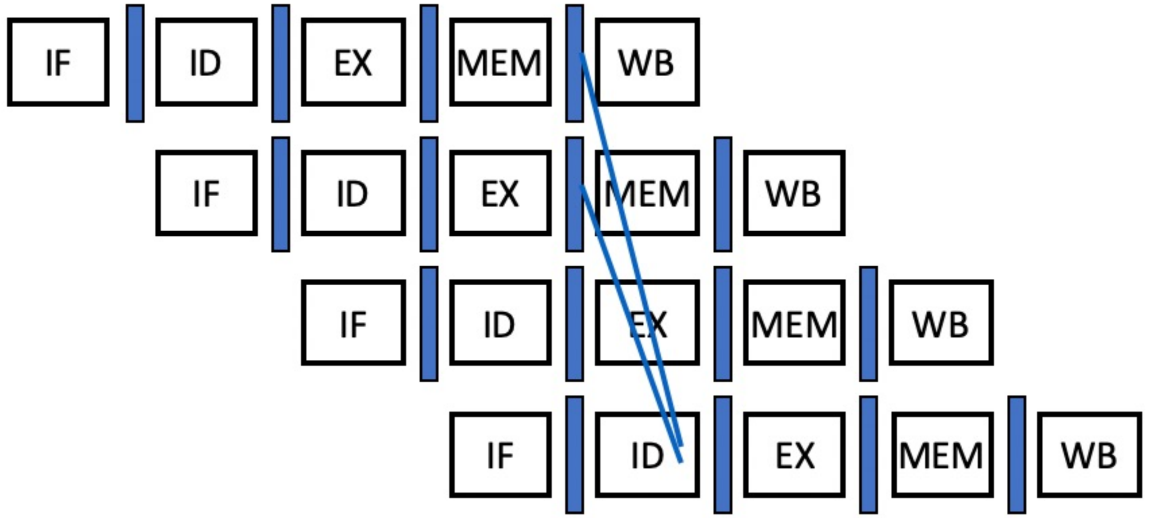
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add x1, x2, x3

add x4, x5, x6

…

beq x1, x4, disp
```



- Can resolve using forwarding

Data Hazards for Branches

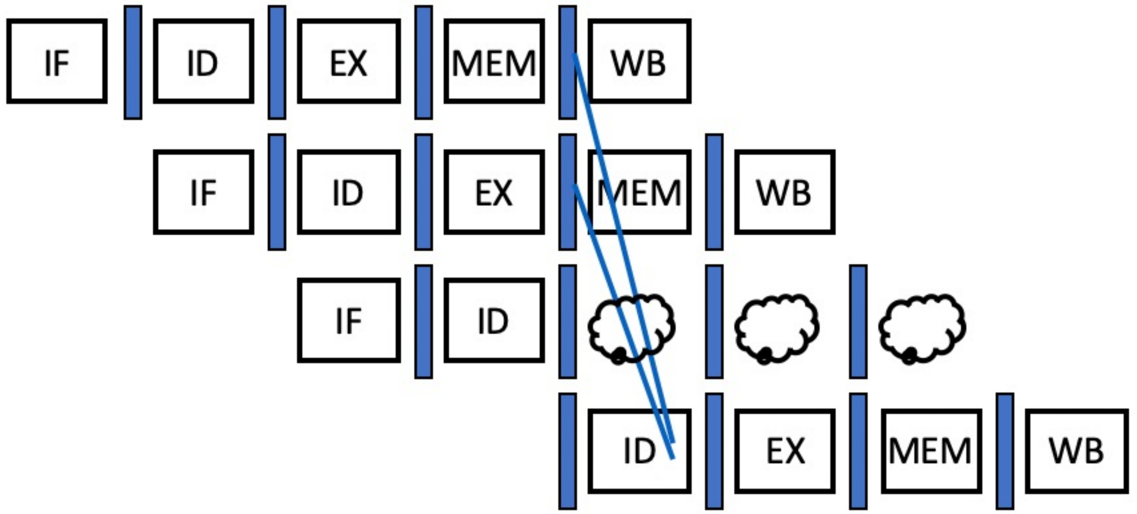
- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

lw x1, addr

add x4, x5, x6

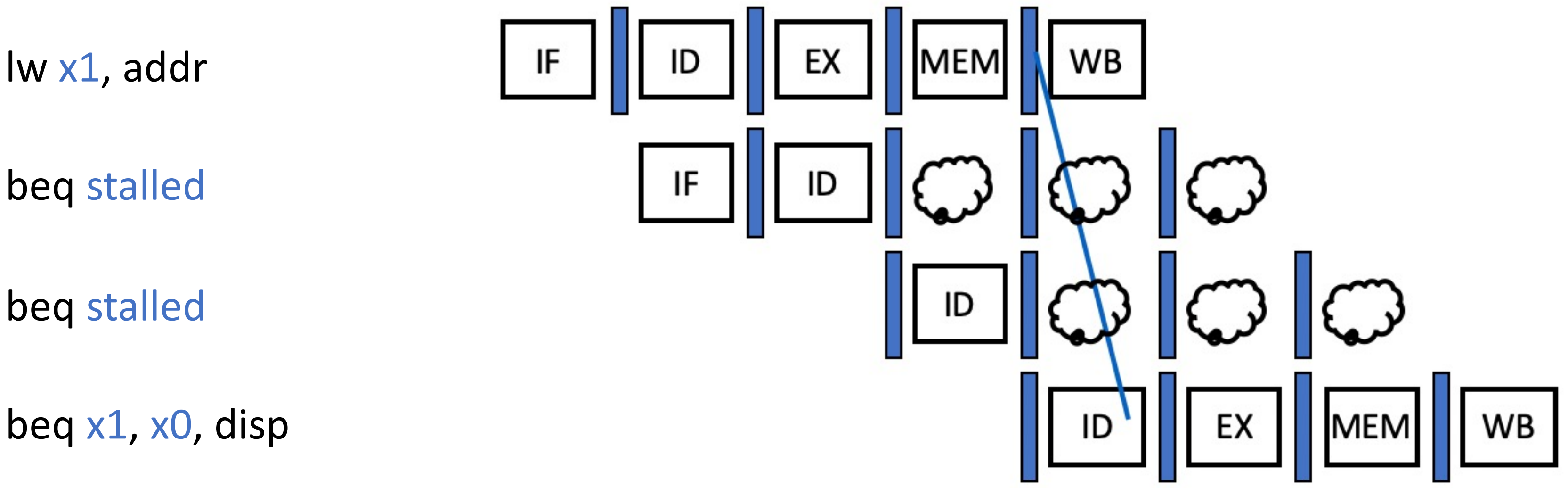
beq stalled

beq x1, x4, disp

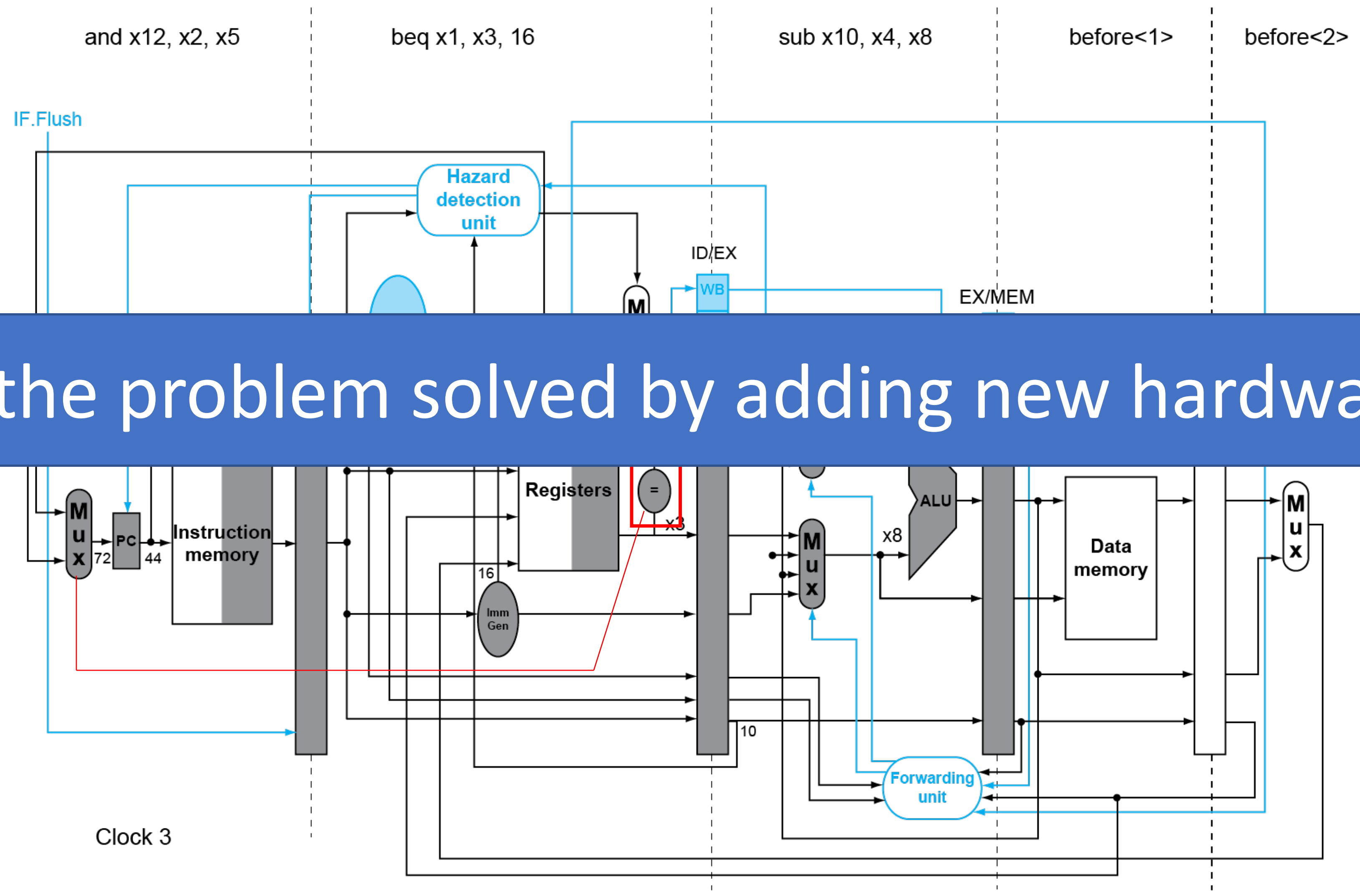


Data Hazards for Branches

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



Forwarding Branch to Earlier Stage



Is the problem solved by adding new hardware?

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

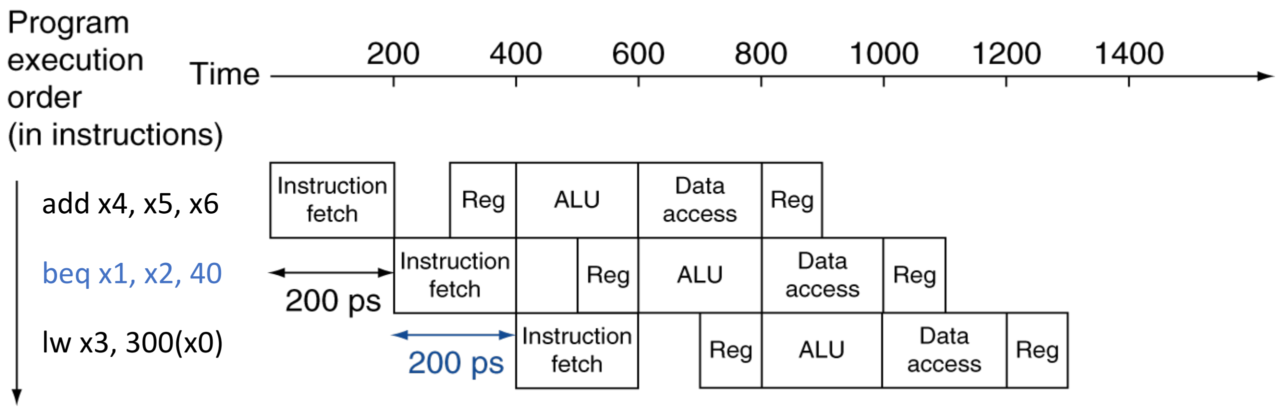
Predict branch

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

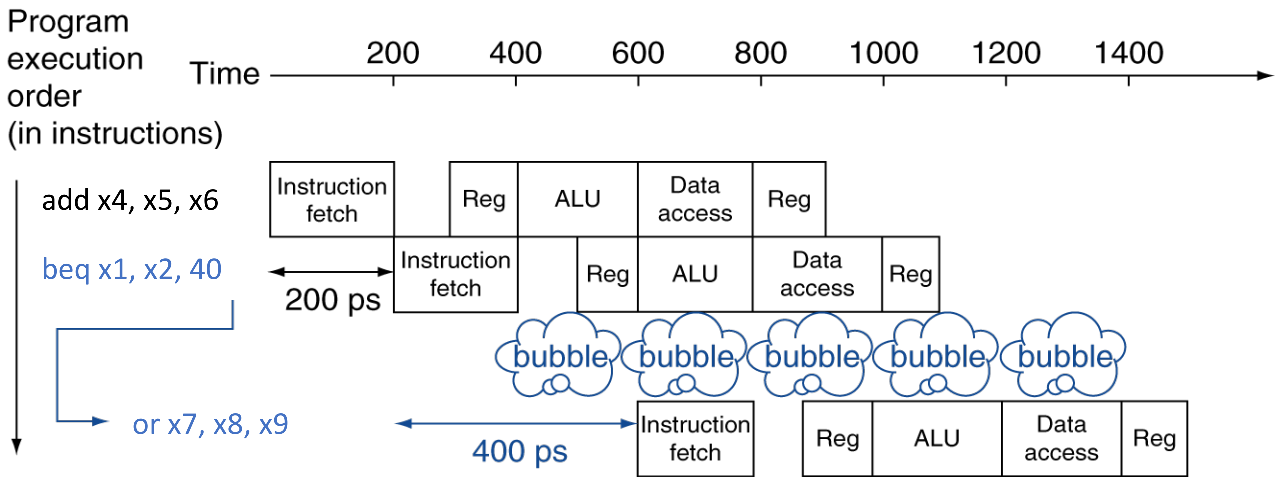
Branch i (Untaken)	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

RISC-V with Predict Not Taken

Prediction correct



Prediction incorrect

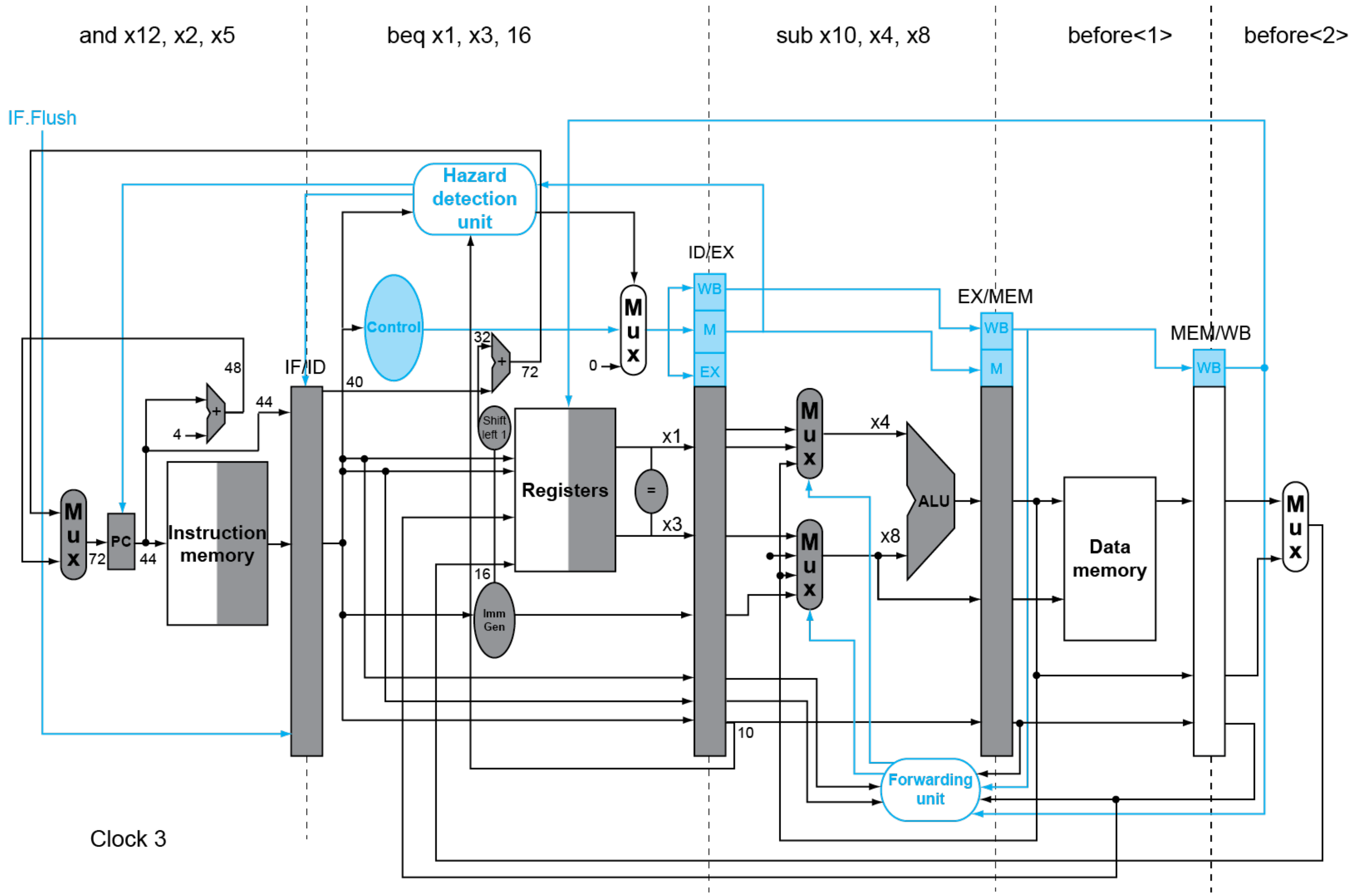


How to Reduce Branch Delay

- Example: branch taken

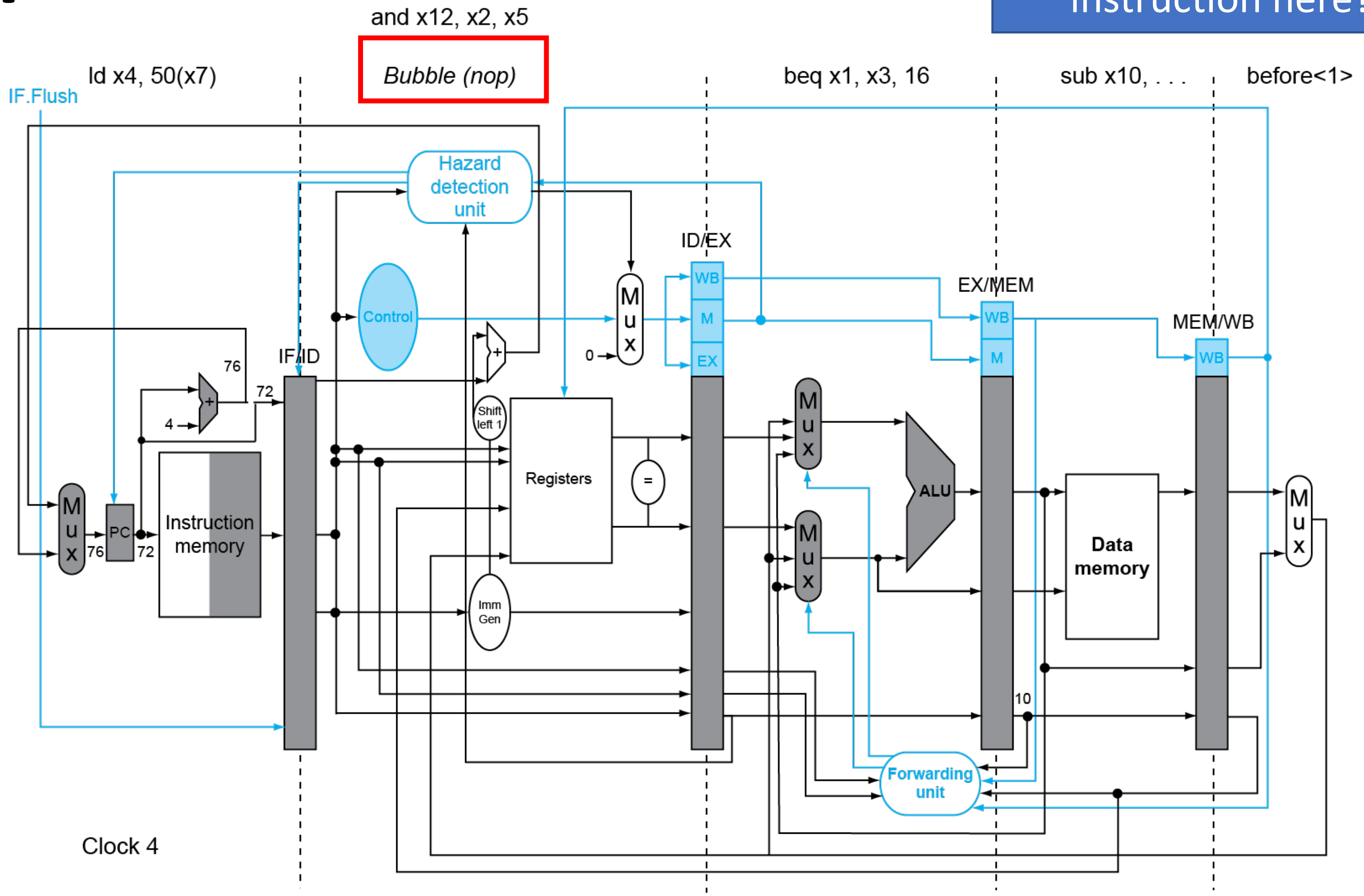
```
36:  sub  x10, x4, x8
40:  beq  x1,  x3, 16  // PC-relative branch
                       // to 40+16*2=72
44:  and  x12, x2, x5
48:  or   x13, x2, x6
52:  add  x14, x4, x2
56:  sub  x15, x6, x7
     ...
72:  ld   x4, 50(x7)
```

Example: Branch Taken



Example: Branch Taken

What is the original instruction here?



Reducing Branch Delay

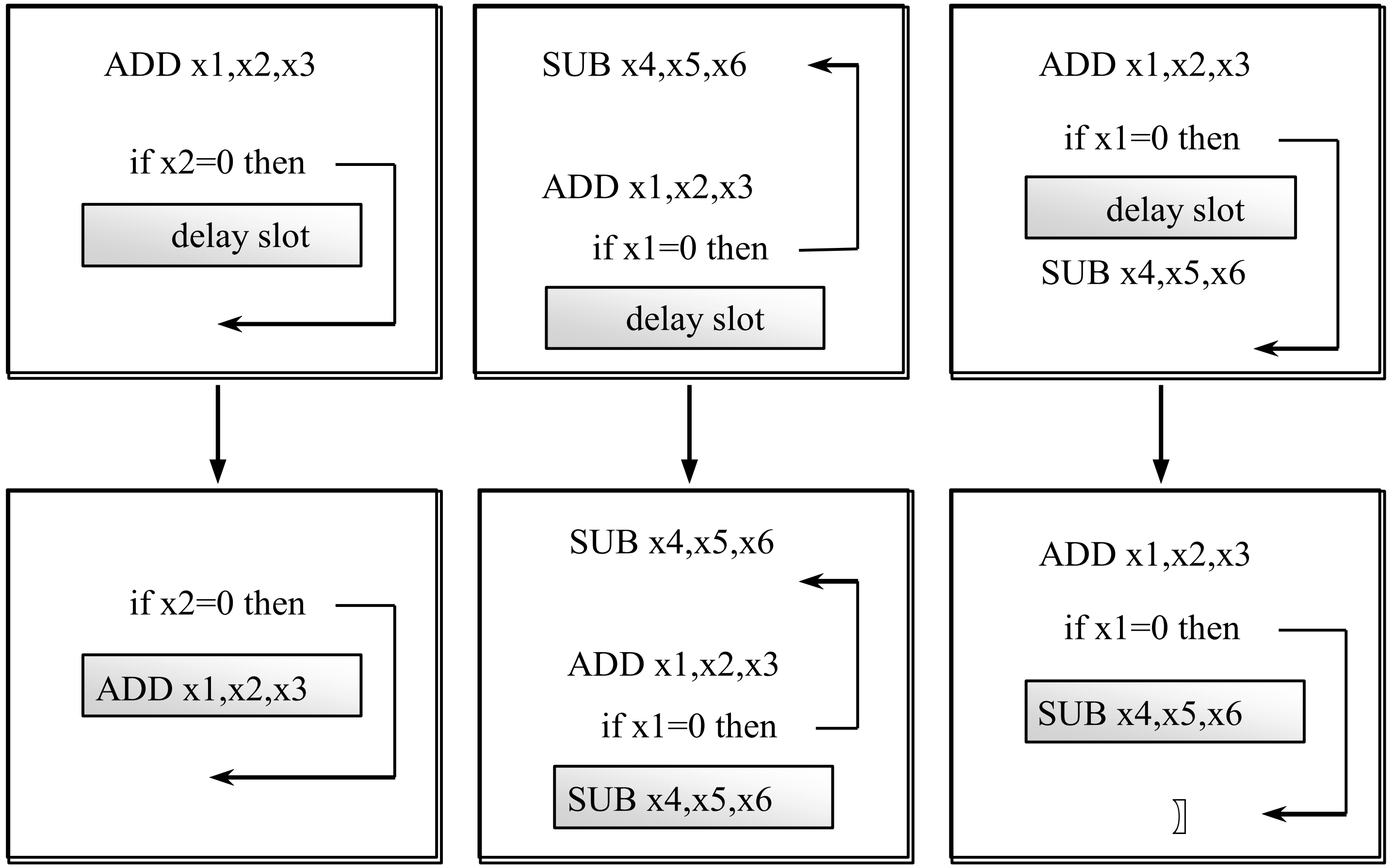
- Predict branch success
- Predict branch failure
- Delay Branch

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

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

Question: Is delay slot a really good design?

Review

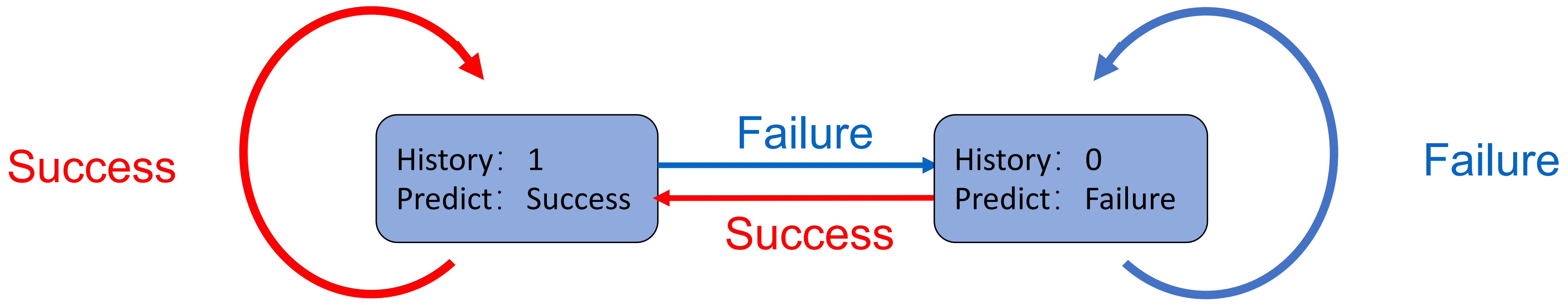
- "A ***RISC-V ISA*** is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA.
- The base integer ISAs are very similar to that of the early RISC processors except ***with no branch delay slots*** and with support for optional variable-length instruction encodings. "

——The RISC-V Instruction Set Manual Volume I

Dynamic Branch Prediction

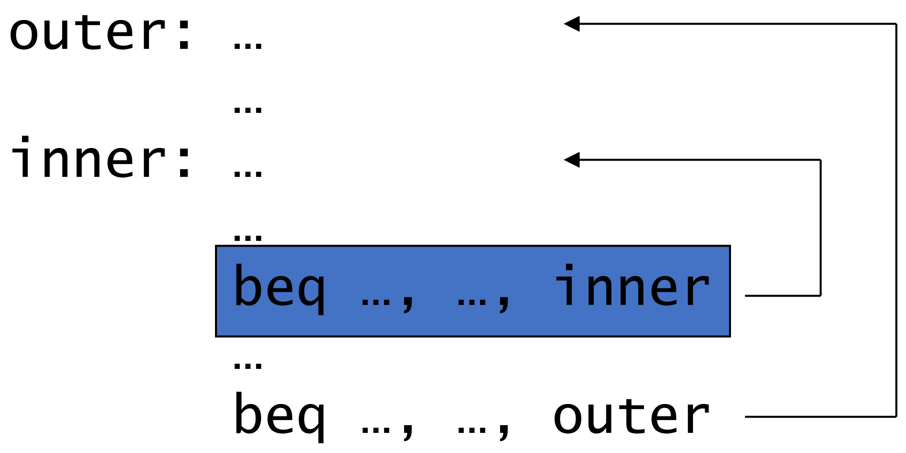
- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same 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!

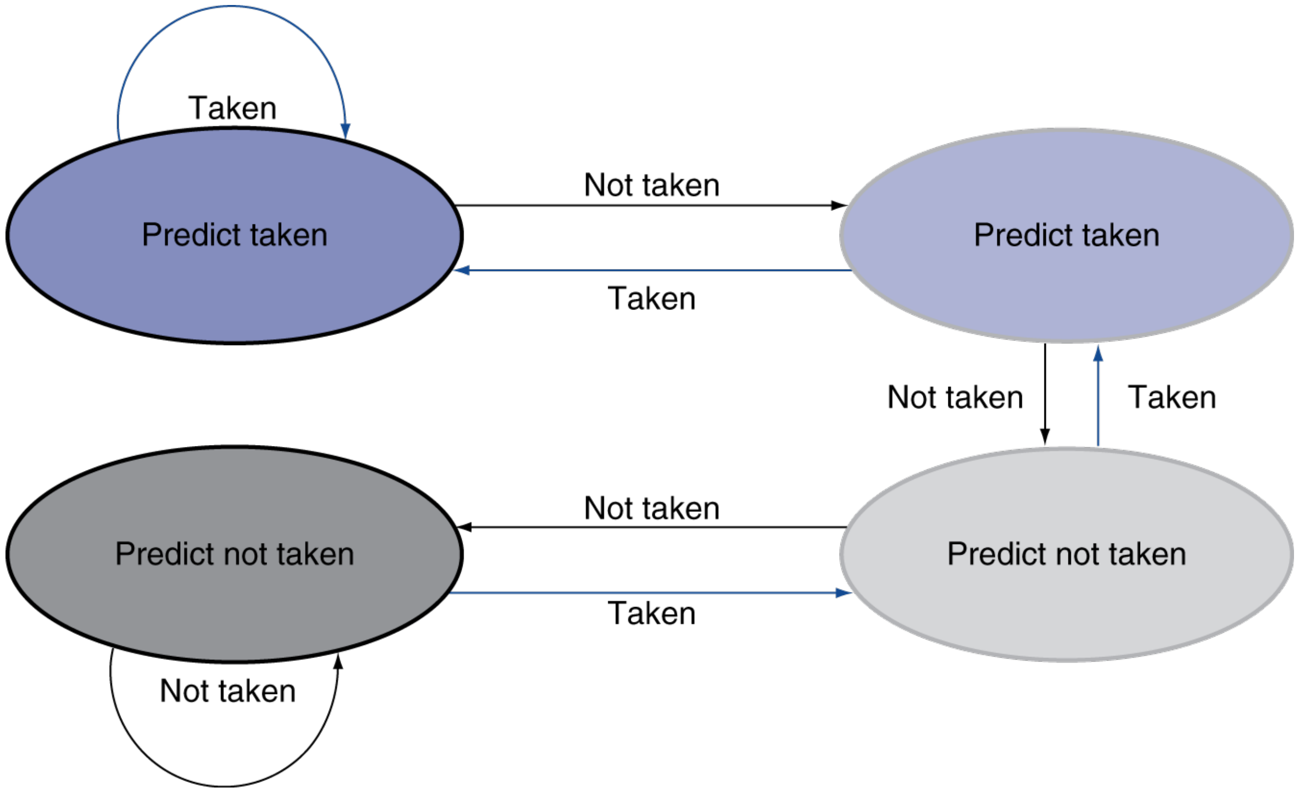


```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

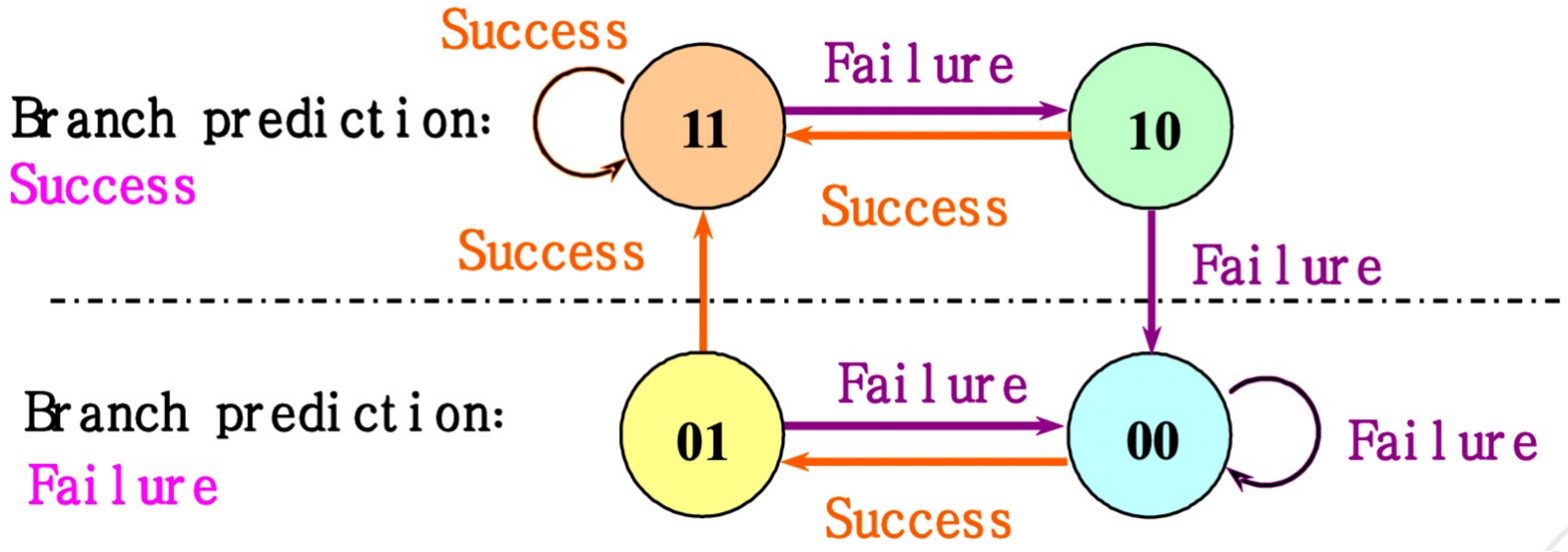
- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

- Only change prediction on two successive mispredictions

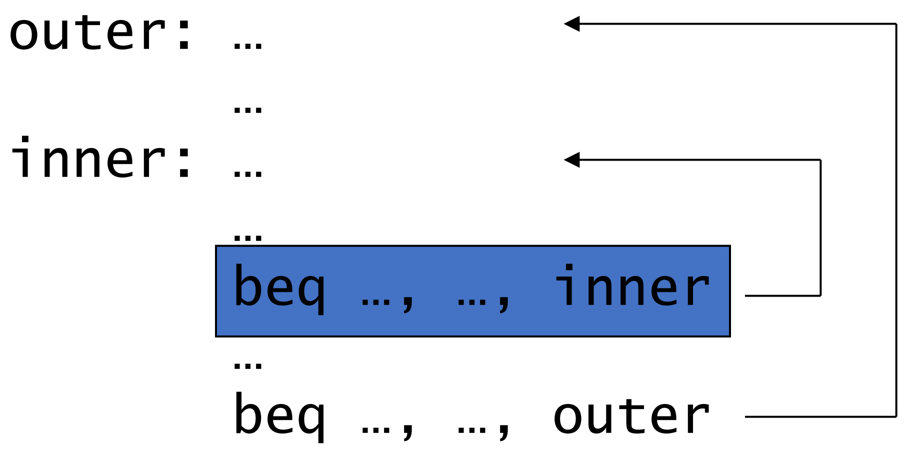


Branch History Table



2-Bit Predictor: Example

- Inner loop branches mispredicted only once!



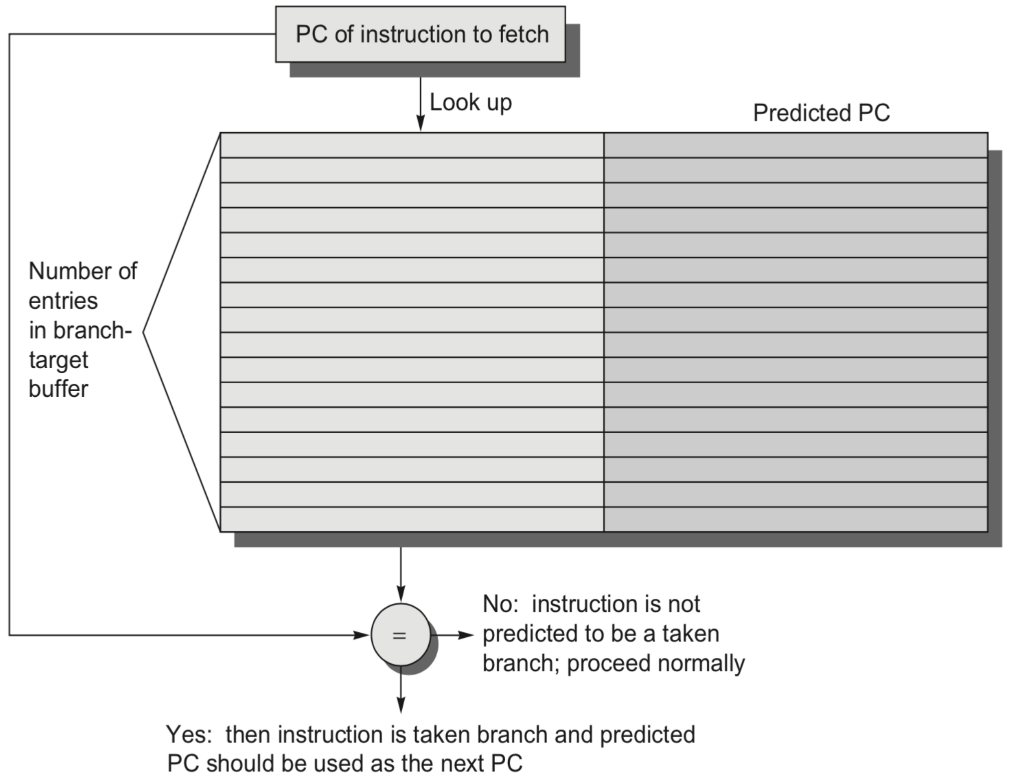
```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

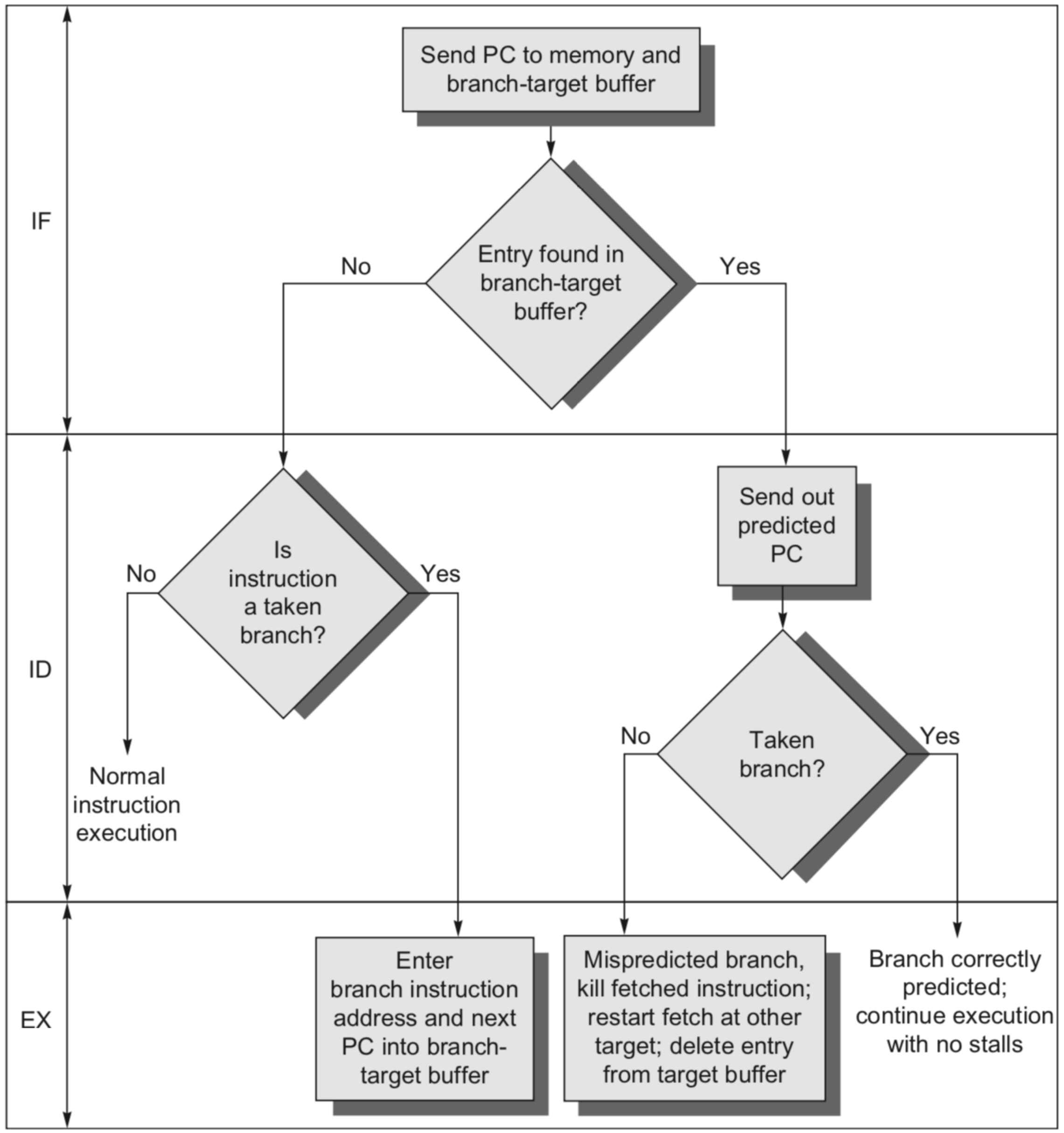
- Only mispredict as taken on last iteration of inner loop

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

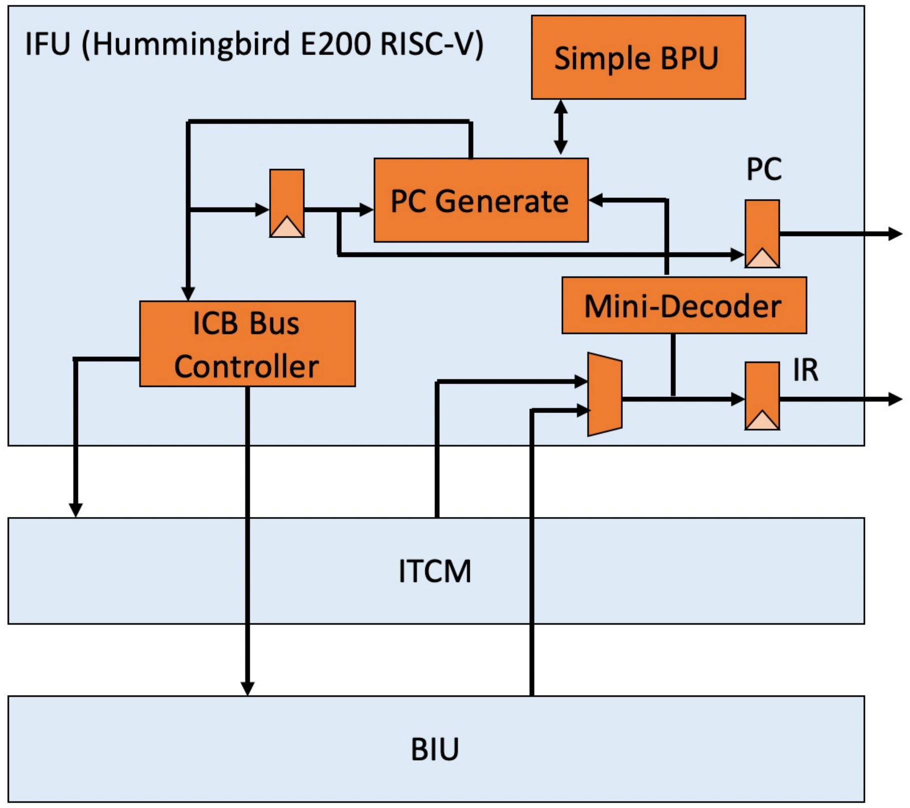
Branch-Target Buffers


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be a taken branch; proceed normally
Yes: then instruction is taken branch and predicted PC should be used as the next PC


IF
ID
EX
Send PC to memory and branch-target buffer
Entry found in branch-target buffer?
No
Yes
Is instruction a taken branch?
No
Yes
Normal instruction execution
Send out predicted PC
Taken branch?
No
Yes
Enter branch instruction address and next PC into branch-target buffer
Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
Branch correctly predicted; continue execution with no stalls

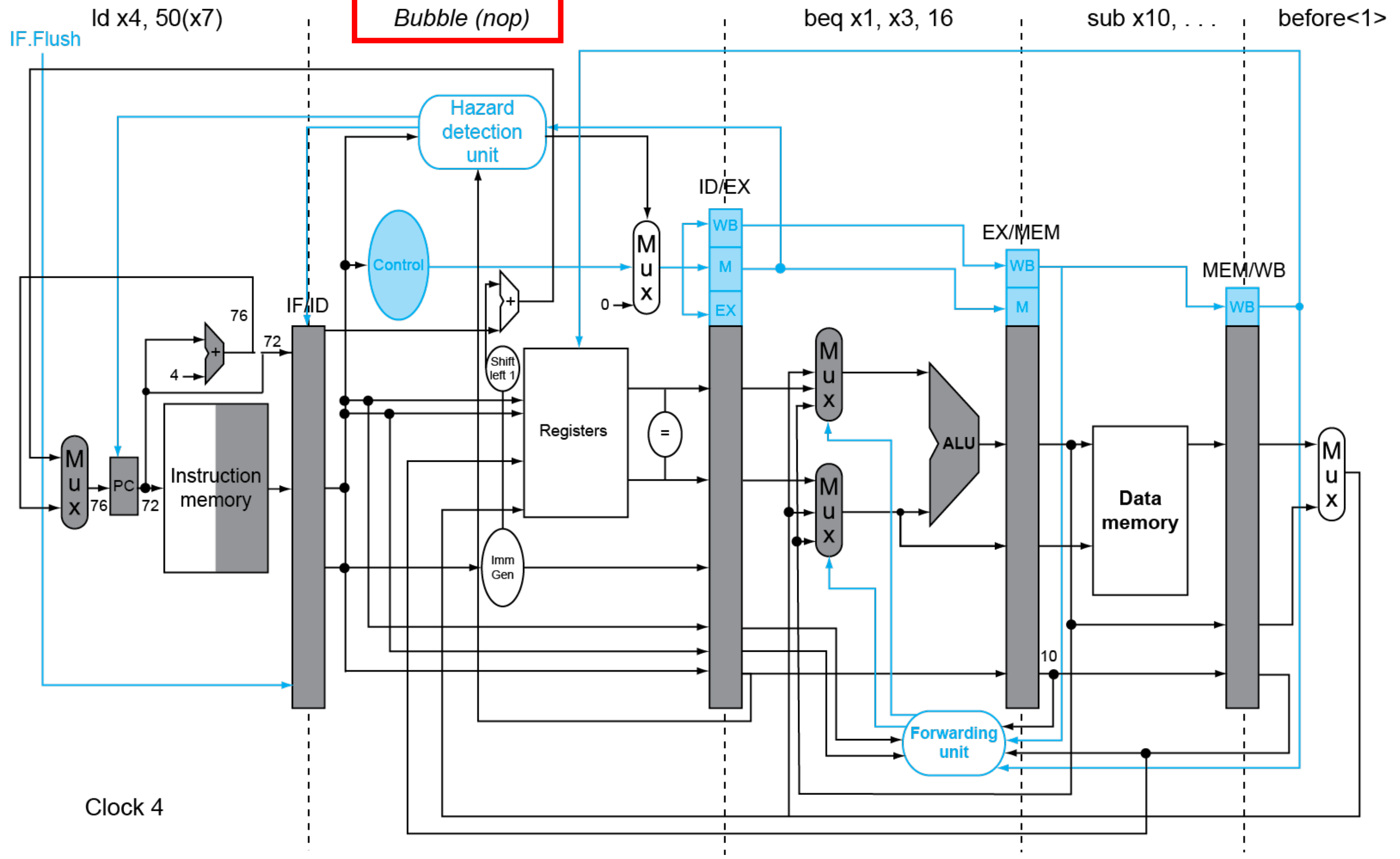
Integrated Instruction Fetch Units

- 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

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