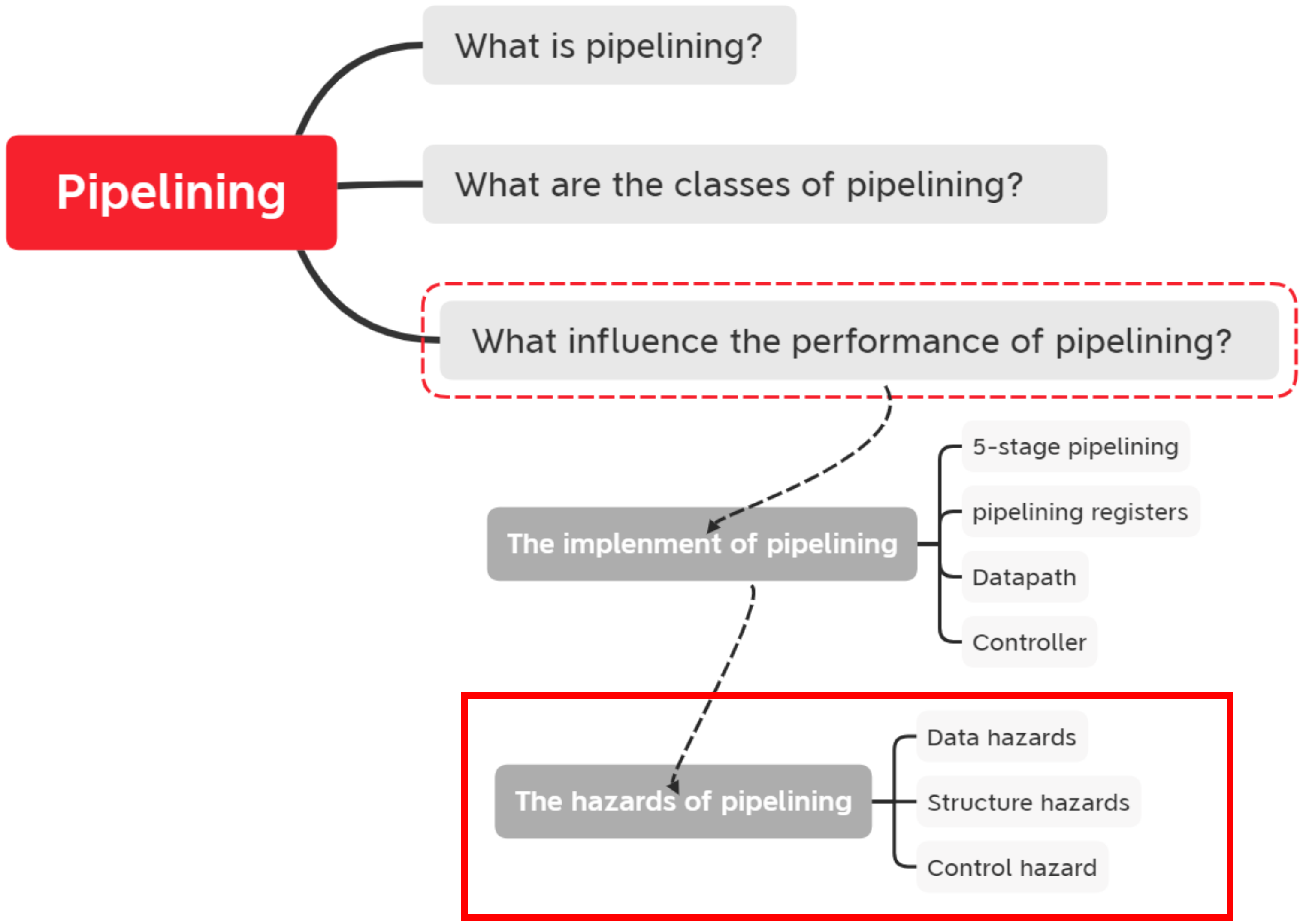
Chapter 2

Pipelining

§ Review


Pipelining
What is pipelining?
What are the classes of pipelining?
What influence the performance of pipelining?
The implenment of pipelining
5-stage pipelining
pipelining registers
Datapath
Controller
The hazards of pipelining
Data hazards
Structure hazards
Control hazard

§2.5 Hazards of Pipelining

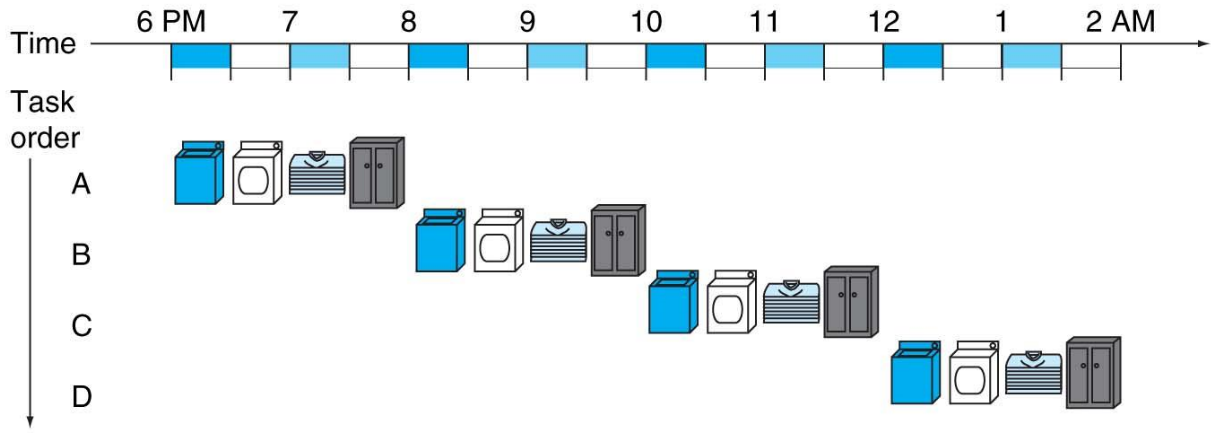
Ben

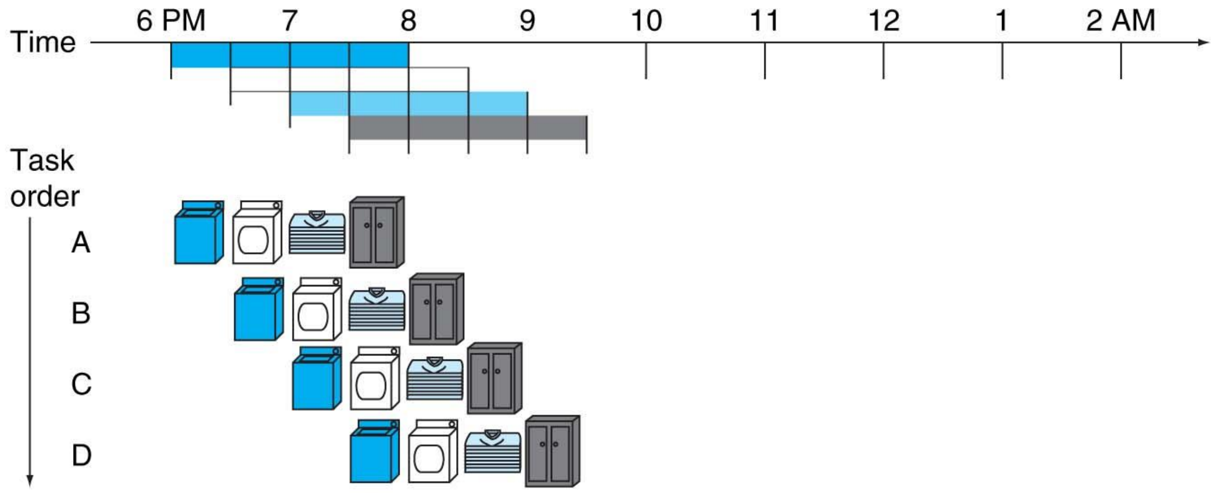
Peter

Jim

Tom

§2.5 Hazards of Pipelining





Data Dependences

- FLD f0, 0(x1)
- FADD.D f4, f0, f2

Name Dependences

```
FDIV.D      f2,  f6,  f4
FADD.D      f6,  f0,  f12
FSUB.D      f8,  f6,  f14
```

DIV&ADD: ***Anti-dependence***

Change f6 as S:

```
FDIV.D      f2,  f6,  f4
FADD.D      S,   f0,  f12
FSUB.D      f8,  S,   f14
```

```
FDIV.D      f2,  f6,  f4
FADD.D      f6,  f0,  f12
FSUB.D      f2,  f6,  f14
```

DIV&SUB: ***Output-dependence***

Change f2 as S:

```
FDIV.D      f2,  f0,  f4
FADD.D      f6,  f0,  f12
FSUB.D      S,   f6,  f14
```

Control Dependences

```
if p1 {
        Statement 1
    }
Statement
if p2 {
        Statement 2
    }
```

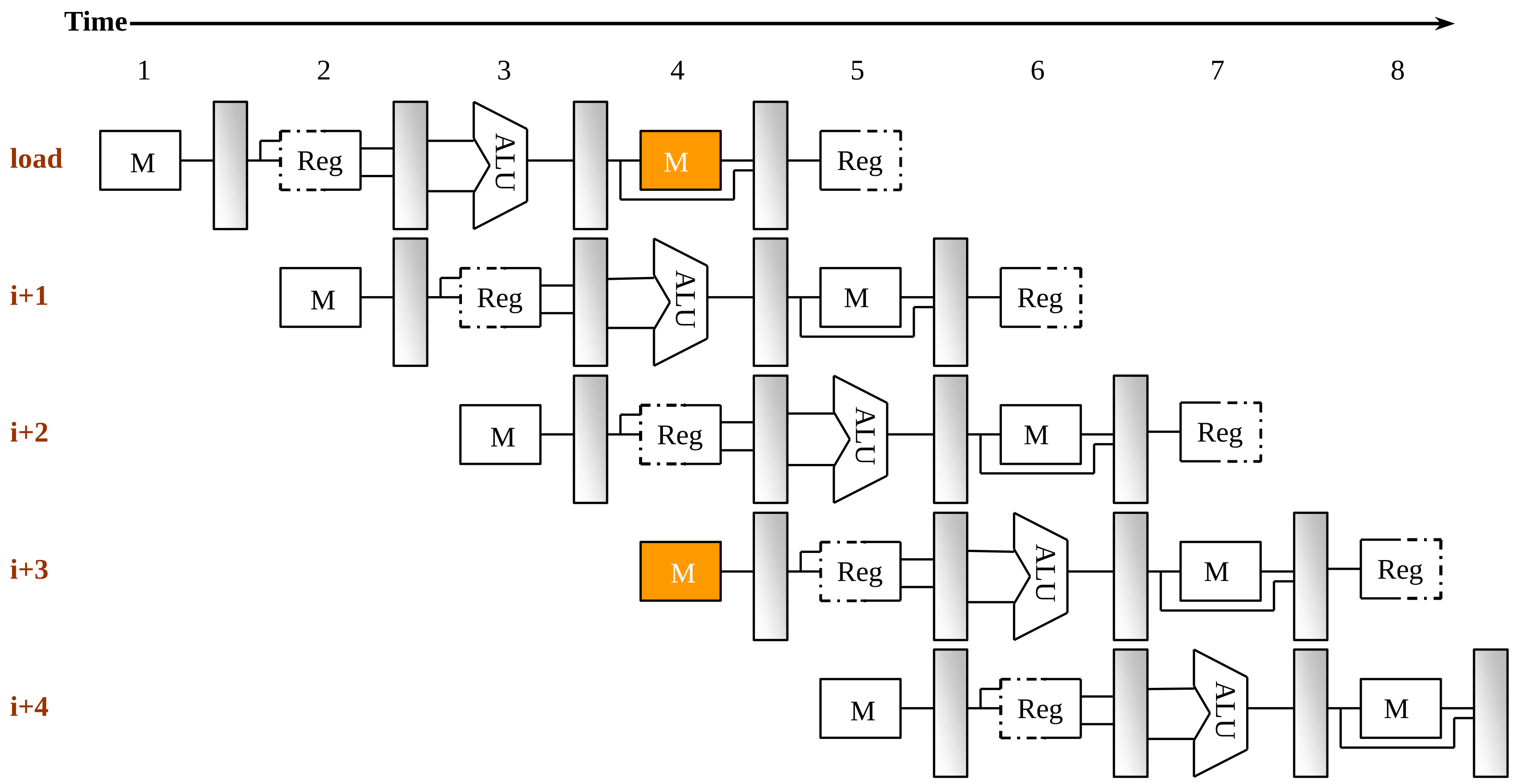
Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

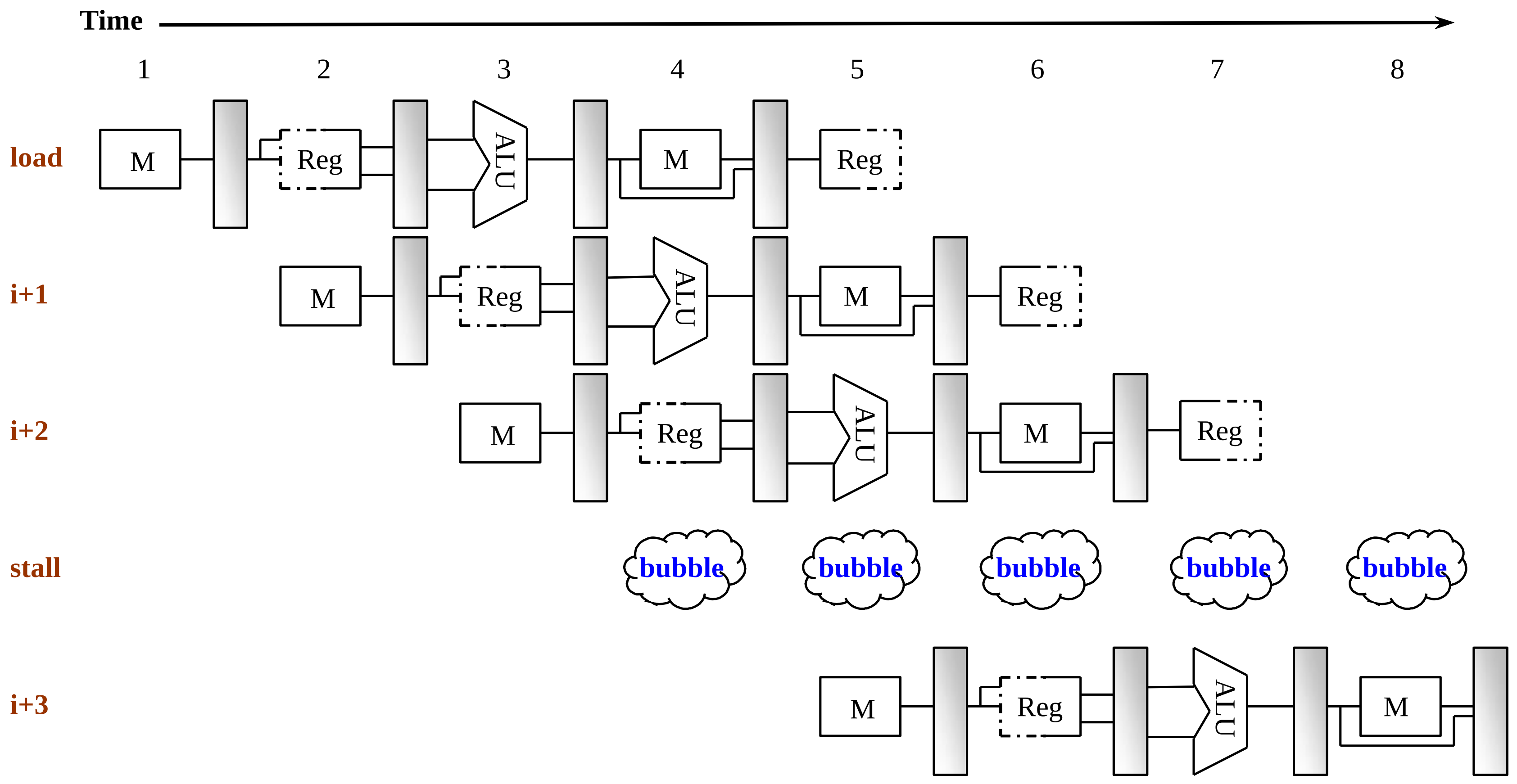
Structure Hazards

- Conflict for use of a resource
- In a pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches

Structure Hazards



Structure Hazards



Structure Hazards

instruction	Clock cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	MEM	WB					
i+1		IF	ID	EX	MEM	WB				
i+2			IF	ID	EX	MEM	WB	WB		
i+3				stall	IF	ID	EX	MEM	WB	
i+4						IF	ID	EX	MEM	WB
i+5							IF	ID	EX	MEM

How to Deal with Structural Hazard?

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

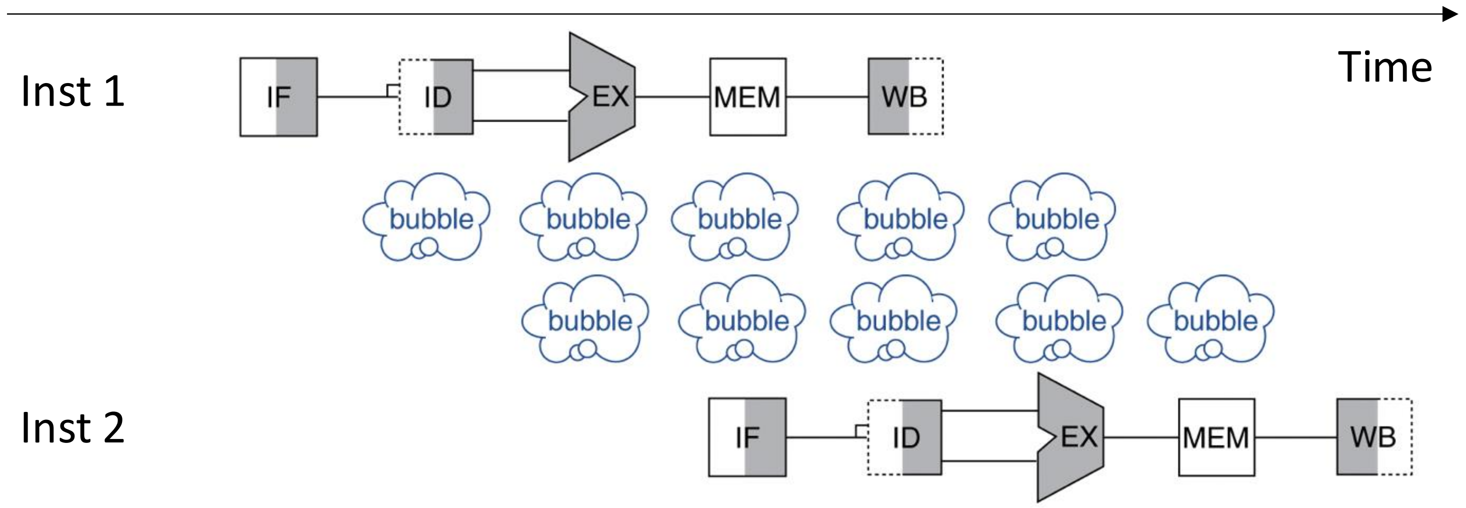
- Solution 1: Instructions take it in turns to use resource, some instructions have to stall

- Solution 2: Add more hardware to machine

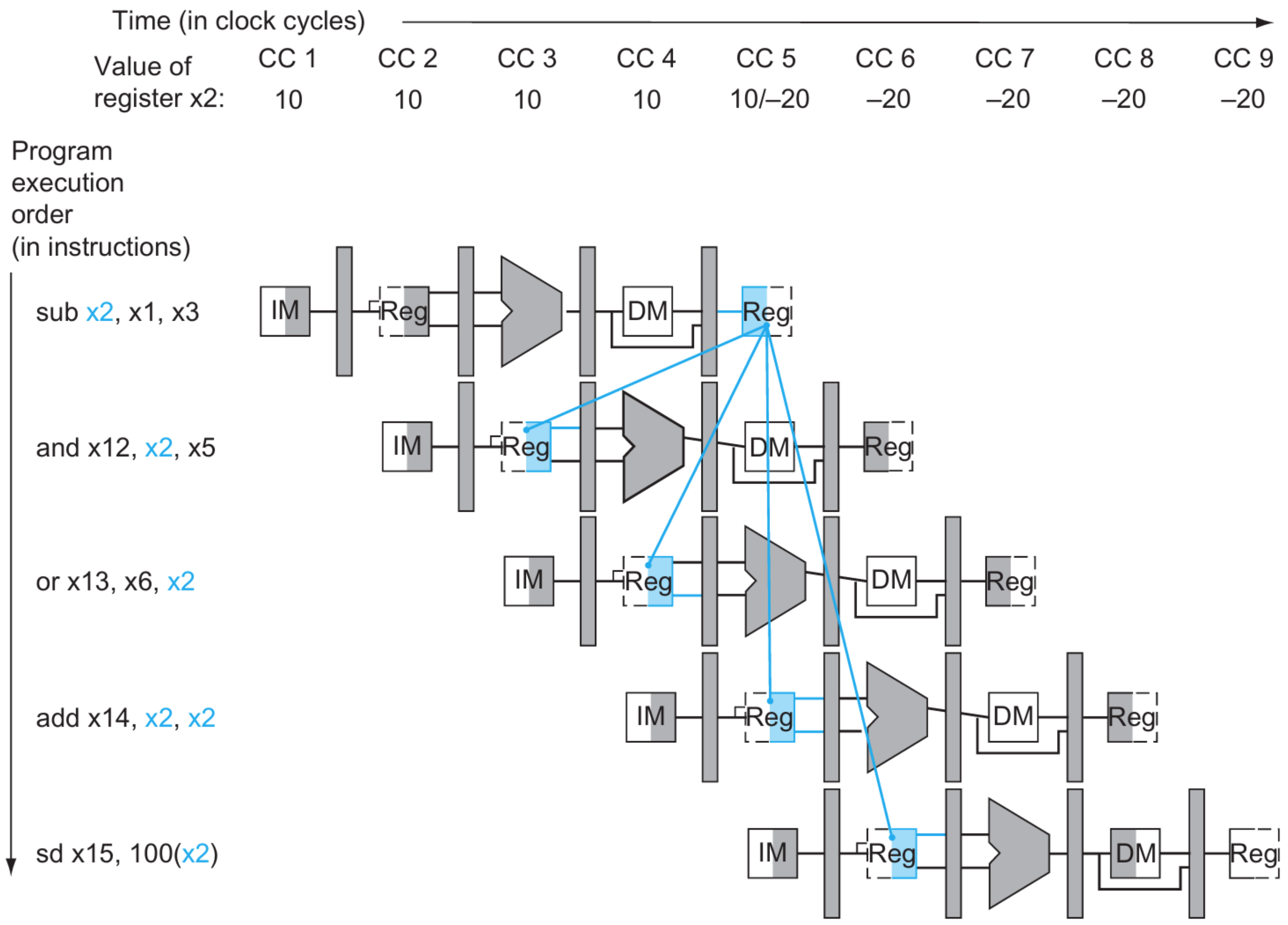
Can always solve a structural hazard by adding more hardware

Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - Inst1: add x5, x28, x29
  - Inst2: sub x30, x5, x31



Data Hazards


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Value of register x2: 10 10 10 10 10/–20 –20 –20 –20 –20
Program execution order (in instructions)
sub x2, x1, x3
and x12, x2, x5
or x13, x6, x2
add x14, x2, x2
sd x15, 100(x2)
IM
Reg
DM

Data Hazards

- Read after write: RAW

```
FADD.D    f6, f0, f12
FSUB.D    f8, f6, f14
```

- Write after read: WAR

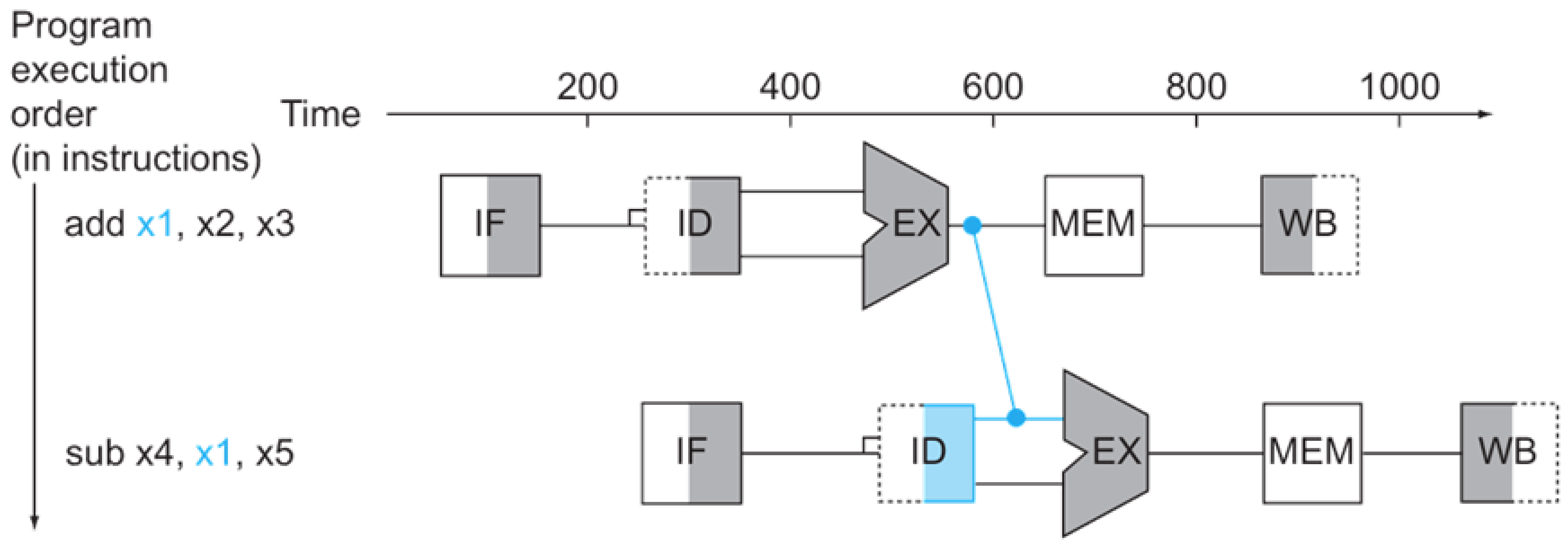
```
FDIV.D    f2, f6, f4
FADD.D    f6, f0, f12
```

- Write after write: WAW

```
FDIV.D    f2, f0, f4
FSUB.D    f2, f6, f14
```

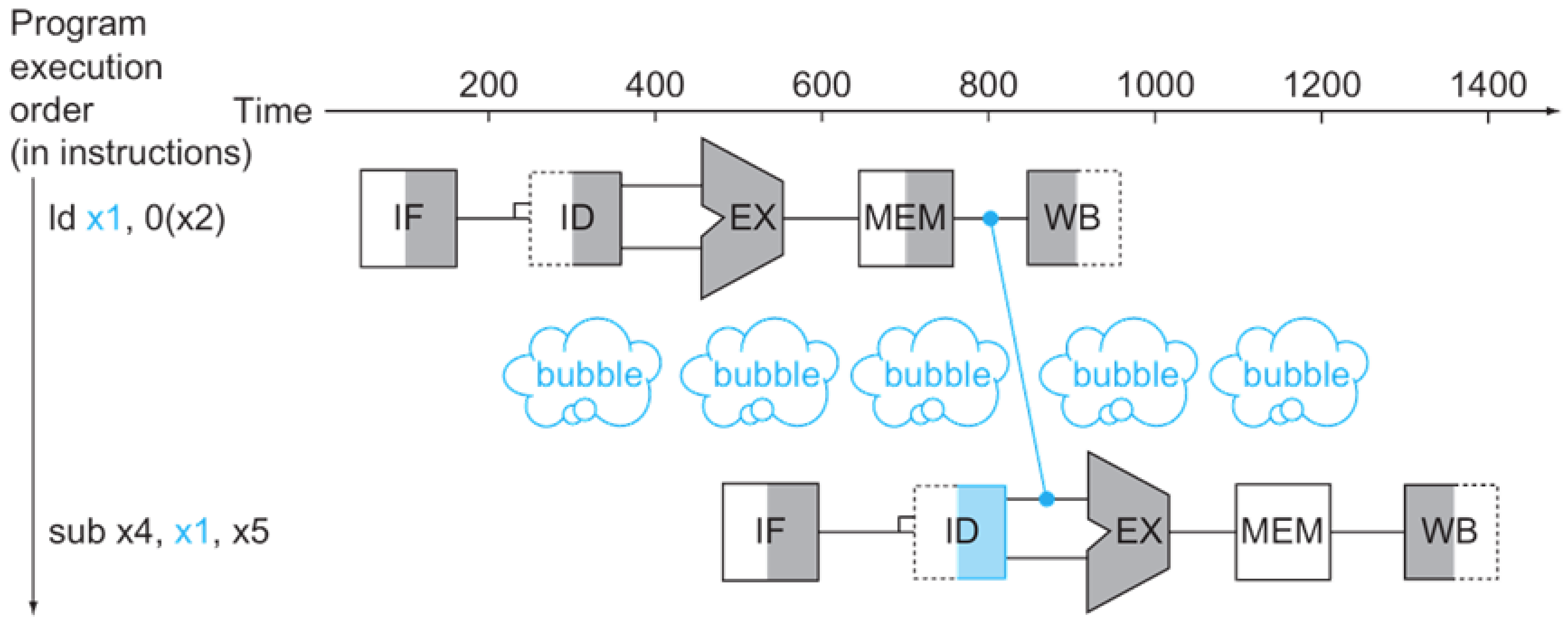
Forwarding (aka Bypassing)

- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires extra connections in the datapath



Load-Use Data Hazard

- Can’t always avoid stalls by forwarding
  - If value not computed when needed
  - Can’t forward backward in time!



Forwarding (aka Bypassing)

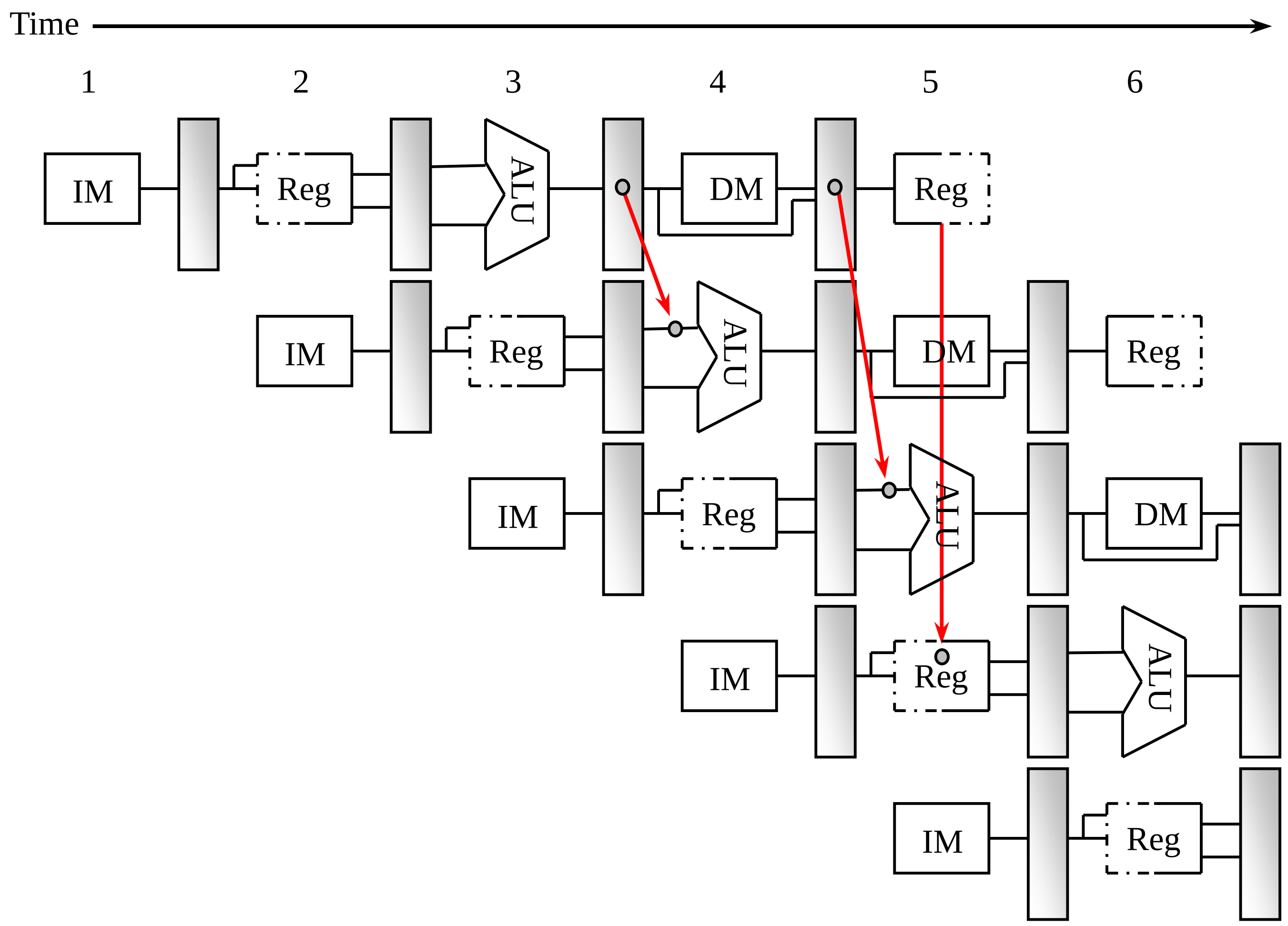
add **x1**， x2， x3

sub x4， **x1**， x5

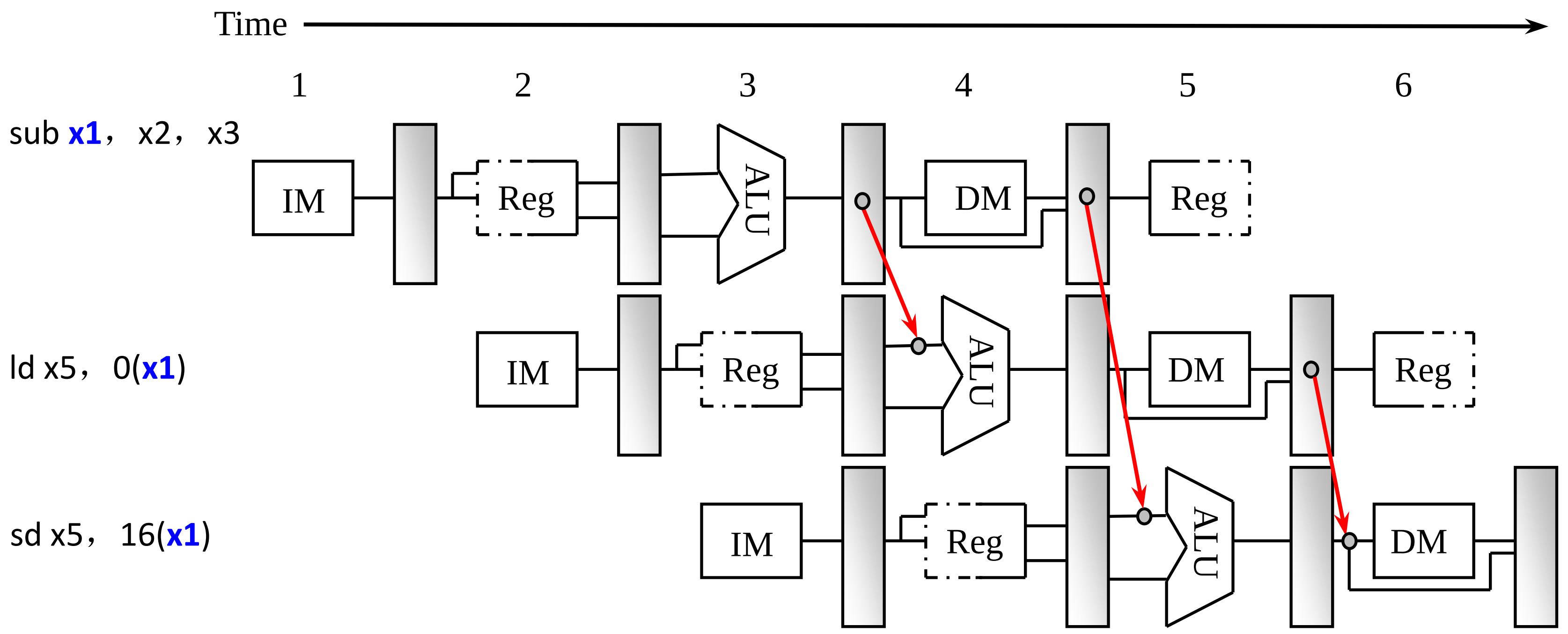
xor x6， **x1**， x7

and x8， **x1**， x9

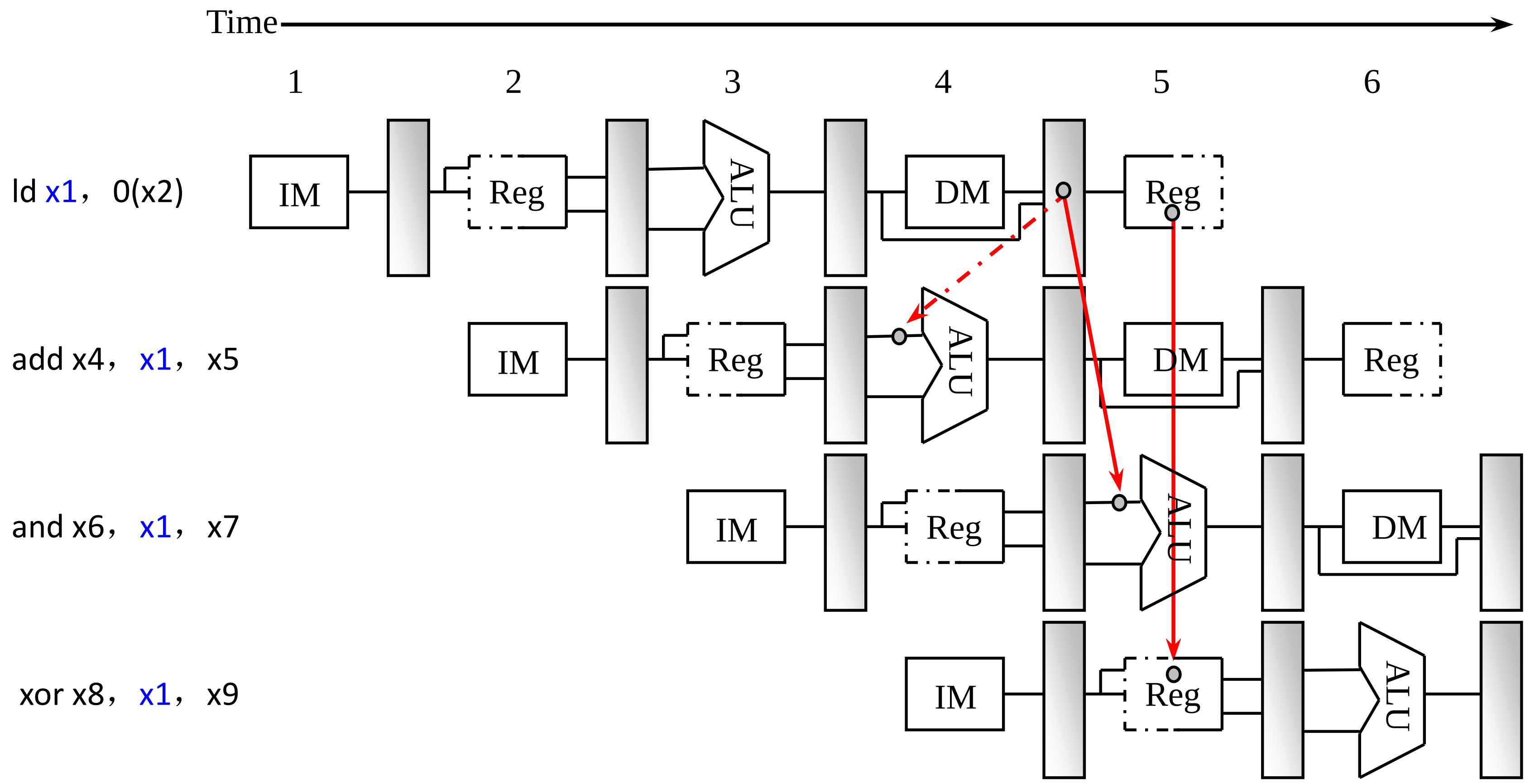
or x10， **x1**， x11



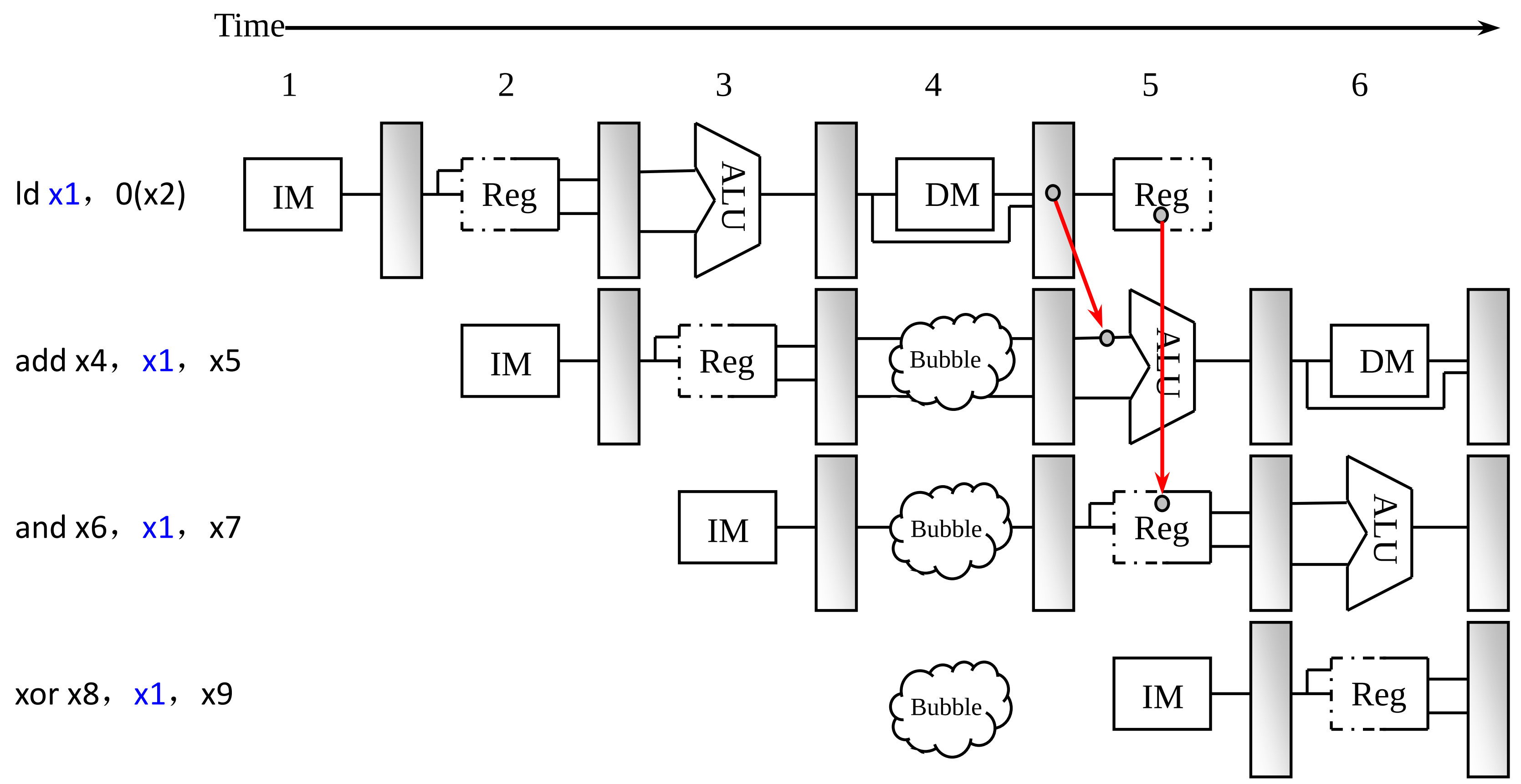
Forwarding (aka Bypassing)



Forwarding (aka Bypassing)



Forwarding with bubble



Forwarding with bubble

ld x1, 0(x2)	IF	ID	EX	MEM	WB			
add x4, x1, x5		IF	ID	EX	MEM	WB		
and x6, x1, x7			IF	ID	EX	MEM	WB	
xor x8, x1, x9				IF	ID	EX	MEM	WB

ld x1, 0(x2)	IF	ID	EX	MEM	WB			
add x4, x1, x5		IF	ID	stall	EX	MEM	WB	
and x6, x1, x7			IF	stall	ID	EX	MEM	WB
xor x8, x1, x9				stall	IF	ID	EX	MEM

Code Scheduling to Avoid Stalls

A = B + C

ld Rb， B	IF	ID	EX	MEM	WB				
ld Rc， C		IF	ID	EX	MEM	WB			
add Ra， Rb， Rc			IF	ID	stall	EX	MEM	WB	
sd Ra， A				IF	stall	ID	EX	MEM	WB

Code Scheduling to Avoid Stalls

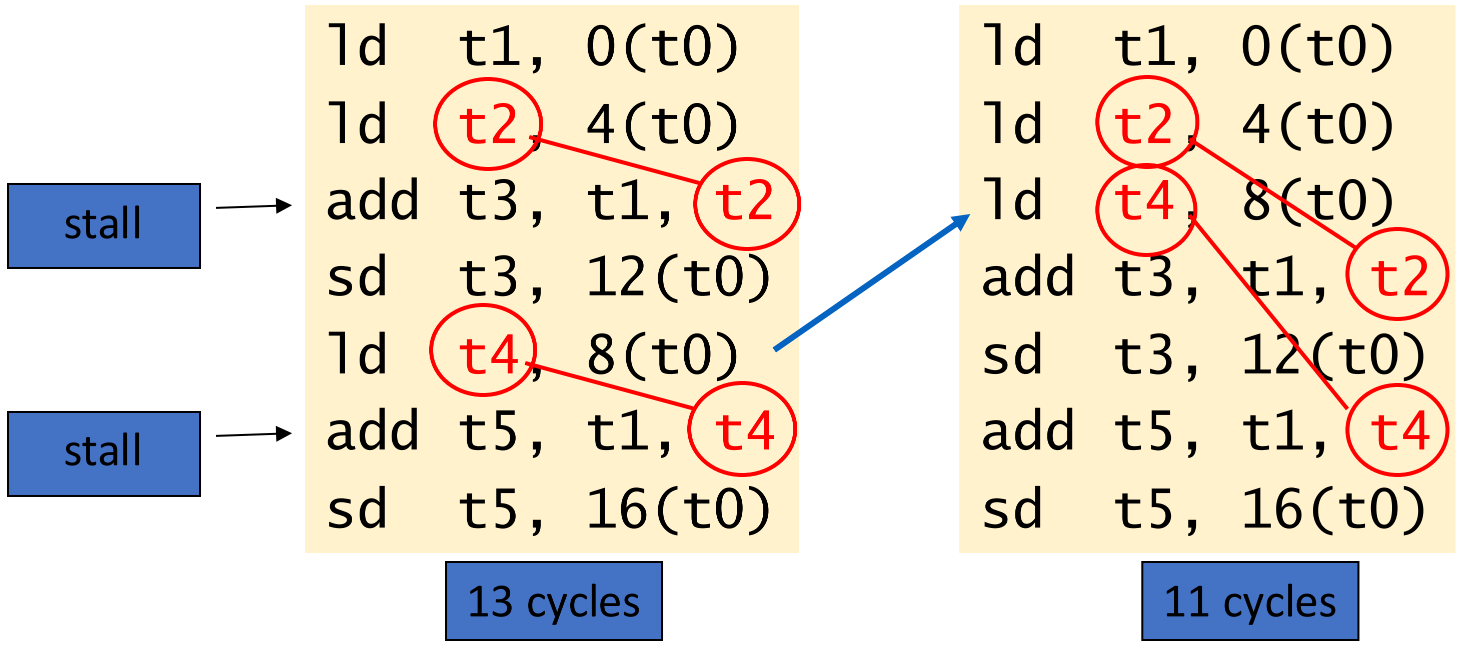
A = B + C

D = E - F

Before Scheduling	After Scheduling
`ld  Rb, B`	
`ld  Rc, C`	
`add Ra, Rb, Rc`	
`sd  Ra, A`	
`ld  Re, E`	
`ld  Rf, F`	
`sub Rd, Re, Rf`	
`Sd  Rd, D`	

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for A = B + E; C = B + F;



```
ld  t1, 0(t0)
ld  t2, 4(t0)
add t3, t1, t2
sd  t3, 12(t0)
ld  t4, 8(t0)
add t5, t1, t4
sd  t5, 16(t0)
```

```
ld  t1, 0(t0)
ld  t2, 4(t0)
ld  t4, 8(t0)
add t3, t1, t2
sd  t3, 12(t0)
add t5, t1, t4
sd  t5, 16(t0)
```

Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

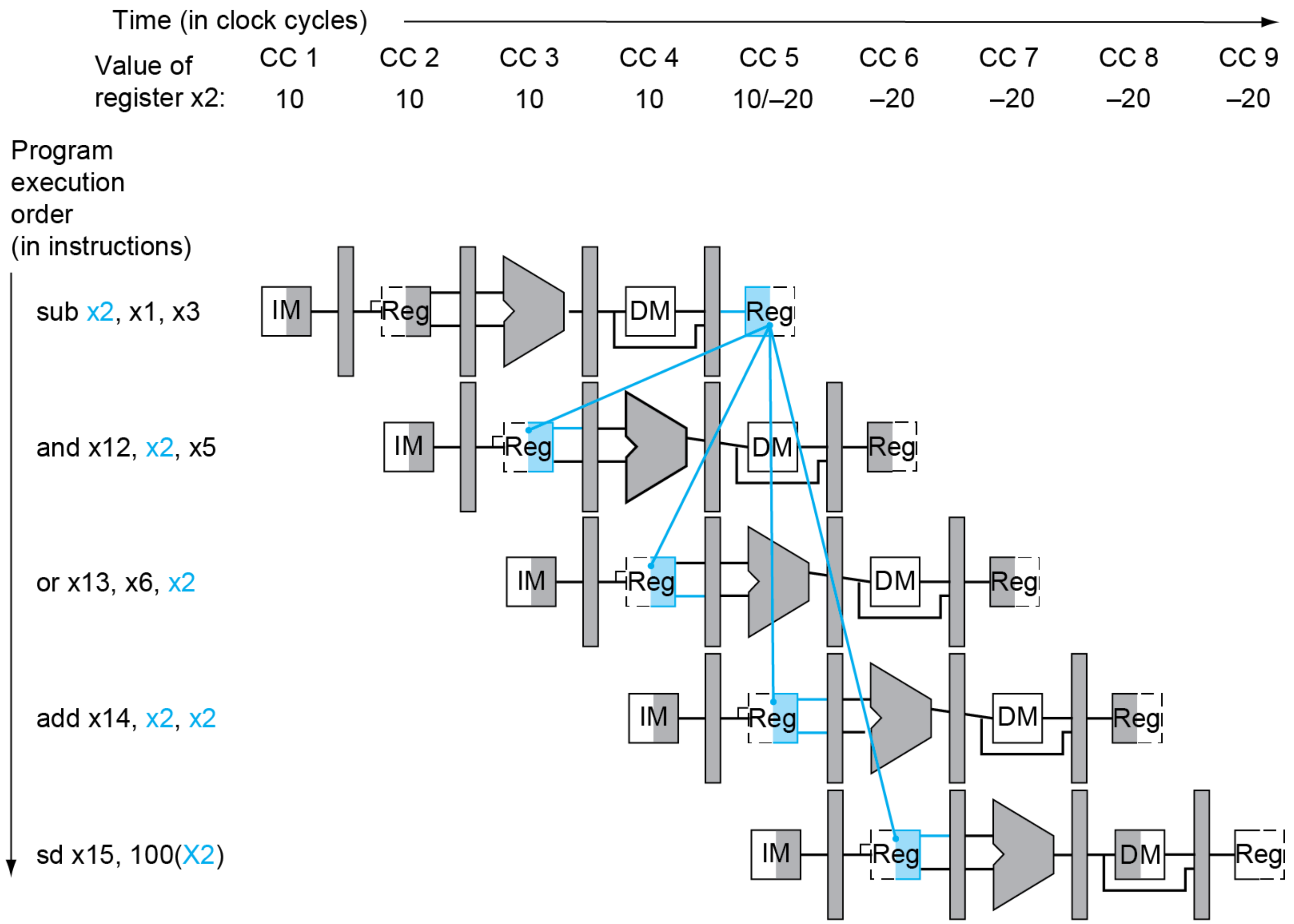
Data Hazards in ALU Instructions

- 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



Data Hazards in ALU Instructions

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

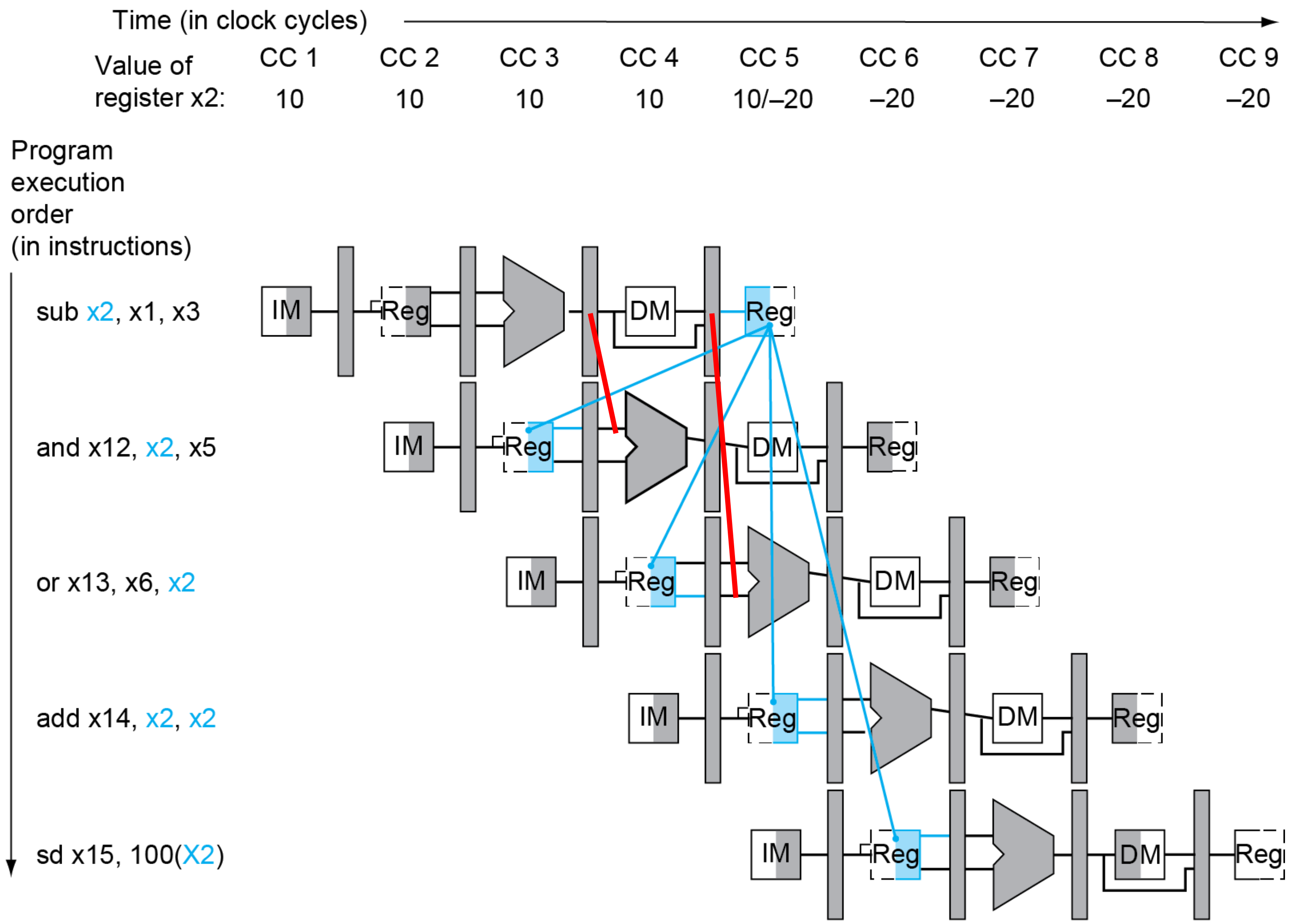
  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  } Fwd from EX/MEM pipeline reg

  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRt
  } Fwd from MEM/WB pipeline reg

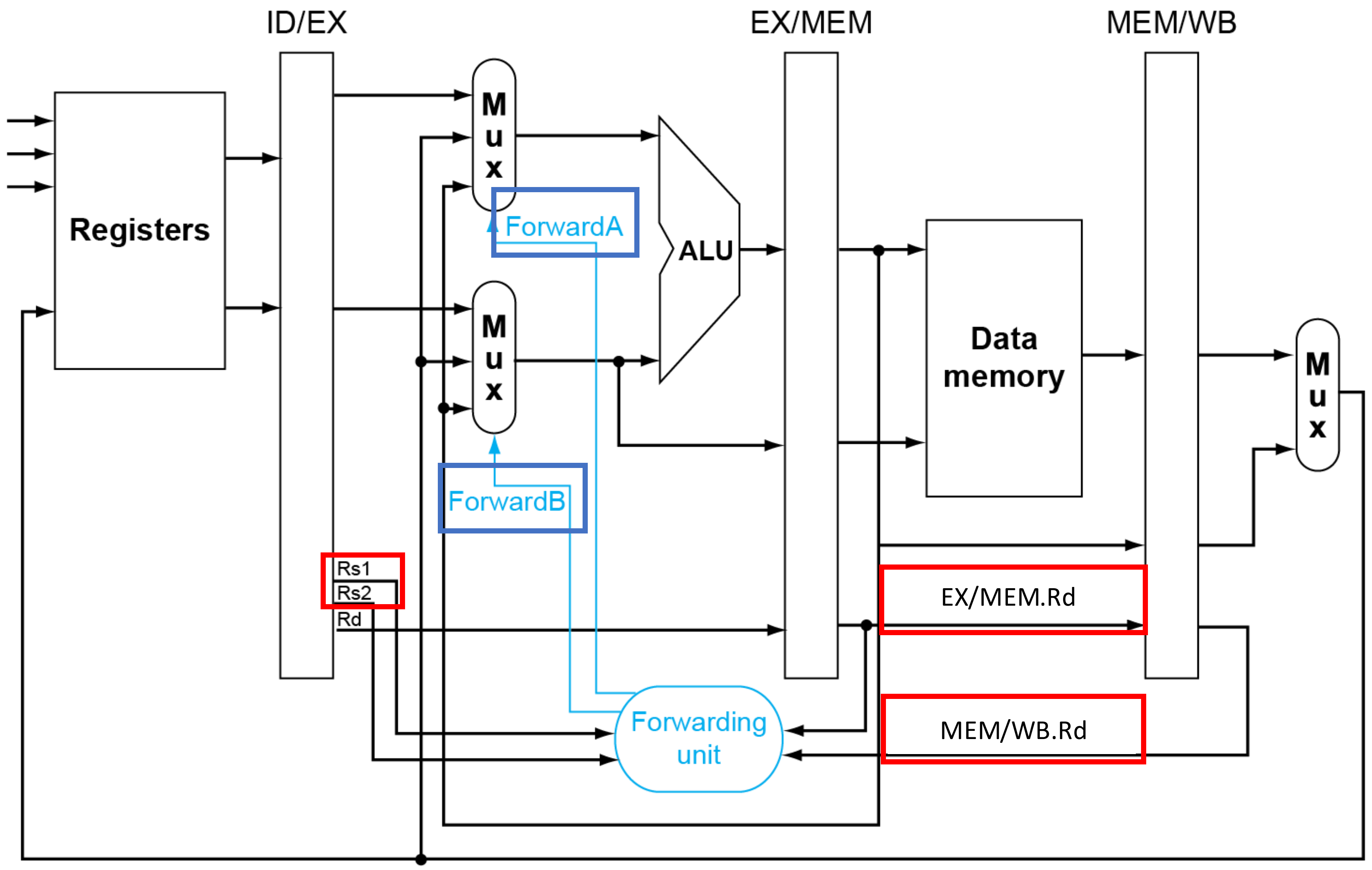
Detecting the Need to Forward

- 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 $zero
  - EX/MEM.RegisterRd ≠ 0,
    MEM/WB.RegisterRd ≠ 0

Dependencies & Forwarding



Forwarding Paths

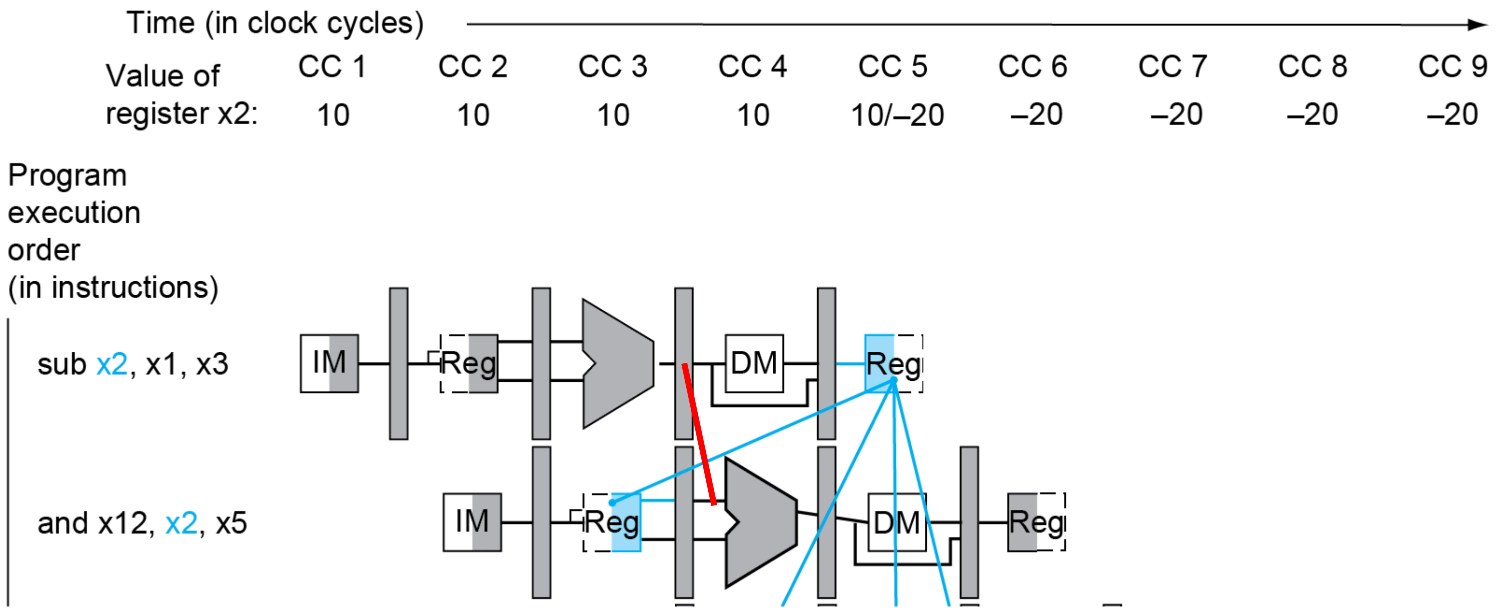


Forwarding Conditions

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

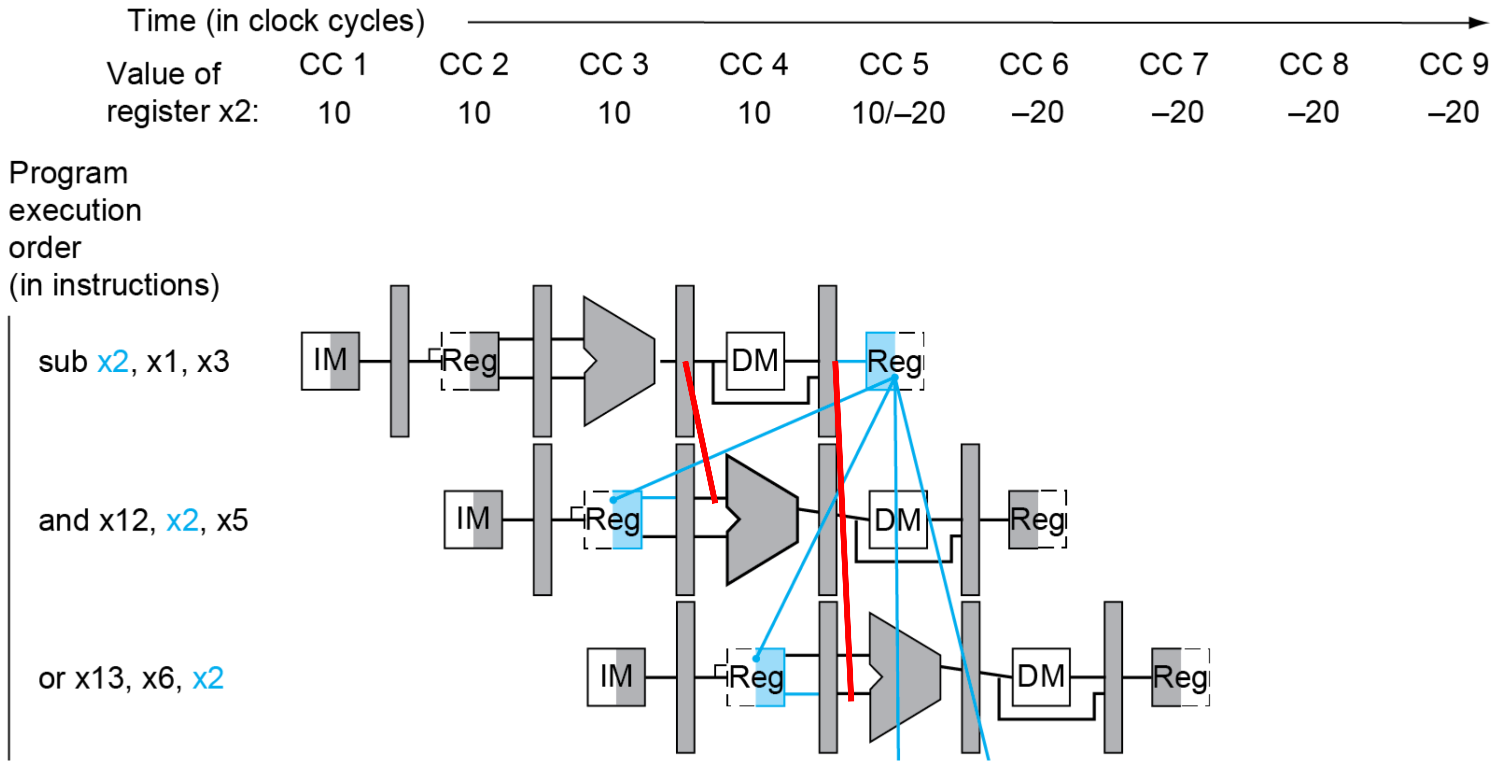
What is the Data hazards condition in the following case?



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.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

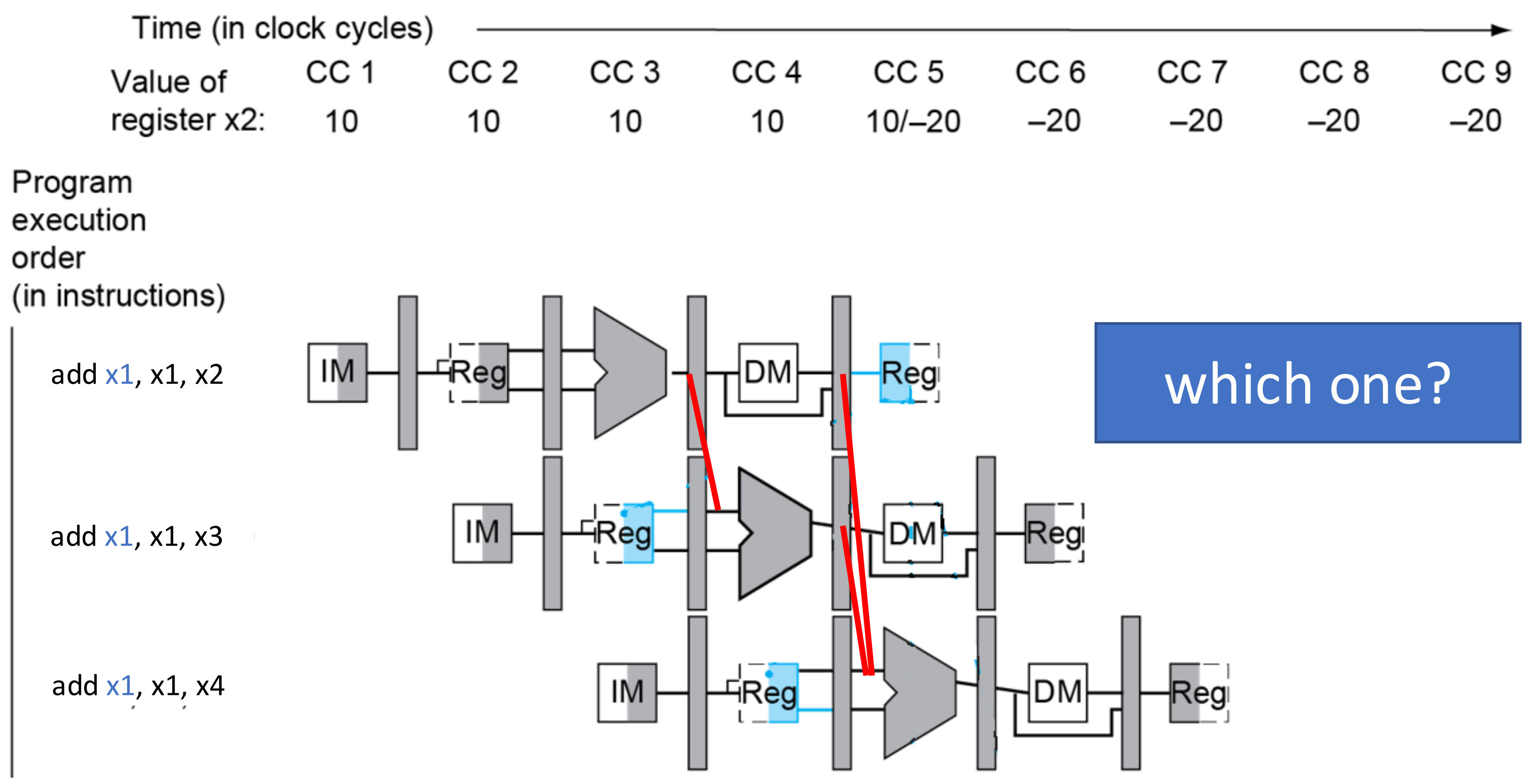
Double Data Hazard

- Consider the sequence:

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

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

Double Data Hazard



which one?

Such an exception should be added into MEM hazards!

Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    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))

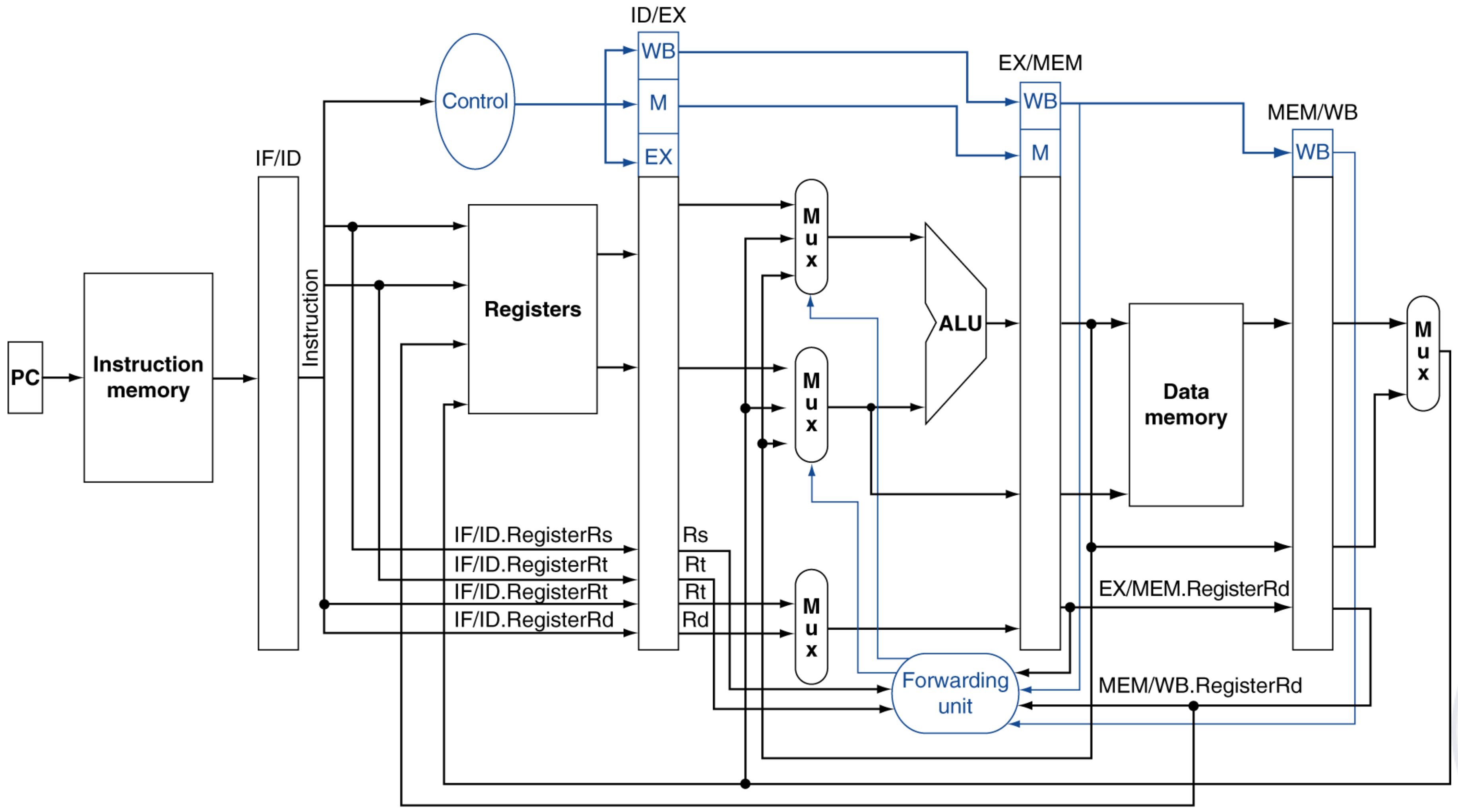
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB. Rd ≠ 0)

    and not(EX hazard)

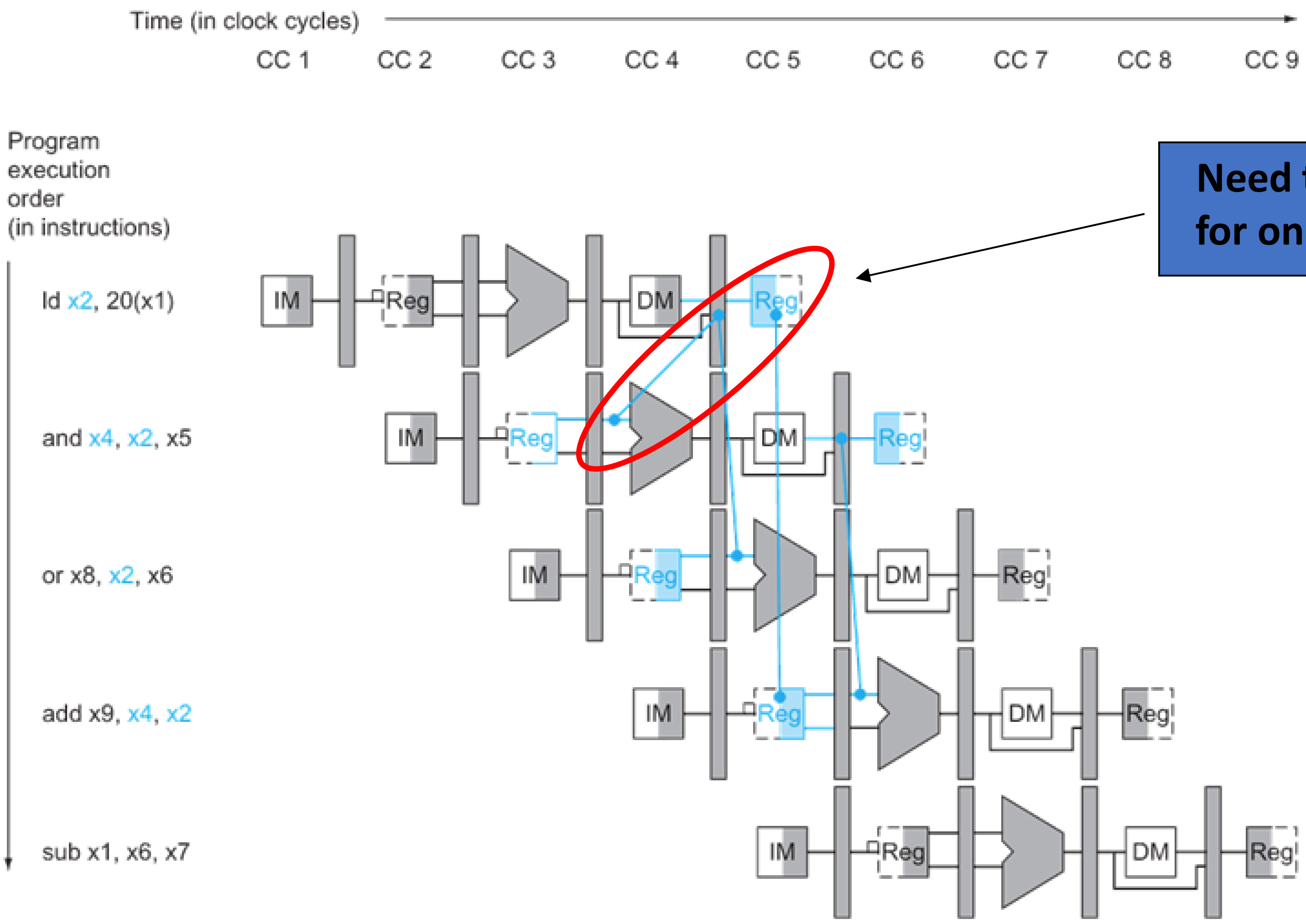
    and (MEM/WB. Rd = ID/EX. Rs2))

    ForwardB = 01

Datapath with Forwarding



Load-Use Data Hazard

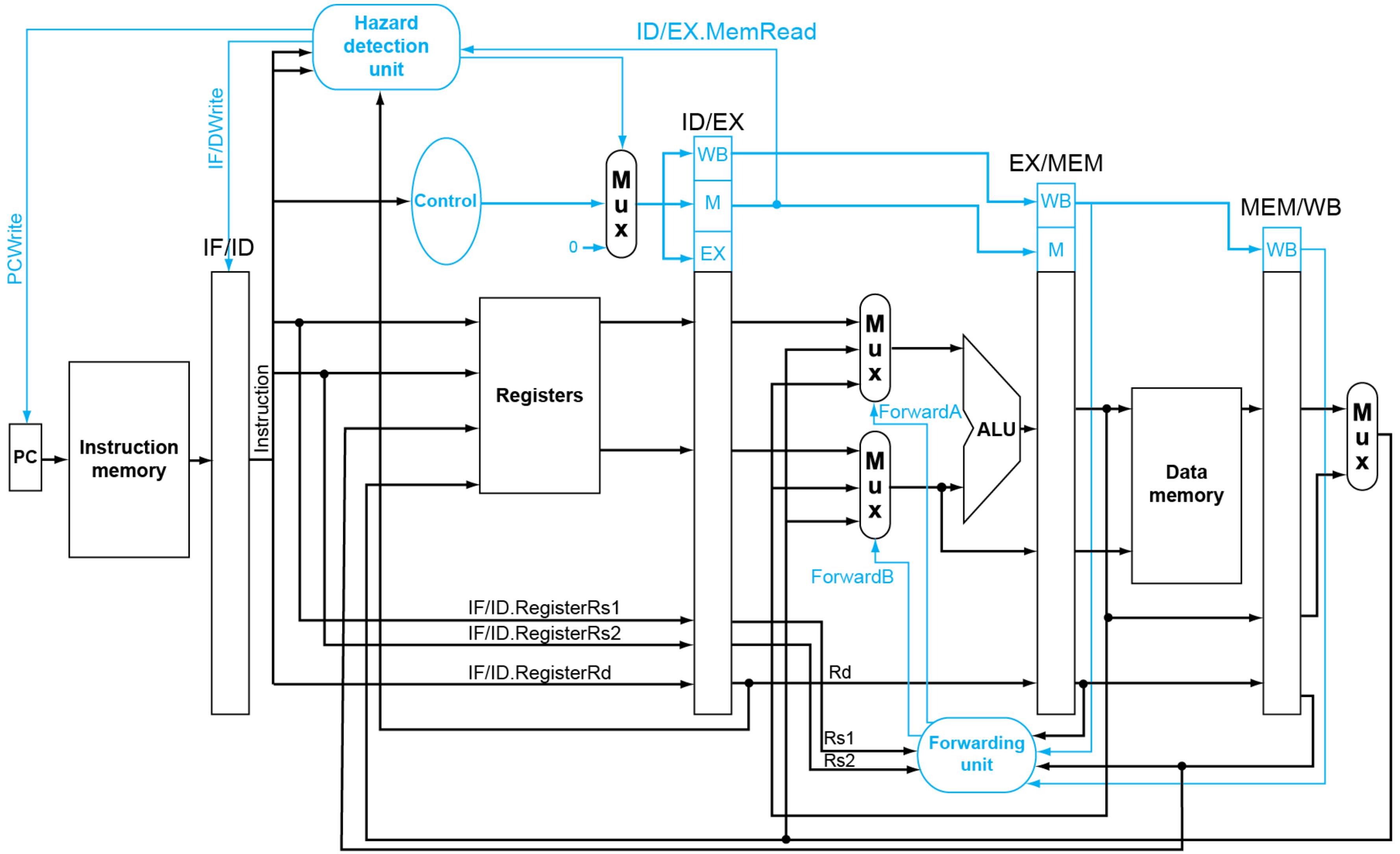


Need to stall for one cycle

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.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

Datapath with Hazard Detection



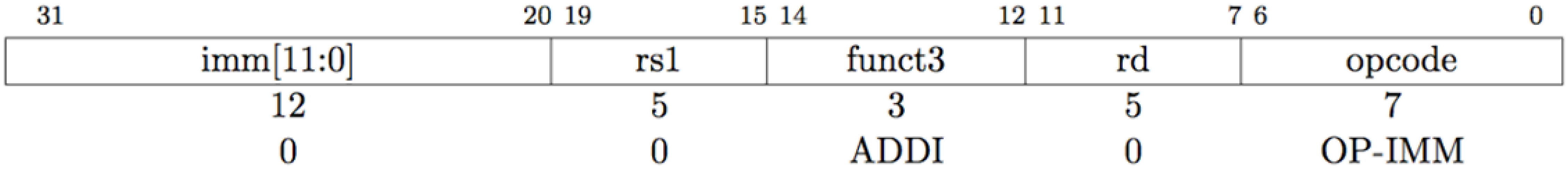
Summary of Data Hazards

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

How to Stall ?

NOP instruction

ADDI x0, x0, 0



31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
0	0	ADDI	0	OP-IMM

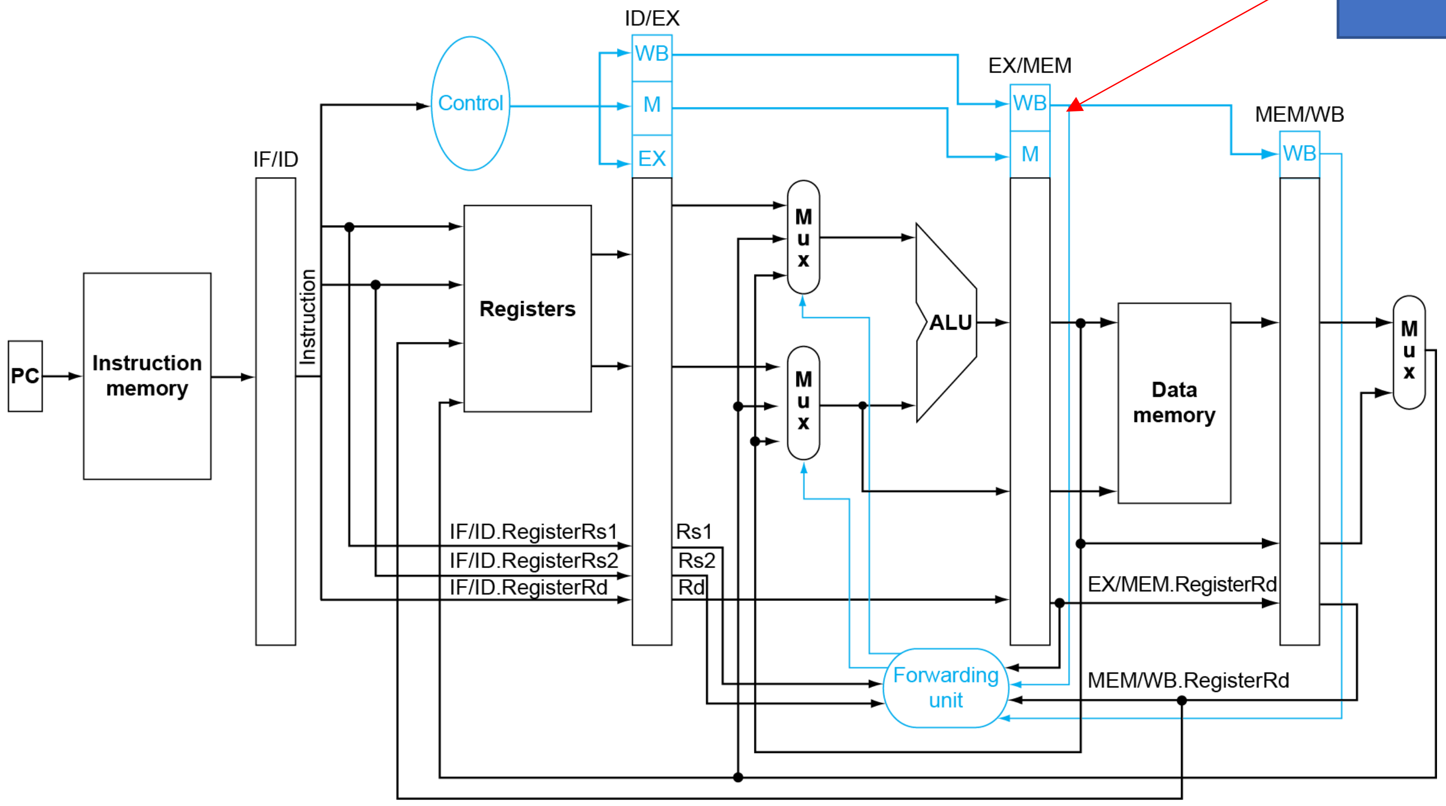
How to Stall the Pipeline

- 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

How to Stall the Pipeline

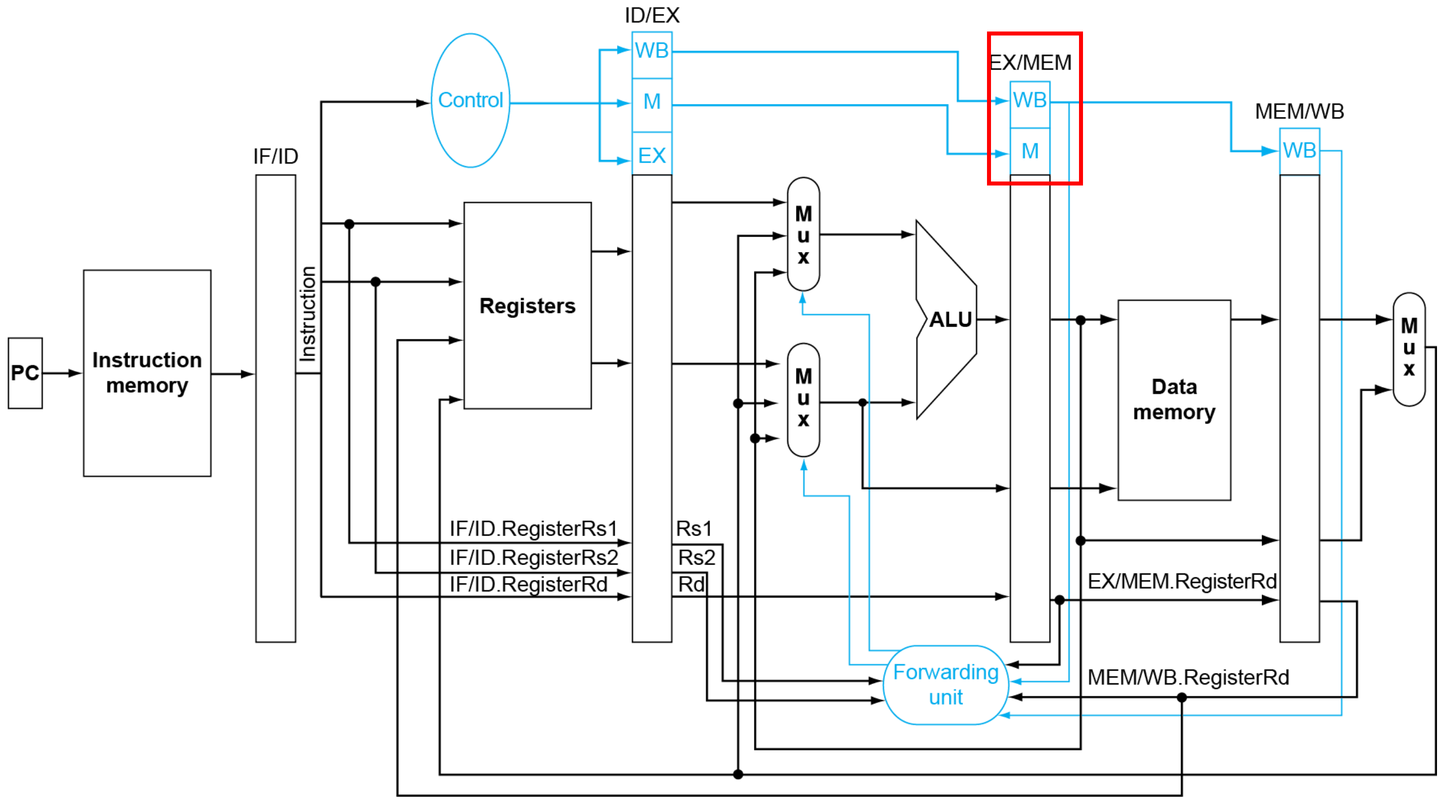
- Force control values in EX/MEM register to 0
  - MEM and WB do nop (no-operation)

Which one?



How to Stall the Pipeline

- Force control values in EX/MEM register to 0
  - MEM and WB do nop (no-operation)



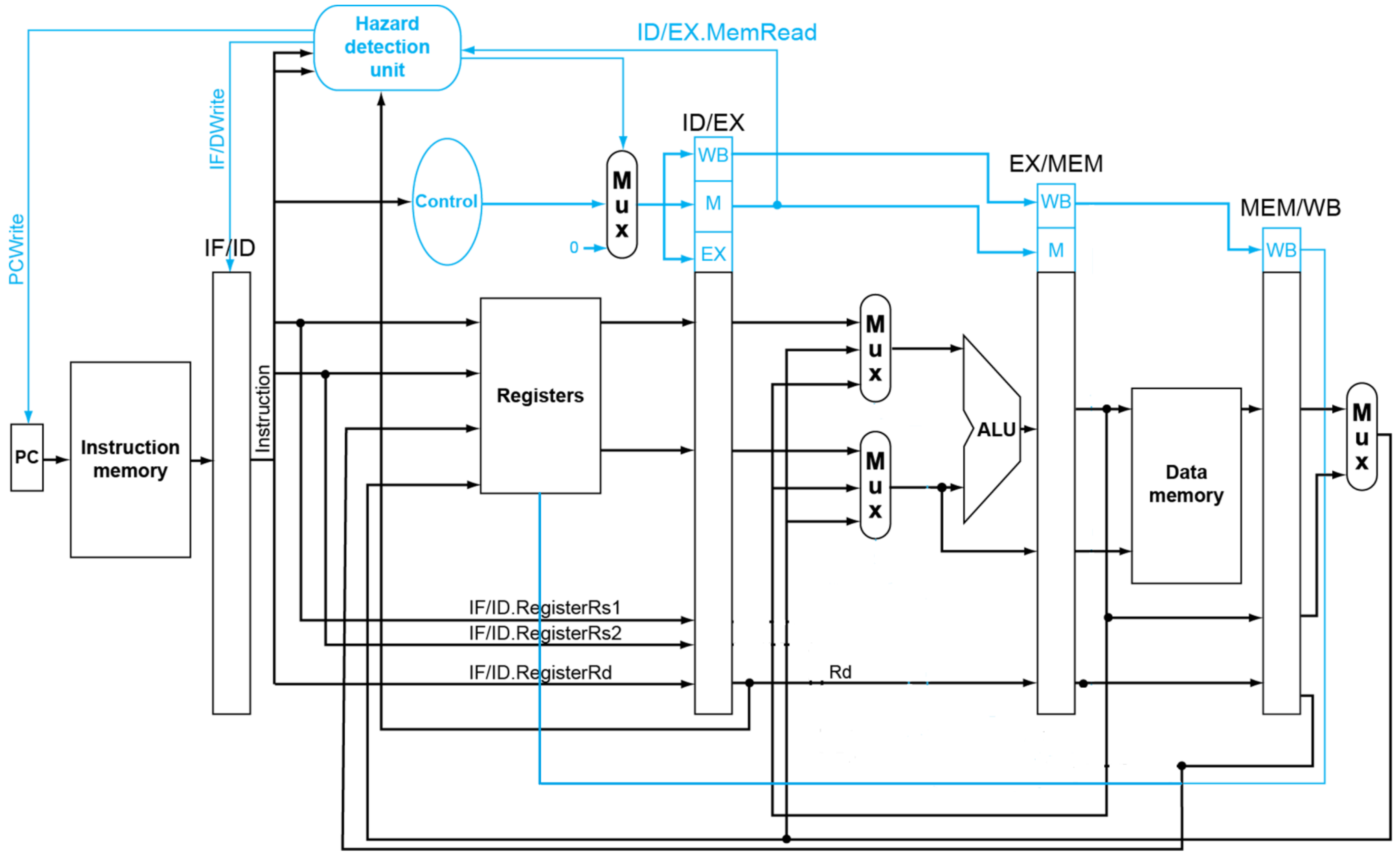
What's more?

How to Stall the Pipeline

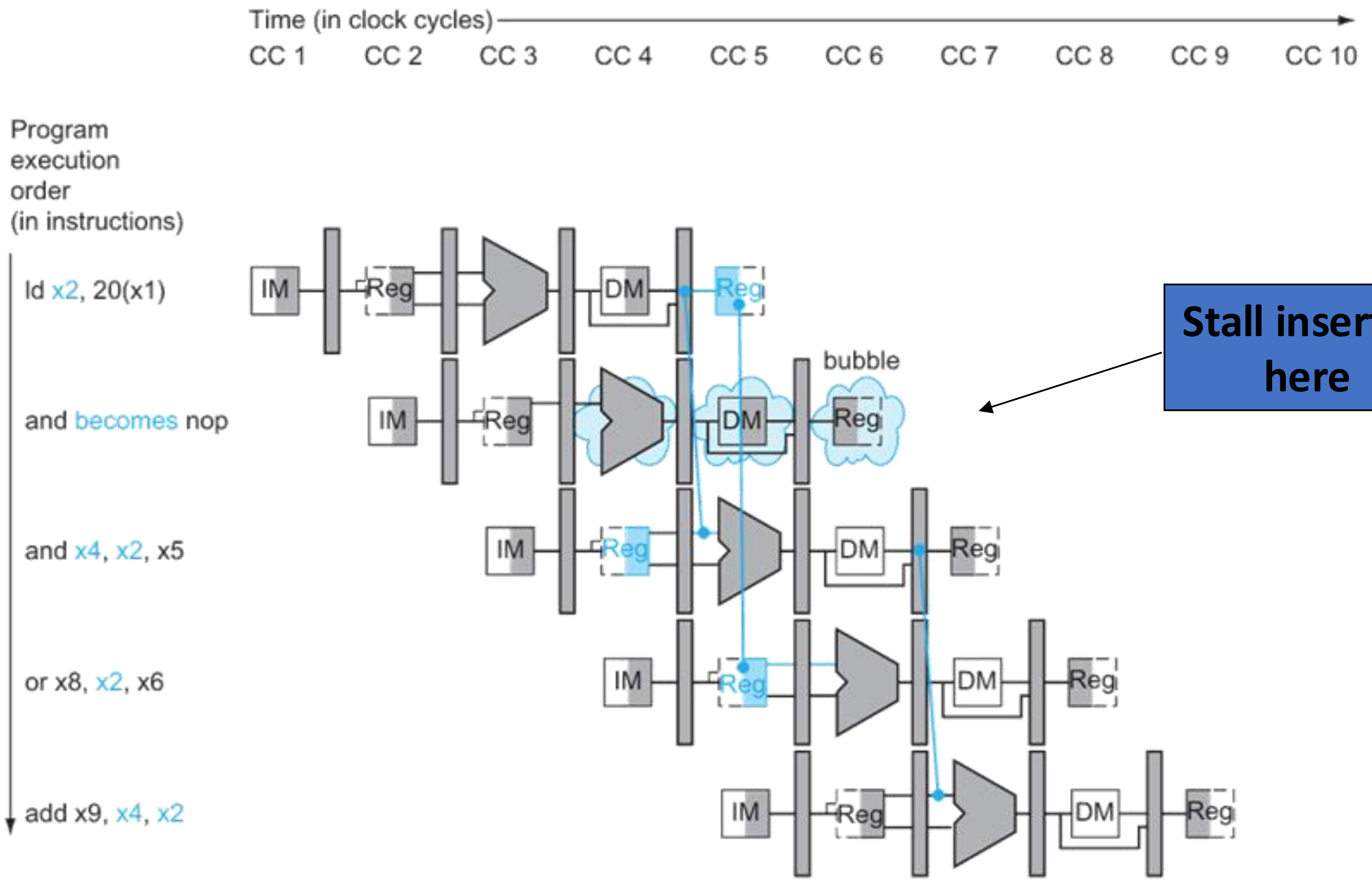
- Force control values in EX/MEM register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC, IF/ID and ID/EX register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `ld`
    - Can subsequently forward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

Any other solution?

Datapath with Stall

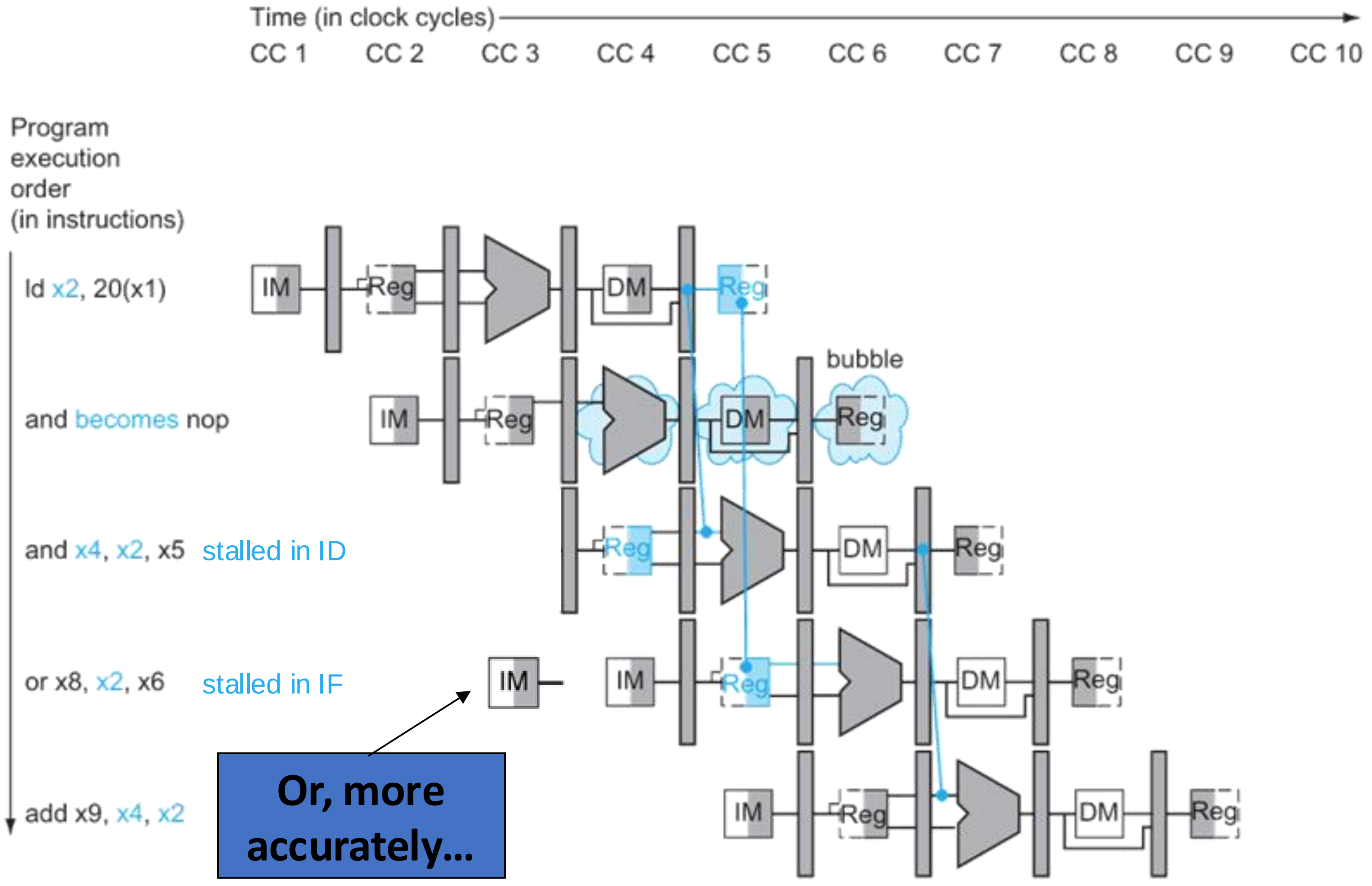


Stall/Bubble in the Pipeline



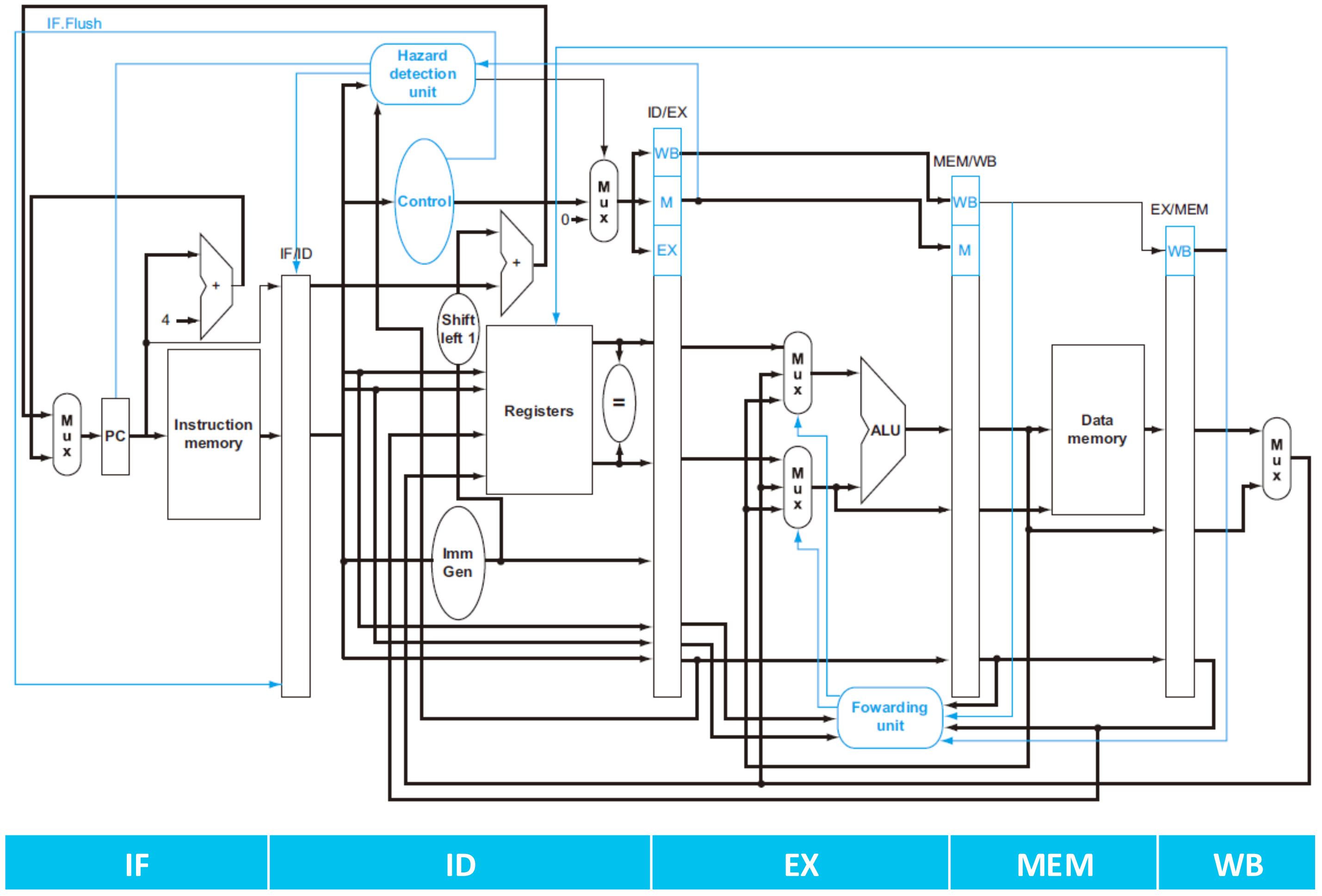
Stall inserted here

Stall/Bubble in the Pipeline



Or, more accurately…

Datapath with Hazard Detection

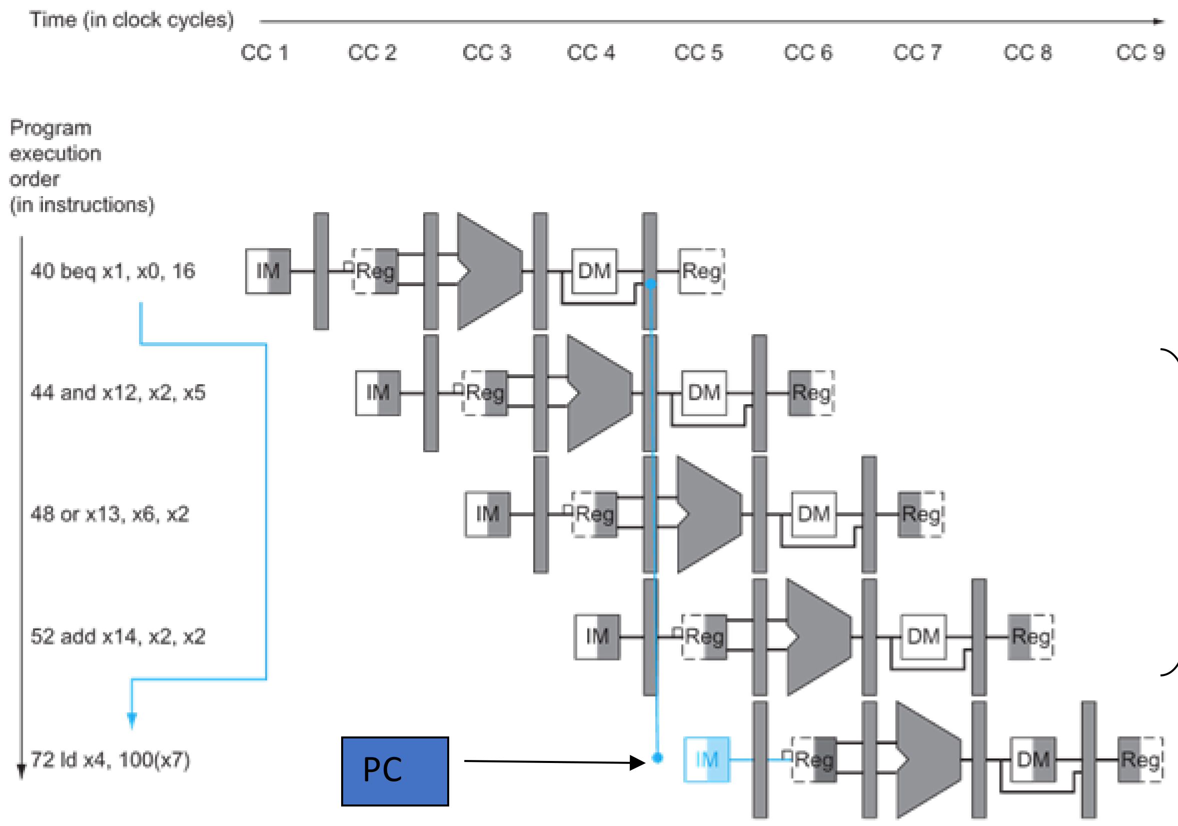


Stalls and Performance

- 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

Branch Hazards

- If branch outcome determined in MEM



Flush these instructions (Set control values to 0)

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

§2.7 Control Hazards

Unconditional Jump

Jal - Jump and Link

Jalr - Jump and Link-Register

Conditional Branch

Stall on Branch

Branch success

Branch	IF	ID	EX	MEM	WB					
Target		IF	stall	stall	IF	ID	EX	MEM	WB	
Target+1						IF	ID	EX	MEM	WB
Target+2							IF	ID	EX	MEM
Target+3								IF	ID	EX

Branch failure

Branch	IF	ID	EX	MEM	WB					
Instruction i		IF	stall	stall	IF	ID	EX	MEM	WB	
Instruction i+1						IF	ID	EX	MEM	WB
Instruction i+2							IF	ID	EX	MEM
Instruction i+3								IF	ID	EX

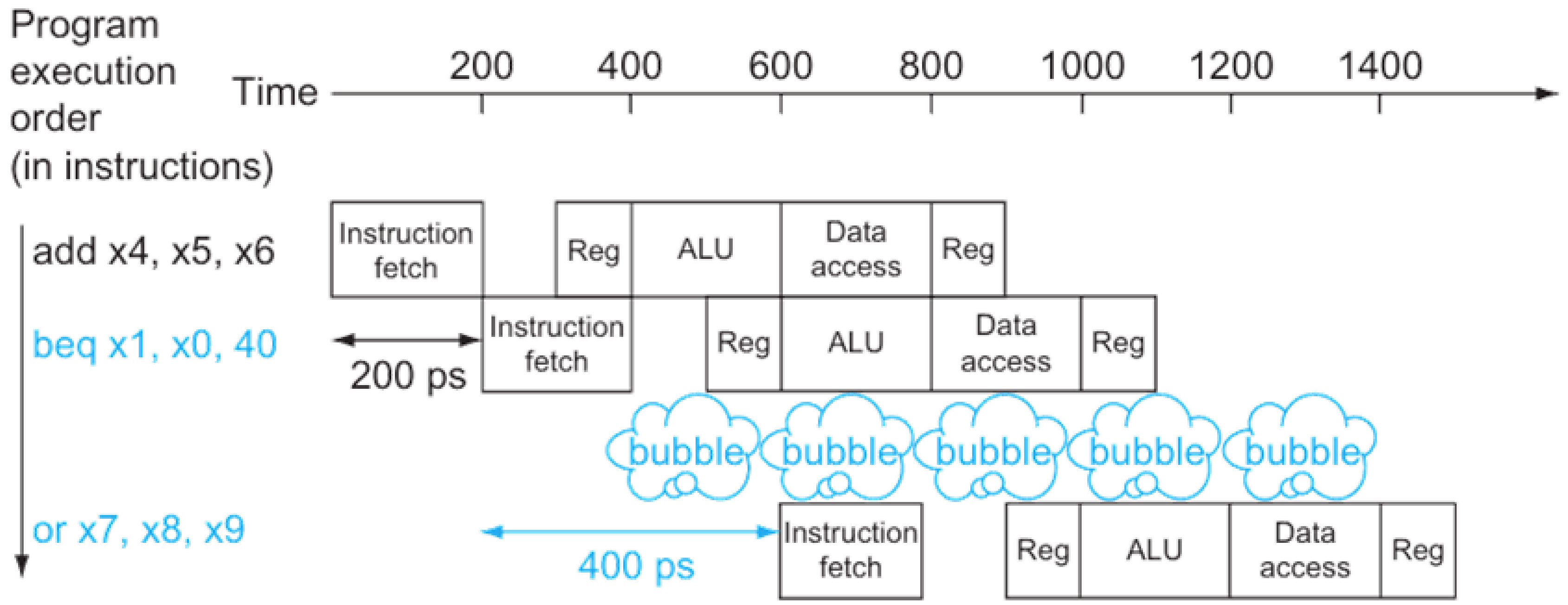
Stall on Branch

Branch causes a stall

Branch	IF	ID	EX	MEM	WB					
Instruction i		IF	IF	ID	EX	MEM	WB			
Instruction i+1				IF	ID	EX	MEM	WB		
Instruction i+2					IF	ID	EX	MEM	WB	
Instruction i+3						IF	ID	EX	MEM	WB

Control Hazards

- Wait until branch outcome determined before fetching next instruction



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

- Prediction taken
- Prediction not taken
- Delayed Branch

Predict not taken

Branch i（taken）	IF	ID	EX	MEM	WB				
Instruction i+1		IF	stall	stall	stall	stall			
Branch target j			IF	ID	EX	MEM	WB		
Branch target j+1				IF	ID	EX	MEM	WB	
Branch target j+2					IF	ID	EX	MEM	WB

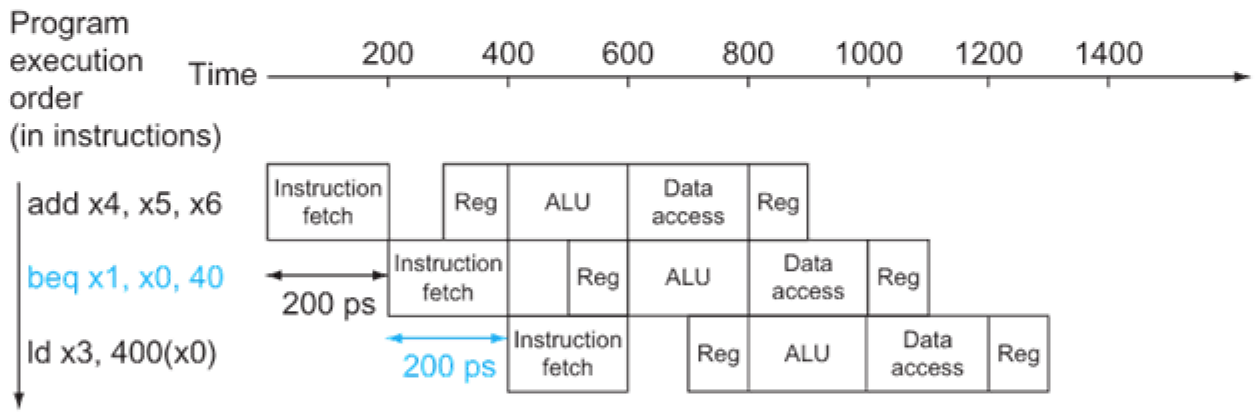
Branch i（not taken）	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

The behavior of a delayed branched is the same whether or not the branch is taken.

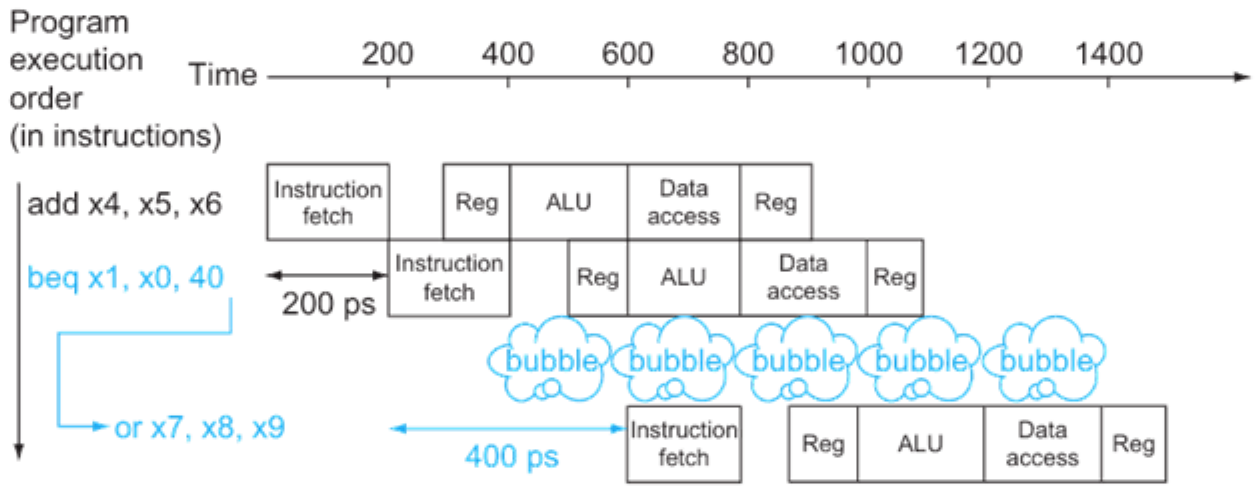
branch not taken instruction	Branch i	IF	ID	EX	MEM	WB				
	Branch Delay instruction i+1		IF	ID	EX	MEM	WB			
	instruction i+2			IF	ID	EX	MEM	WB		
	instruction i+3				IF	ID	EX	MEM	WB	
	instruction i+4					IF	ID	EX	MEM	WB
branch taken Instruction	Branch i	IF	ID	EX	MEM	WB				
	Branch Delay instruction i+1		IF	ID	EX	MEM	WB			
	Branch target j			IF	ID	EX	MEM	WB		
	Branch target j+1				IF	ID	EX	MEM	WB	
	Branch target j+2					IF	ID	EX	MEM	WB

Predict Not Taken

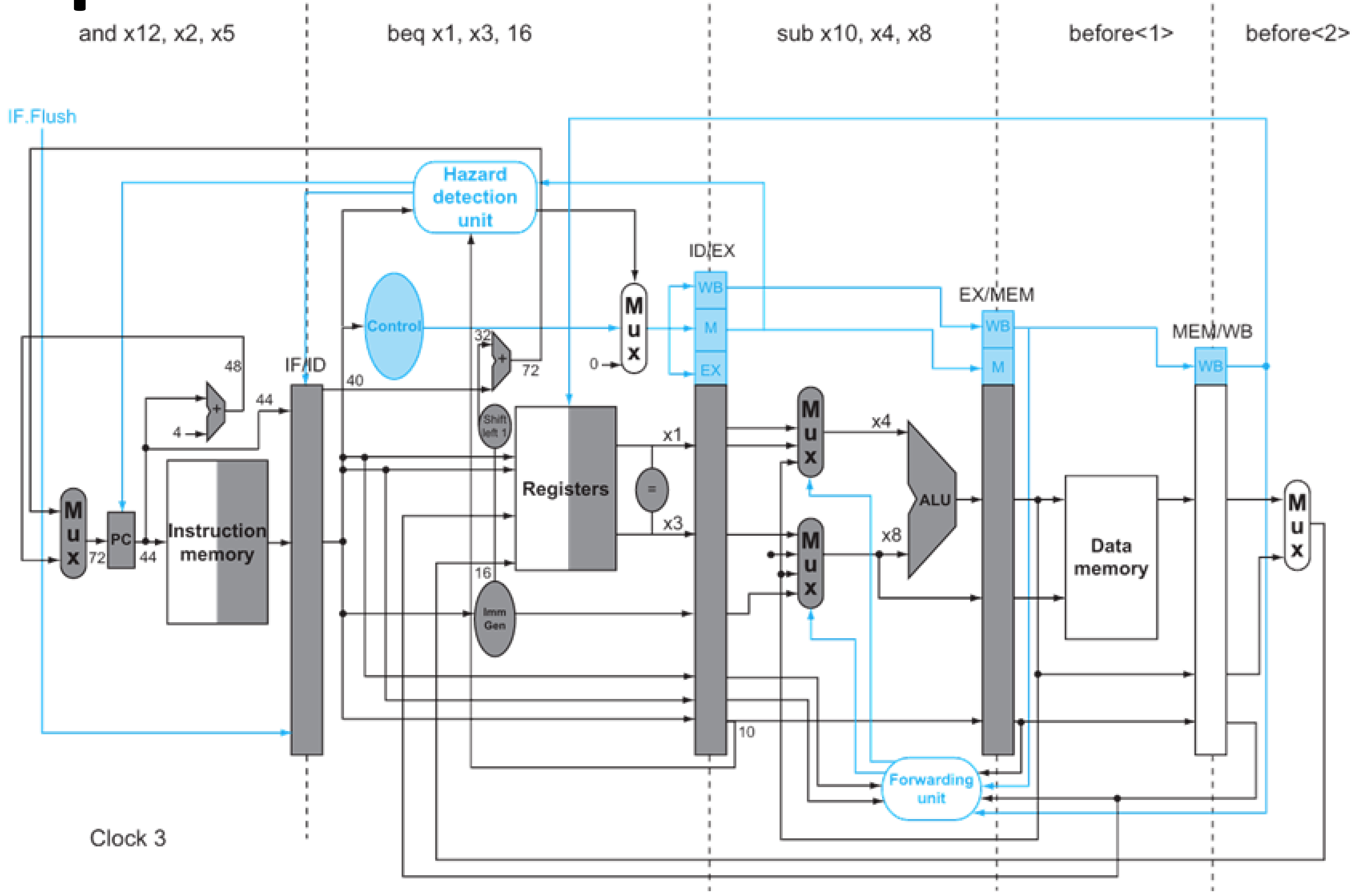
Branch not taken



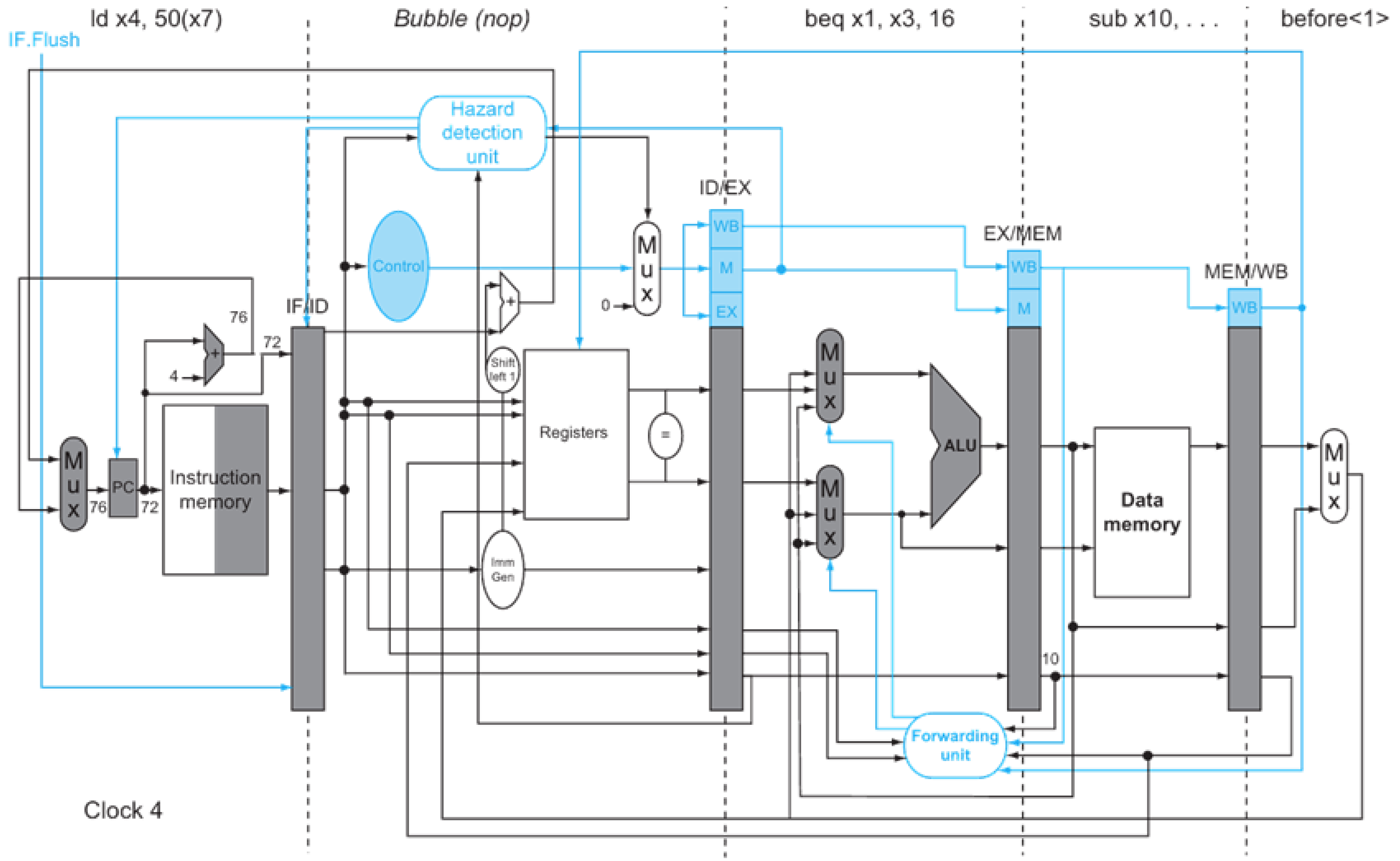
Branch taken



Example: Branch Taken



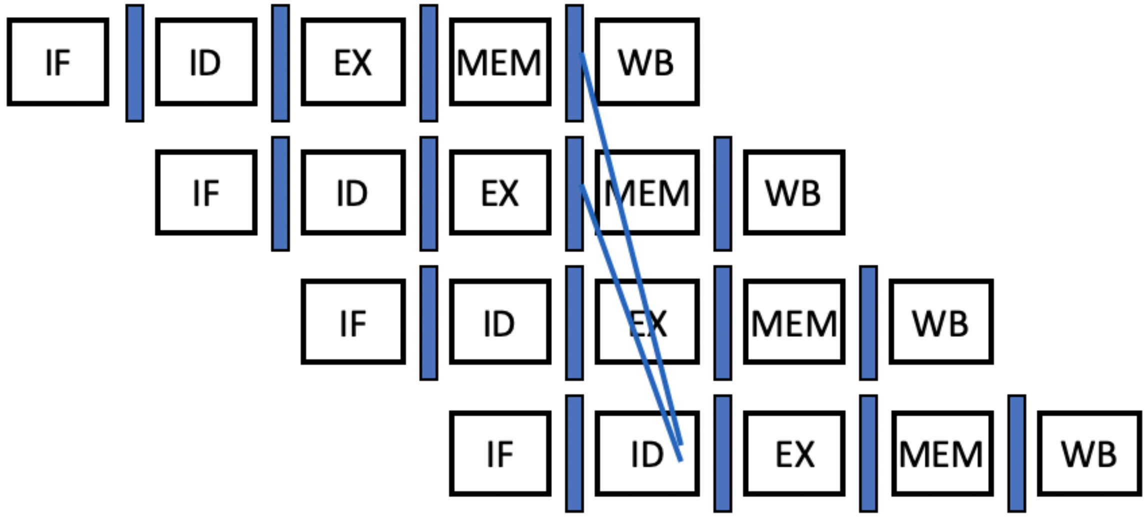
Example: Branch Taken



Data Hazards for Branches

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

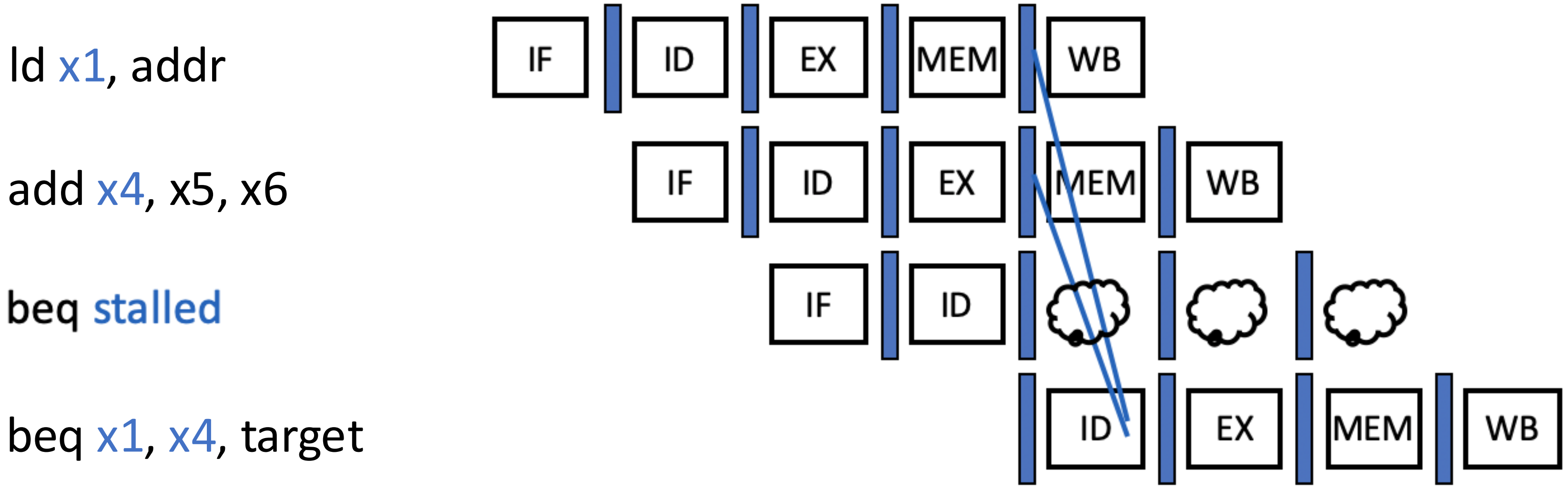
```
add x1, x2, x3

add x4, x5, x6

…

beq x1, x4, target
```



- Can resolve using forwarding

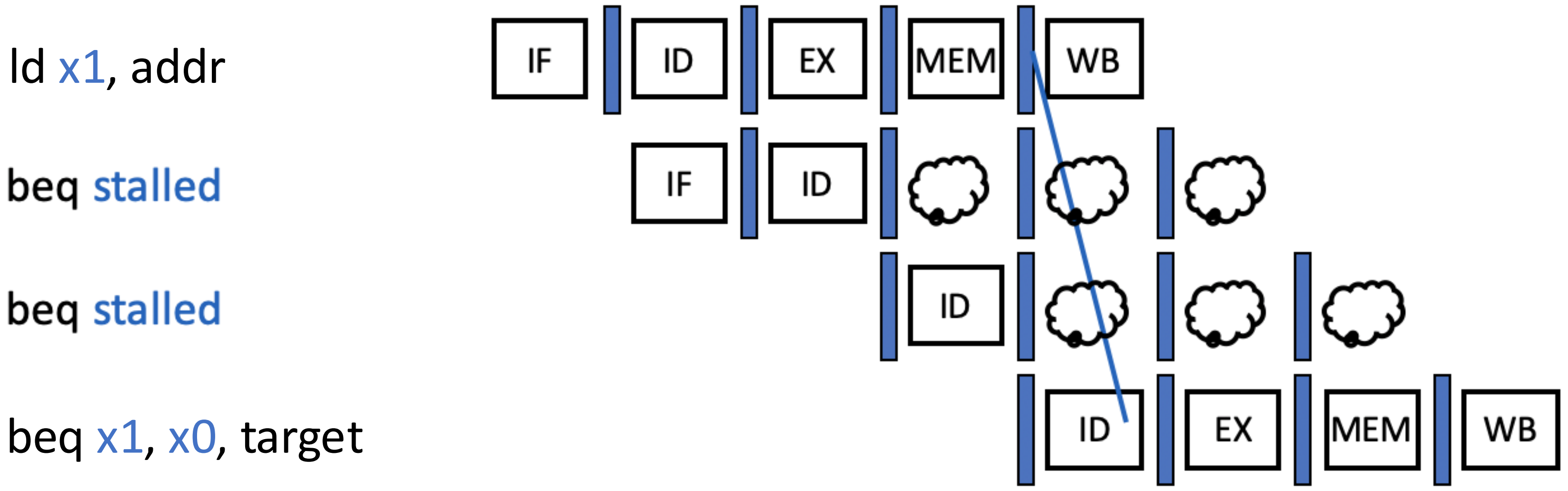
Data Hazards for Branches

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



Data Hazards for Branches

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



Question: Is delay slot a really good design?

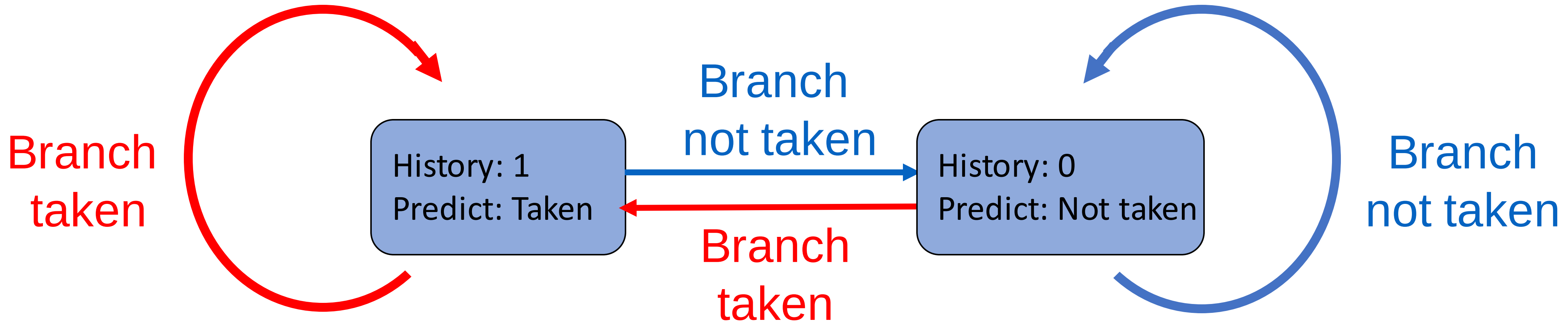
- “A ***RISC-V ISA*** is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA.
- The base integer ISAs are very similar to that of the early RISC processors except ***with no branch delay slots*** and with support for optional variable-length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

——The RISC-V Instruction Set Manual Volume I

Dynamic Branch Prediction

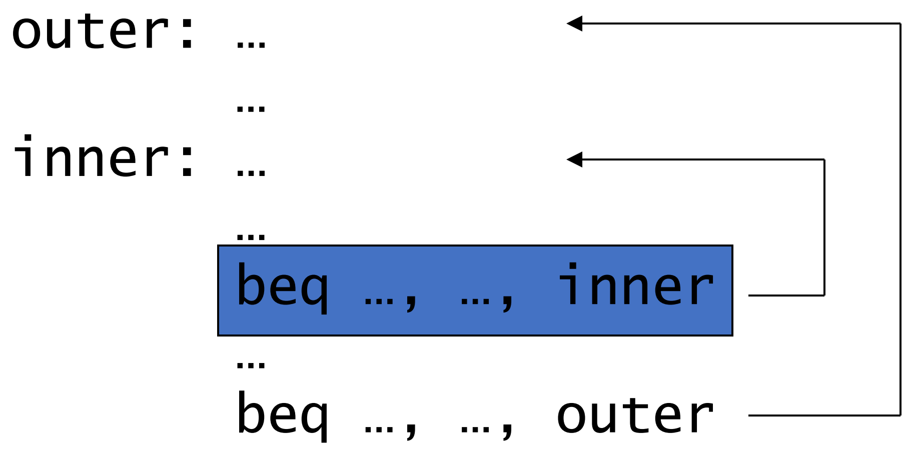
- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same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

Branch History Table(BHT)



1-Bit Predictor: Shortcoming

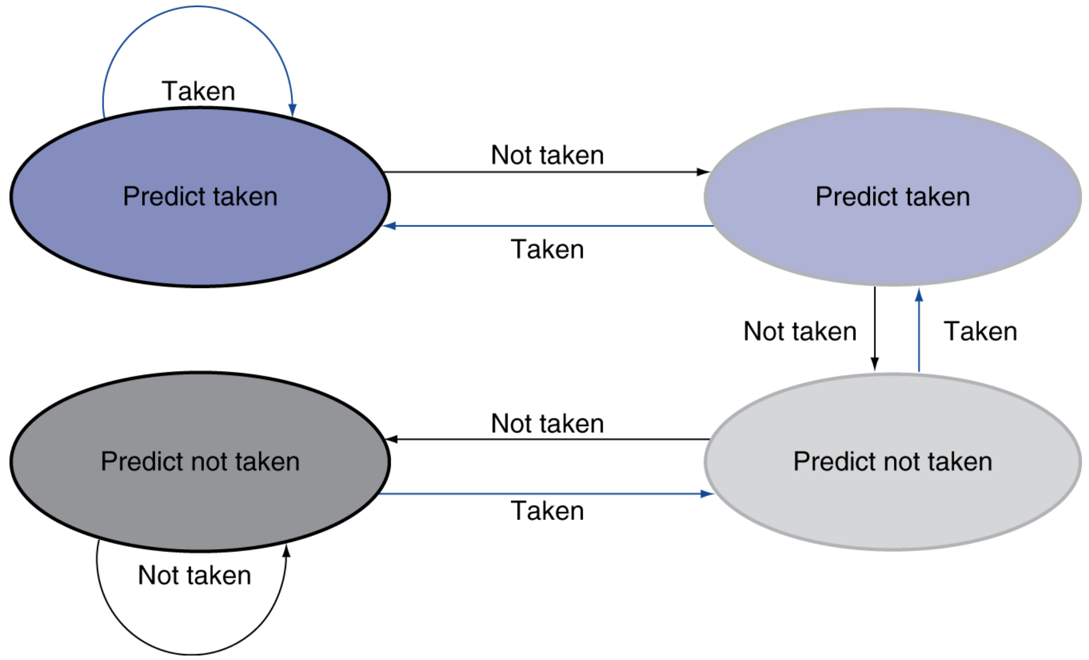
- Inner loop branches mispredicted twice!



```
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



Taken

Not taken

11 10

Taken

Taken

Not taken

Not taken

01 00

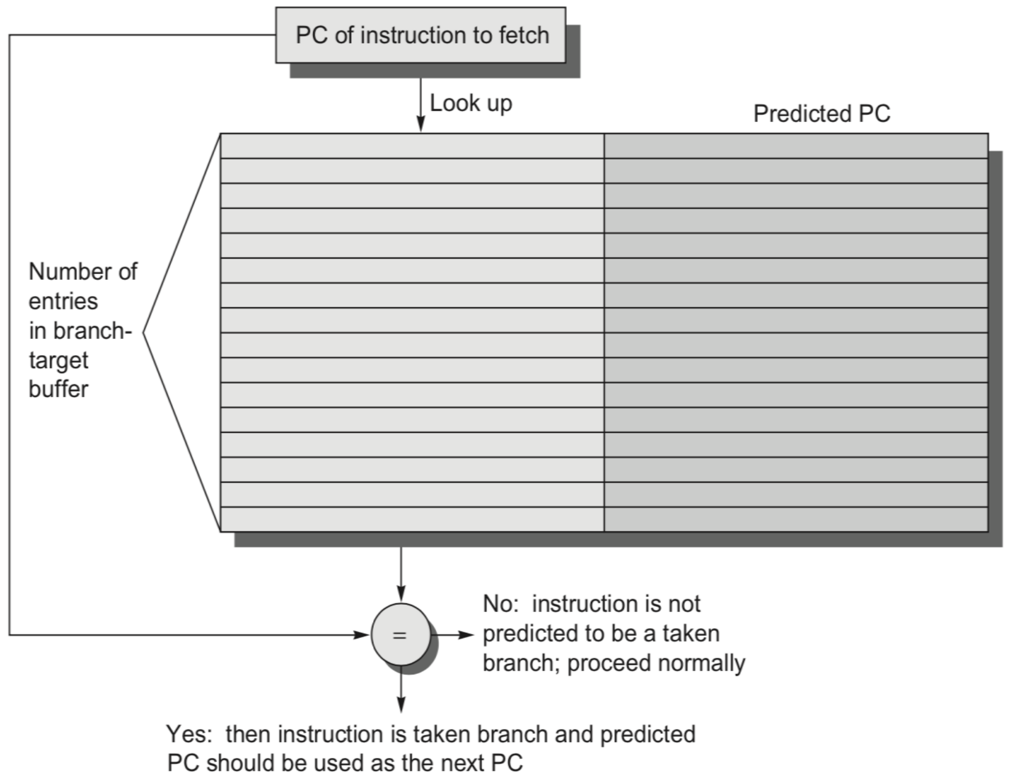
Not taken

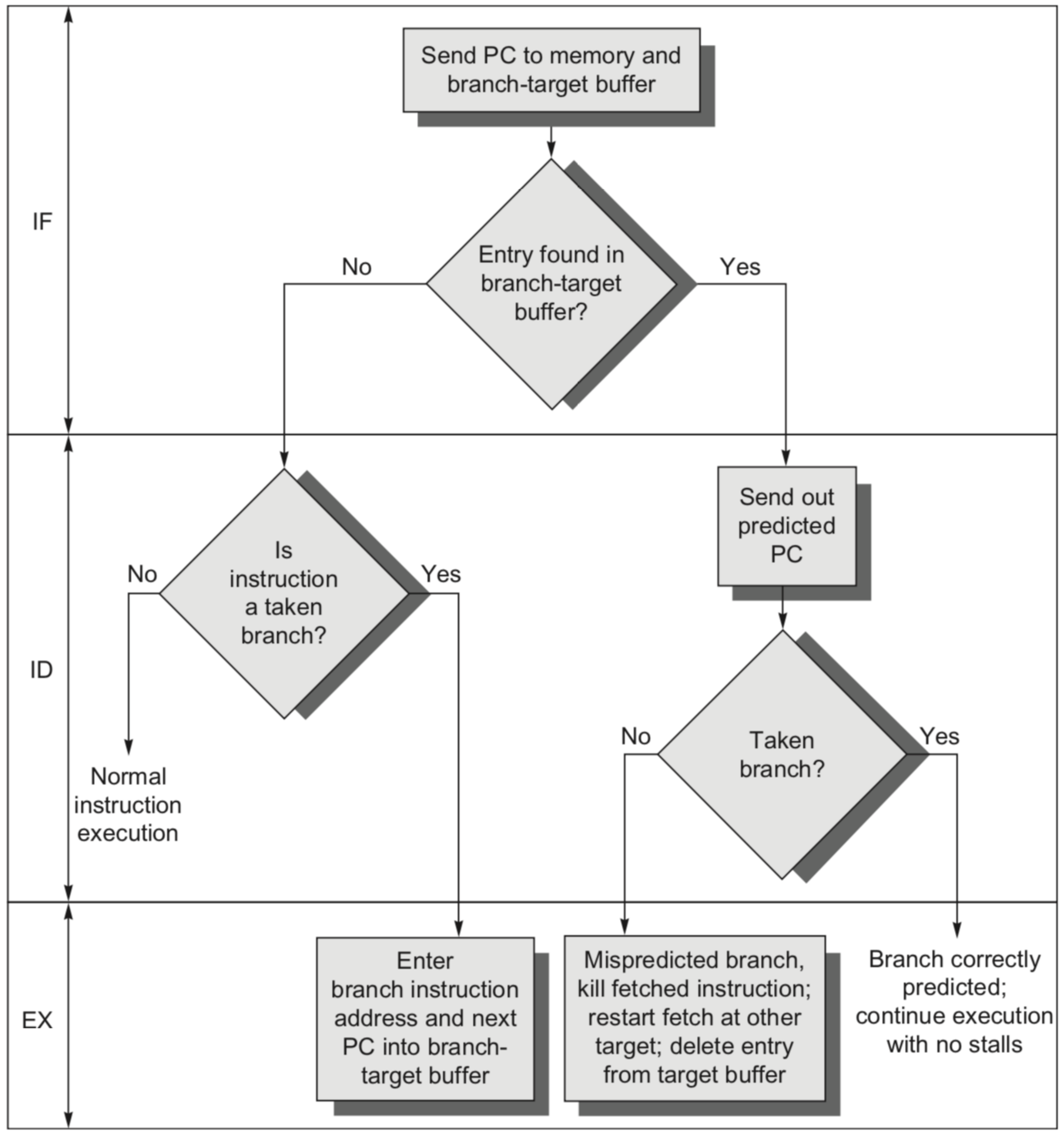
Taken

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

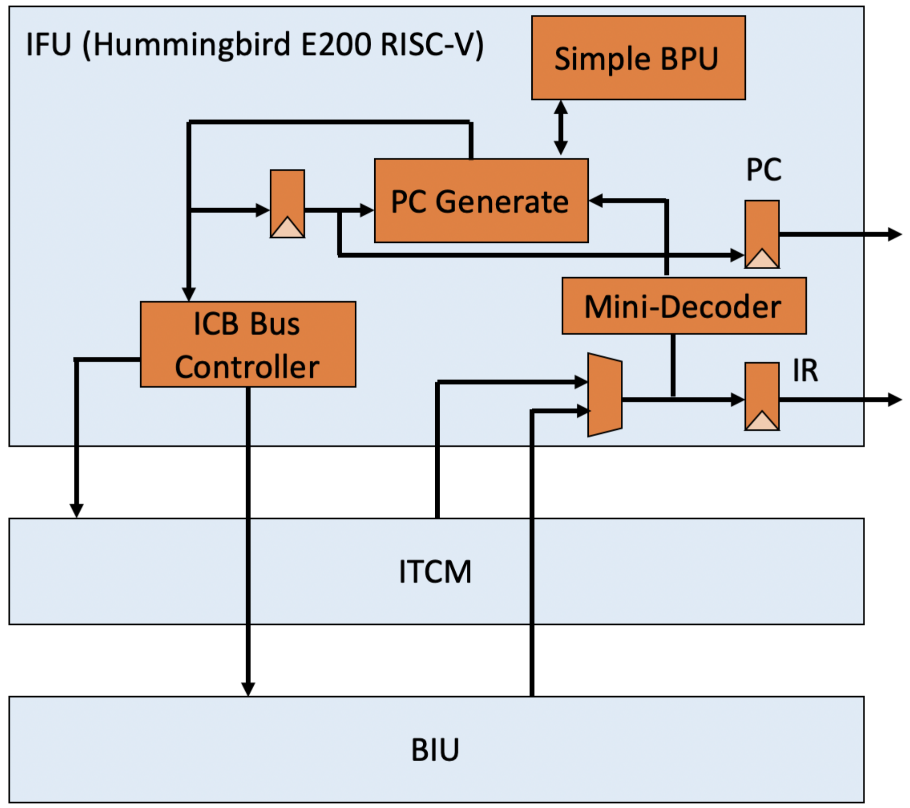
Branch-Target Buffers


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

- 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

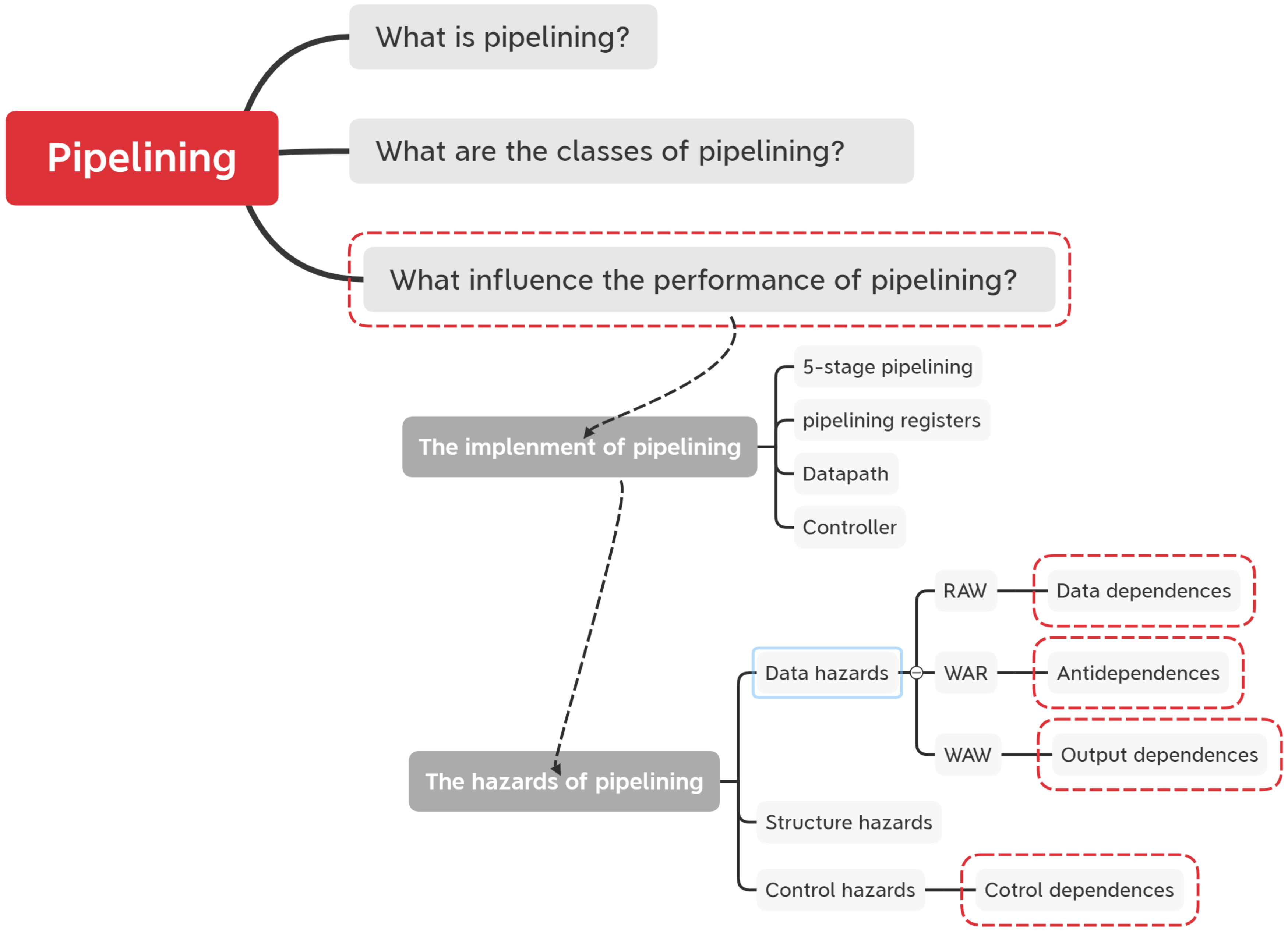
Branch-Target Buffer/Branch-Target Cache

Is instruction in BTB？	Predict	Reality	Delay cycle
Yes	Taken	Taken	0
Yes	Taken	Not taken	2
No	Not taken	Taken	2
No	Not taken	Not taken	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.


Pipelining
What is pipelining?
What are the classes of pipelining?
What influence the performance of pipelining?
The implenment of pipelining
5-stage pipelining
pipelining registers
Datapath
Controller
The hazards of pipelining
Data hazards
RAW
Data dependences
WAR
Antidependences
WAW
Output dependences
Structure hazards
Control hazards
Cotrol dependences

Linear vs. Nonlinear Pipelining

Linear pipelining: Each section of the pipelining is connected serially without feedback loop. When data passes through each segment in the pipelining, each segment can only flow once at most

What about nonlinear pipelining?

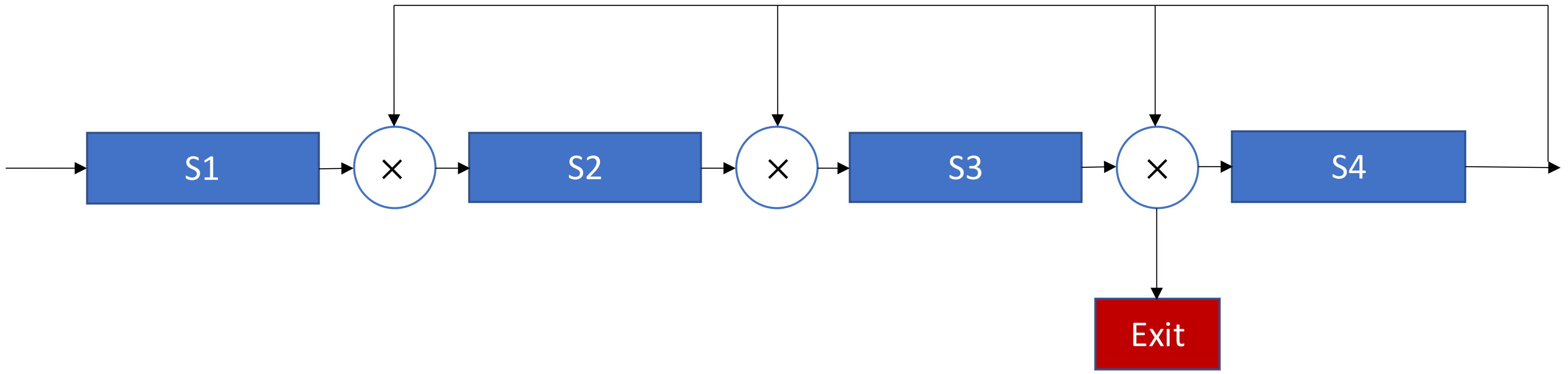
Linear vs. Nonlinear Pipelining

Nonlinear pipelining: In addition to the serial connection, there is also a feedback loop in the pipelining

Scheduling problem of nonlinear pipelining

Determine when to introduce a new task to the pipelining, so that the task will not conflict with the task previously entering the pipelining

Nonlinear pipelining

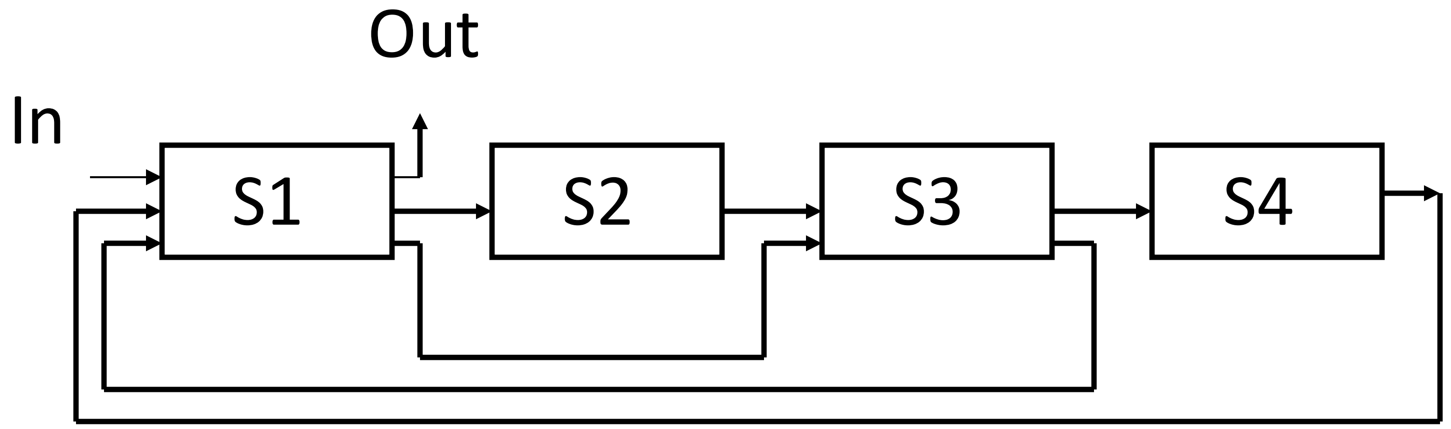


Task: → S1 → S2 → S3 → S4 → S2 → S3 → S4 → S3 →

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**		**2**	**1**	**3**	**2**	**1 4**	**3**	**2 5**	**4**	**3 6**	
S2		**1**		**2**	**1**	**3**	**2**	**4**	**3**	**5**	**4**	
S3		**1**		**2**		**1 3**		**2 4**		**3 5**		**…**
S4			**1**		**2**		**3**		**4**		**5**	

§2.9 Schedule of Nonlinear pipelining



Reservation Table:

	1	2	3	4	5	6	7
S1	√			√			√
S2		√			√		
S3		√				√	
S4			√				

Why Need to Scheduling?

Let's extend the clock cycle!

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**			**1**			**1**					
S2		**1**			**1**							
S3		**1**				**1**						**…**
S4			**1**									

Following Instructions Scheduling

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**	**2**		**1**	**2**		**1**	**2**				
S2		**1**	**2**		**1**	**2**						
S3		**1**	**2**			**1**	**2**					**…**
S4			**1**	**2**								

Following Instructions Scheduling

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	1	2	3	1	2	3	1	2	3			
S2		1	2	3	1	2	3					
S3		1	2	3		1	2	3				…
S4			1	2	3							

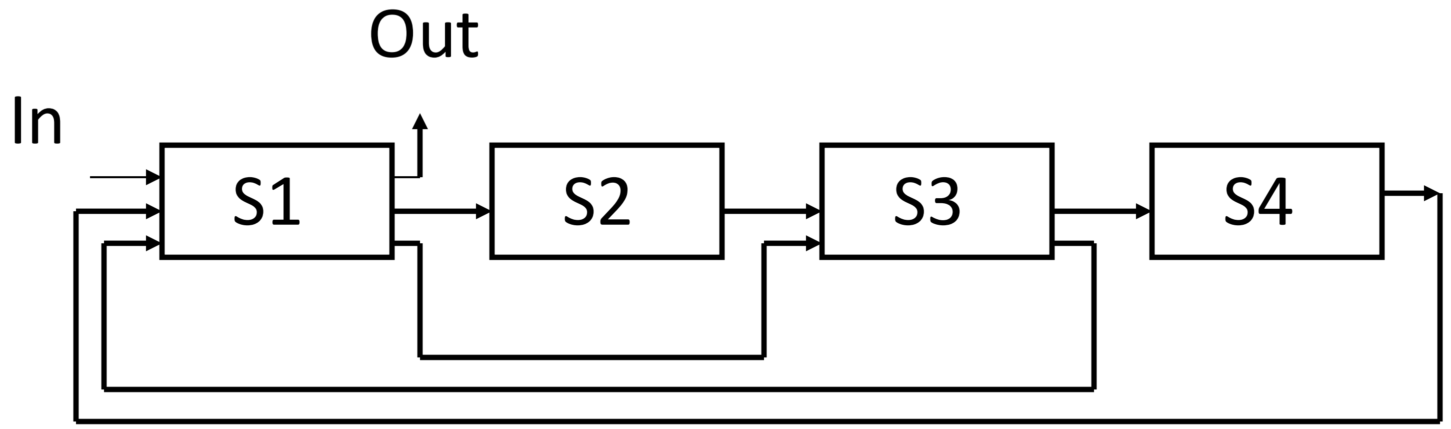
Following Instructions Scheduling

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**	**2**	**3**	**14**	**25**	**3**	**14**	**2 5**	**3**	**4**	**5**	
S2		**1**	**2**	**3**	**14**	**25**	**3**	**4**	**5**			
S3		**1**	**2**	**3**	**4**	**15**	**2**	**3**	**4**	**5**		**…**
S4			**1**	**2**	**3**	**4**	**5**					

Hazard! Any other scheduling?

§2.9 Schedule of Nonlinear pipelining



Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**		**2**	**1**		**2**	**1**		**2**			
S2		**1**		**2**	**1**		**2**					
S3		**1**		**2**		**1**		**2**				**…**
S4			**1**		**2**							

Another Scheduling

Reservation Table:

	1	2	3	4	5	6	7	8	9	10	11	…
S1	**1**		**2**	**1**	**3**	**2**	**1 4**	**3**	**2 5**	**4**	**3 6**	
S2		**1**		**2**	**1**	**3**	**2**	**4**	**3**	**5**	**4**	
S3		**1**		**2**		**1 3**		**2 4**		**3 5**		**…**
S4			**1**		**2**		**3**		**4**		**5**	

Still hazard! Need scheduling

How to Schedule Non-linear Pipeline?

Schedule of Nonlinear pipelining without hazards

Initial conflict vector $\rightarrow$ Conflict vector $\rightarrow$ State transition graph

$\rightarrow$ Circular queue $\rightarrow$ Shortest average interval

Initial conflict vector

	n									
k		1	2	3	4	5	6	7	8	9
	1	√								√
	2		√	√					√	
	3				√					
	4					√	√			
	5							√	√	

Initial conflict vector

$$\begin{tikzcd}
\text{Initial conflict vector} \arrow[d] \\
\text{Conflict vector}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Prohibit sets } F=\{1,5,6,8\} \arrow[d] \\
\text{Initial conflict vector } C_0=(10110001) \arrow[d] \\
\text{Conflict vector } (C_{N-1}C_{N-2}\ldots C_i\ldots C_2C_1)
\end{tikzcd}$$

Conflict vector

- Initial conflict vector C0=(10110001) CCV=Current Conflict vector

Interval	Initial	
CCV	10110001	
1→	10110001	

Conflict vector

- Initial conflict vector C0=(10110001) CCV=Current Conflict vector

Interval	Initial	2
CCV	101100$\underline{0}$1	10111101
1→	10110001	00101100
2→		10110001

Conflict vector

- Initial conflict vector C0=(10110001) CCV=Current Conflict vector

Interval	Initial	2	2
CCV	101100$\underline{0}$1	101111$\underline{0}$1	10111111
1→	10110001	00101100	00001011
2→		10110001	00101100
3→			10110001

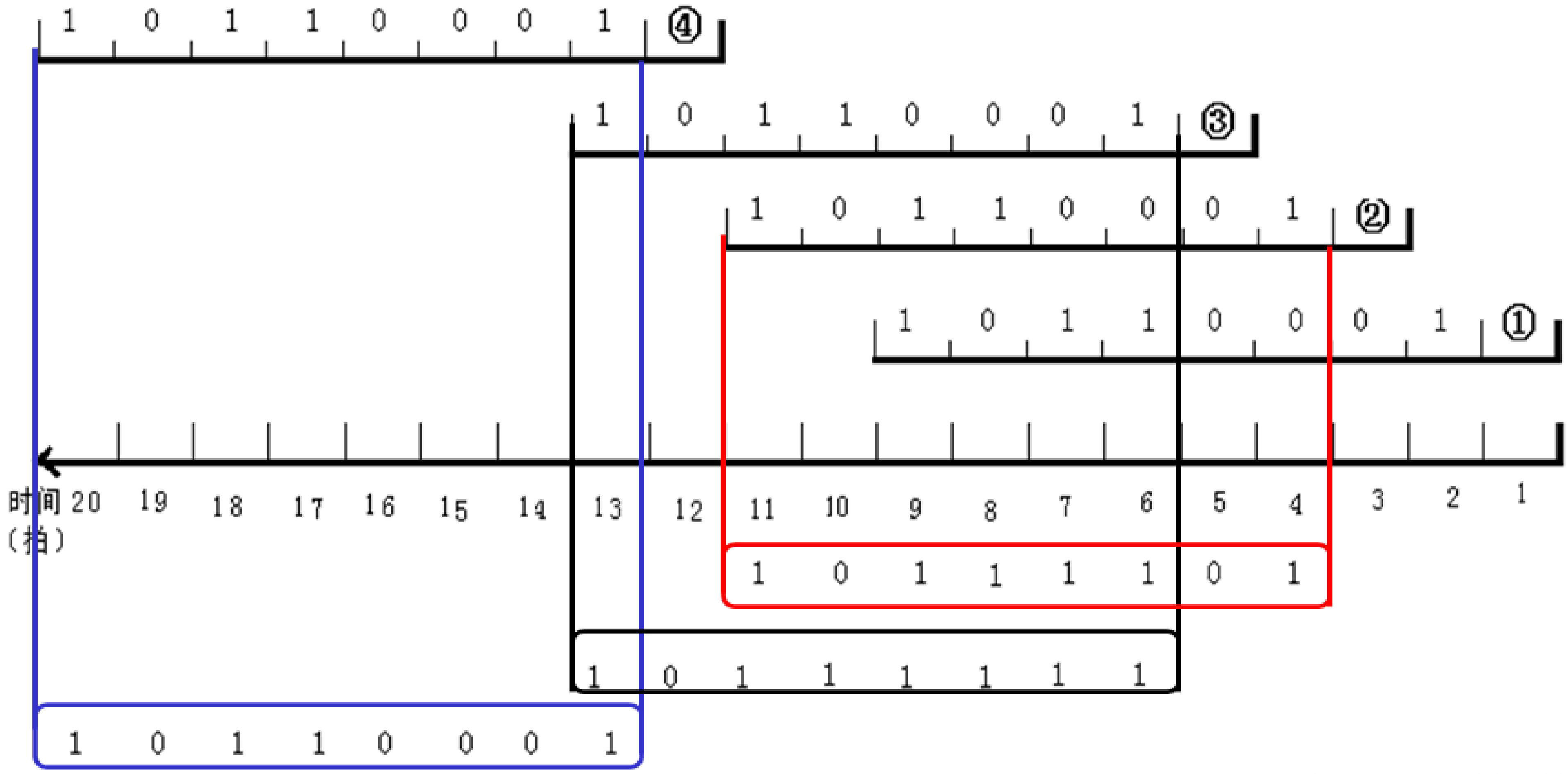
Conflict vector

- Initial conflict vector C0=(10110001) CCV=Current Conflict vector

Interval	Initial	2	2	7
CCV	101100$\underline{0}$1	101111$\underline{0}$1	1$\underline{0}$111111	10110001
1→	10110001	00101100	00001011	00000000
2→		10110001	00101100	00000000
3→			10110001	00000001
4→				10110001

Conflict vector

Any other scheduling?

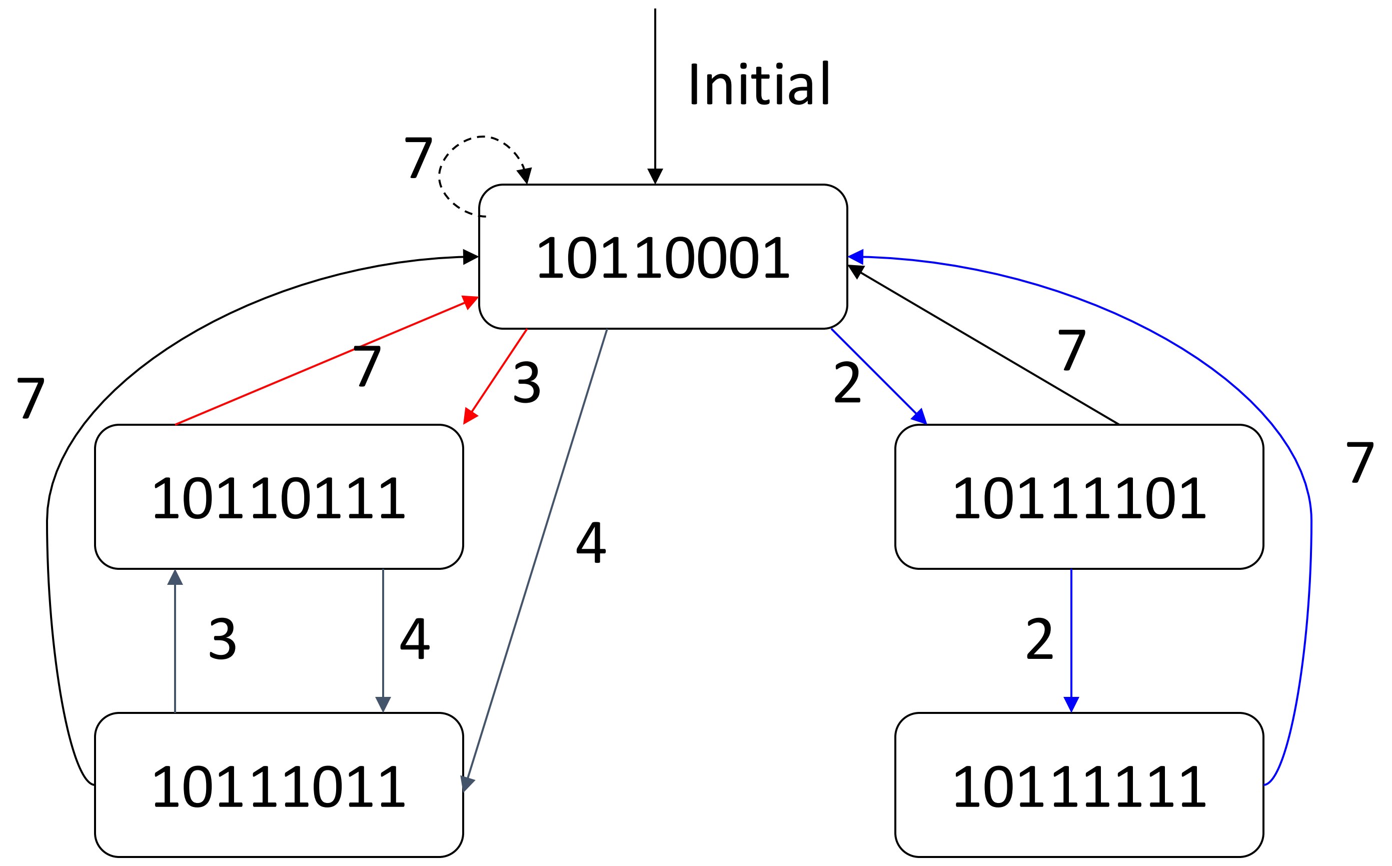


Conflict vector

- Initial conflict vector C0=(10110001) CCV=Current Conflict vector

Interval	Initial	2	7
CCV	101100$\underline{0}$1	1$\underline{0}$111101	10110001
1→	10110001	00101100	00000000
2→		10110001	00000001
3→			10110001

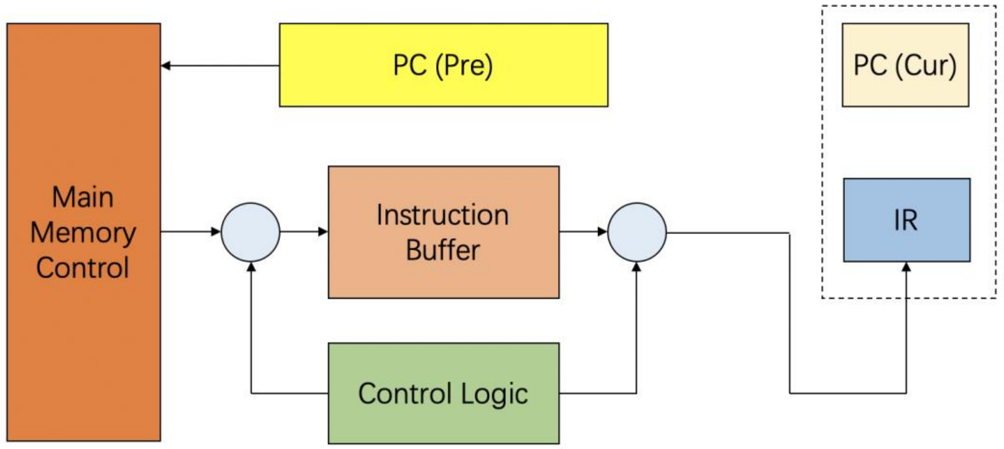
State transition graph

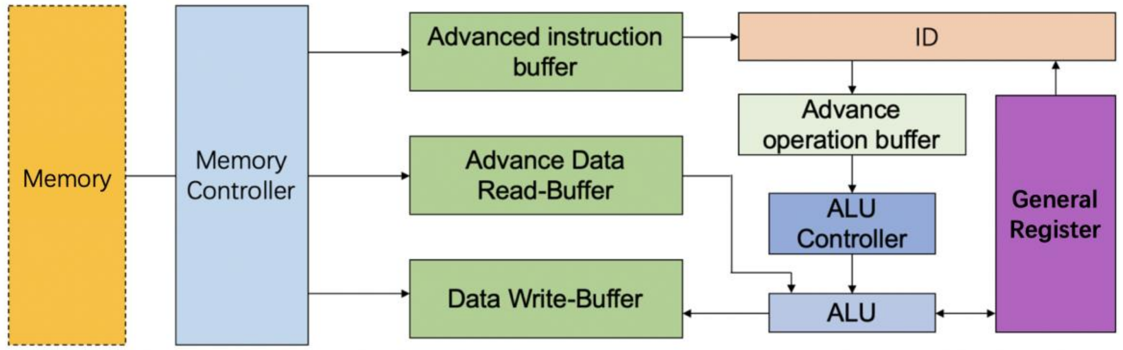


Circular queue	Shortest average interval
2,2,7	3.67
2,7	4.5
3,4	3.5
4,3	3.5
3,4,7	4.67
3,7	5
4,3,7	4.67
4,7	5.5
7	7

Summary

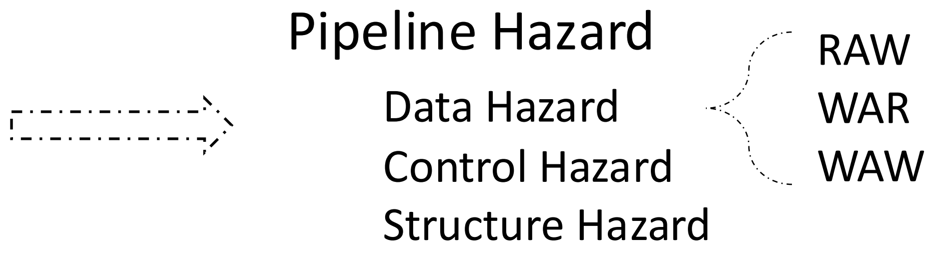
- 1. How the instruction is executed
  - Sequential execution
  - Overlap once
  - Second overlap
  - Pipeline
- 2. Classification of pipelines
  - Single function, multi-function
  - Static, dynamic
  - Linear, non-linear
  - In-order, out-of-order
- 3. Performance indicators of the pipeline
  - Throughput rate
  - Speedup ratio effectiveness





Summary

- 4. Factors affecting the performance of the pipeline
  - Pipeline design
  - Type of instructions
  - Instructions related
    - Data dependence
    - Name dependence
    - Control dependence



- 5. Dynamic Branch Prediction
  - Branch History Table (BHT)
  - Branch-Target Buffer (BTB)
- 6. Non-linear pipeline scheduling problem