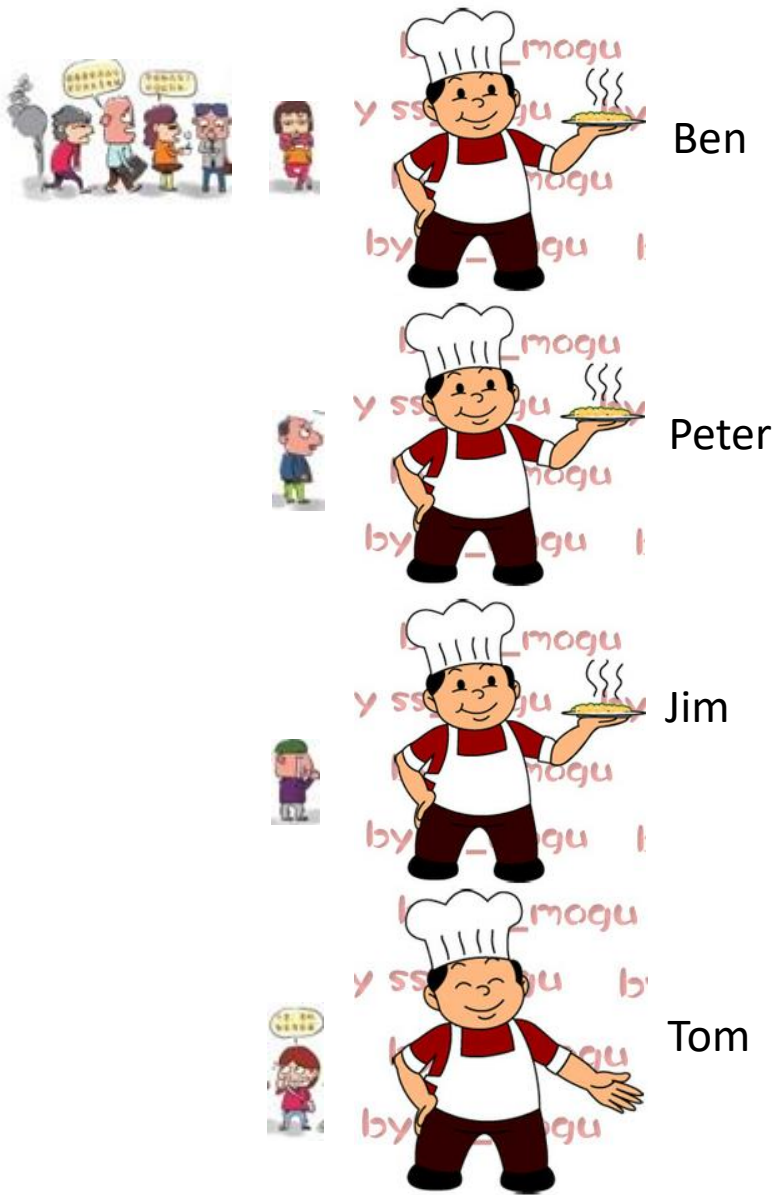


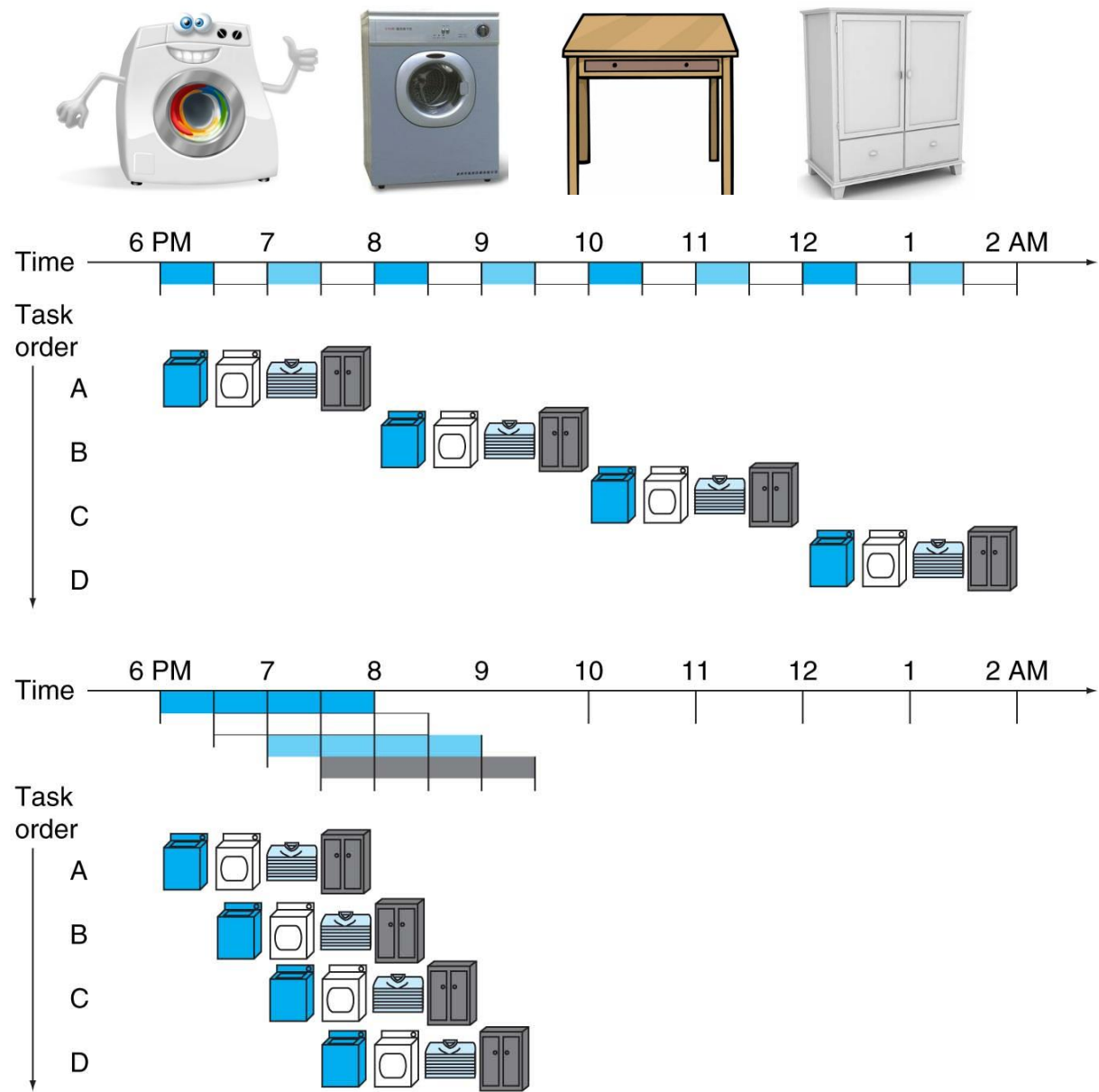
Chapter 2

Pipelining





§2.5 Hazards of Pipelining



Data Dependences

- FLD F0, 0(R1)
- 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

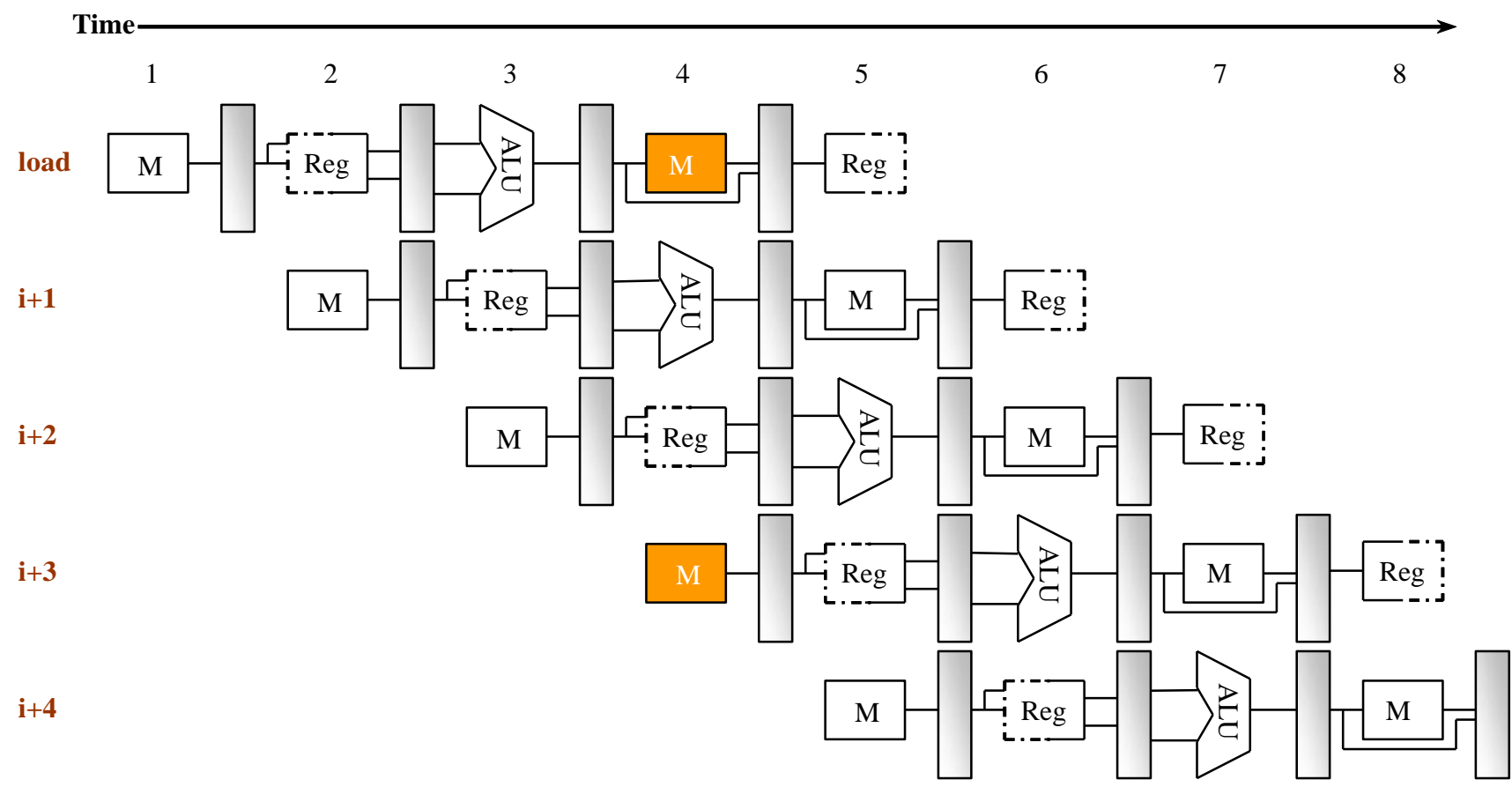


Structure Hazards

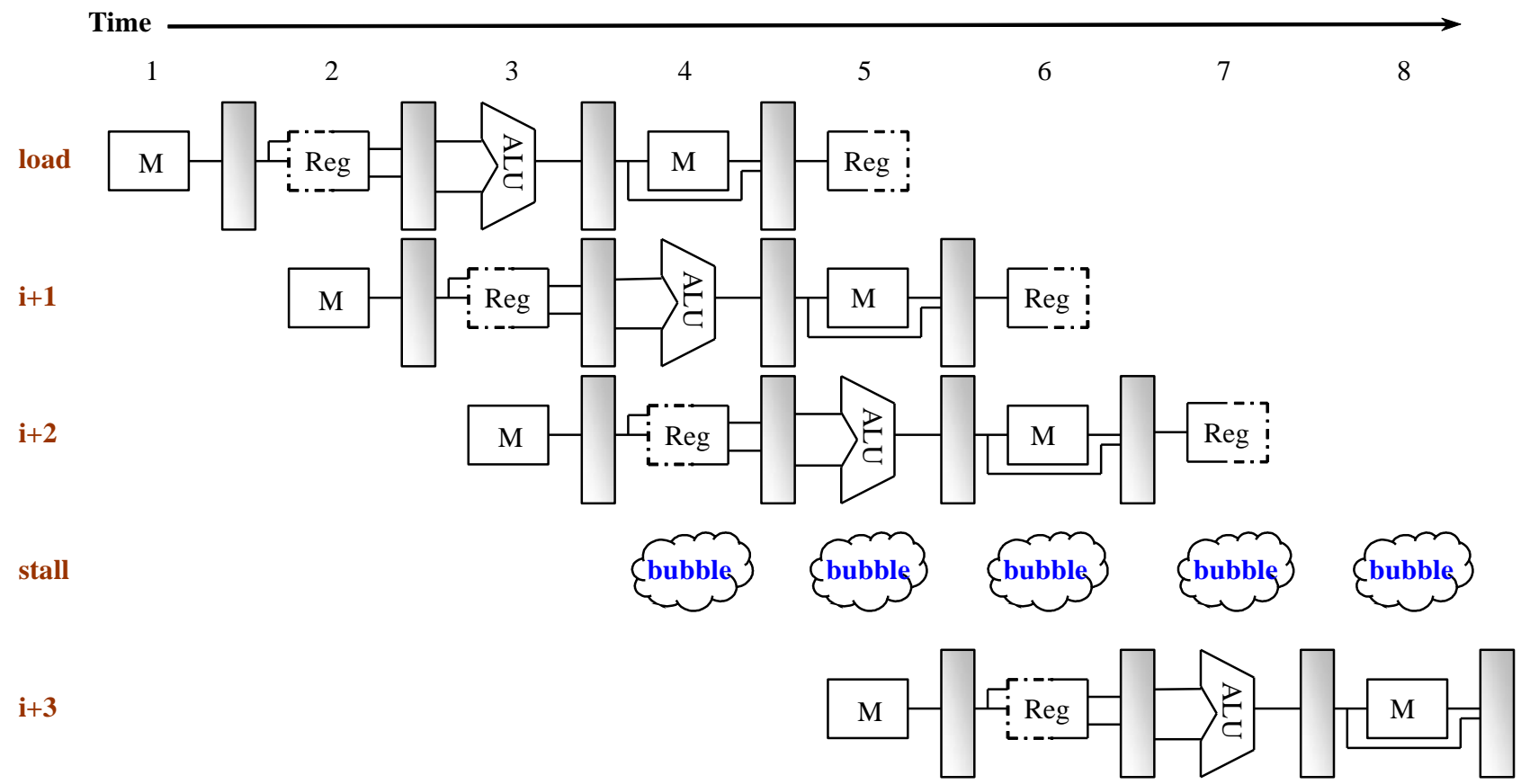
- Conflict for use of a resource
- In a pipeline with a single memory
 - Load/store requires data access
 - Instruction fetch would have to *stall* for that cycle
 - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
 - Or separate instruction/data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



Structure Hazards



Structure Hazards



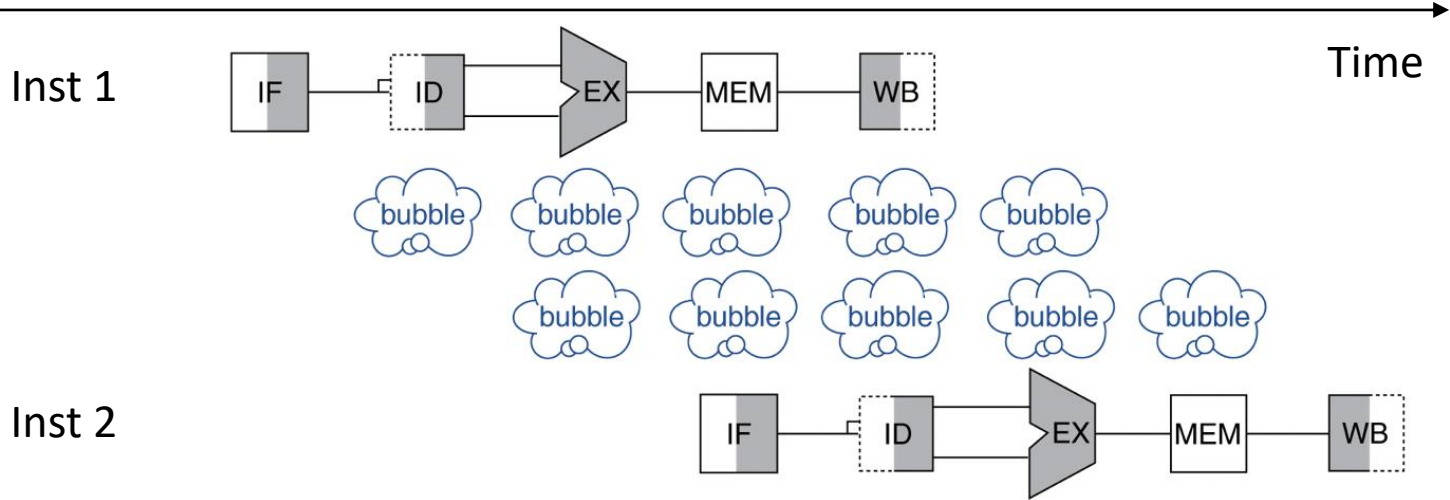
Structure Hazards

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

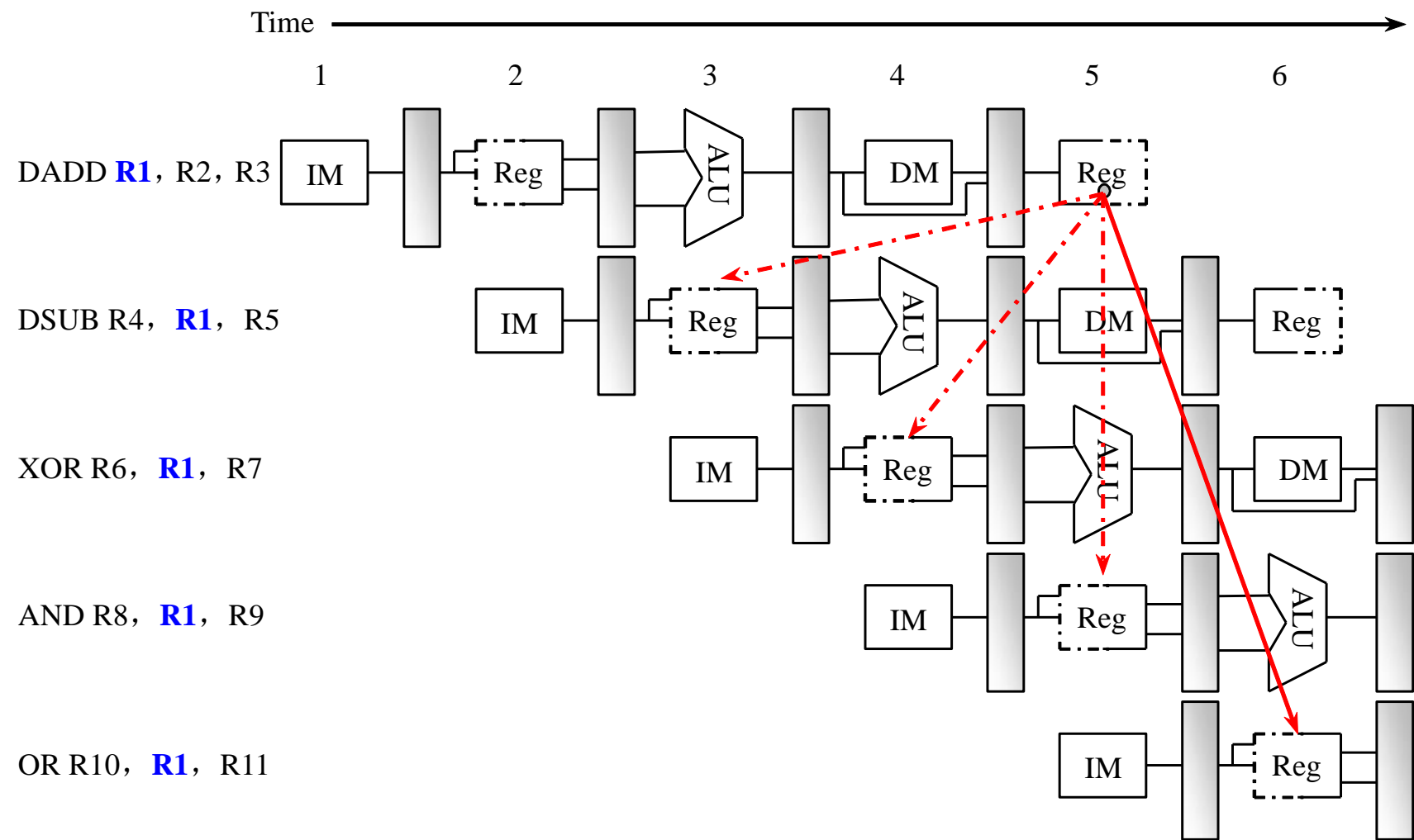


Data Hazards

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



Data Hazards



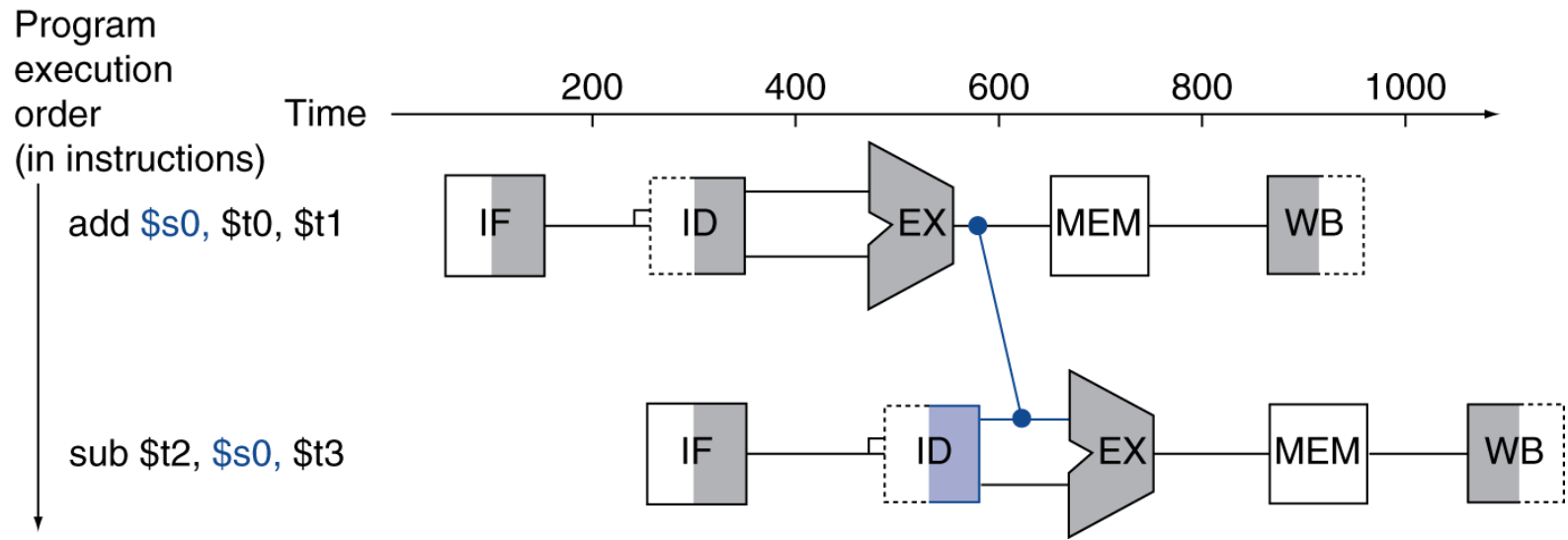
Data Hazards

• 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



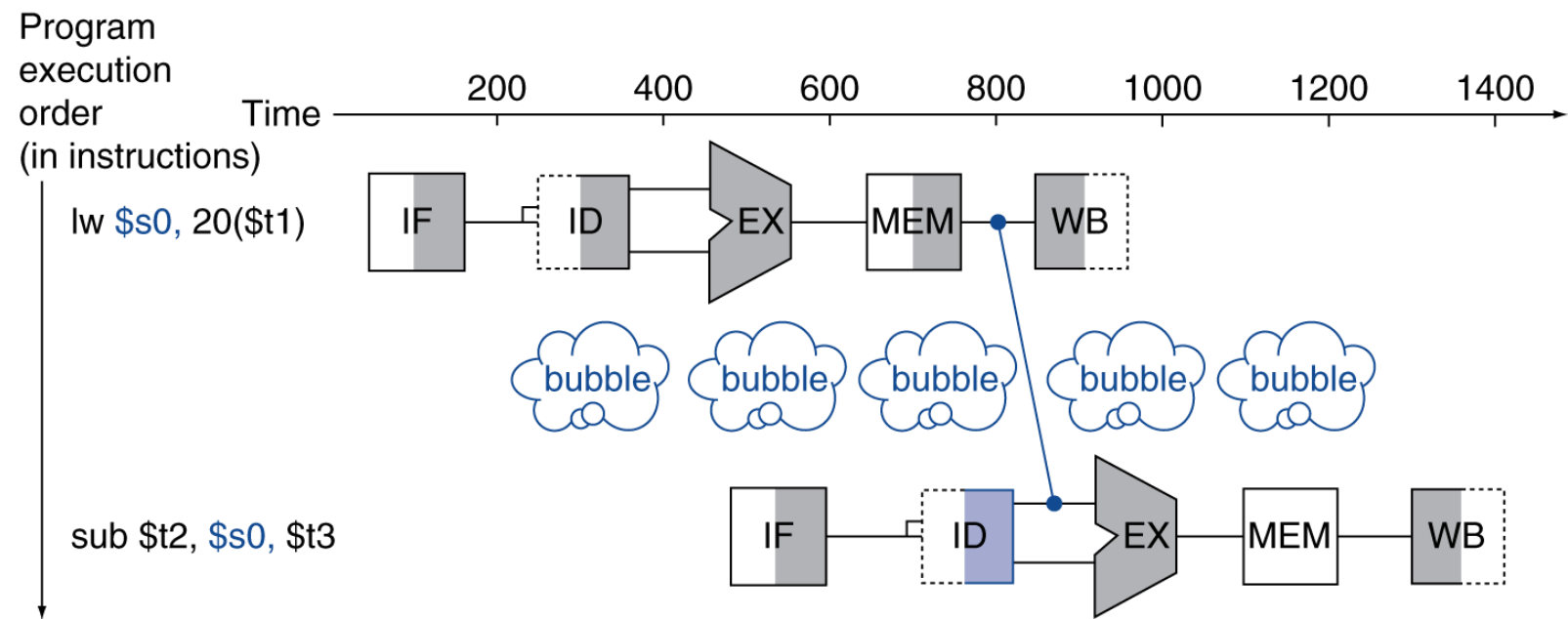
Forwarding (aka Bypassing)

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

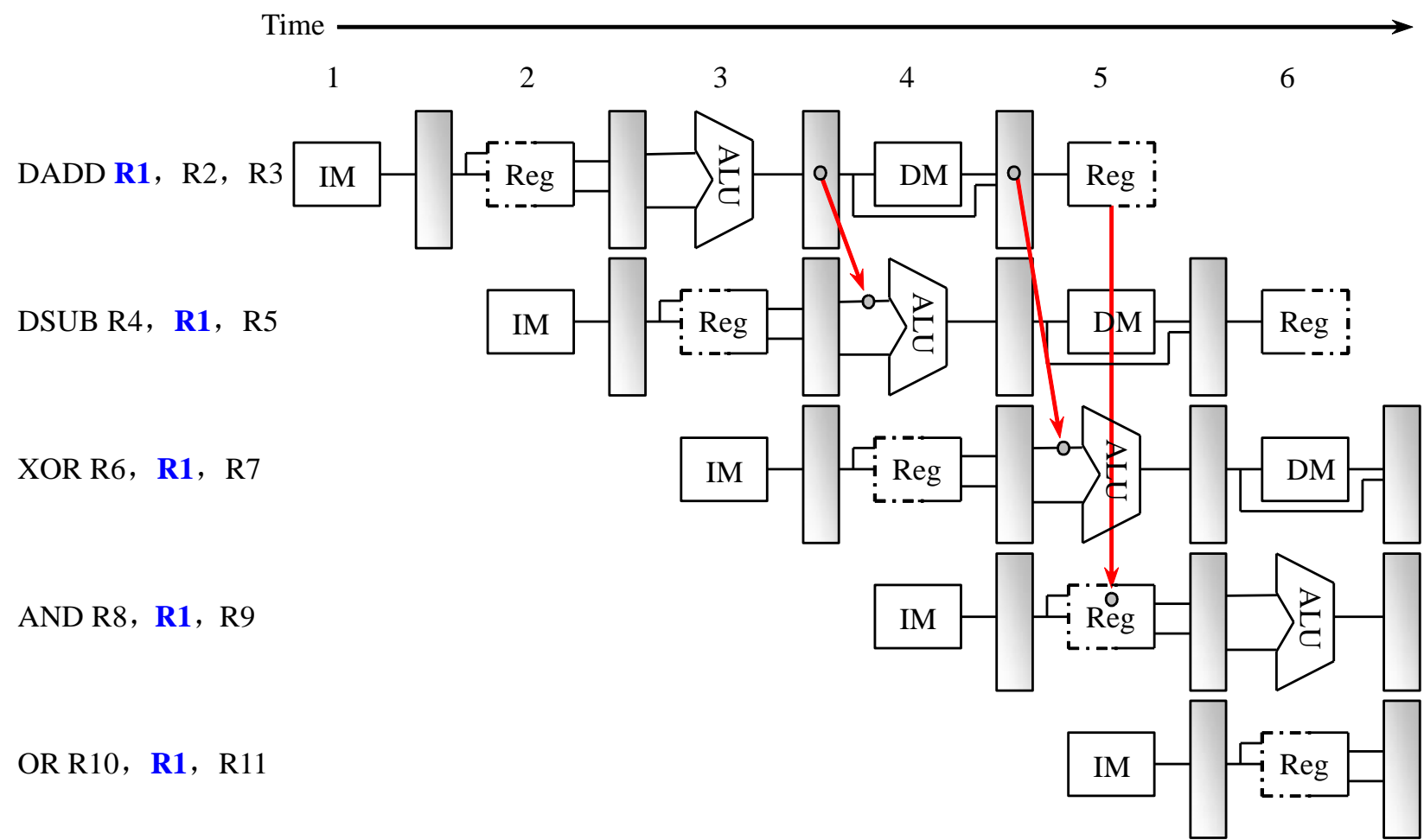


Load-Use Data Hazard

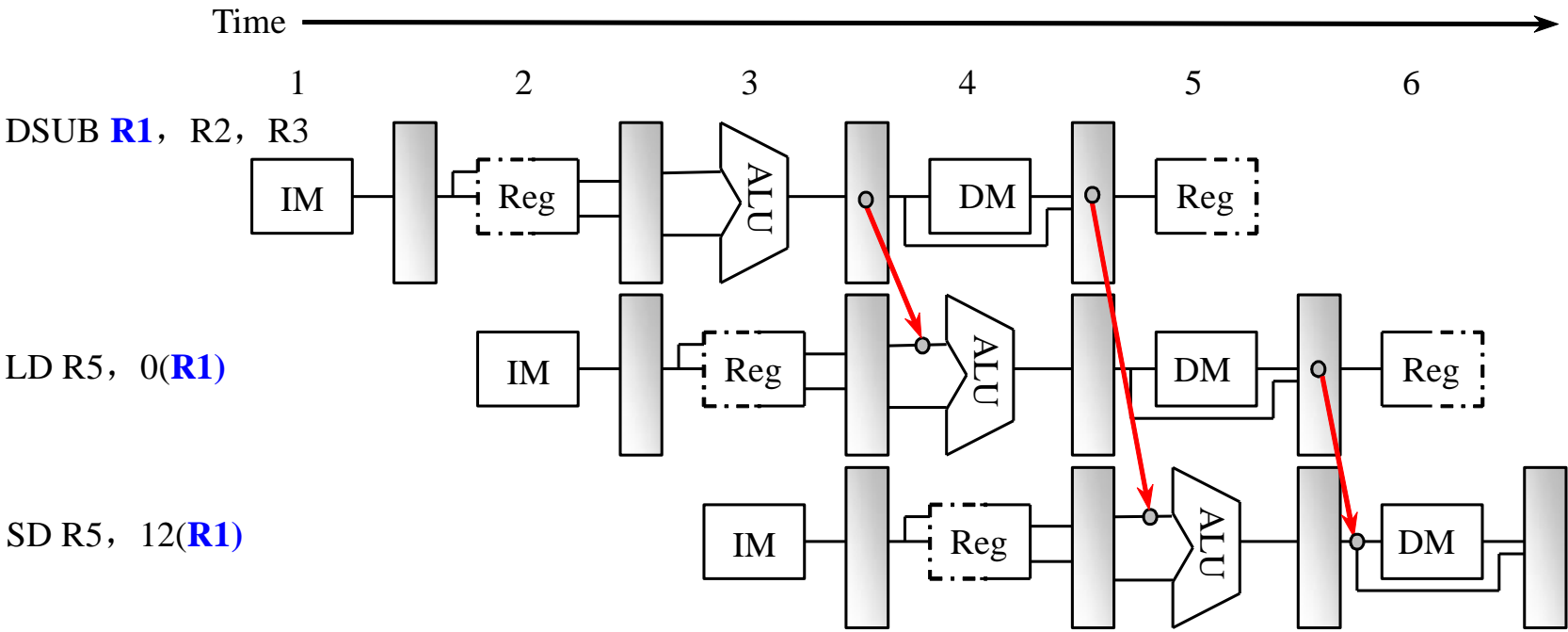
- Can't always avoid stalls by forwarding
 - If value not computed when needed
 - Can't forward backward in time!



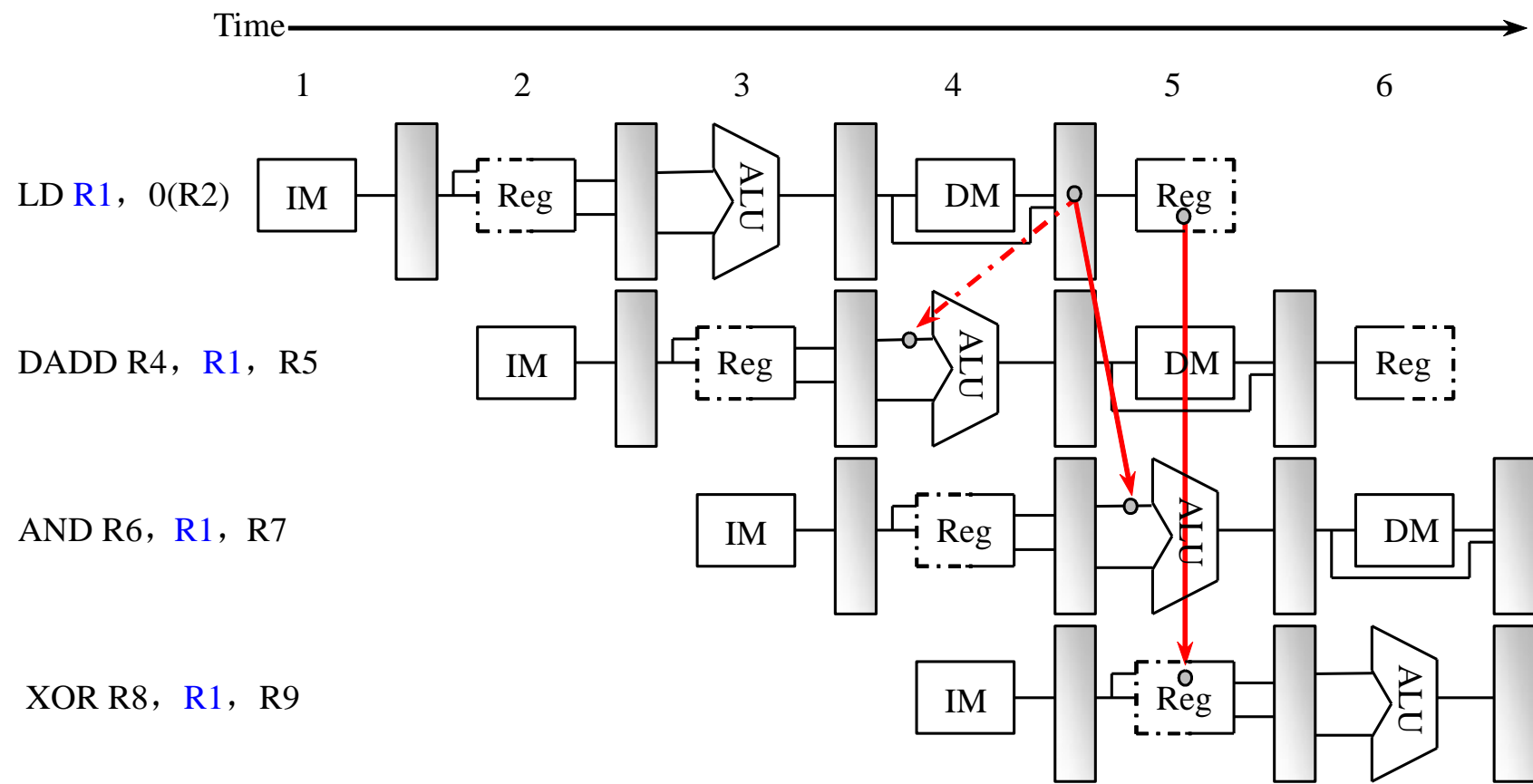
Forwarding (aka Bypassing)



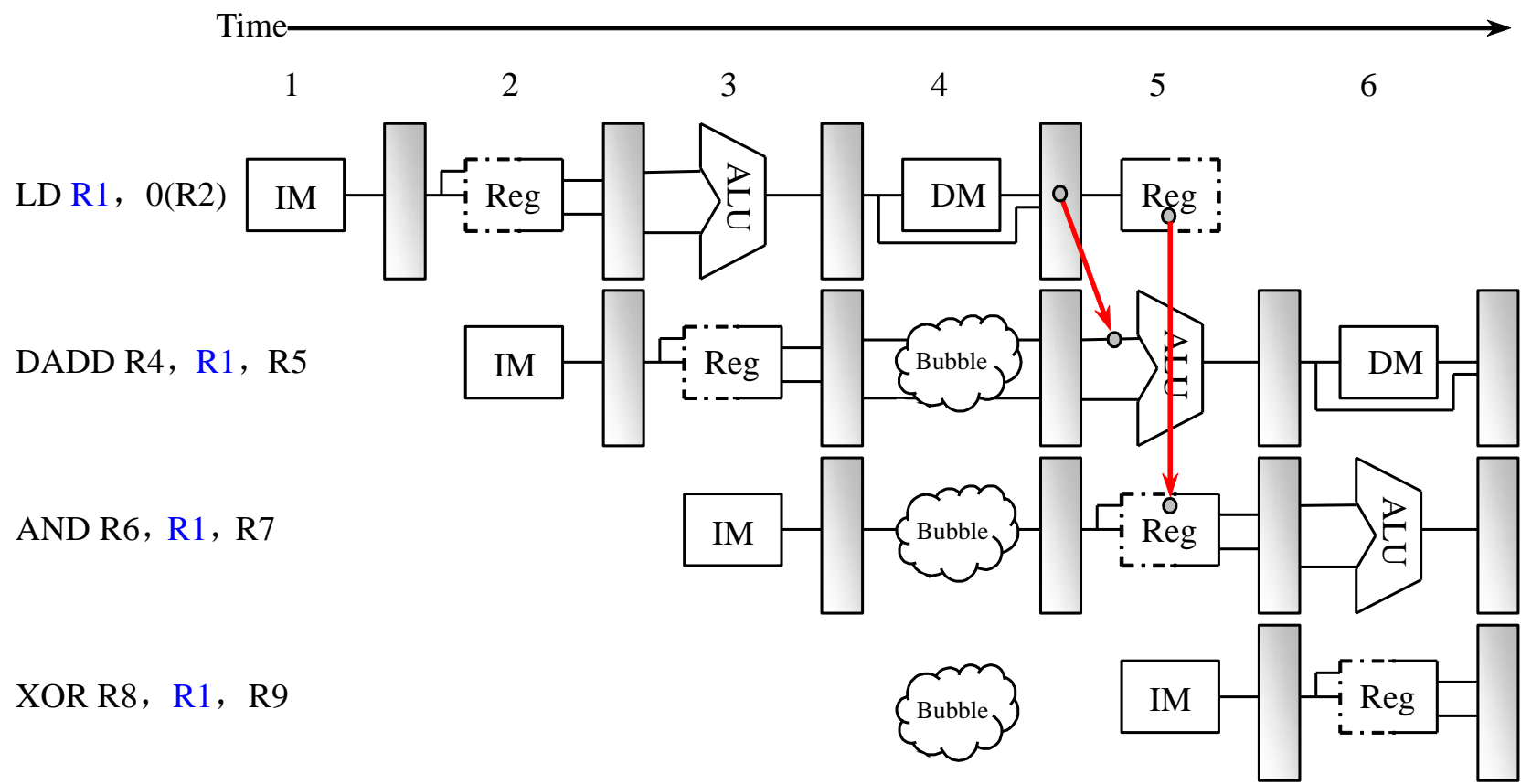
Forwarding (aka Bypassing)



Forwarding (aka Bypassing)



Forwarding with bubble



Forwarding with bubble

LD R1, 0 (R2)	IF	ID	EX	MEM	WB			
DADD R4, R1, R5		IF	ID	EX	MEM	WB		
AND R6, R1, R7			IF	ID	EX	MEM	WB	
XOR R8, R1, R9				IF	ID	EX	MEM	WB

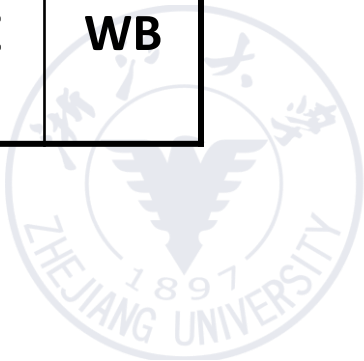
LD R1, 0 (R2)	IF	ID	EX	MEM	WB			
DADD R4, R1, R5		IF	ID	stall	EX	MEM	WB	
AND R6, R1, R7			IF	stall	ID	EX	MEM	WB
XOR R8, R1, R9				stall	IF	ID	EX	MEM



Code Scheduling to Avoid Stalls

A = B + C

LD Rb, B	IF	ID	EX	MEM	WB				
LD Rc, C		IF	ID	EX	EX	ME M	WB	WB	
DADD Ra, Rb, Rc			IF	ID	stall	EX	ME M	WB	
SD Ra , A				IF	stall	ID	EX	ME M	WB



Code Scheduling to Avoid Stalls

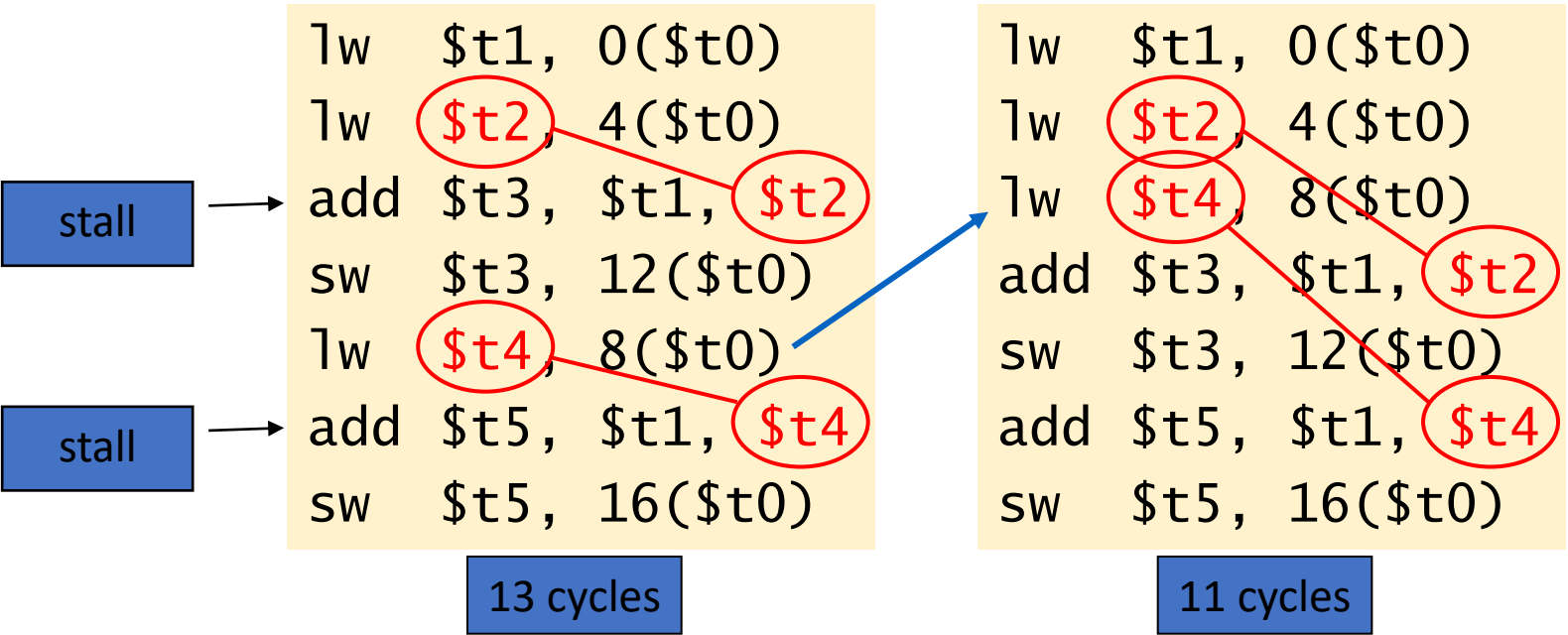
A = B + C
D = E - F

Before Scheduling	After Scheduling
LD Rb, B	
LD Rc, C	
<u>DADD Ra, Rb, Rc</u>	
SD Ra, A	
LD Re, E	
LD Rf, F	
<u>DSUB Rd, Re, Rf</u>	
SD Rd, D	



Code Scheduling to Avoid Stalls

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



More-Realistic Branch Prediction

- Static branch prediction
 - Based on typical branch behavior
 - Example: loop and if-statement branches
 - Predict backward branches taken
 - Predict forward branches not taken
- Dynamic branch prediction
 - Hardware measures actual branch behavior
 - e.g., record recent history of each branch
 - Assume future behavior will continue the trend
 - When wrong, stall while re-fetching, and update history



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

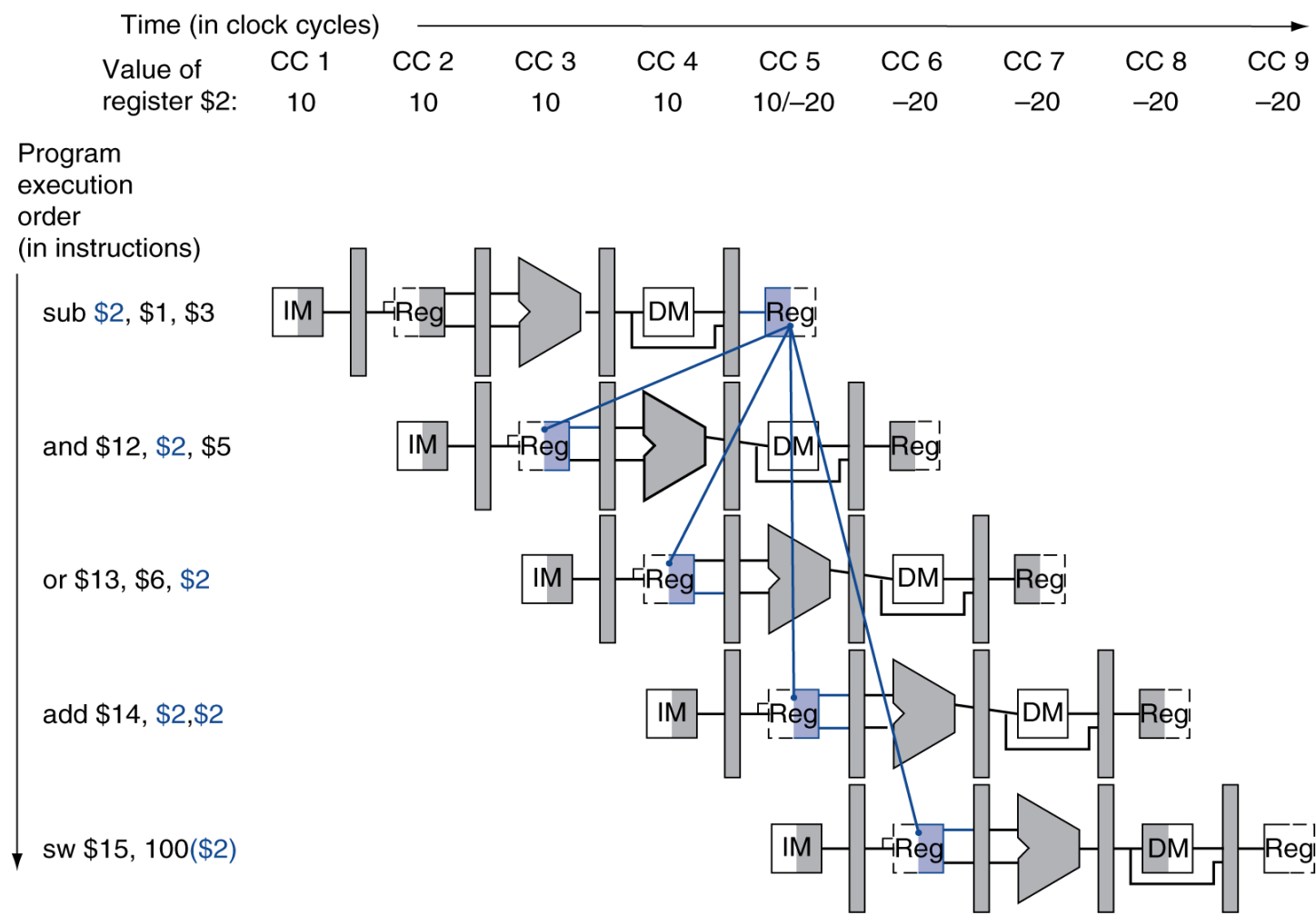
- Consider this sequence:

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

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

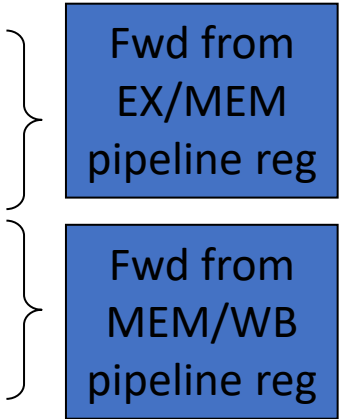


Dependencies & Forwarding



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
 - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
 - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

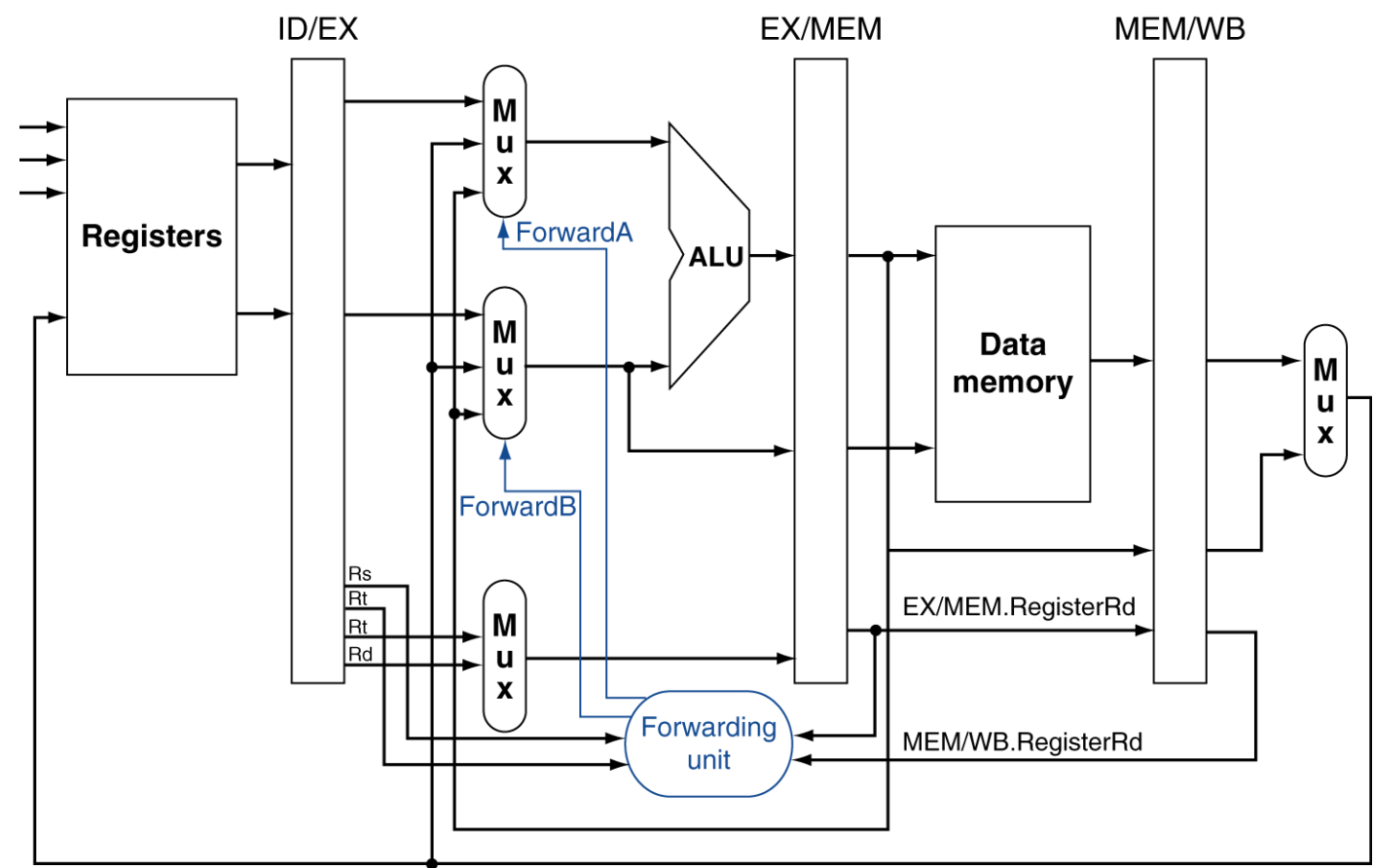


Detecting the Need to Forward

- 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 \neq 0,
MEM/WB.RegisterRd \neq 0



Forwarding Paths



b. With forwarding



Forwarding Conditions

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



Double Data Hazard

- Consider the sequence:
 - add \$1,\$1,\$2
 - add \$1,\$1,\$3
 - add \$1,\$1,\$4
- Both hazards occur
 - Want to use the most recent
- Revise MEM hazard condition
 - Only fwd if EX hazard condition isn't true

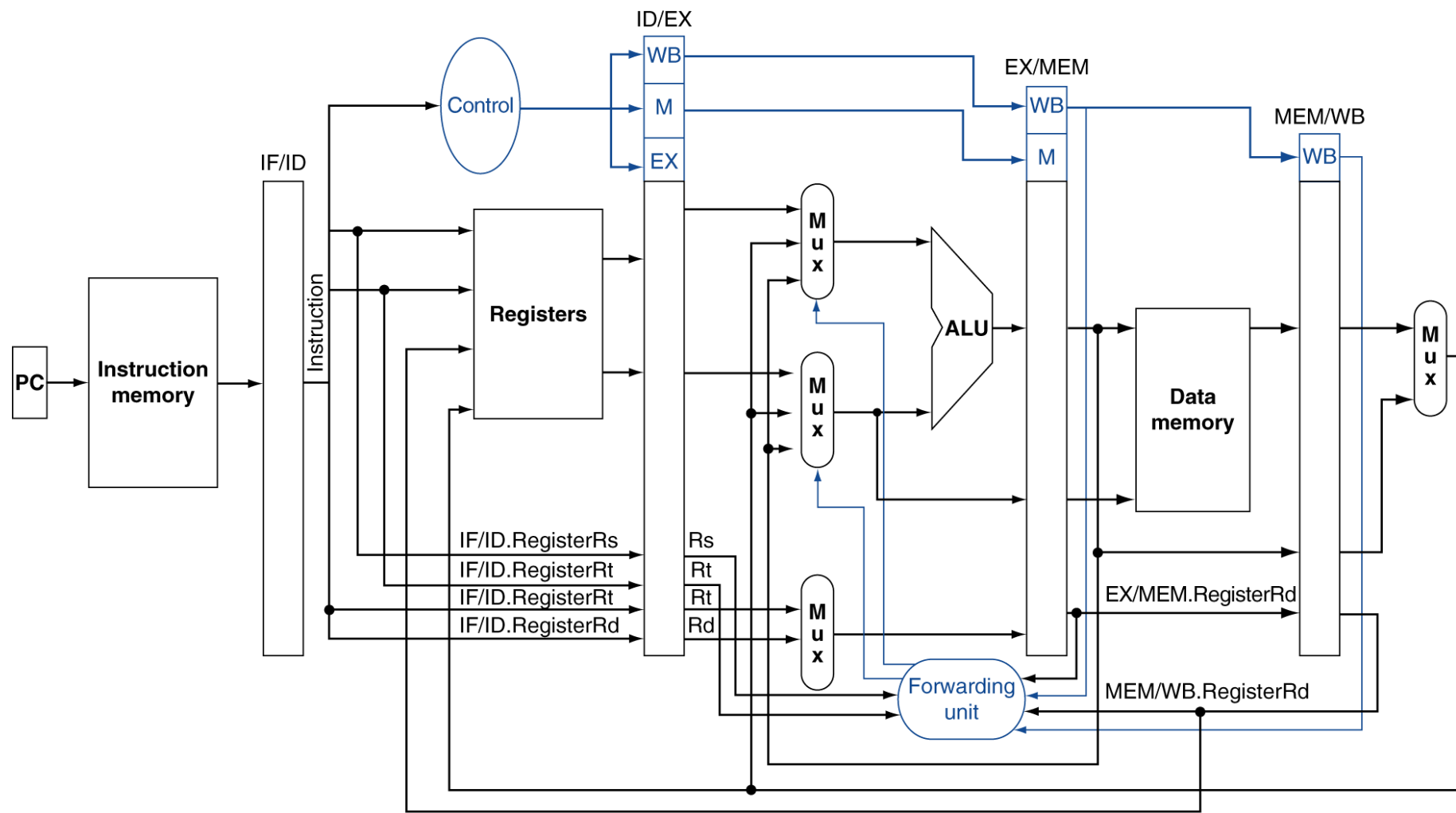


Revised Forwarding Condition

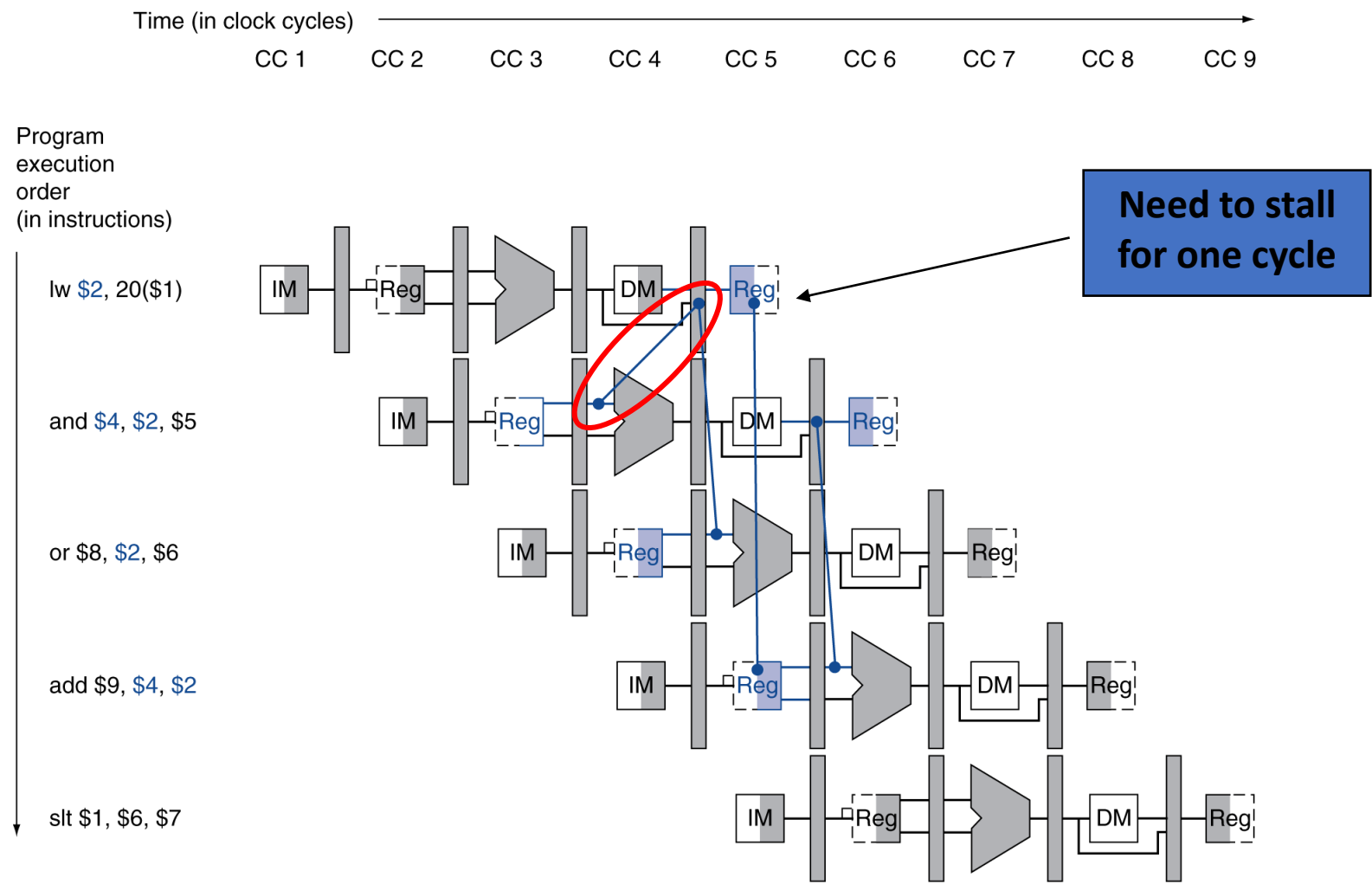
- MEM hazard
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
ForwardA = 01
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and not (EX/MEM.RegWrite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



Datapath with Forwarding



Load-Use Data Hazard



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

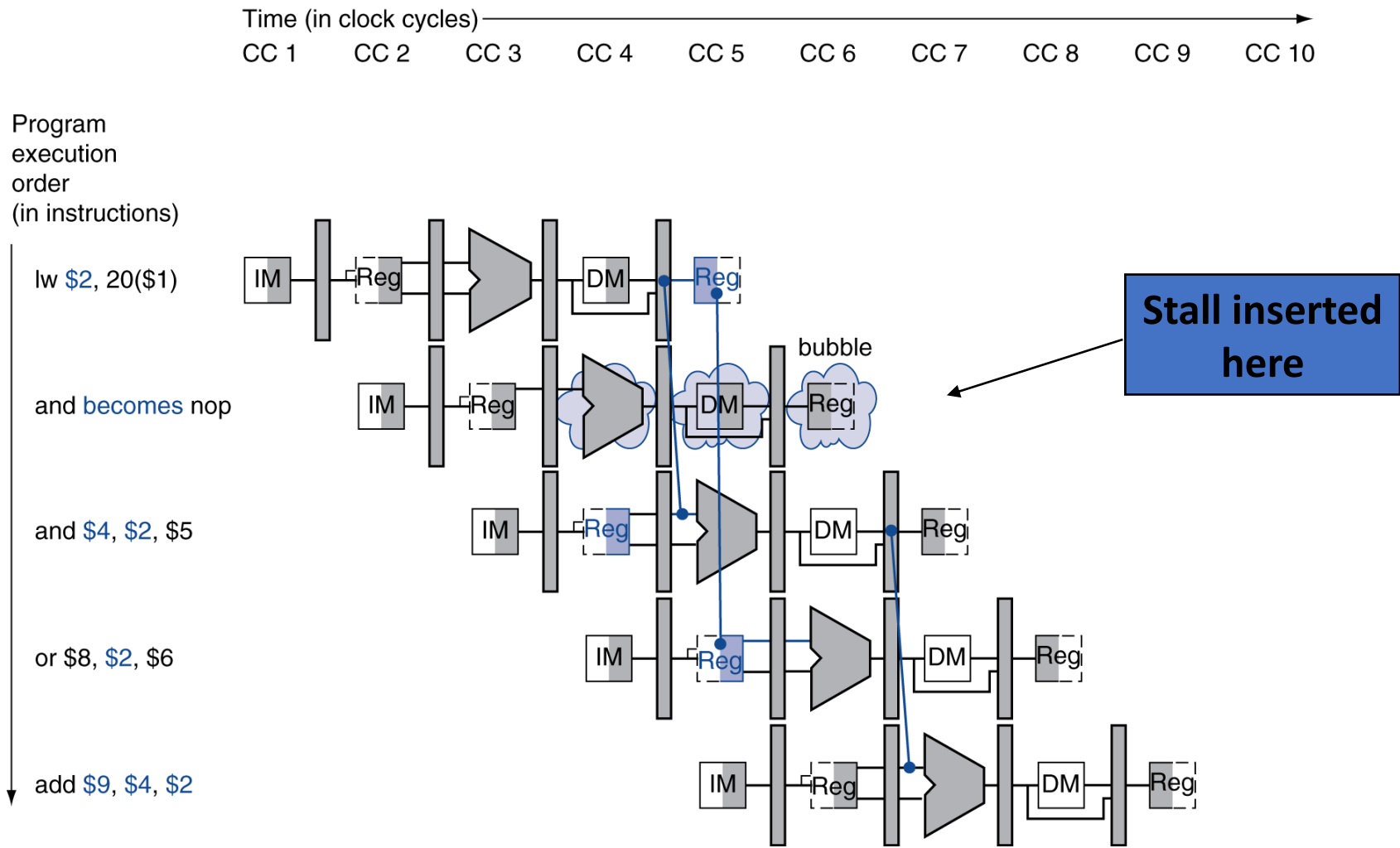


How to Stall the Pipeline

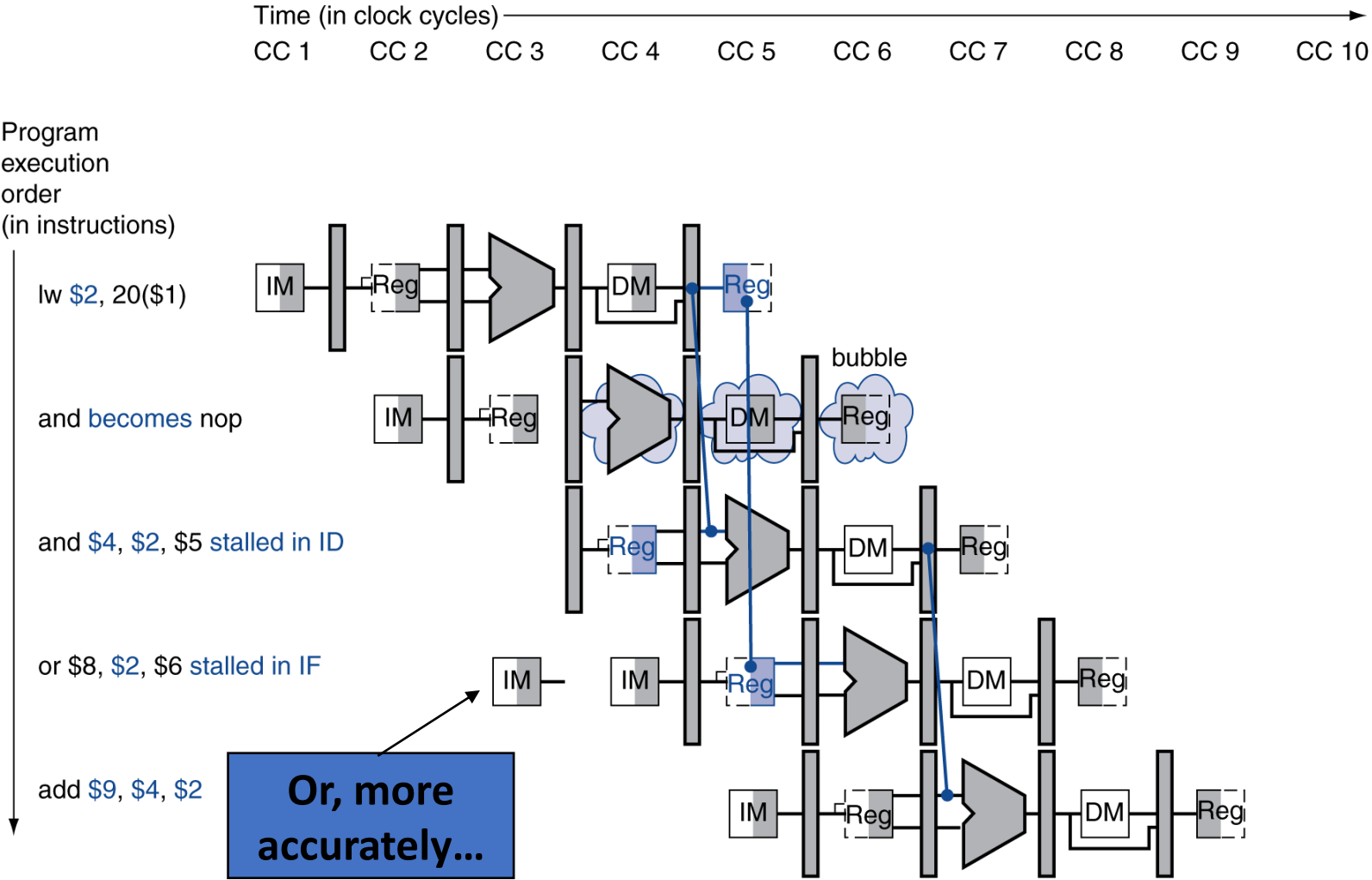
- Force control values in ID/EX register to 0
 - EX, MEM and WB do **nop** (no-operation)
- Prevent update of PC and IF/ID register
 - Using instruction is decoded again
 - Following instruction is fetched again
 - 1-cycle stall allows MEM to read data for 1w
 - Can subsequently forward to EX stage



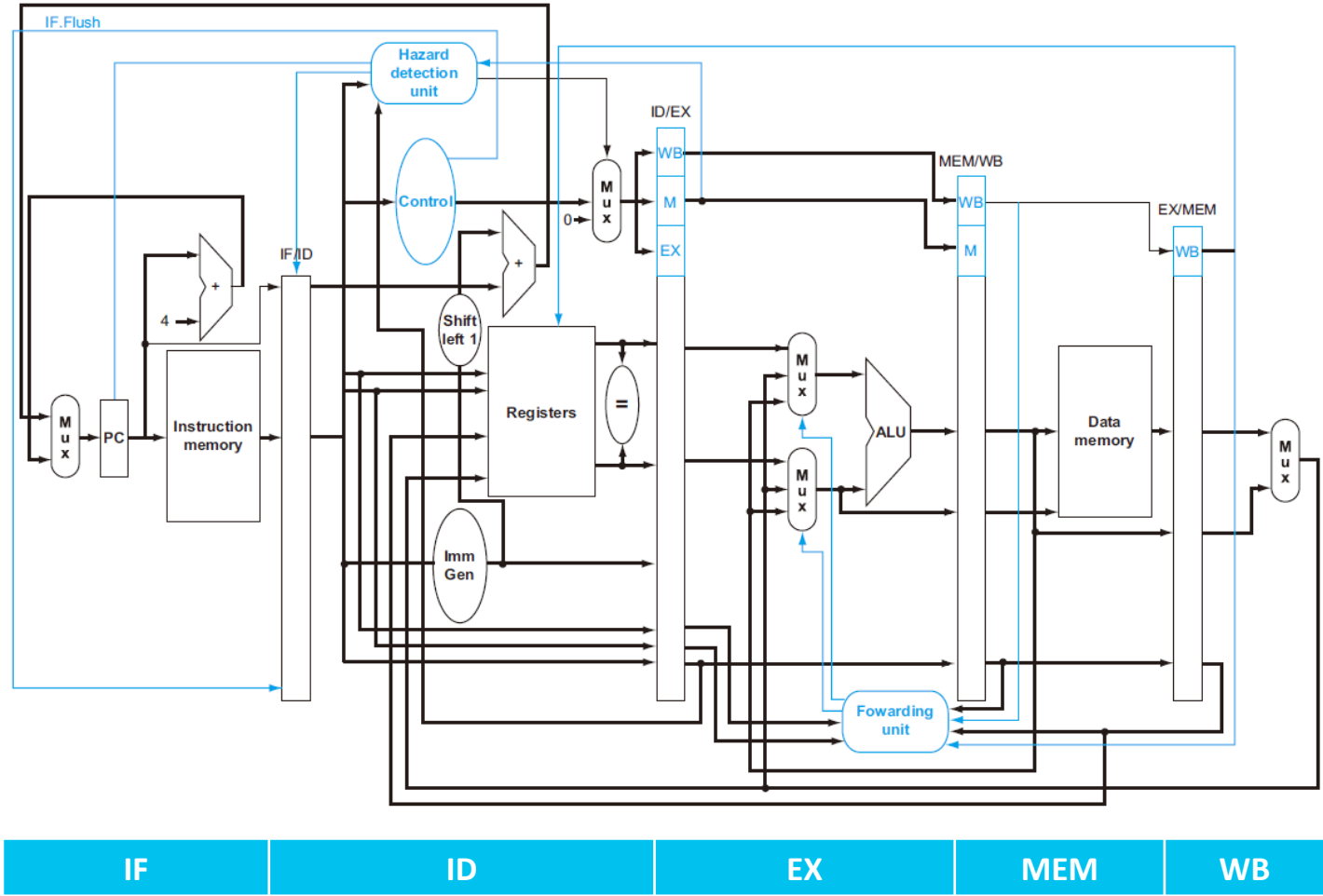
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline



Datapath with Hazard Detection



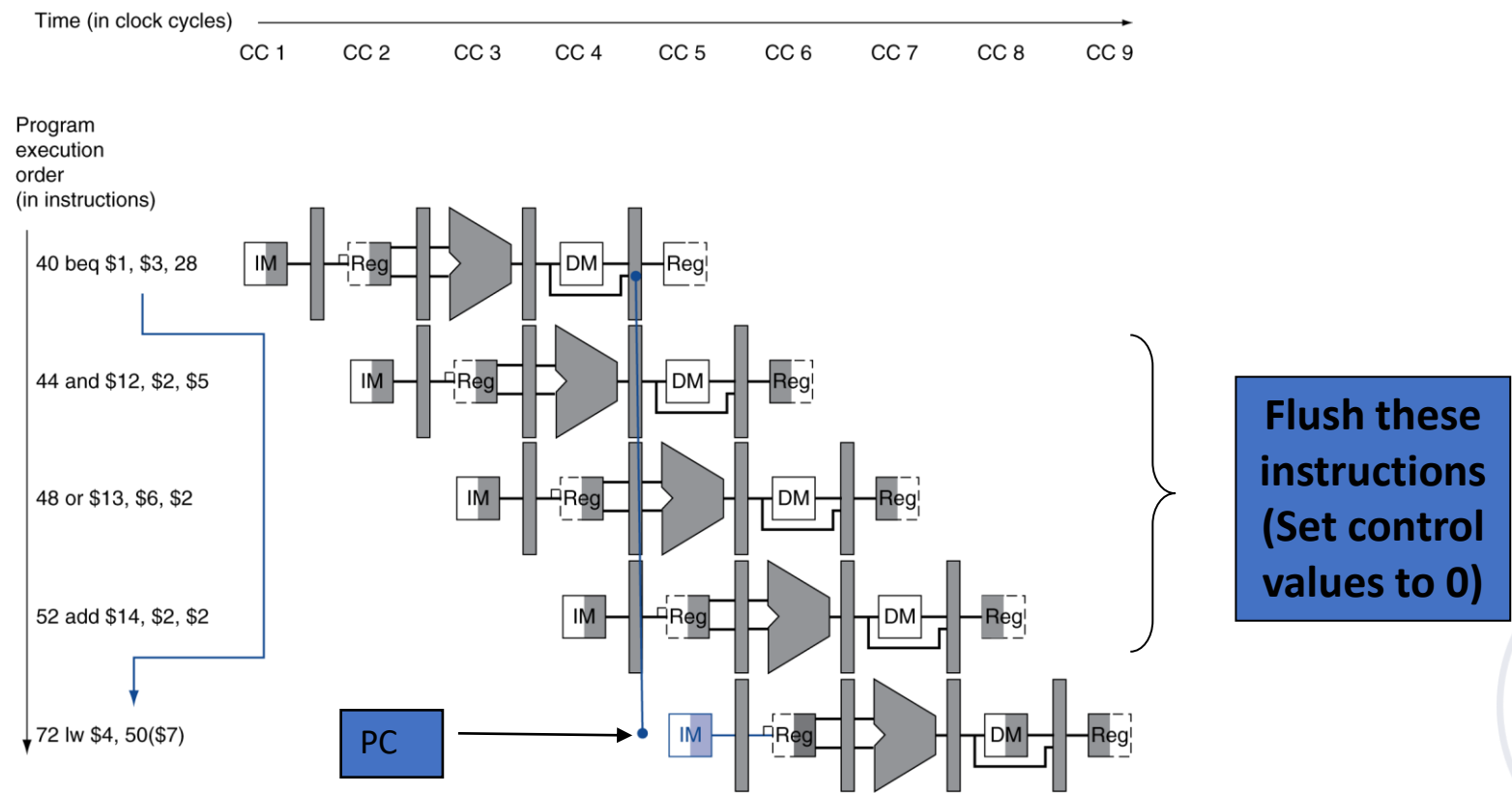
Stalls and Performance

- 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



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



Unconditional Jump

Jal - Jump and Link

Jalr - Jump and Link-Register

Conditional Branch

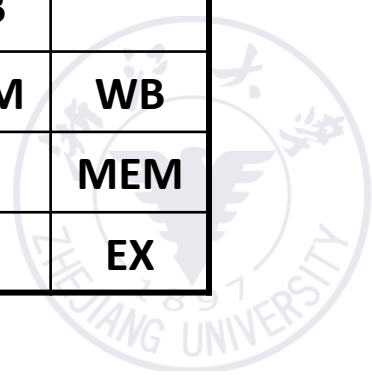
Stall on Branch

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

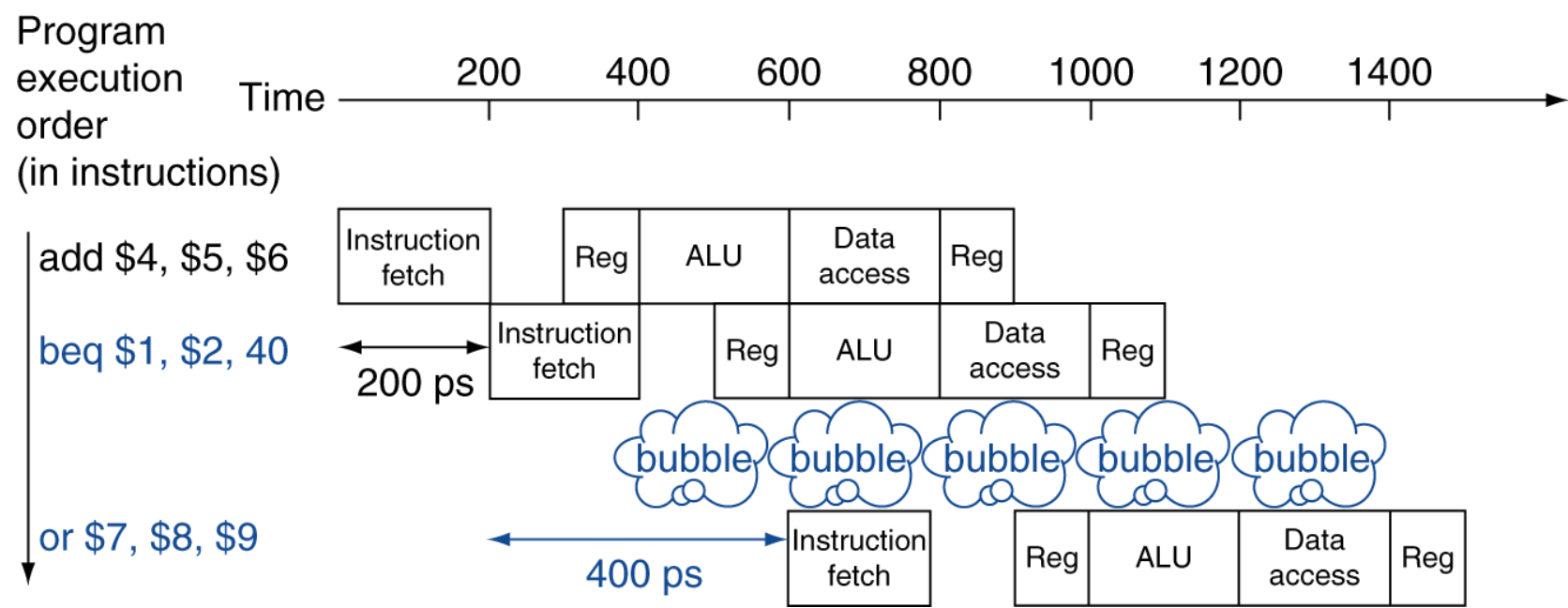
Branch causes a stall

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

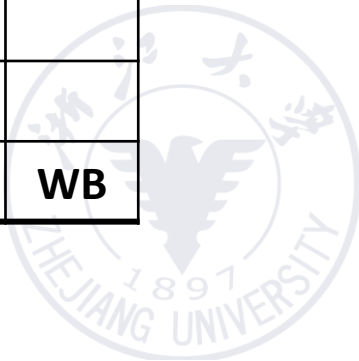
- Prediction taken
- Prediction not taken
- Delayed Branch



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

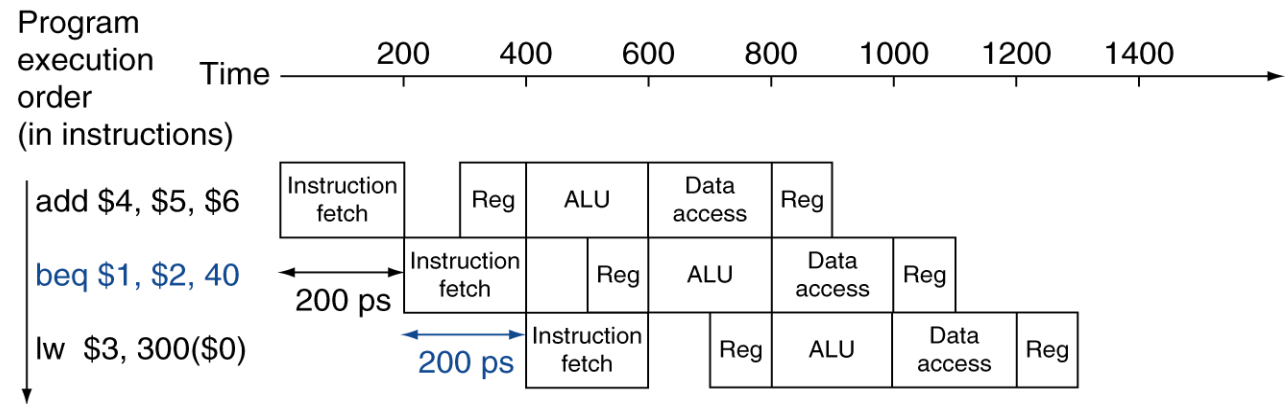


The behavior of a delayed branched is the same whether or not the branch is taken.

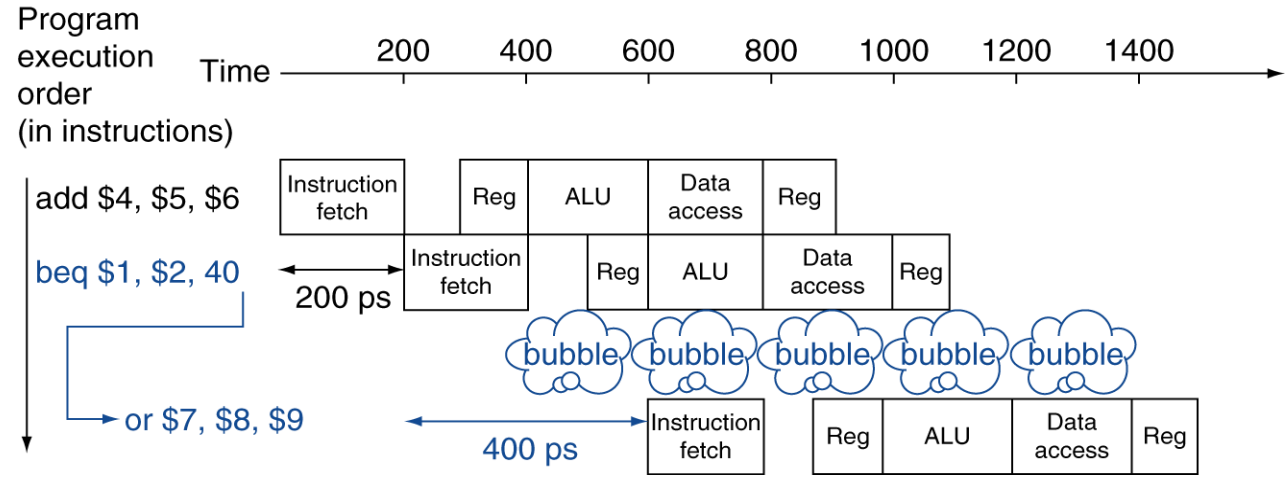
branch not taken instruction	Branch i	IF	ID	EX	MEM	WB				
	Branch Delay instruction i+1		IF	ID	EX	MEM	WB			
	instruction i+2			IF	ID	EX	MEM	WB		
	instruction i+3				IF	ID	EX	MEM	WB	
	instruction i+4					IF	ID	EX	MEM	WB
branch taken Instruction	Branch i	IF	ID	EX	MEM	WB				
	Branch Delay instruction i+1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

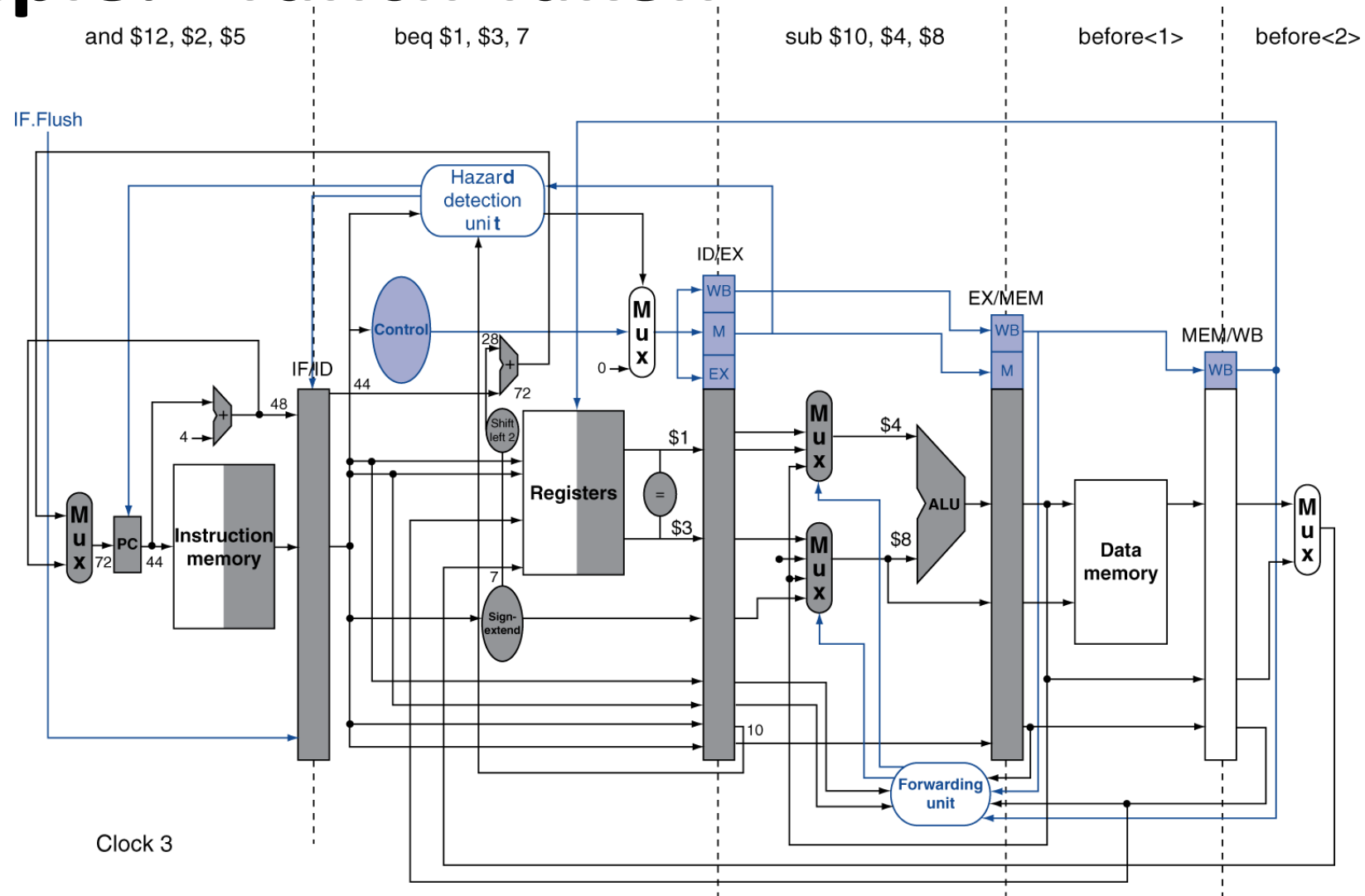
Branch not taken



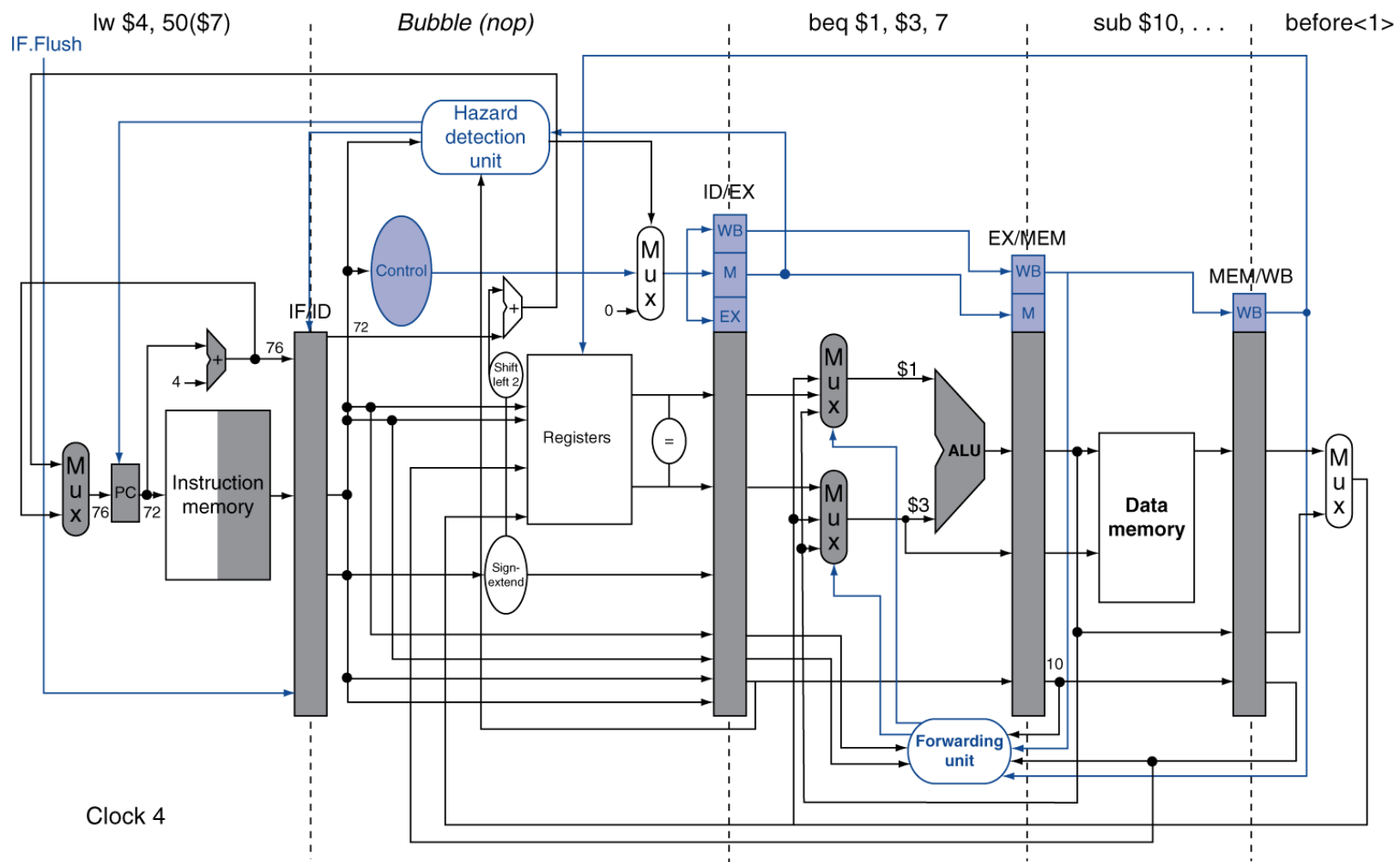
Branch taken



Example: Branch Taken

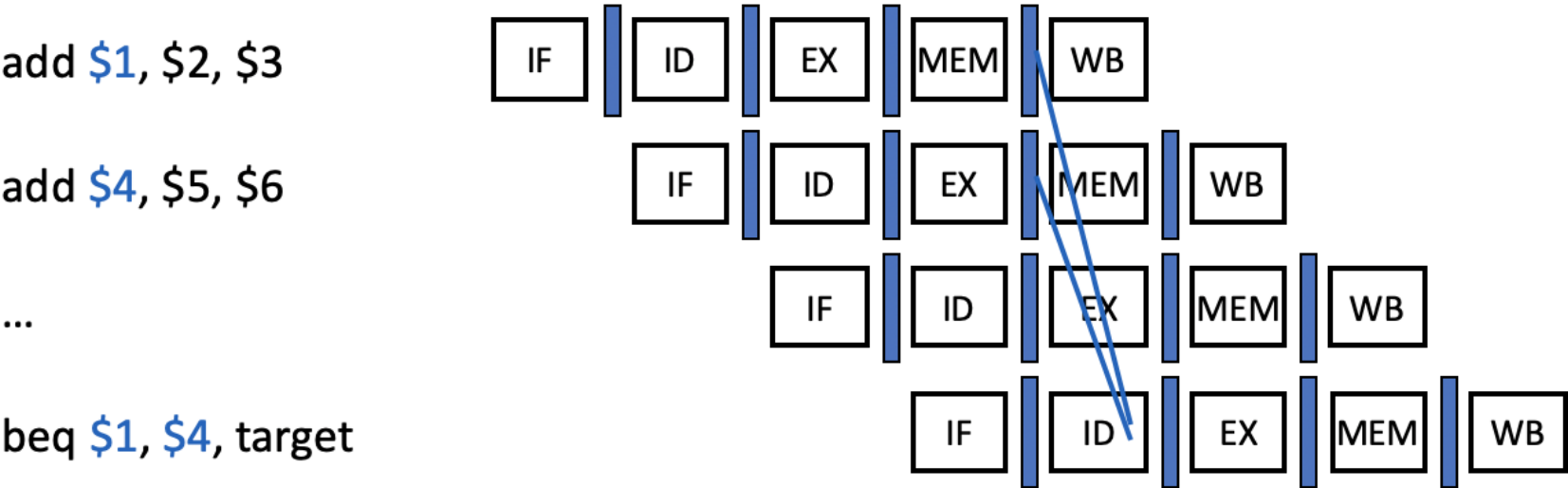


Example: Branch Taken



Data Hazards for Branches

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

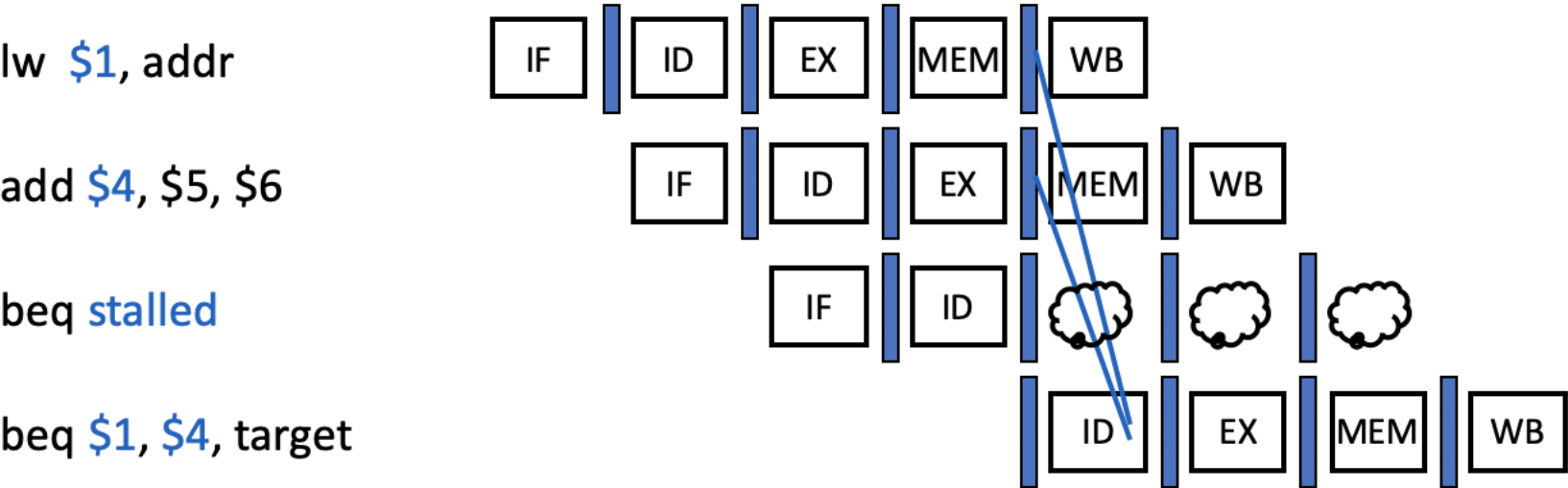


- Can resolve using forwarding



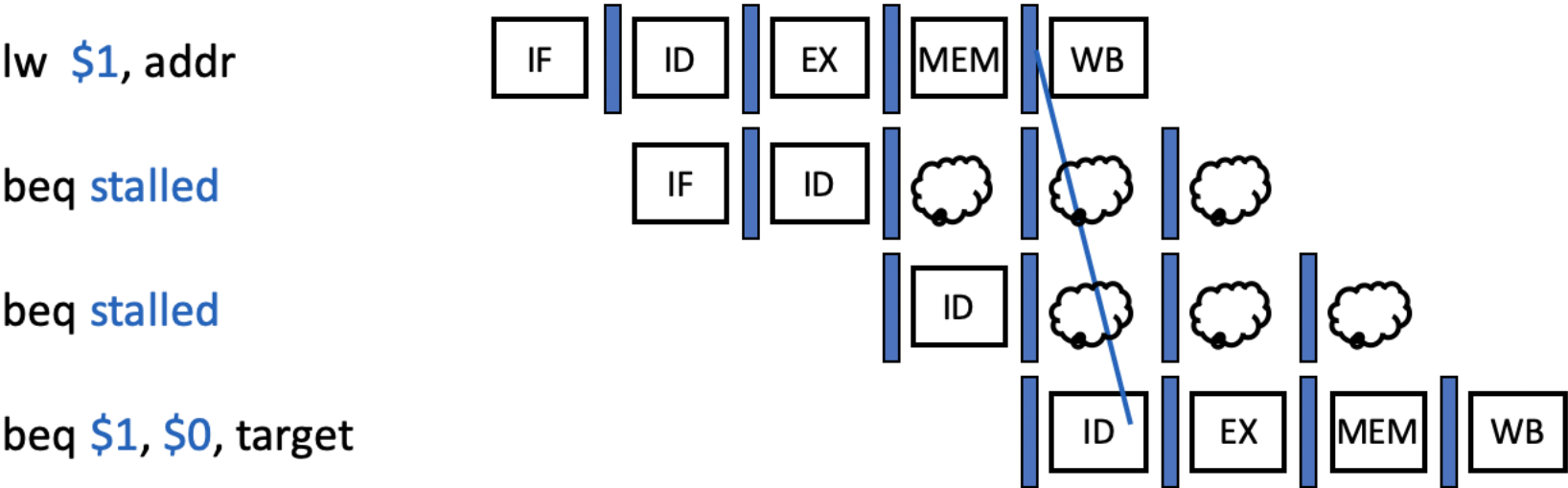
Data Hazards for Branches

- 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

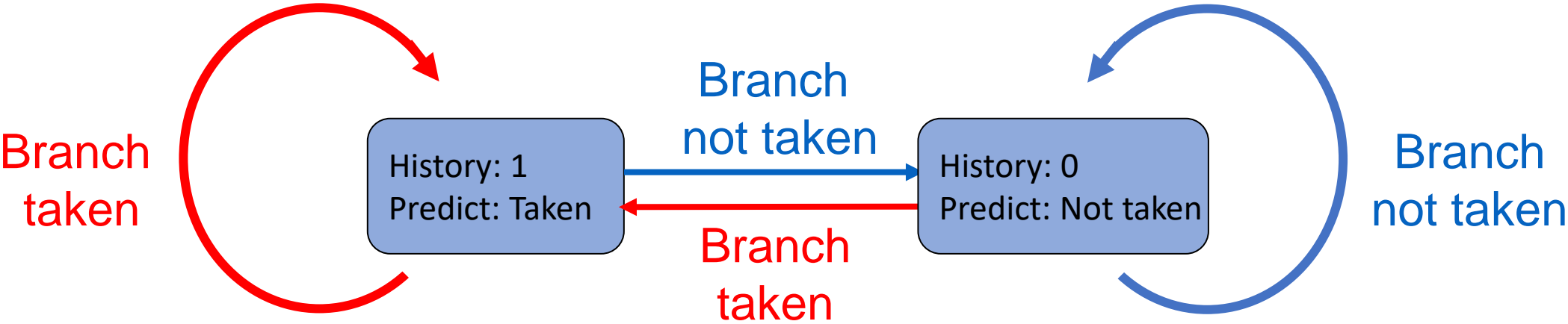


Dynamic Branch Prediction

- 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

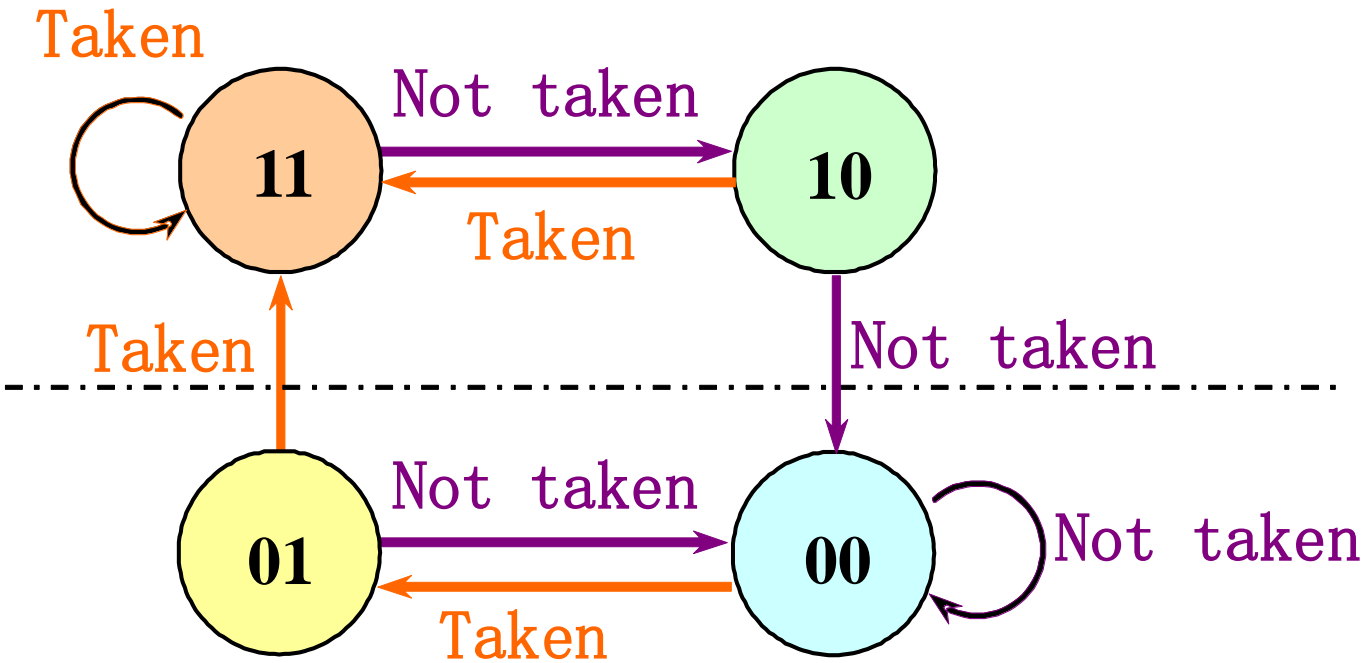
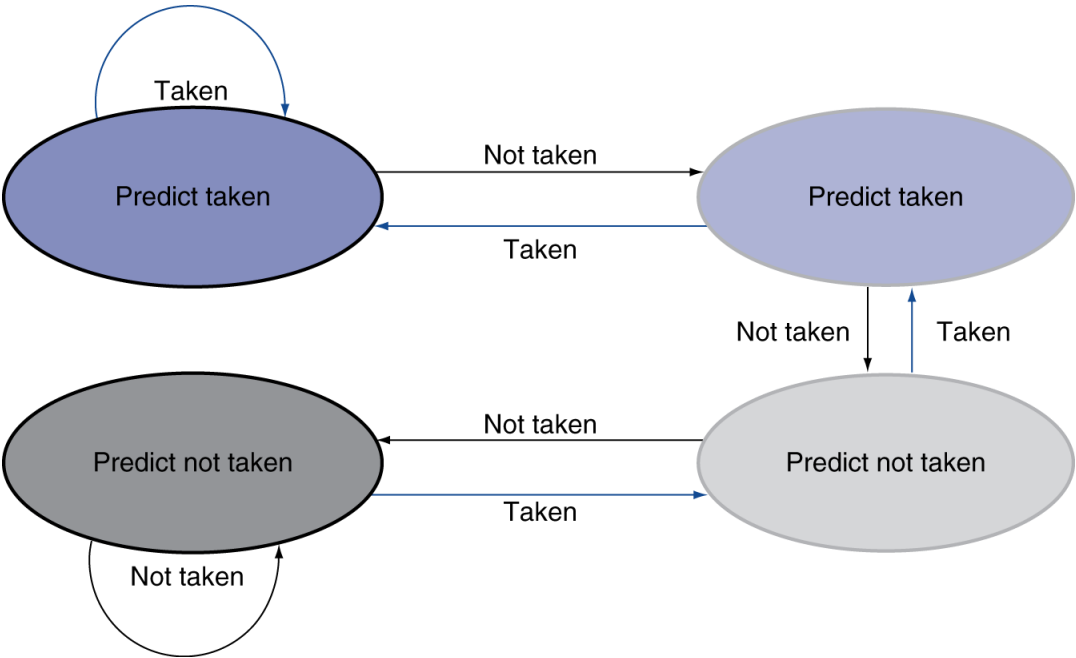
- Inner loop branches mispredicted twice!



- 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

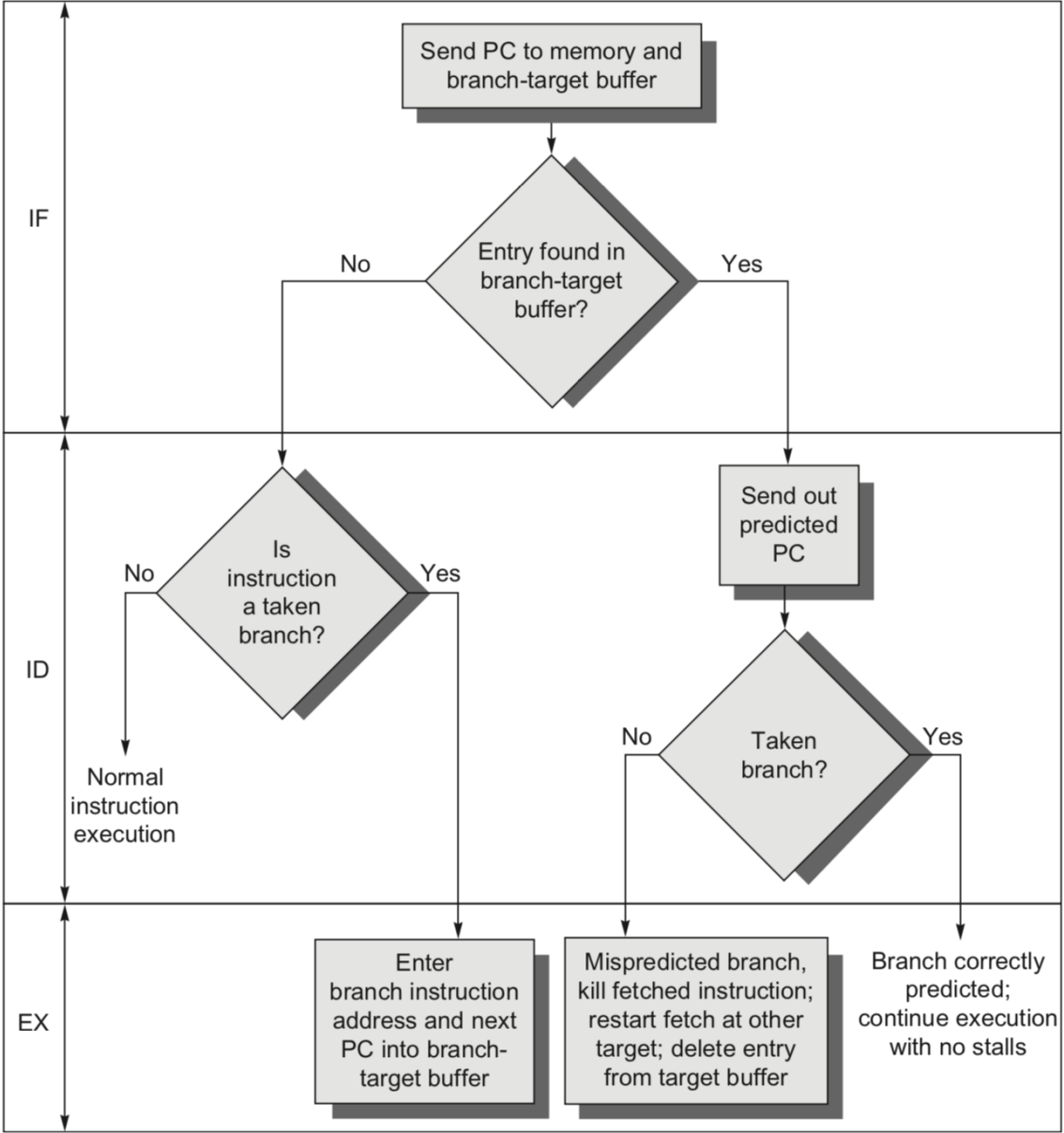
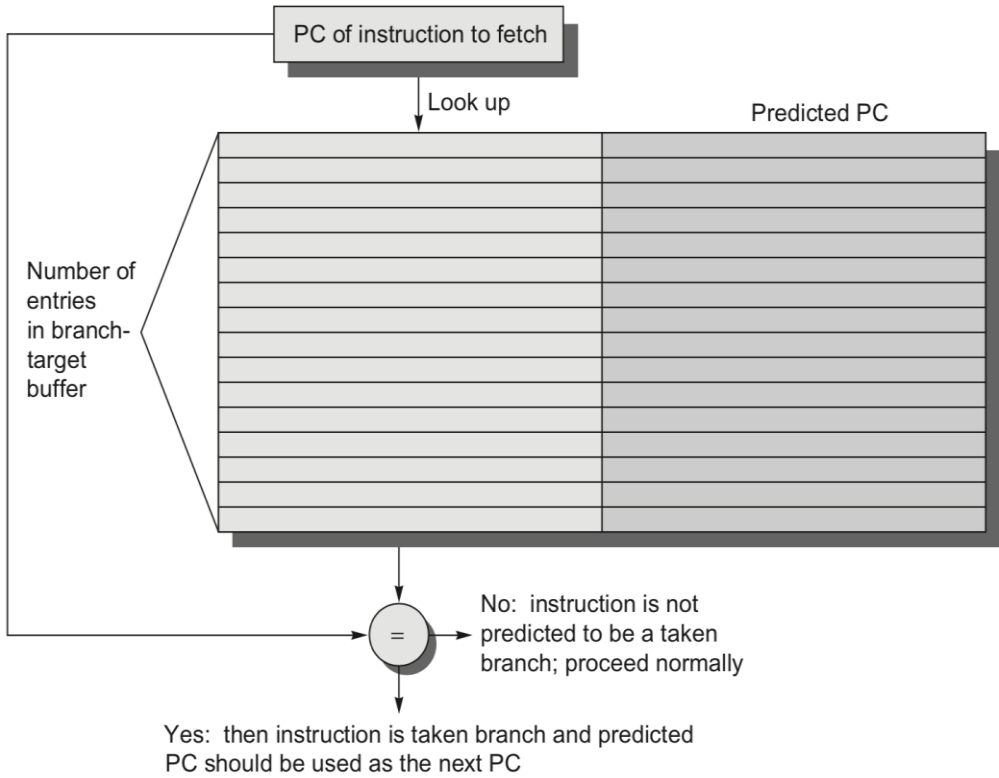


Advanced Techniques for Instruction Delivery and Speculation

- Increasing Instruction Fetch Bandwidth
 - Branch-Target Buffers
- Specialized Branch Predictors: Predicting Procedure Returns, Indirect Jumps, and Loop Branches
 - Integrated Instruction Fetch Units

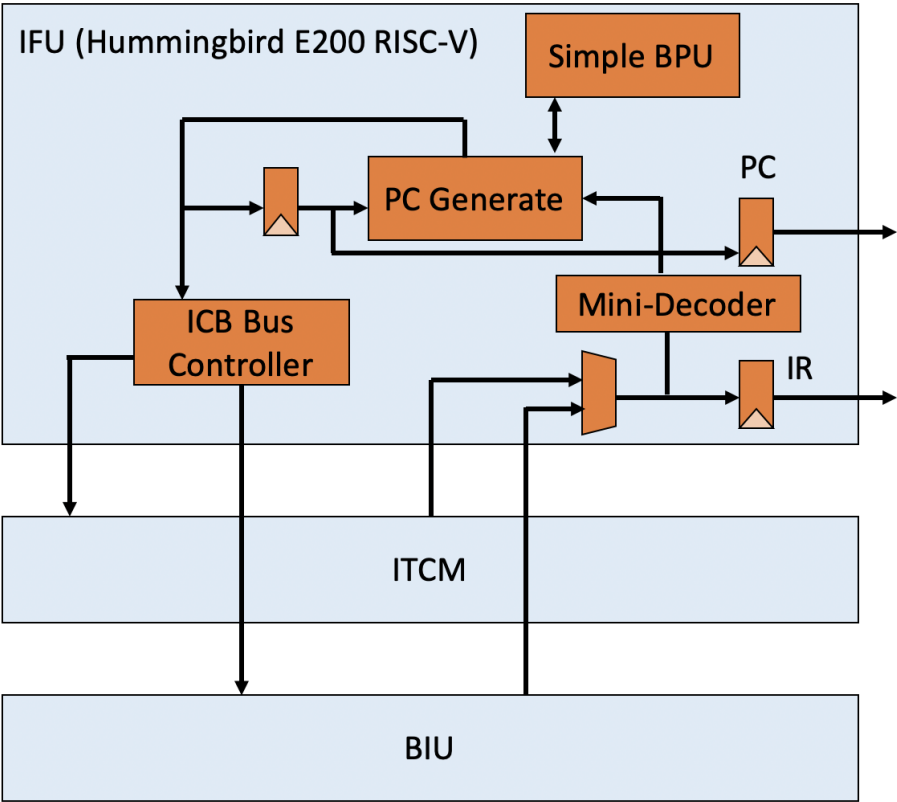


Branch-Target Buffers



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
- 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

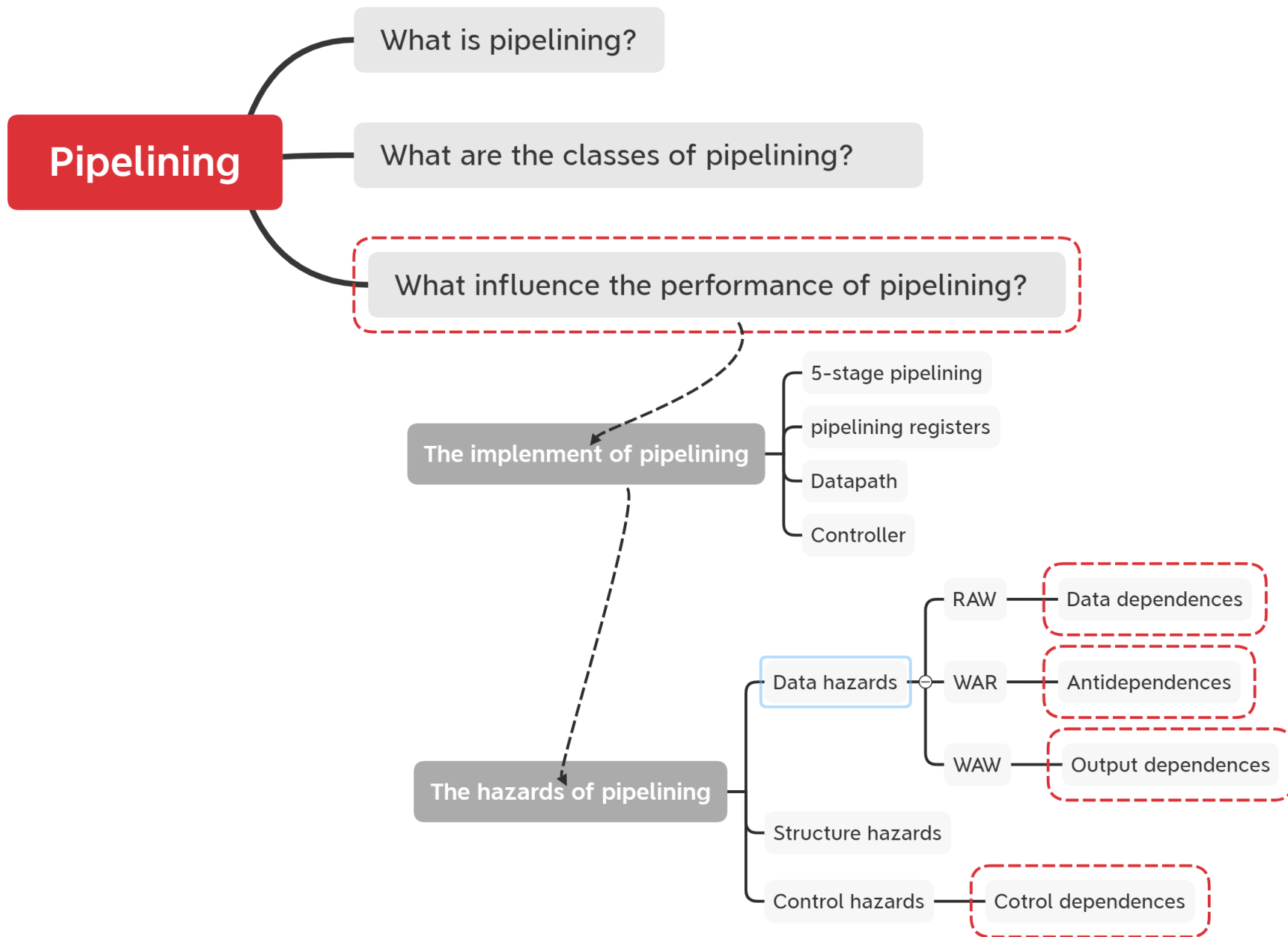


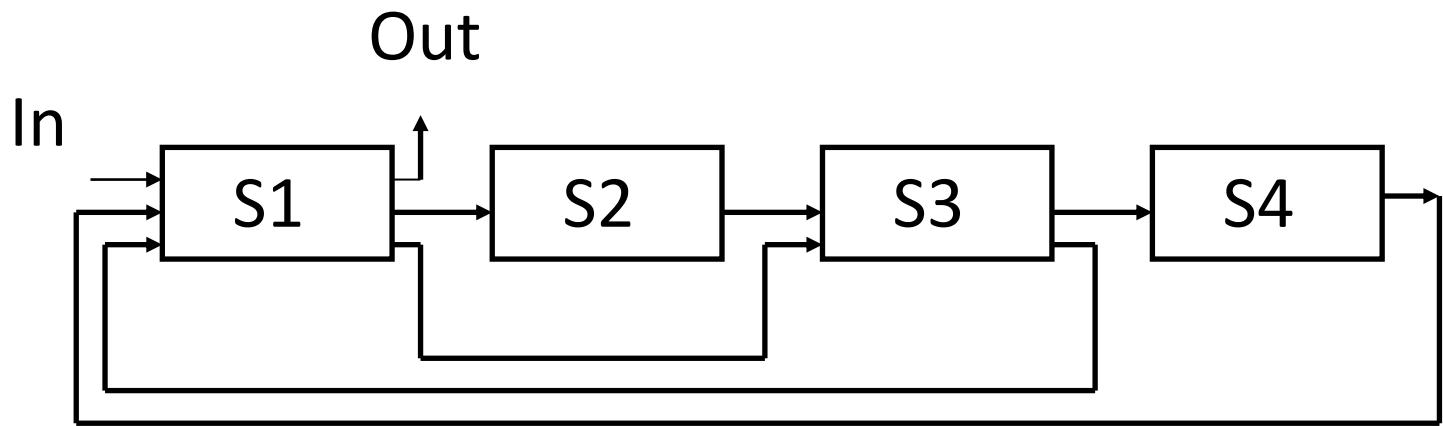
Branch-Target Buffer/Branch-Target Cache

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.







Reservation Table:

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



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									



Reservation Table:

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



Reservation Table:

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



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							

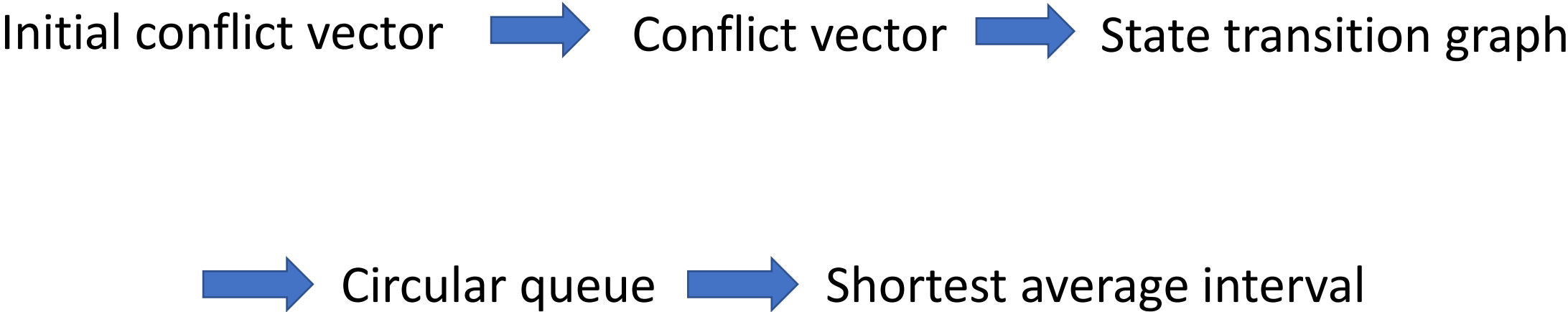


Reservation Table:

	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	



Schedule of Nonlinear pipelining without hazards

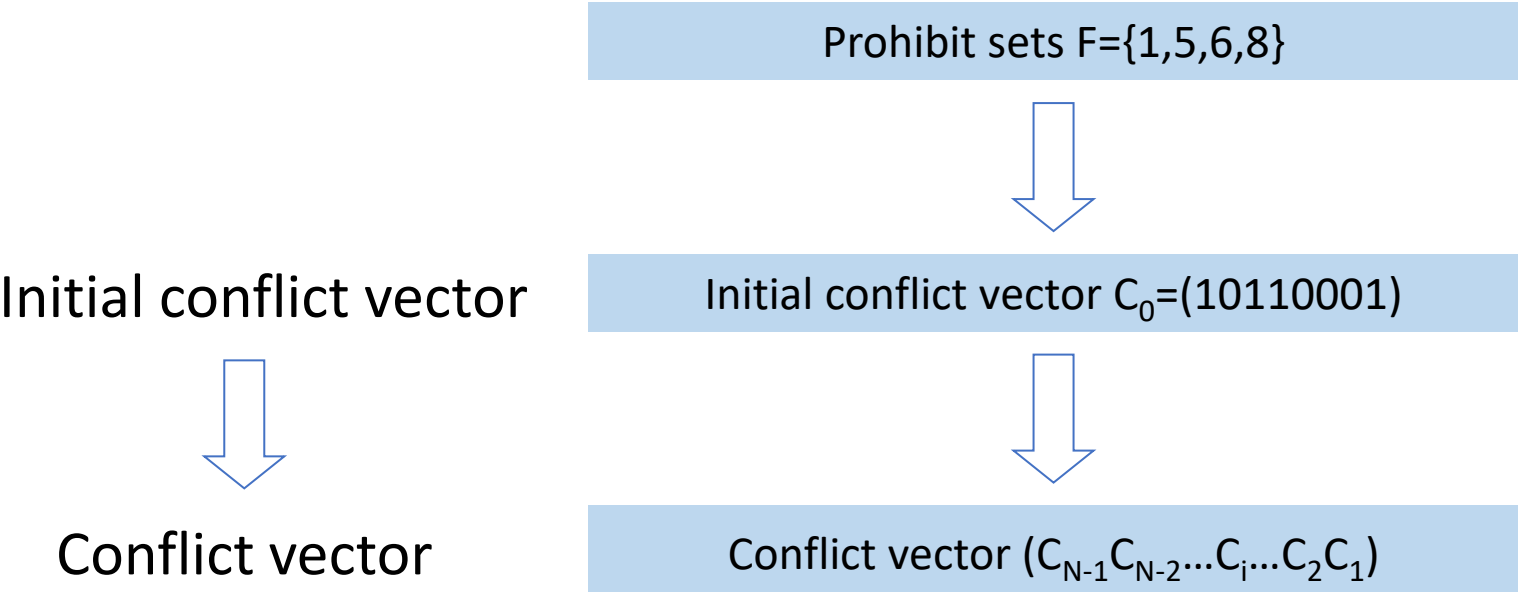


Initial conflict vector

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



Initial conflict vector



Conflict vector

- Initial conflict vector $C_0=(10110001)$ **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	1011000 <u>1</u>	10111101
1→	10110001	00101100
2→		10110001



Conflict vector

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

Interval	Initial	2	2
CCV	1011000 <u>1</u>	101111 <u>0</u> 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	1011000 <u>1</u>	101111 <u>0</u> 1	1 <u>0</u> 111111	10110001
1→	10110001	00101100	00001011	00000000
2→		10110001	00101100	00000000
3→			10110001	00000001
4→				10110001



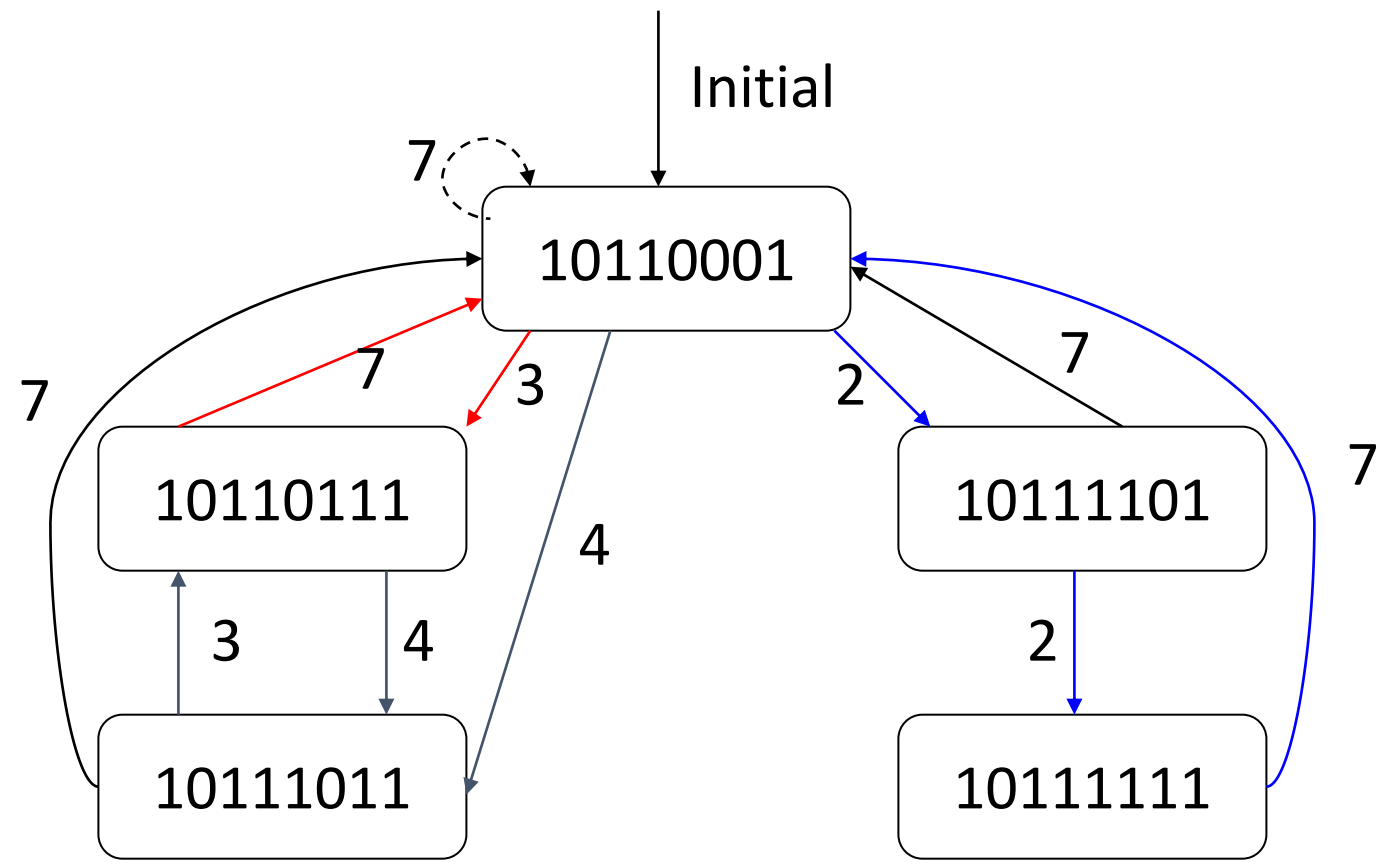
Conflict vector

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

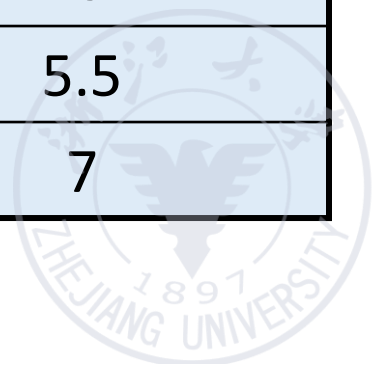
Interval	Initial	2	7
CCV	1011000 <u>1</u>	1 <u>0</u> 111101	10110001
1→	10110001	00101100	00000000
2→		10110001	00000001
3→			10110001



State transition graph

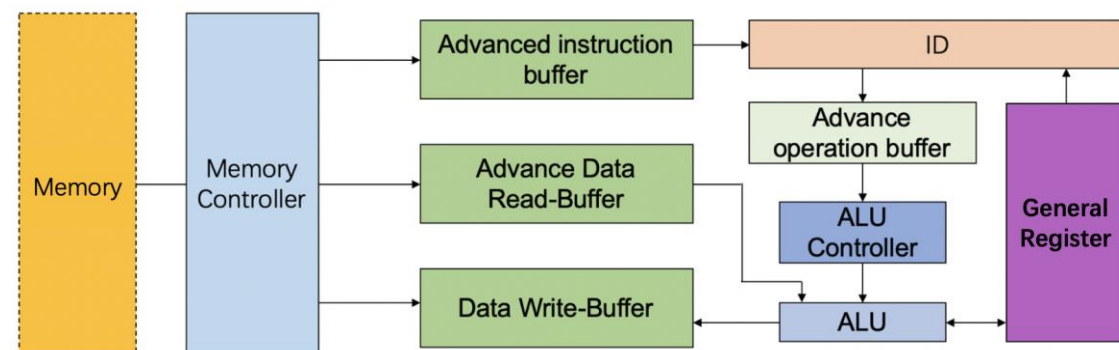
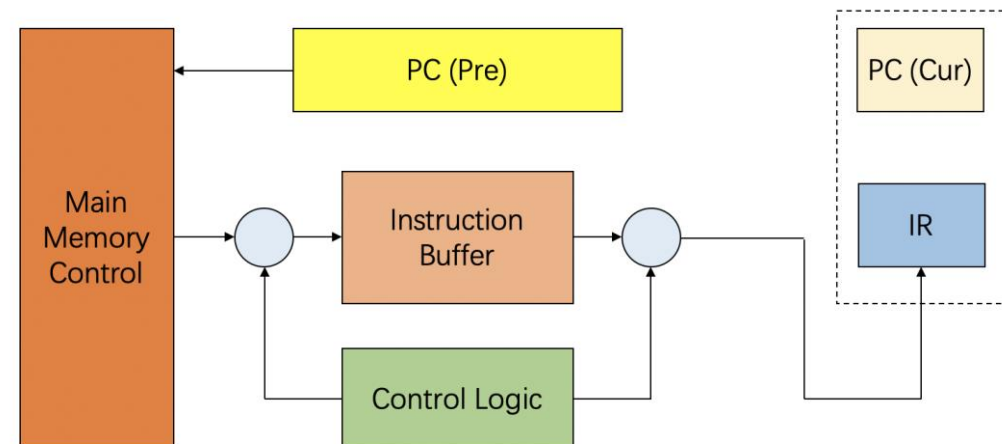


Circular queue	Shortest average interval
2,2,7	3.67
2,7	4.5
3,4	3.5
4,3	3.5
3,4,7	4.67
3,7	5
4,3,7	4.67
4,7	5.5
7	7



Summary

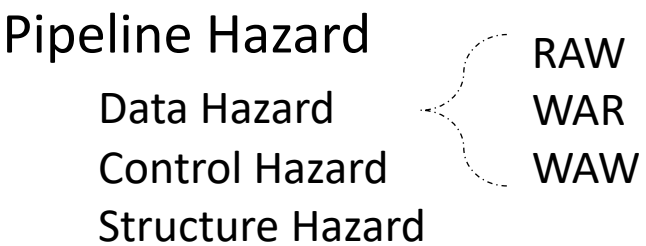
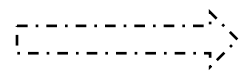
- 1. How the instruction is executed
 - Sequential execution
 - Overlap once
 - Second overlap
 - Pipeline
- 2. Classification of pipelines
 - Single function, multi-function
 - Static, dynamic
 - Linear, non-linear
 - In-order, out-of-order
- 3. Performance indicators of the pipeline
 - Throughput rate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



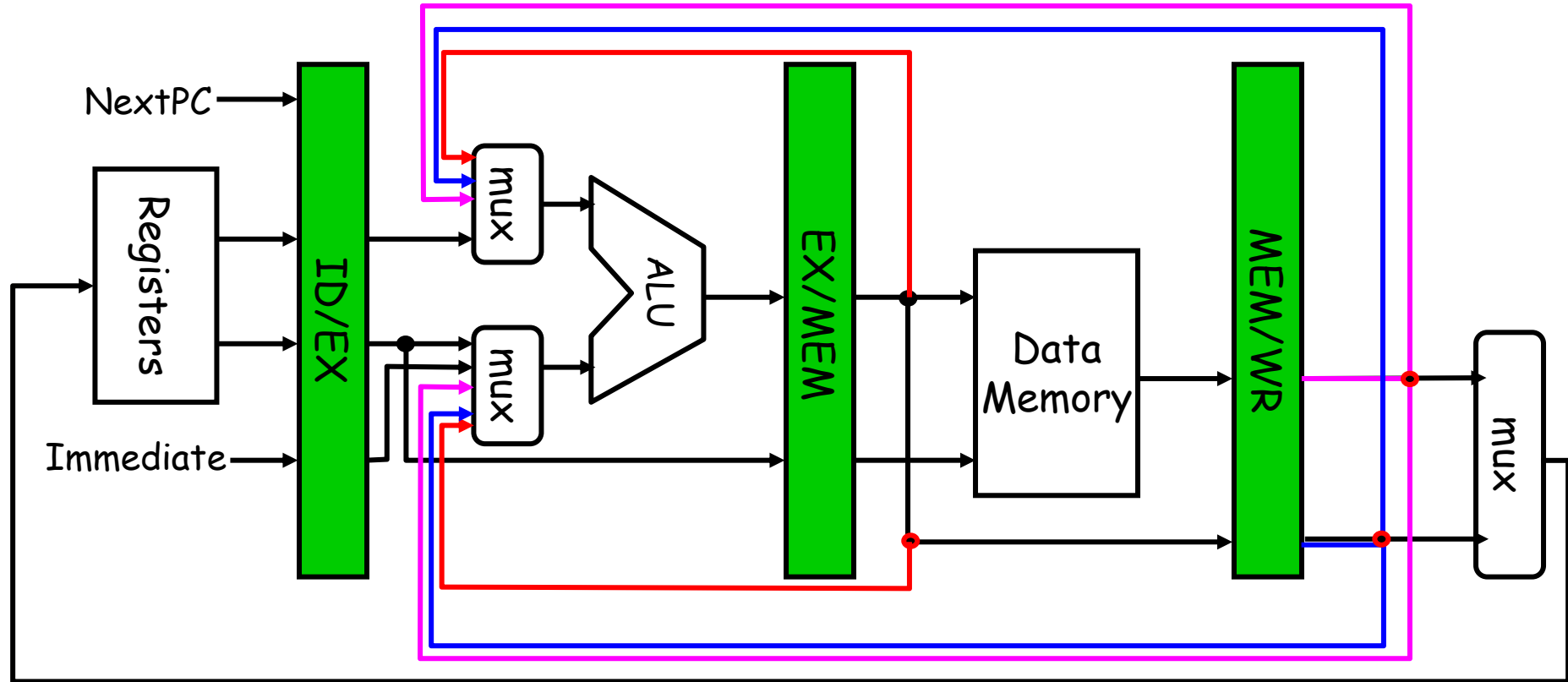
- 5. Dynamic Branch Prediction

- Branch History Table (BHT)
- Branch-Target Buffer (BTB)

- 6. Non-linear pipeline scheduling problem



Hardware Change for Forwarding



- **EX/Mem.ALUoutput → ALU input**
- **MEM/WB.ALUoutput → ALU input**
- **MEM/WB.LMD → ALU input**