



## **Topic 8**

---

# **Data Hazards**

# Hazards

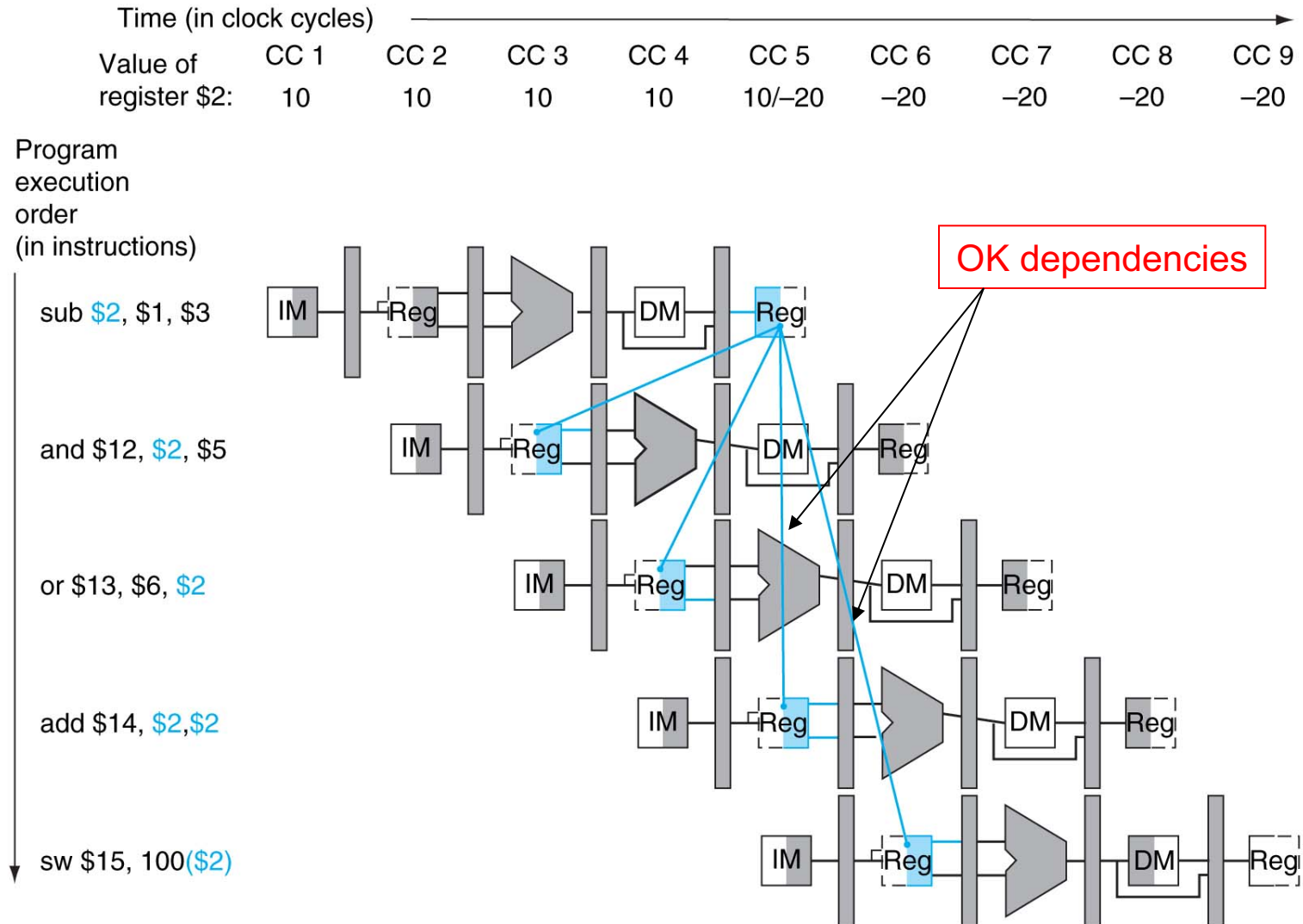
- Situations that prevent starting the next instruction in the next cycle
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Decision on control action depends on previous instruction
- Structure hazards
  - A required resource is busy

# Data Hazards in ALU Instructions

- Consider this sequence:

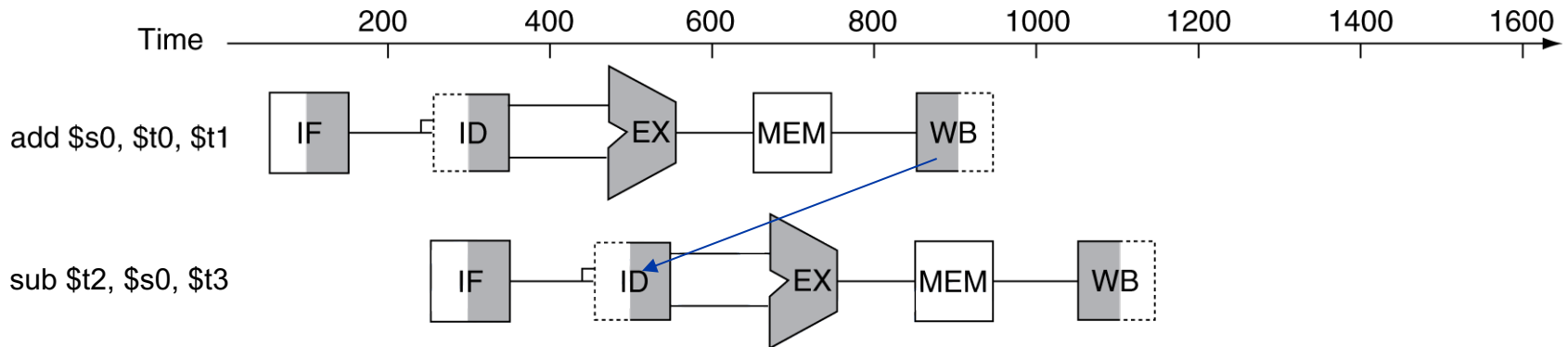
```
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
add  $14, $2, $2
sw   $15, 100($2)
```

# Data Dependencies



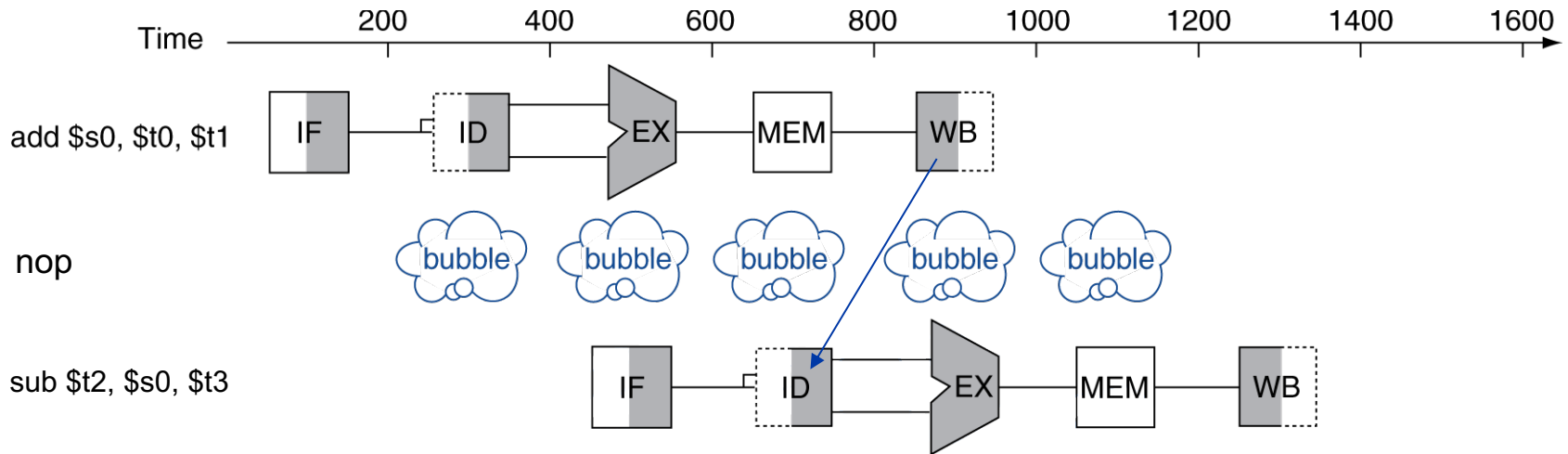
# Data Hazards

- An instruction must be delayed
  - If an instruction depends on completion of a data by a previous instruction
  - By inserting **bubbles** or **stalls** (nop instruction)
  - Example  
add    **\$s0**, \$t0, \$t1  
sub    \$t2, **\$s0**, \$t3



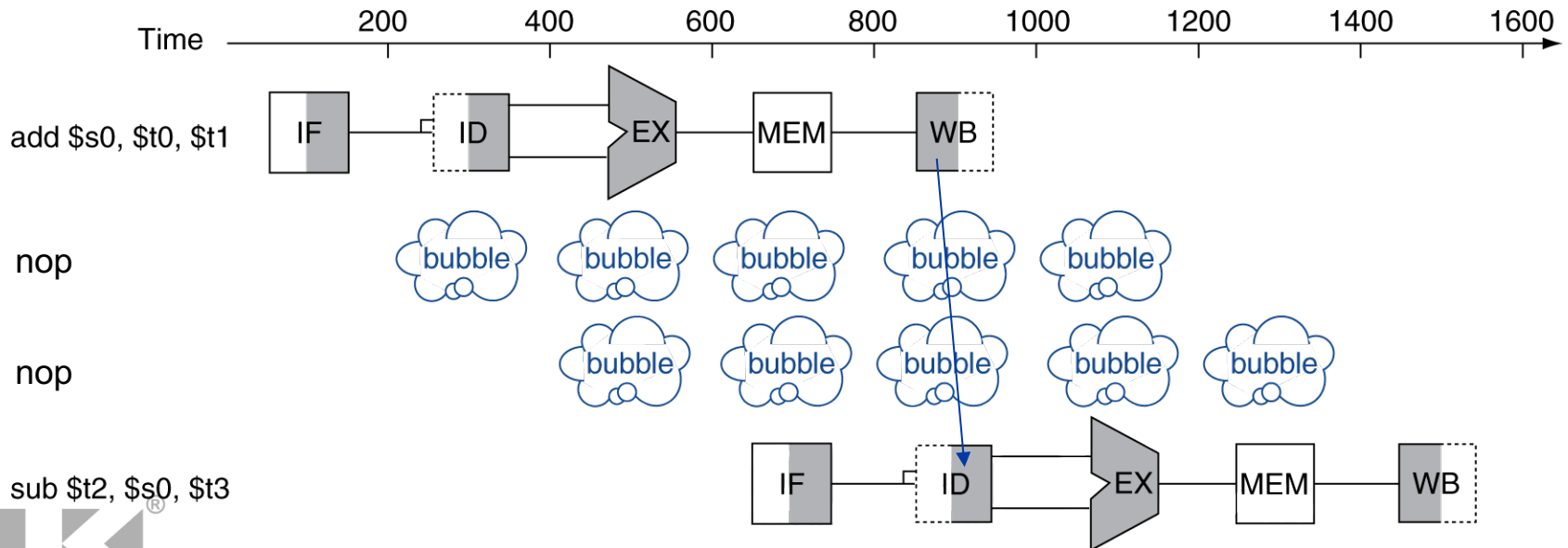
# Data Hazards

- An instruction must be delayed
    - If an instruction depends on completion of a data by a previous instruction
    - By inserting **bubbles** or **stalls**
- nop     # no operation  
         # machine code: 0x00000000



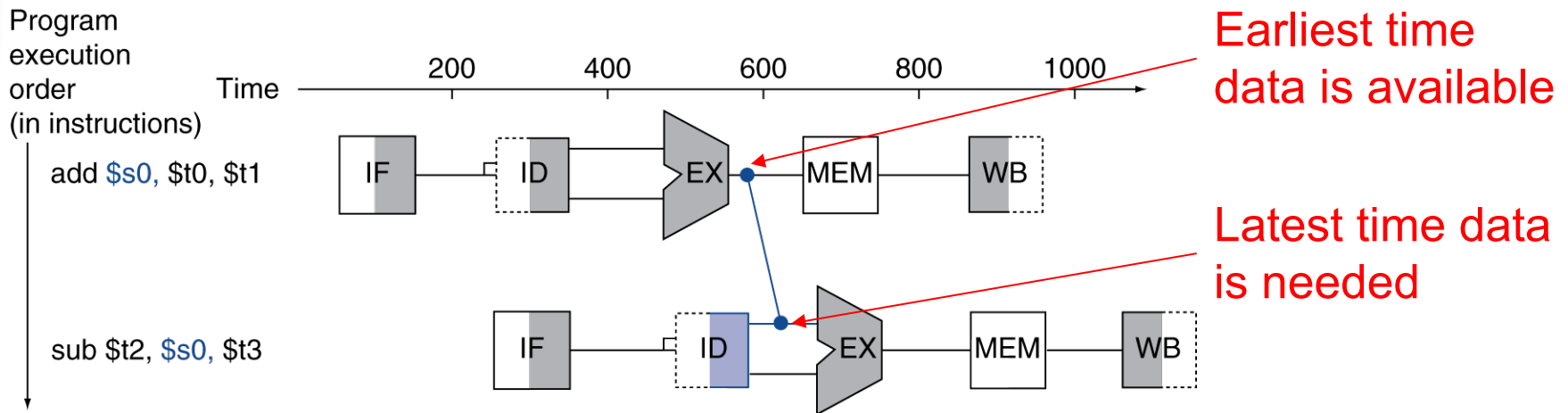
# Data Hazards

- An instruction must be delayed
    - If an instruction depends on completion of a data by a previous instruction
    - By inserting **bubbles** or **stalls**
- nop     # no operation  
         # machine code: 0x00000000



# Forwarding (aka Bypassing)

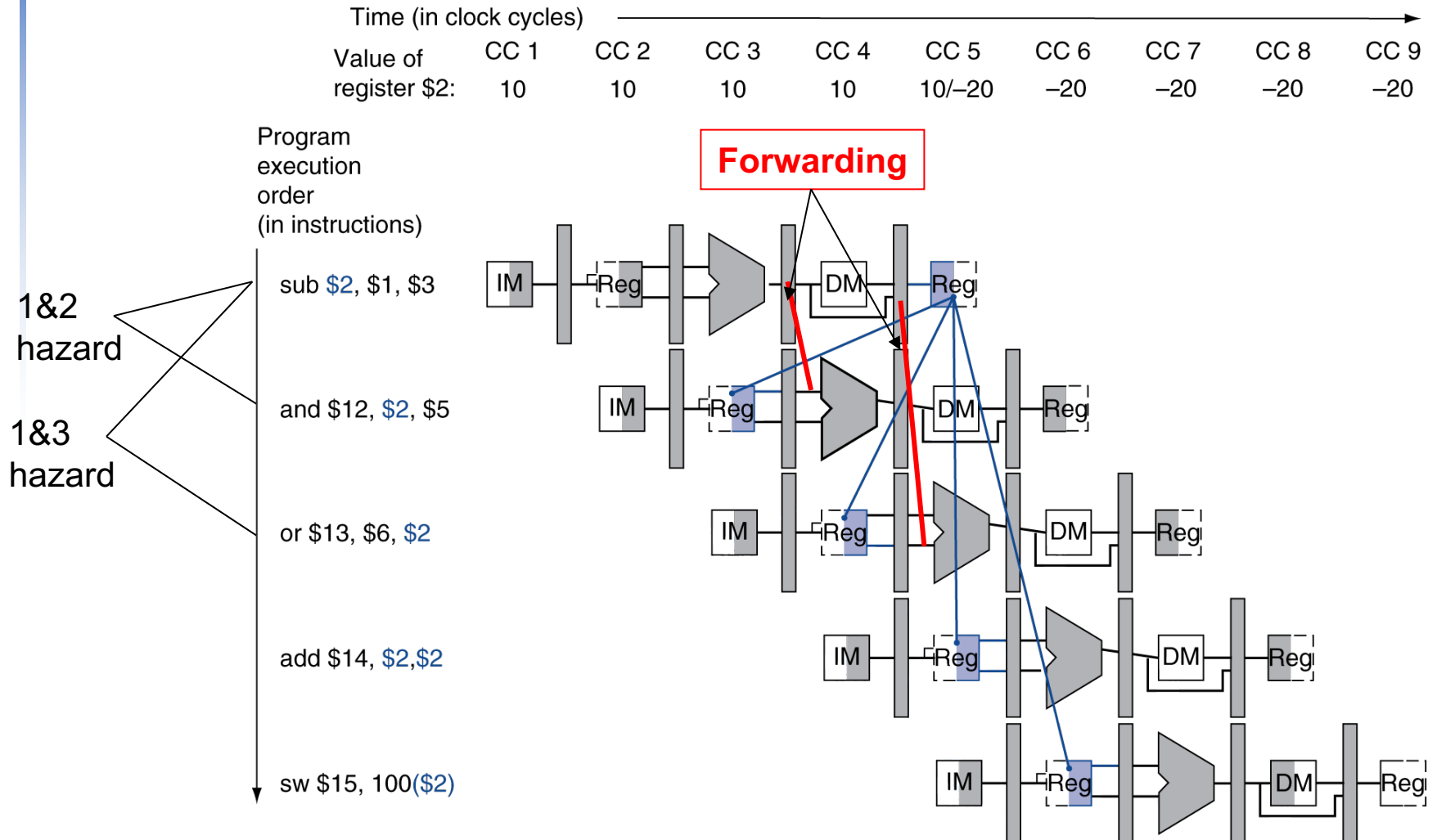
- Inserting nop (delay) compromises performance
- Data hazard can be solved with **Forwarding**
- Use result as soon as it is available
  - Don't wait for it to be stored in a register
  - Requires extra hardware in the datapath



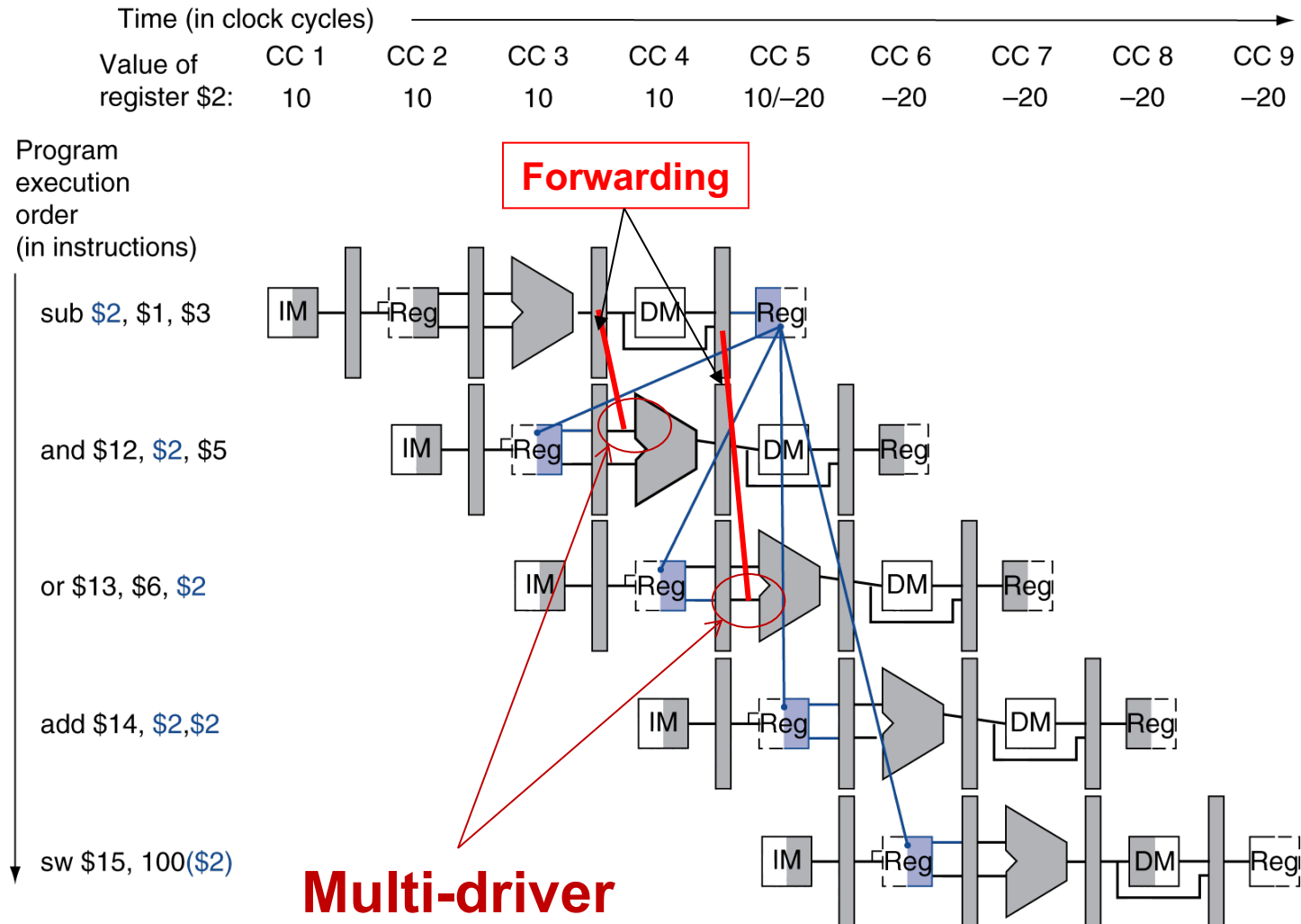
**Can't connect input and output of a component directly**



# Forwarding from Registers

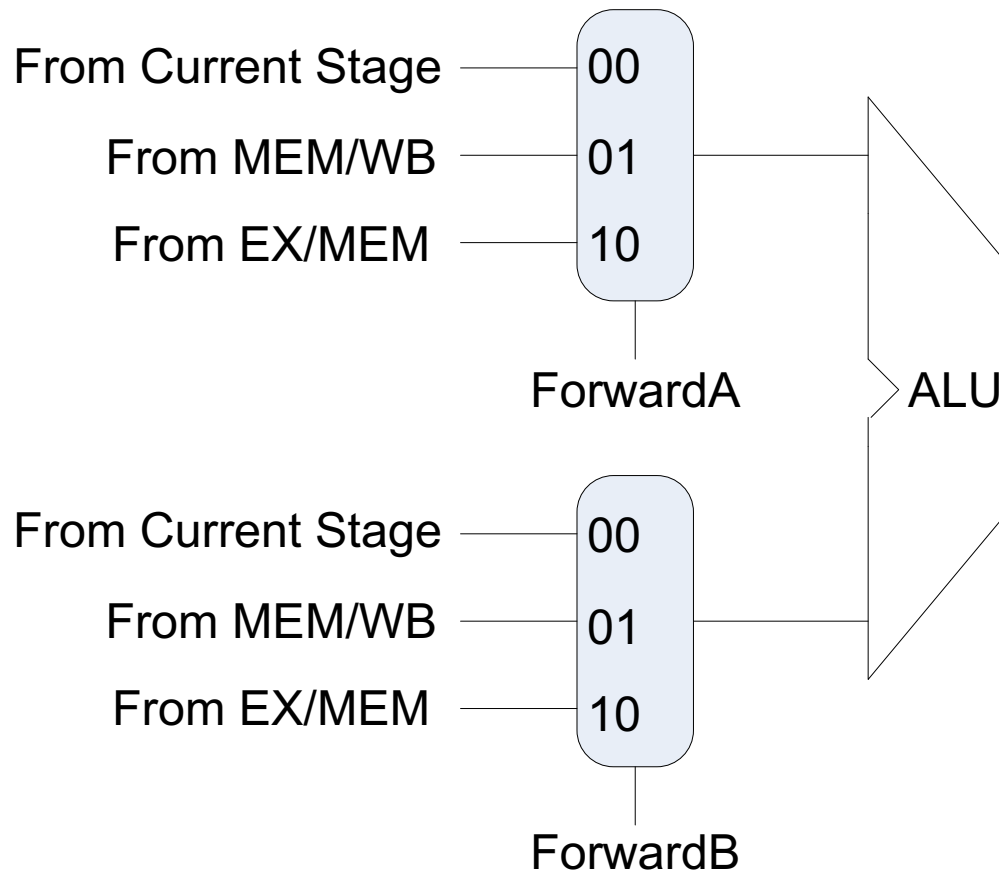


# Dependencies & Forwarding

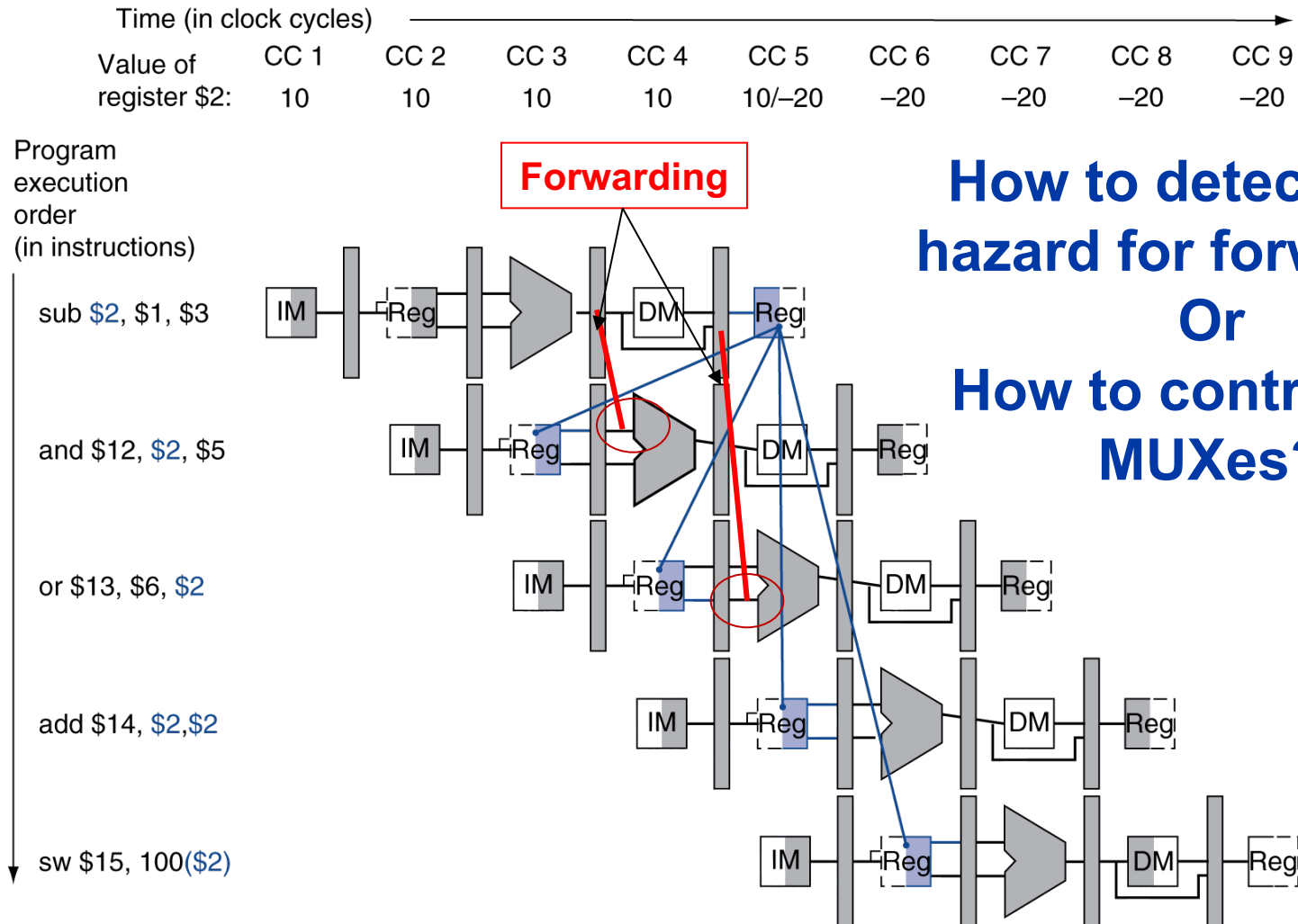


# Forwarding Paths

- Forwarding paths are created between stage pipeline registers and ALU inputs
  - By using MUXes



# Dependencies & Forwarding



# Detect the Condition for Forwarding

- Denotation:
  - E.g.  $ID/EX.RegisterRs = \text{register number}$  for Rs residing in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - $ID/EX.RegisterRs, ID/EX.RegisterRt$
- Data hazard conditions include

Hazard  
between  
1&2

- 1a.  $EX/MEM.RegisterRd == ID/EX.RegisterRs$
- 1b.  $EX/MEM.RegisterRd == ID/EX.RegisterRt$

Fwd from  
EX/MEM  
pipeline reg

Hazard  
between  
1&3

- 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

- Only if an earlier instruction will write back to a register, determined by
  - EX/MEM.RegWrite
  - MEM/WB.RegWrite
- And only if Rd for that instruction is not \$zero (constant register), determined by
  - EX/MEM.RegisterRd  $\neq$  0
  - MEM/WB.RegisterRd  $\neq$  0

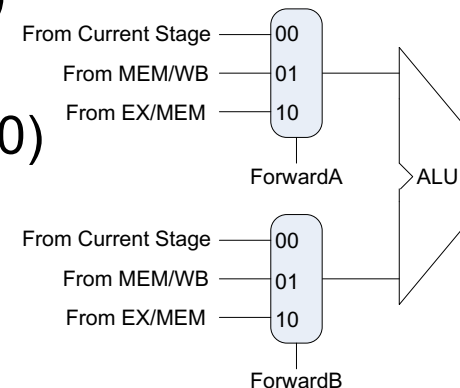
# Revised Forwarding Conditions

## 1&2 hazard

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd == ID/EX.RegisterRs))  
MUX select signal ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd == ID/EX.RegisterRt))  
MUX select signal ForwardB = 10

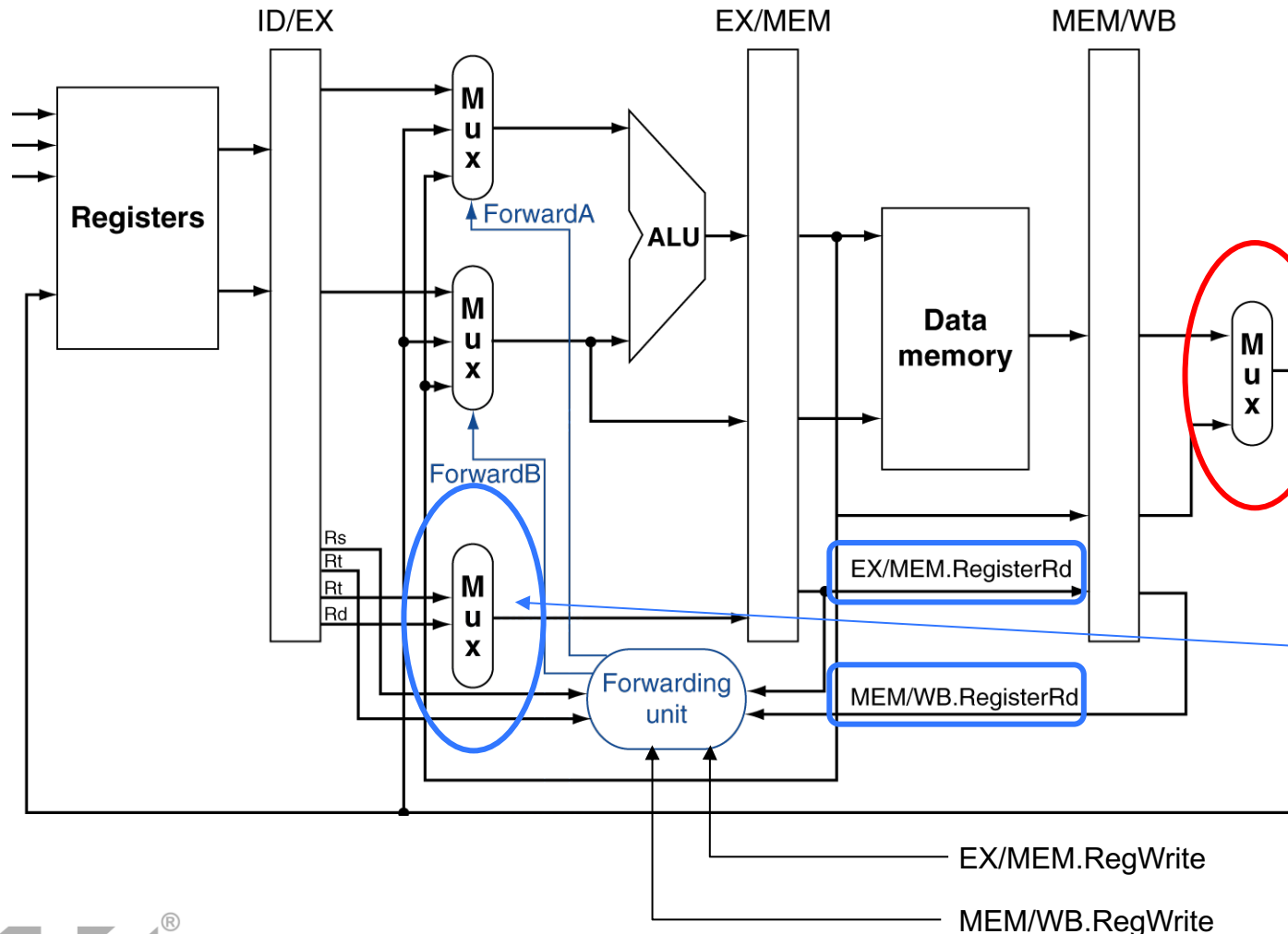
## 1&3 hazard

- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd == ID/EX.RegisterRs))  
MUX select signal ForwardA = 01
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd == ID/EX.RegisterRt))  
MUX select signal ForwardB = 01



# Forwarding Paths

## Forwarding Unit in **EX** stage



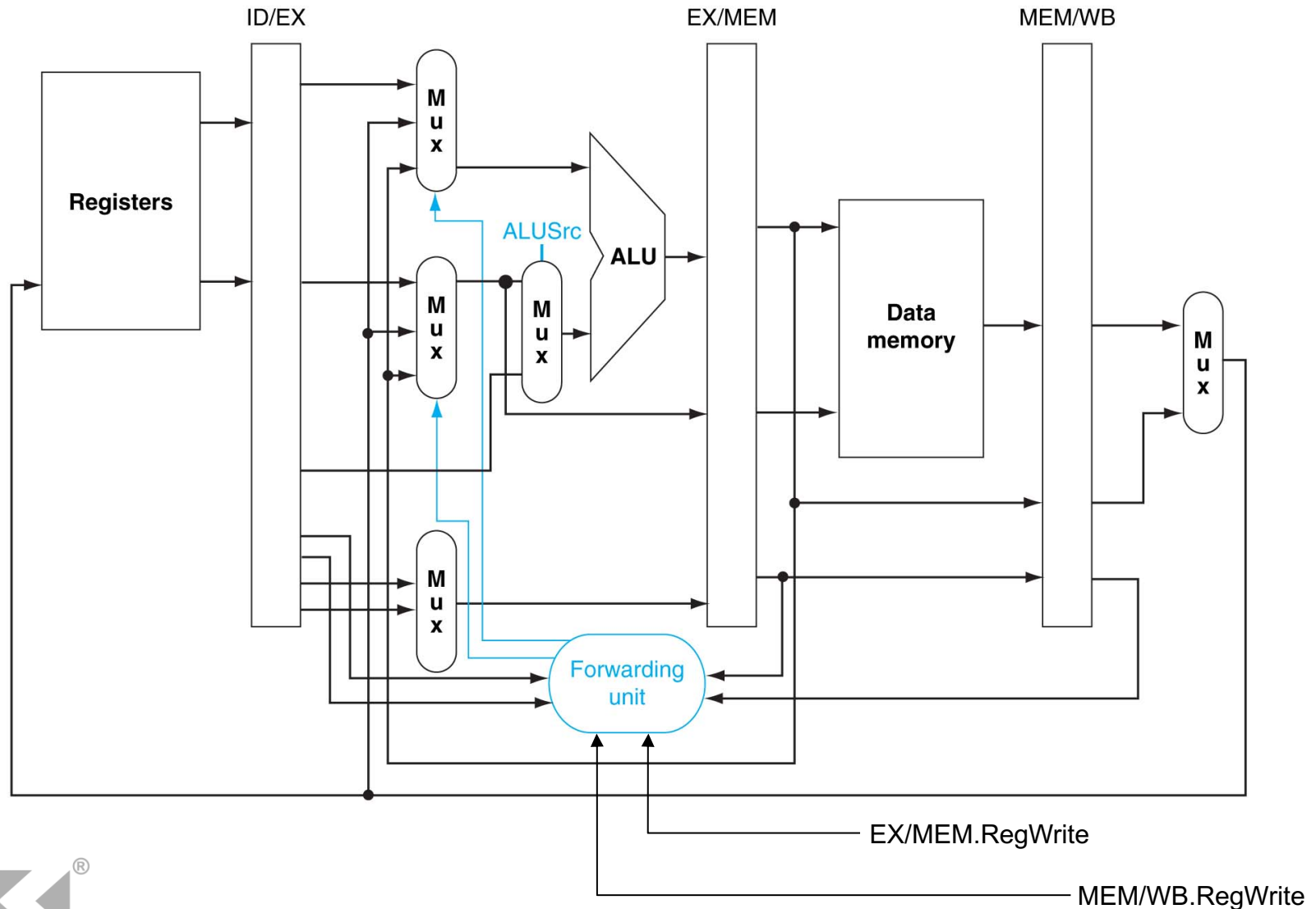
MEM/WB forwarding path actually comes from this MUX

EX/MEM.RegisterRd and  
MEM/WB.RegisterRd  
actually  
come from  
this MUX

RegisterRd should really be called RegisterDst



# Forwarding Paths with ALUSrc



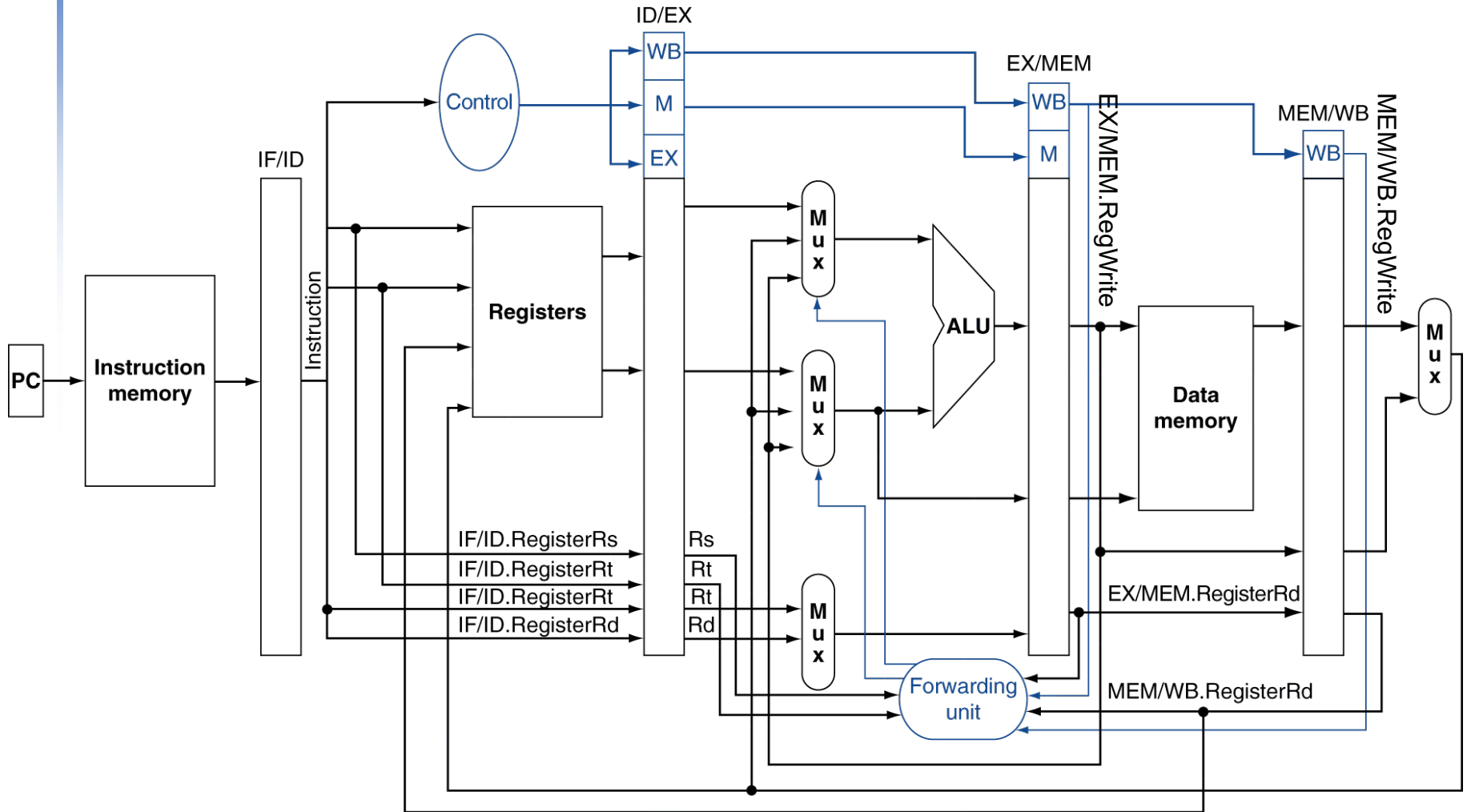
# Double Data Hazard

- Consider the sequence:
  - add \$1, \$1, \$2
  - add \$1, \$1, \$3
  - add \$1, \$1, \$4
- Conditions for both hazards are satisfied
  - Want to use the most recent
  - Forwarding between 1<sup>st</sup> and 3<sup>rd</sup> – MEM hazard condition should be disabled
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

# Revised Forwarding Condition

- 1&3 hazard (not 1&2 hazard)
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)  
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) )  
ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)  
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) )  
ForwardB = 01

# Datapath with Forwarding Unit



# Forwarding May Fail

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

# Load-Use Data Hazard

Program  
execution  
order  
(in instructions)

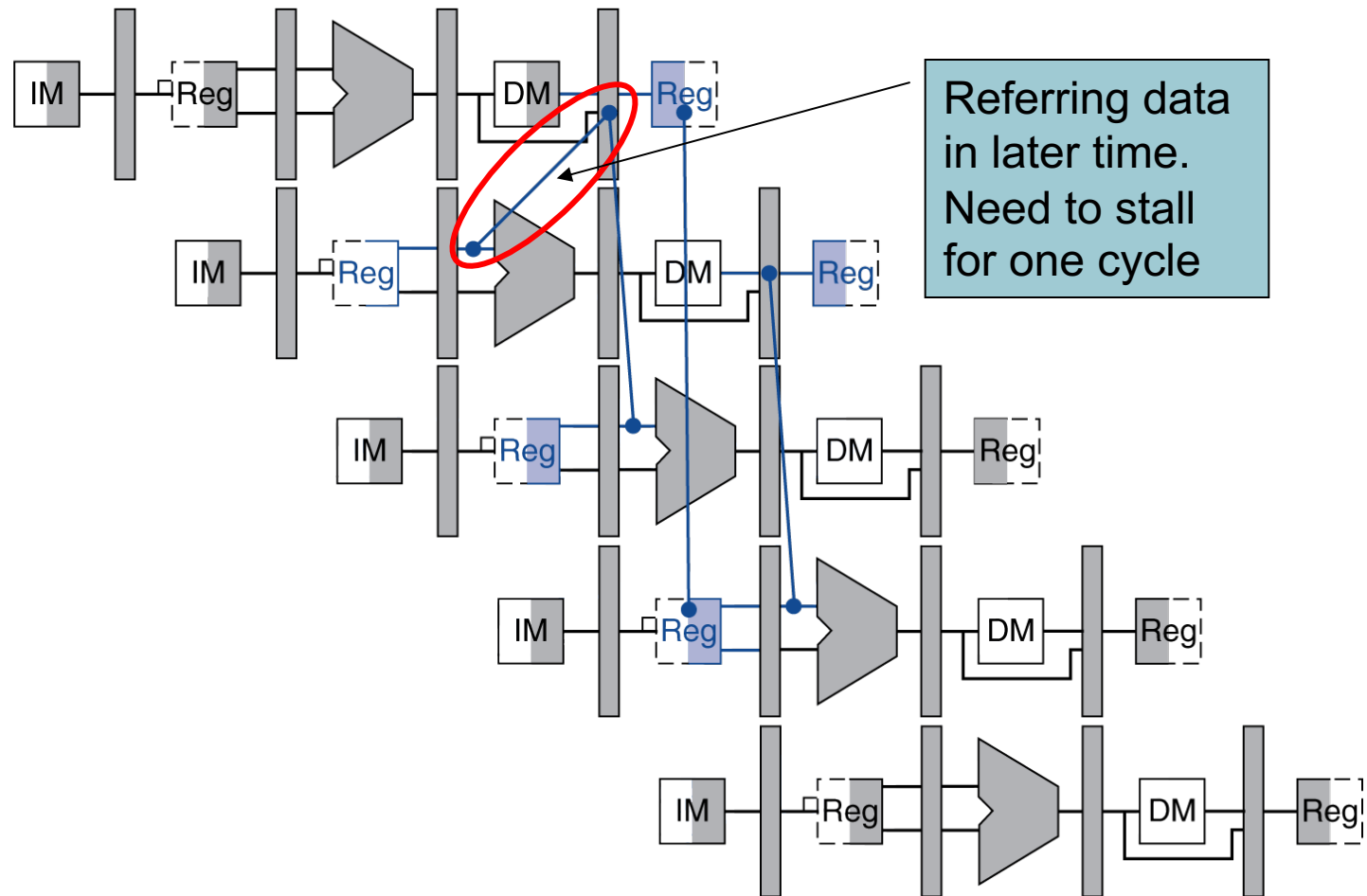
lw \$2, 20(\$1)

and \$4, \$2, \$5

or \$8, \$2, \$6

add \$9, \$4, \$2

slt \$1, \$6, \$7

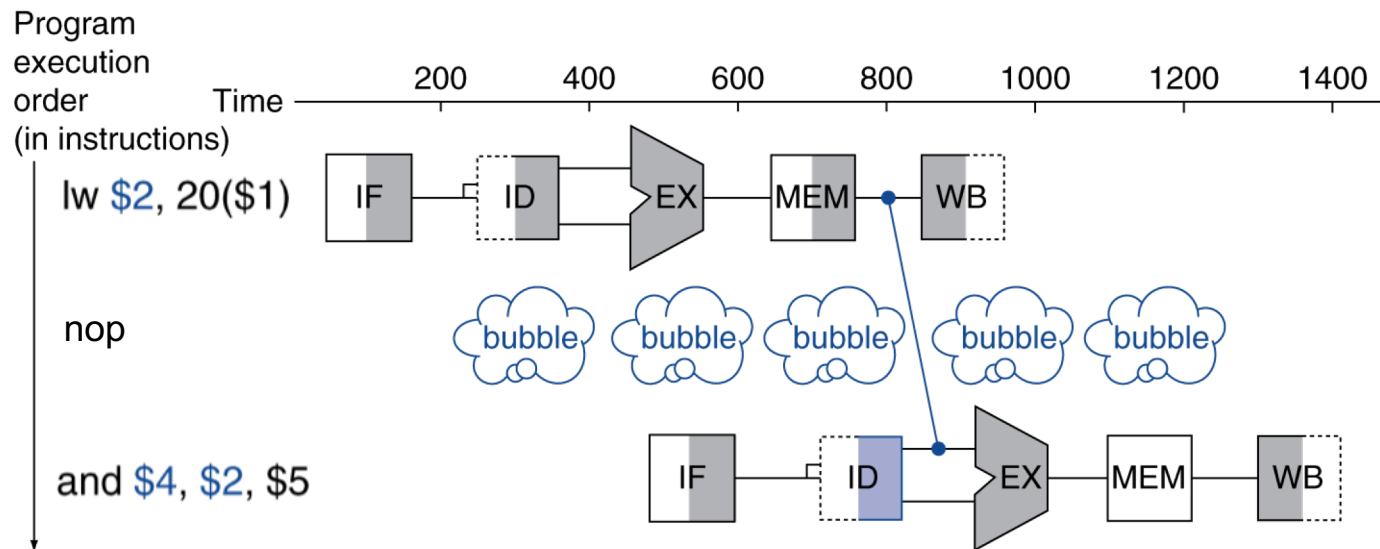


# Load-Use Hazard Detection

- Need to check only for a **load** instruction, determined by
  - ID/EX.MemRead
- Load-use hazard when
  - ID/EX.MemRead and  
((ID/EX.RegisterRt == IF/ID.RegisterRs) or  
(ID/EX.RegisterRt == IF/ID.RegisterRt))
- Hazard Detection is in **ID** stage
- If detected, stall and insert bubble

# Load-Use Data Hazard

- Resolve by inserting stalls

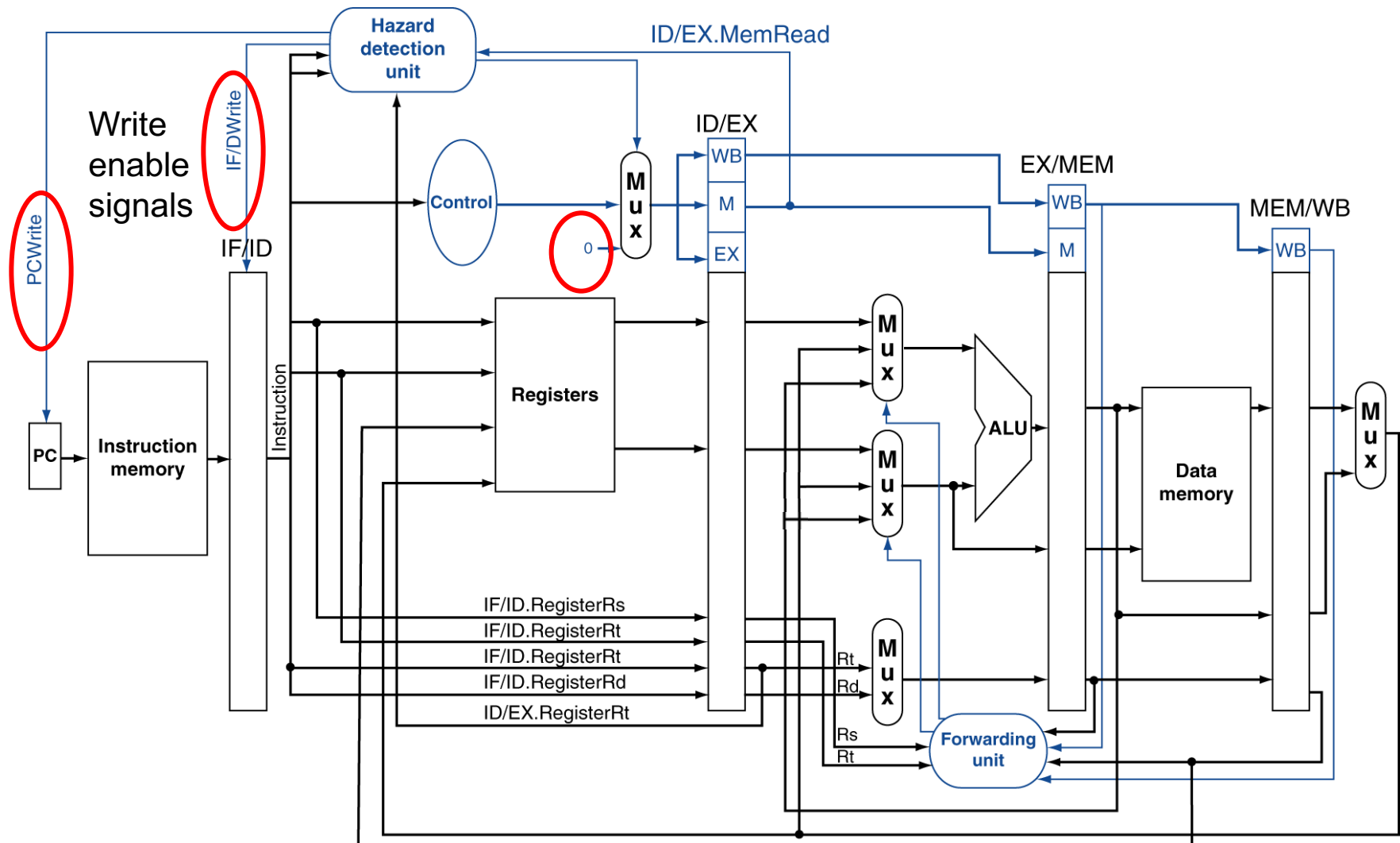




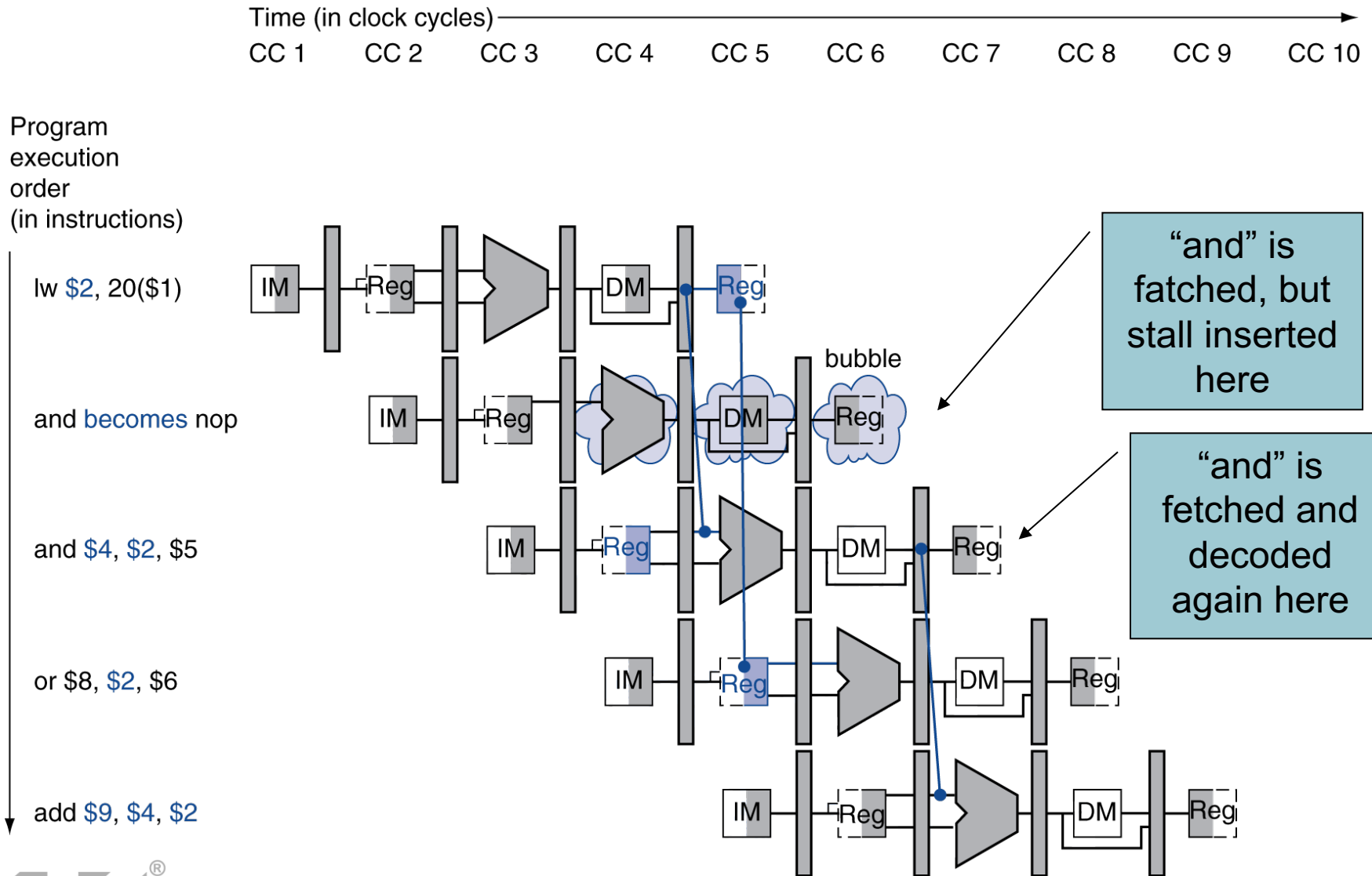
# How to Stall the Pipeline?

- (how to insert bubbles?)
- Force control signals in ID/EX register to 0's
  - EX, MEM and WB in following cycles do no-operation (nop) with those 0 control signals
- Prevent updates of PC and IF/ID registers
  - Instruction in IF/ID is held and decoded again
  - PC is held, so the same instruction is fetched again

# Datapath with Hazard Detection



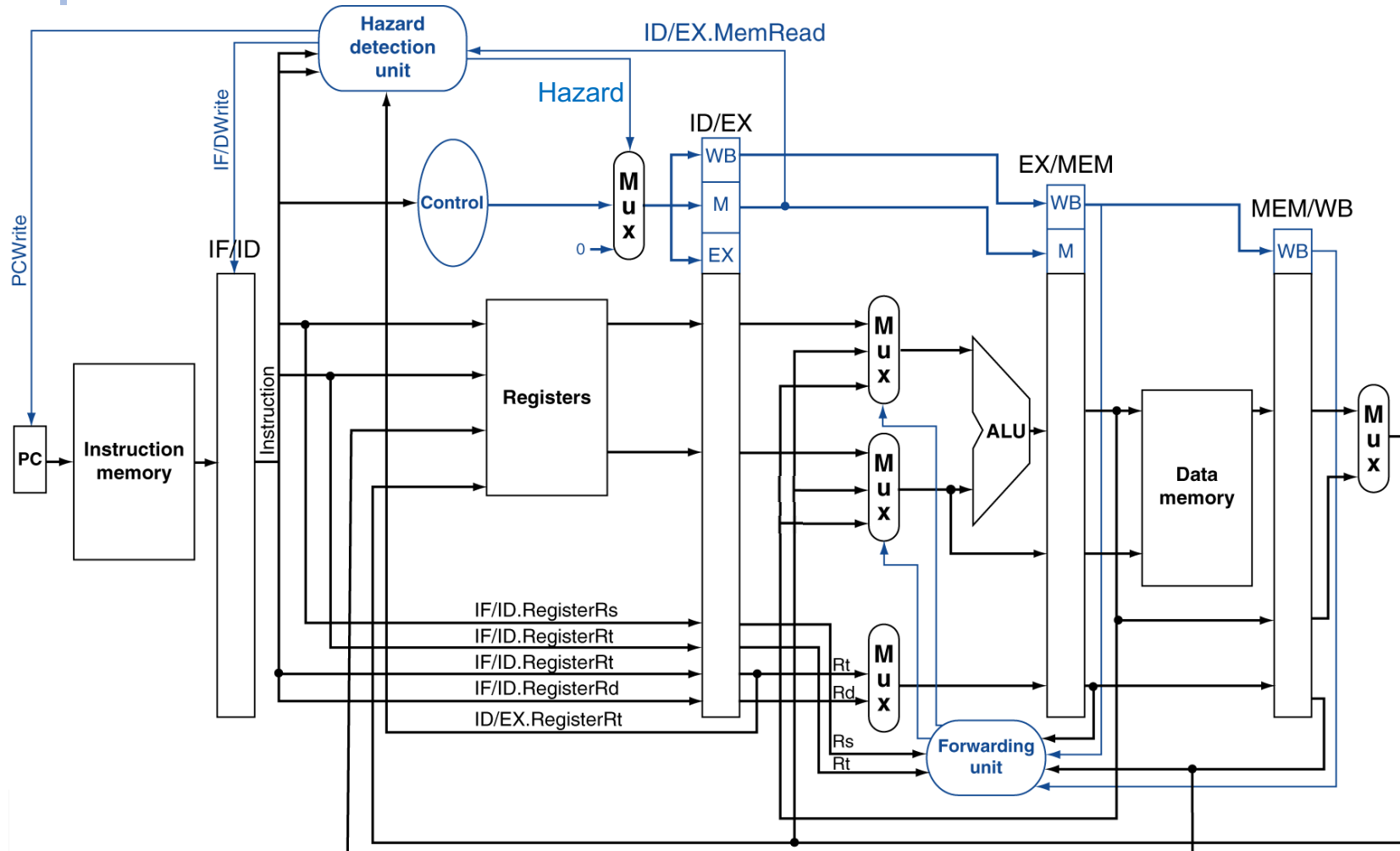
# Stall/Bubble in the Pipeline



# CC1

PC Write	IF/ID Write	Hazard	ID/EX. MemRead	IF/ID .Rs	IF/ID .Rt	ID/EX. Rt
1	1	X	X	X	X	X

lw \$2, 20(\$1)

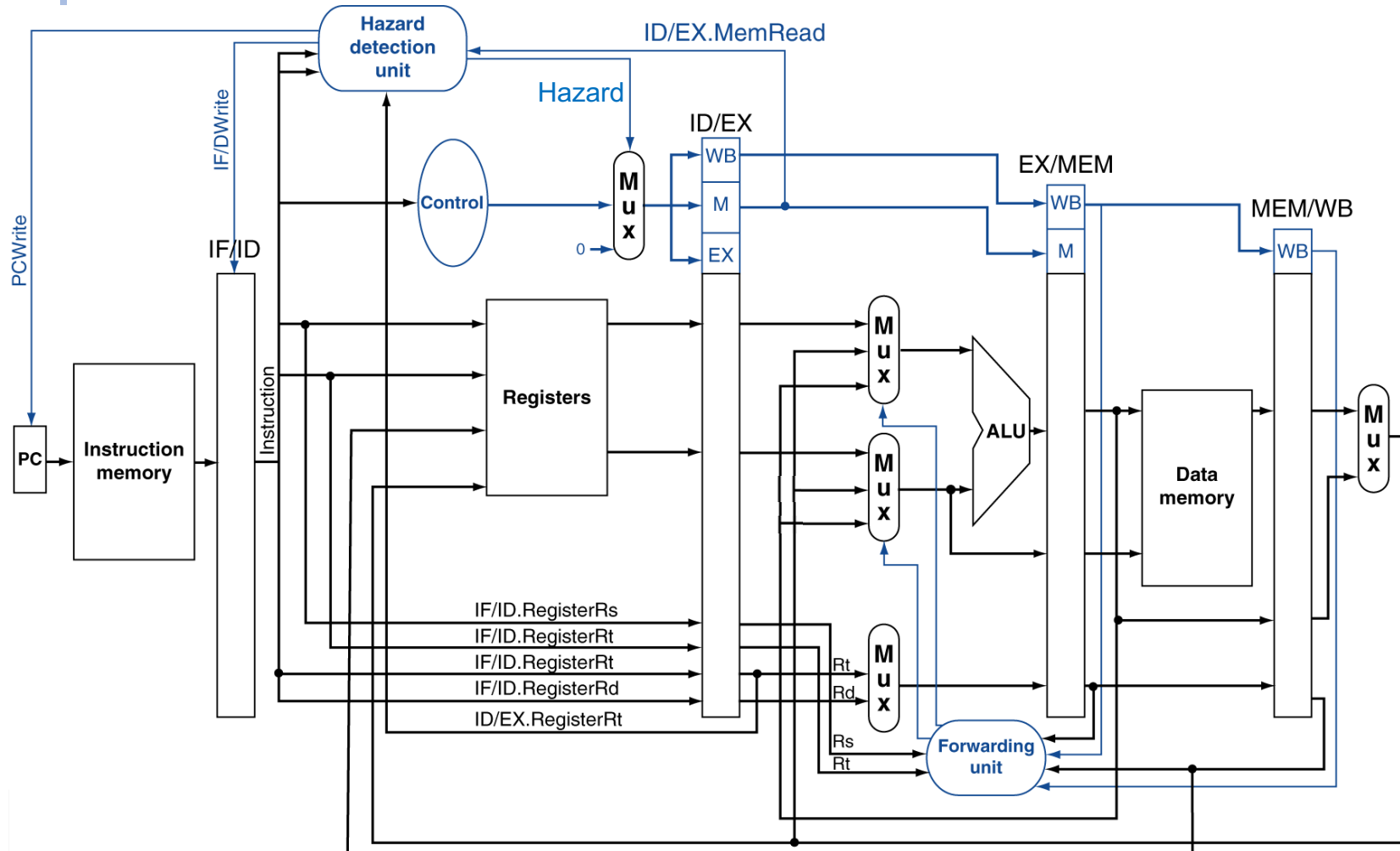


# CC2

PC Write	IF/ID Write	Hazard	ID/EX. MemRead	IF/ID .Rs	IF/ID .Rt	ID/EX. Rt
1	1	x	x	1	2	x

and \$4,\$2,\$5

lw \$2, 20(\$1)



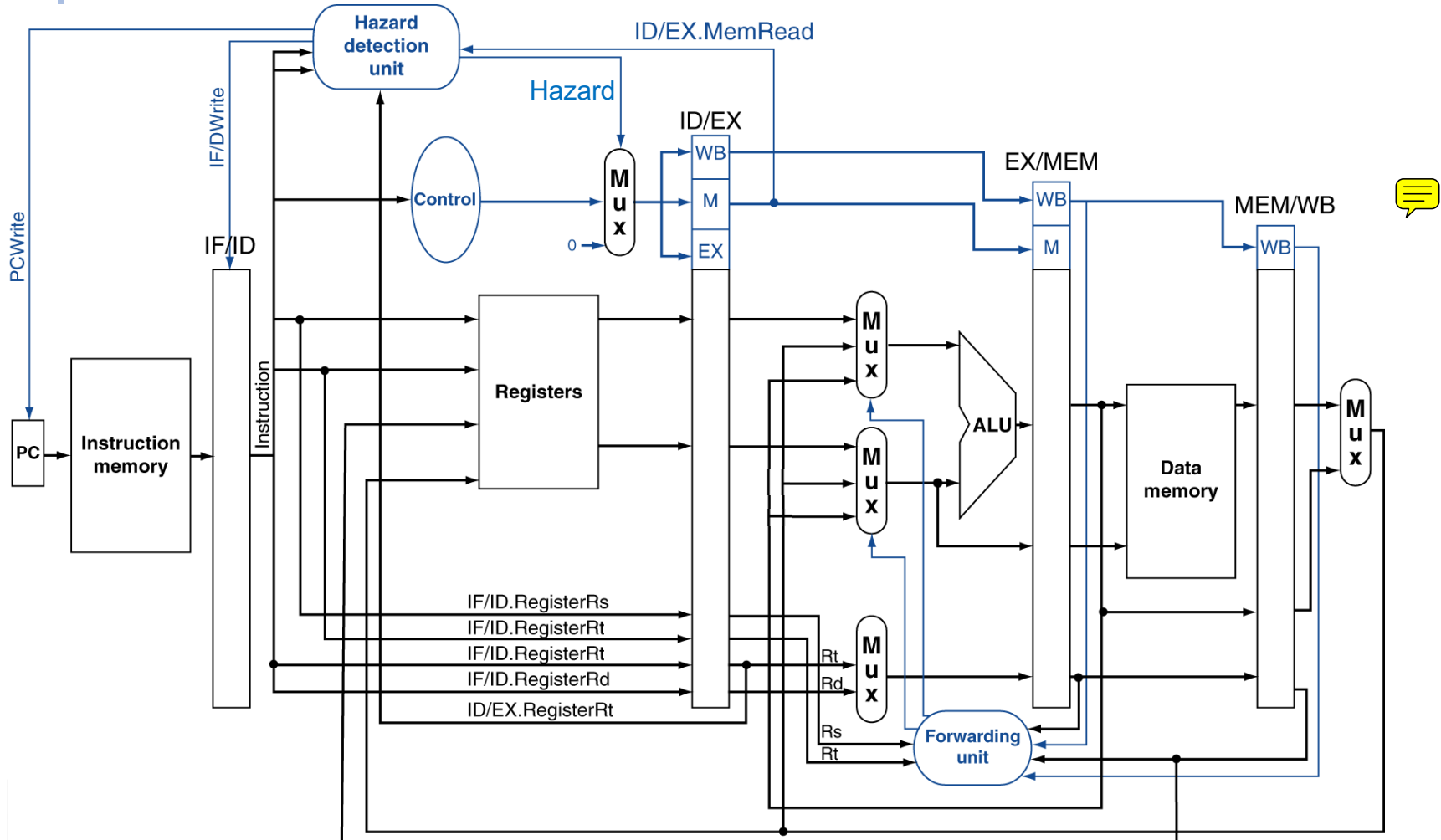
# CC3

PC Write	IF/ID Write	Hazard	ID/EX. MemRead	IF/ID .Rs	IF/ID .Rt	ID/EX. Rt
0	0	1	1	2	5	2

or \$8,\$2,\$6

and \$4,\$2,\$5

lw \$2, 20(\$1)



# CC4

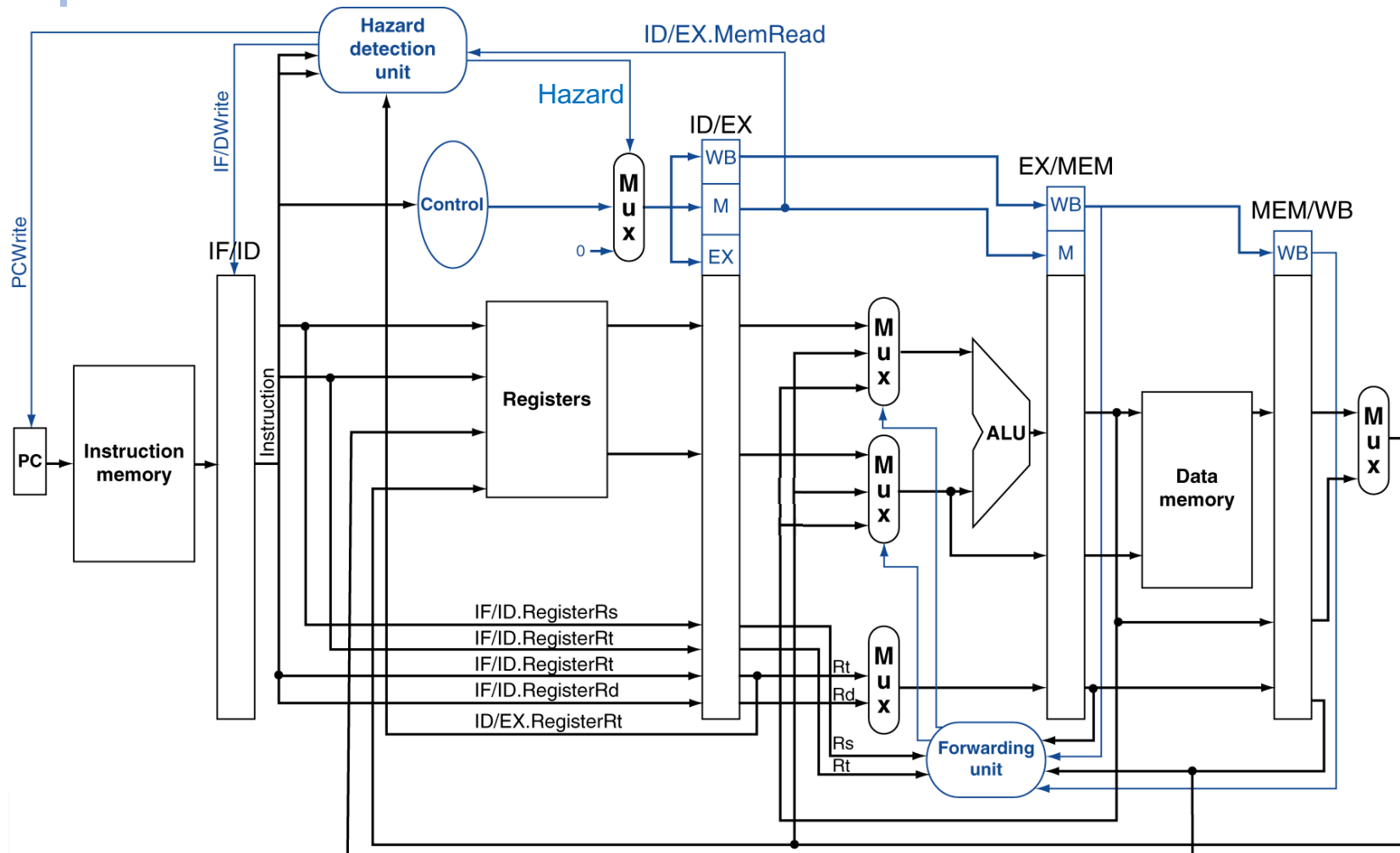
PC Write	IF/ID Write	Hazard	ID/EX. MemRead	IF/ID .Rs	IF/ID .Rt	ID/EX. Rt
1	1	0	0	2	5	5

or \$8,\$2,\$6

and \$4,\$2,\$5

nop

lw \$2, 20(\$1)



ID/EX.Rs	ID/EX.Rt	ID/EX.Dst	EX/MEM.Dst	MEM/WB.Dst
2	5	5	2	x
EX/MEM.RegWrite	MEM/WB.RegWrite	ForwardA	ForwardB	
1	x	10	00	

and \$4,\$2,\$5

lw \$2, 20(\$1)

This diagram illustrates the internal architecture of a 5-stage MIPS processor, focusing on hazard detection and forwarding mechanisms. The stages are Instruction (IF), Decode (ID), Execute (EX), Memory (MEM), and Writeback (WB).

- PC and Instruction Memory:** The Program Counter (PC) provides the address for Instruction Memory. The PCWrite signal is used to update the PC.
- IF/ID Stage:** The Instruction Memory outputs the instruction to the IF/ID stage. This stage provides the instruction to the Hazard Detection Unit and the Control unit. It also outputs register identifiers (IF/ID.RegisterRs, IF/ID.RegisterRt, IF/ID.RegisterRd) to the Register File.
- Hazard Detection Unit:** This unit checks for data hazards by comparing the register file with the register identifiers from the IF/ID stage. It outputs a Hazard signal to the Mux in the ID/EX stage.
- ID/EX Stage:** The instruction is decoded, and the register file is updated with the register identifiers (ID/EX.RegisterRt). The Hazard signal is used by a Mux to select between the register file output and a forwarded value.
- EX/MEM Stage:** The ALU performs operations on the register values. The ALU result is forwarded to the MEM/WB stage via a Mux. The MEM stage accesses Data Memory. The MEM/WB stage outputs the result to the Mux in the WB stage.
- Forwarding Unit:** This unit receives the ALU result and the register identifiers from the IF/ID stage. It outputs a forwarded value to the Mux in the ID/EX stage.
- Writeback (WB) Stage:** The result from the MEM/WB stage is written back to the Register File via the Mux.



# CC5

PC Write	IF/ID Write	Hazard	EX/MEM. MemRead	EX/MEM. MemWrite
1	1	0	0	0

No effective operation

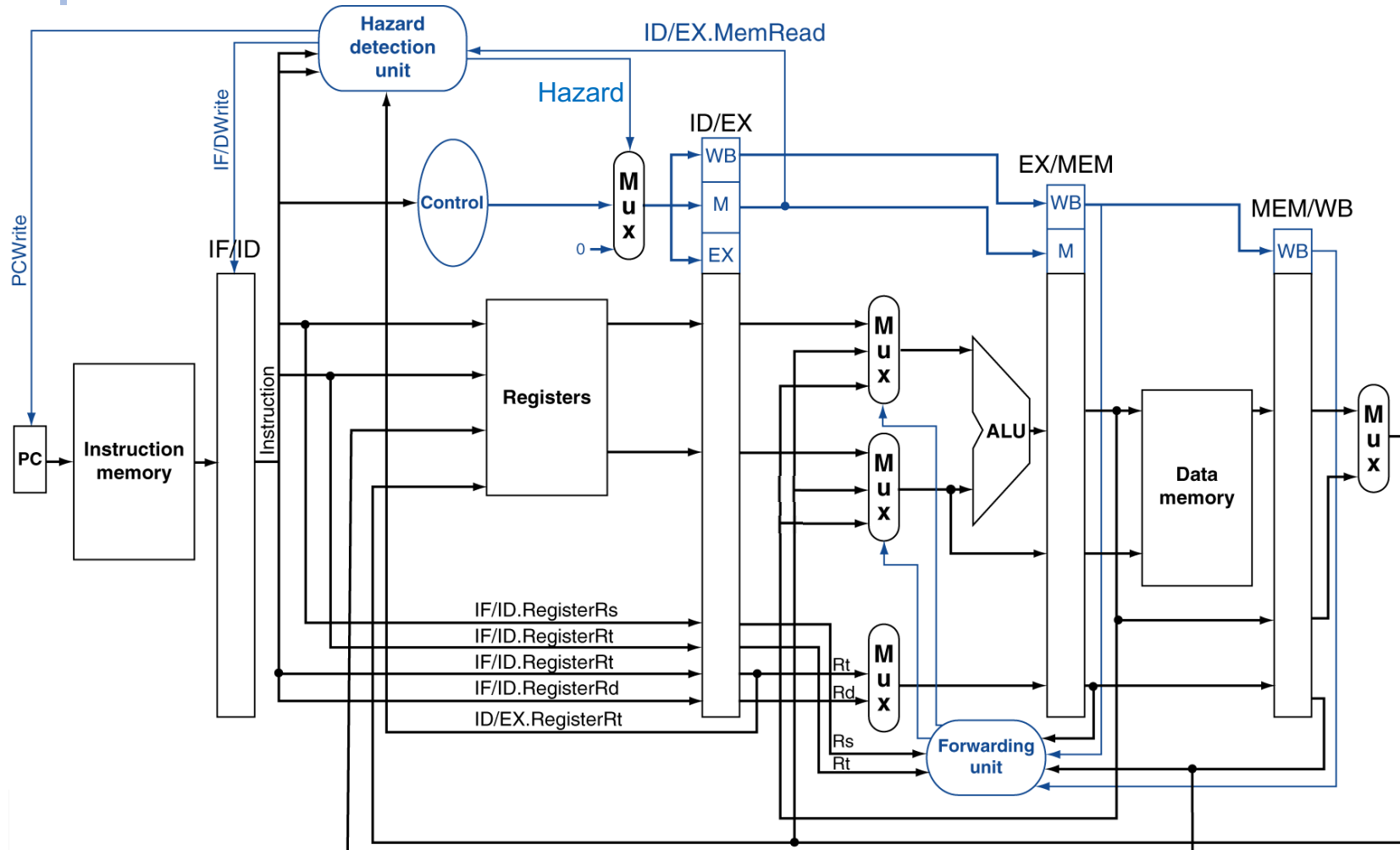
add \$9,\$4,\$2

or \$8,\$2,\$6

and \$4,\$2,\$5

**nop**

lw \$2, 20(\$1)



# CC5

ID/EX.Rs	ID/EX.Rt	ID/EX.Dst	EX/MEM.Dst	MEM/WB.Dst
2	5	4	5	2
EX/MEM.RegWrite	MEM/WB.RegWrite	ForwardA	ForwardB	
0	1	01	00	

add \$9,\$4,\$2

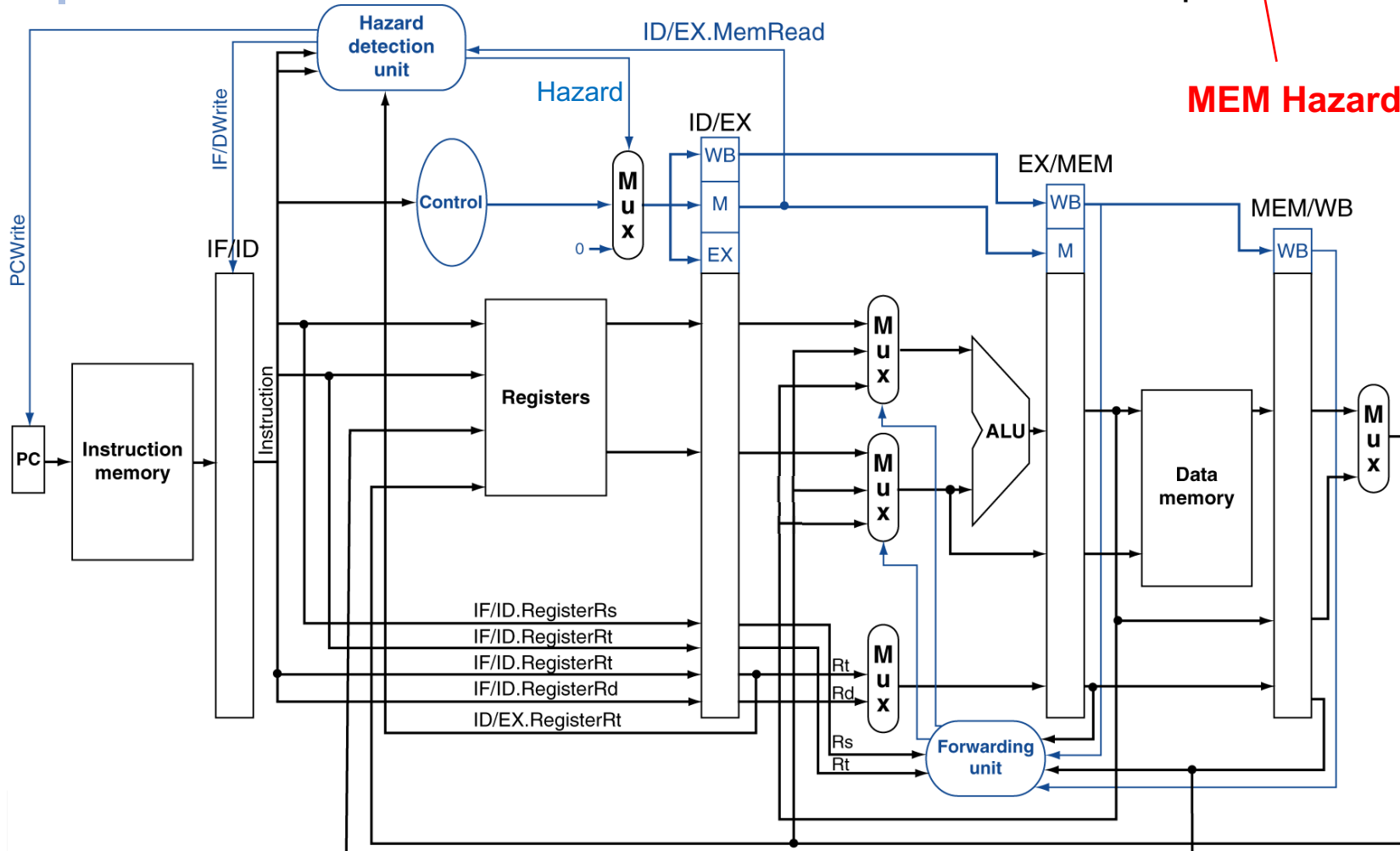
or \$8,\$2,\$6

and \$4,\$2,\$5

nop

lw \$2, 20(\$1)

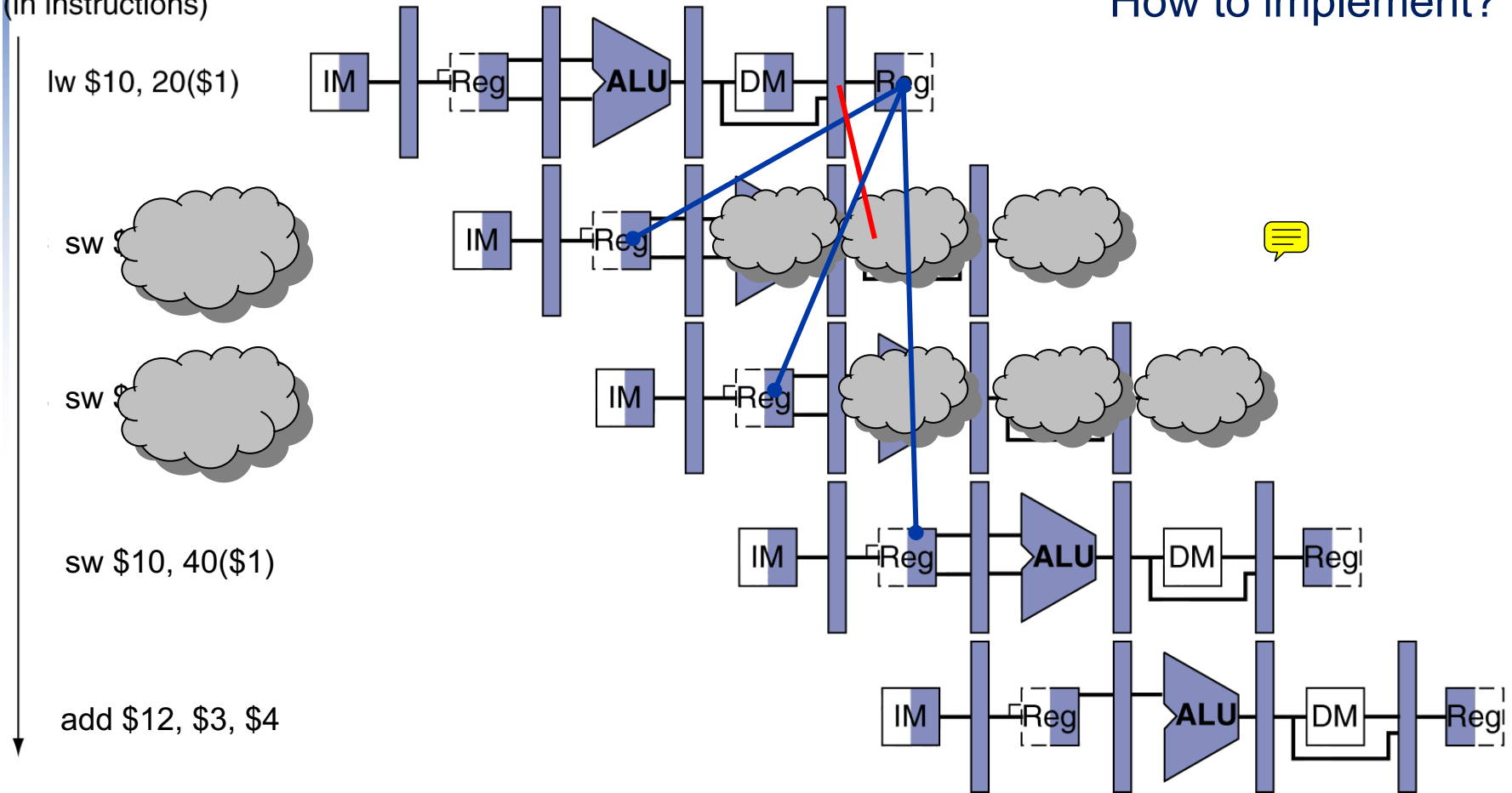
MEM Hazard



# Question

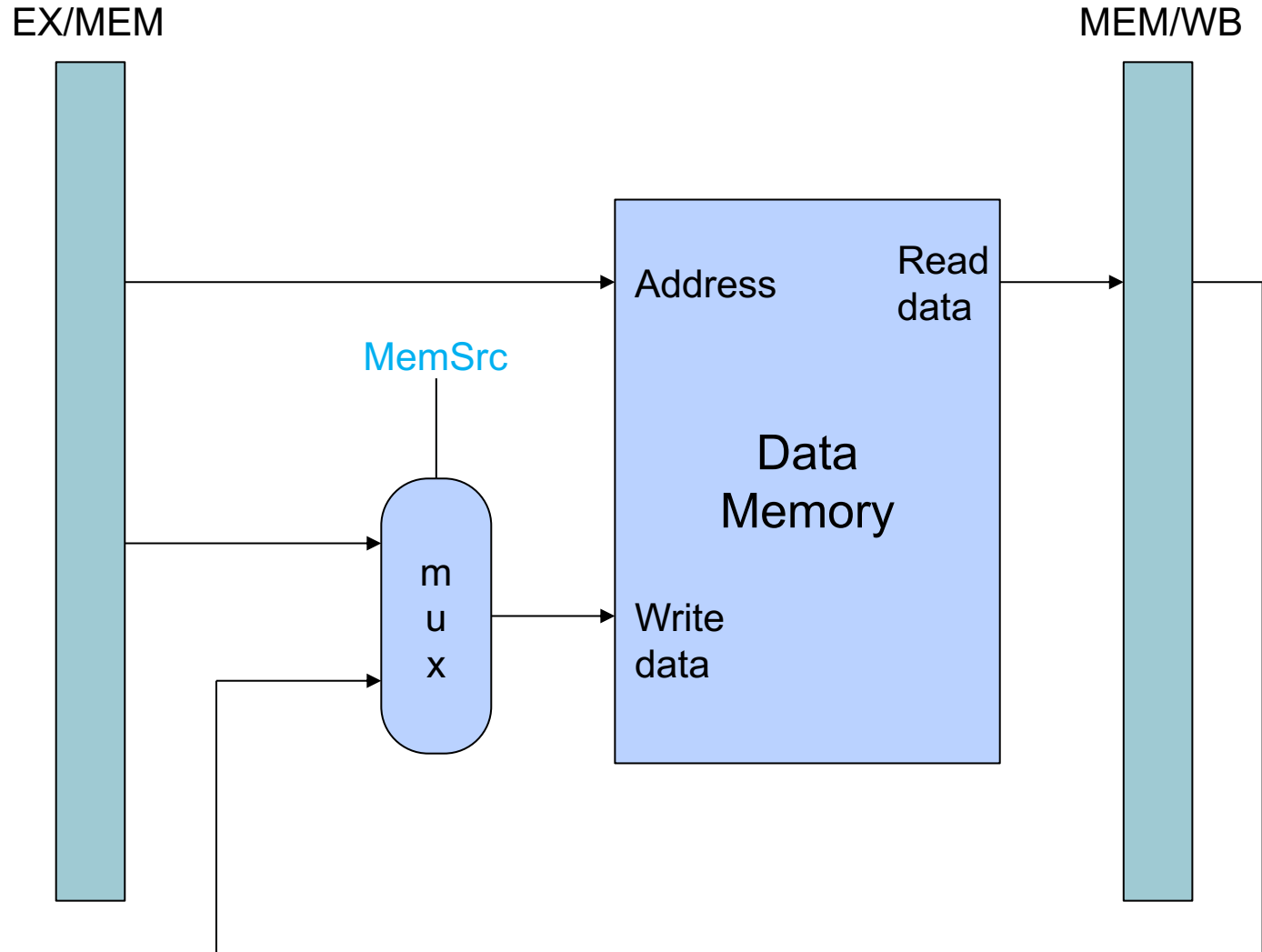
(in instructions)

How to implement?




This delays the execution too much, how to fix?

# Question: how to implement

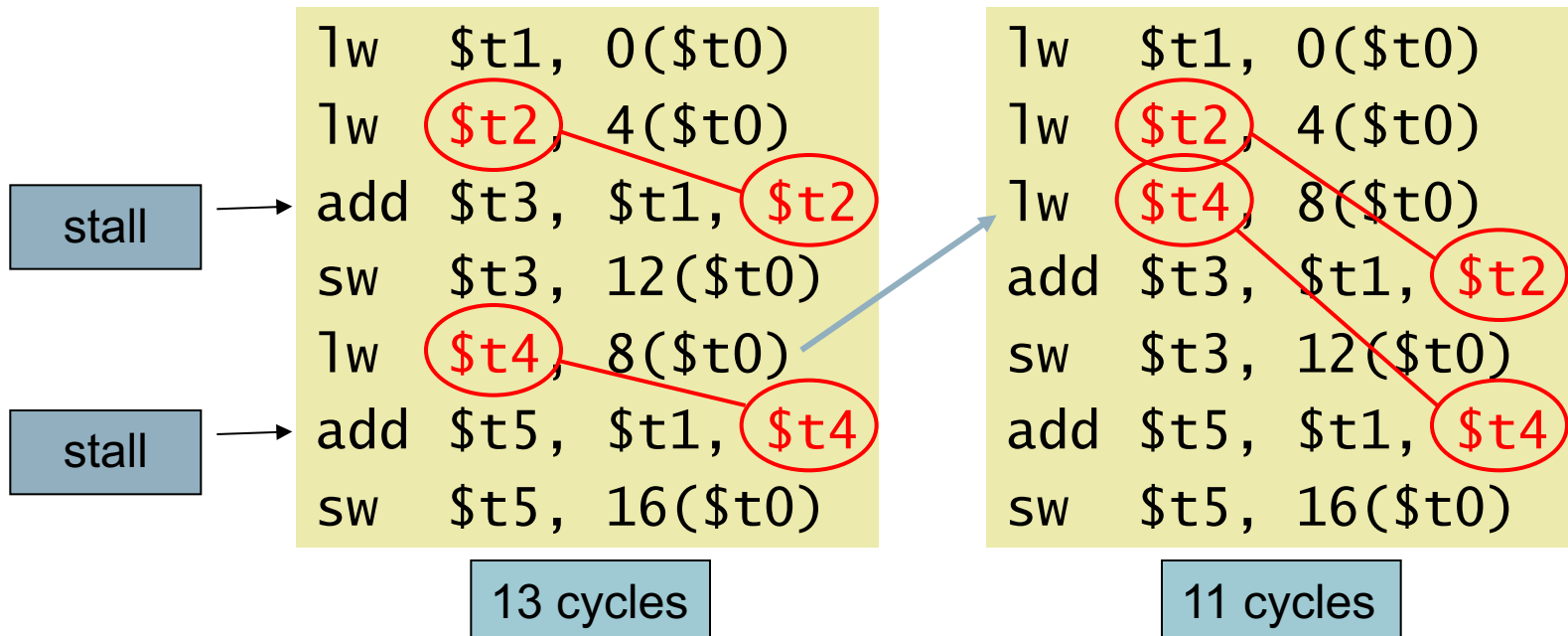


# Question: how to decide MemSrc?

- MemSrc = 1 when  
(MEM/WB.Rt == EX/MEM.Rt) and  
MEM/WB.MemRead (already used) and  
EX/MEM.MemWrite 

# Code Scheduling to Avoid Stalls

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



# Structure Hazards

- Conflict for use of a resource
- If common memory for program and data
  - 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
  - separate instruction/data caches (Harvard)
  - Common low-level memory (Von Neumann)



# Stalls and Performance

## The BIG Picture

- Stalls reduce performance
  - But are required to get correct results
- Hardware can be improved to take care of the hazards automatically
  - forwarding
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure