CMPE 411
Computer Architecture

Lecture 15

Handling Pipeline Hazards

October 19, 2017

www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm

Lecture's Overview

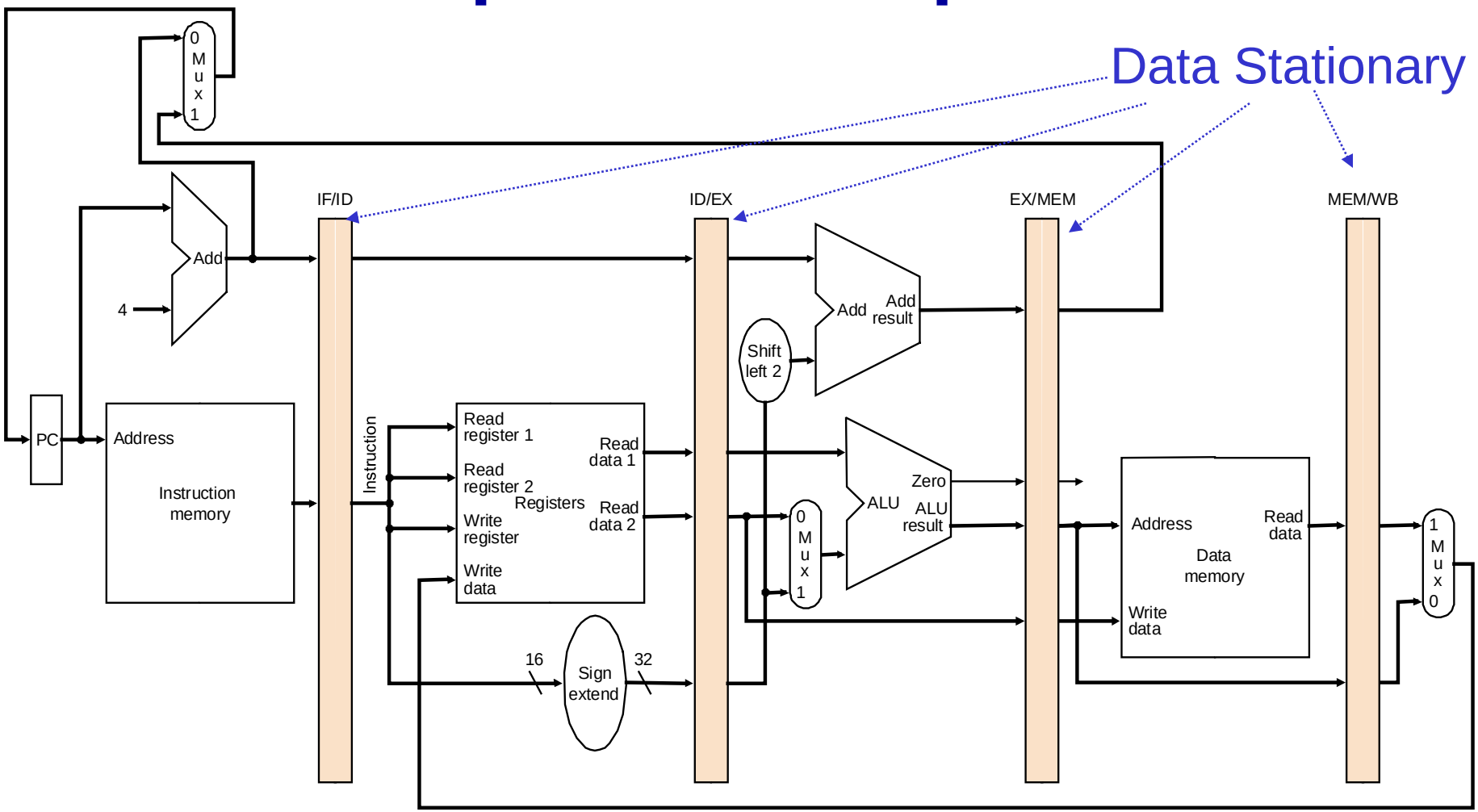
❑ *Previous Lecture:*

- Designing a pipelined datapath
  - ➔ Standardized multi-stage instruction execution
  - ➔ Unique resources per stage
- Controlling pipeline operations
  - ➔ Simplified stage-based control
  - ➔ Detailed working example

❑ *This Lecture:*

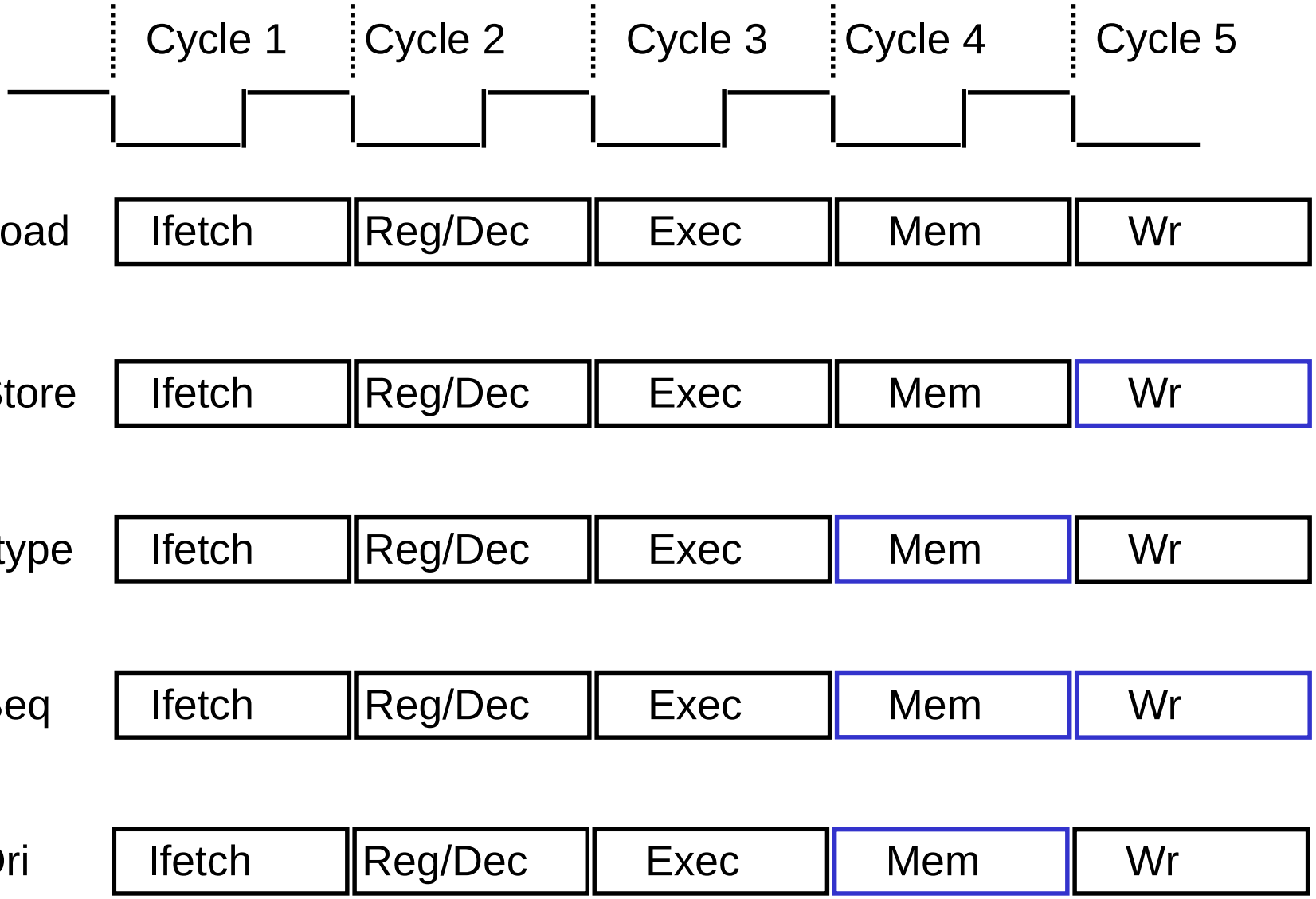
- Pipeline hazard detection
- Handling hazard in the pipeline design
- Supporting exceptions

Pipelined Datapath



❶ ❷ ❸ ❹ ❺

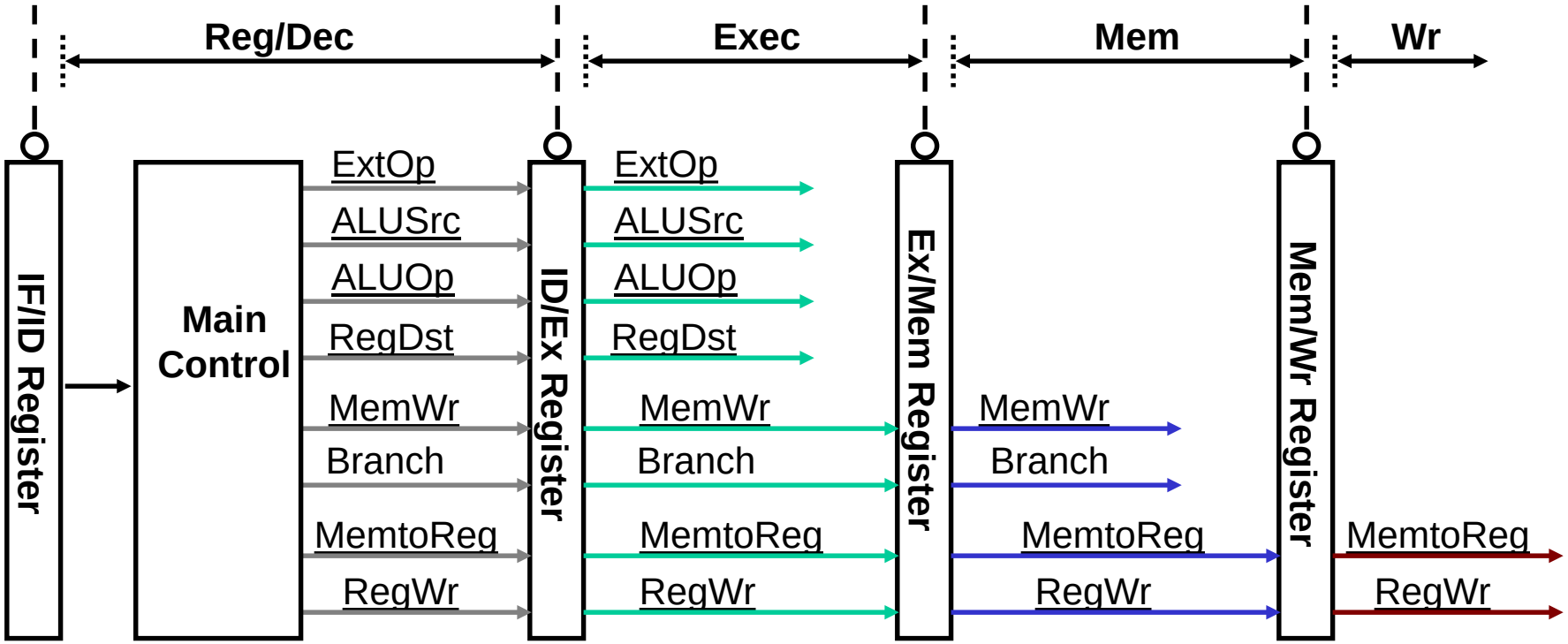
Standardized Five Stages Instructions



Data Stationary Control

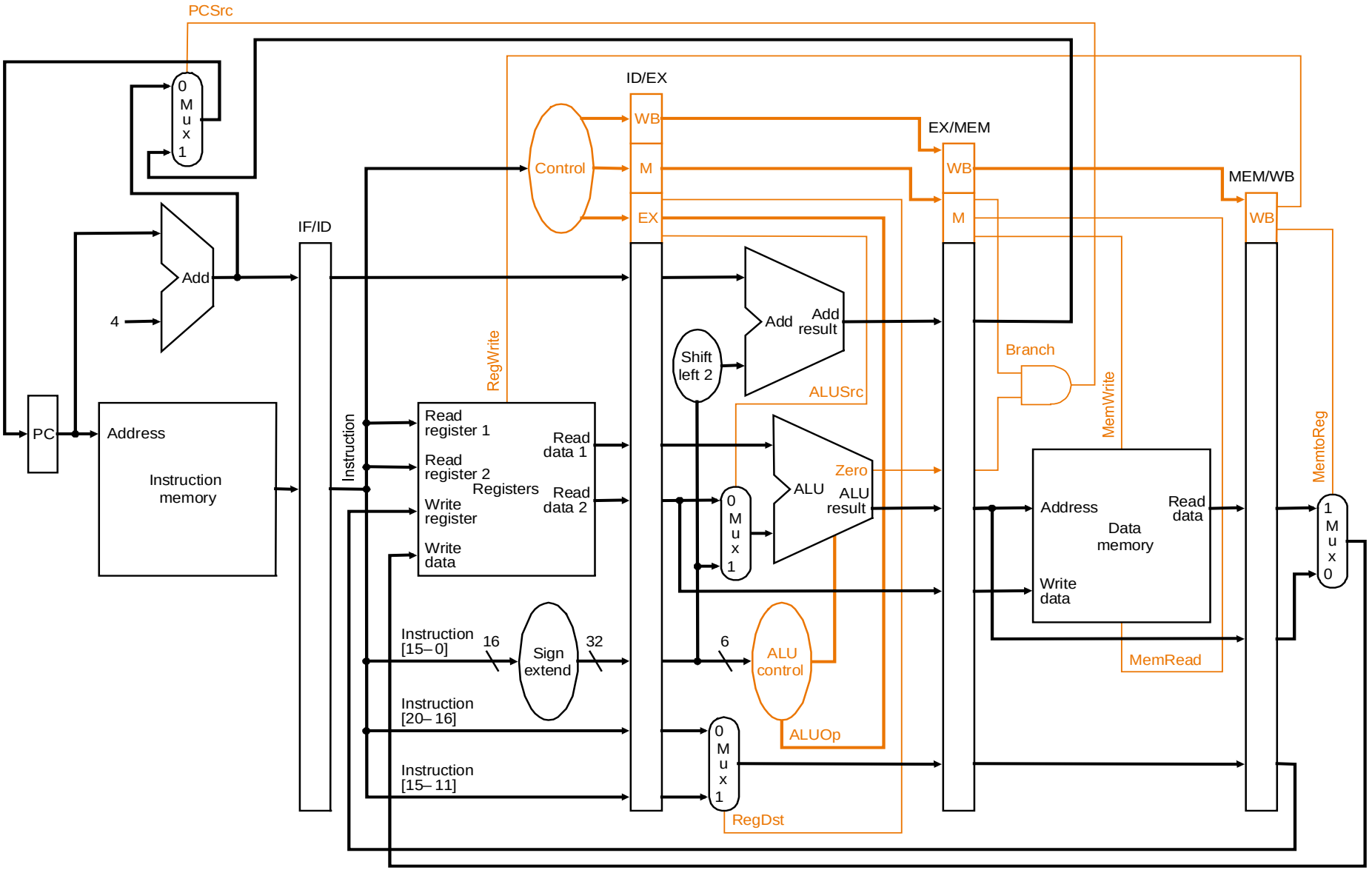
The Main Control generates the control signals during Reg/Dec

- ➔ Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
- ➔ Control signals for Mem (MemWr Branch) are used 2 cycles later
- ➔ Control signals for Wr (MemtoReg RegWr) are used 3 cycles later

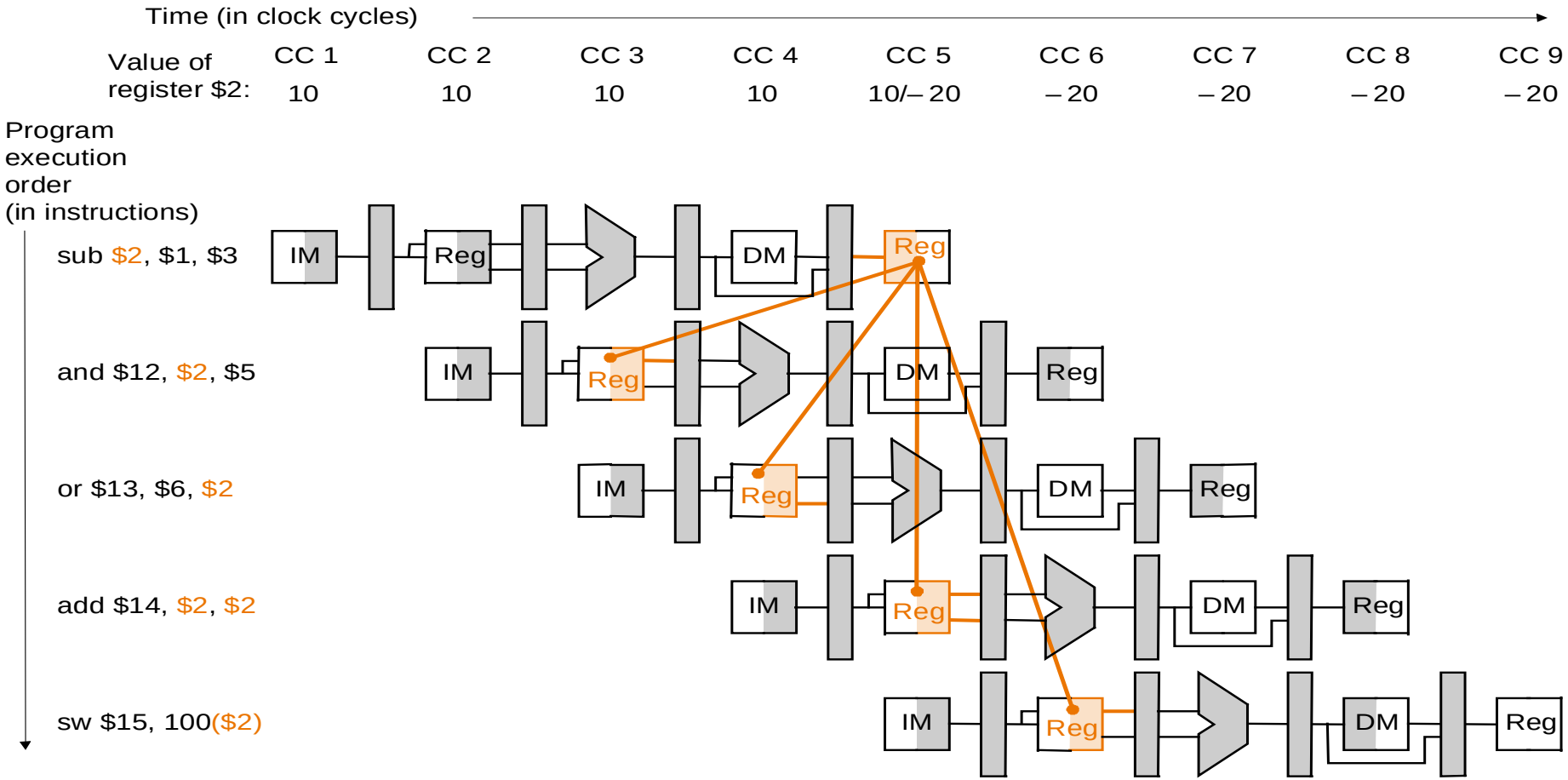


* Slide is courtesy of Dave Patterson

Datapath + Data Stationary Control

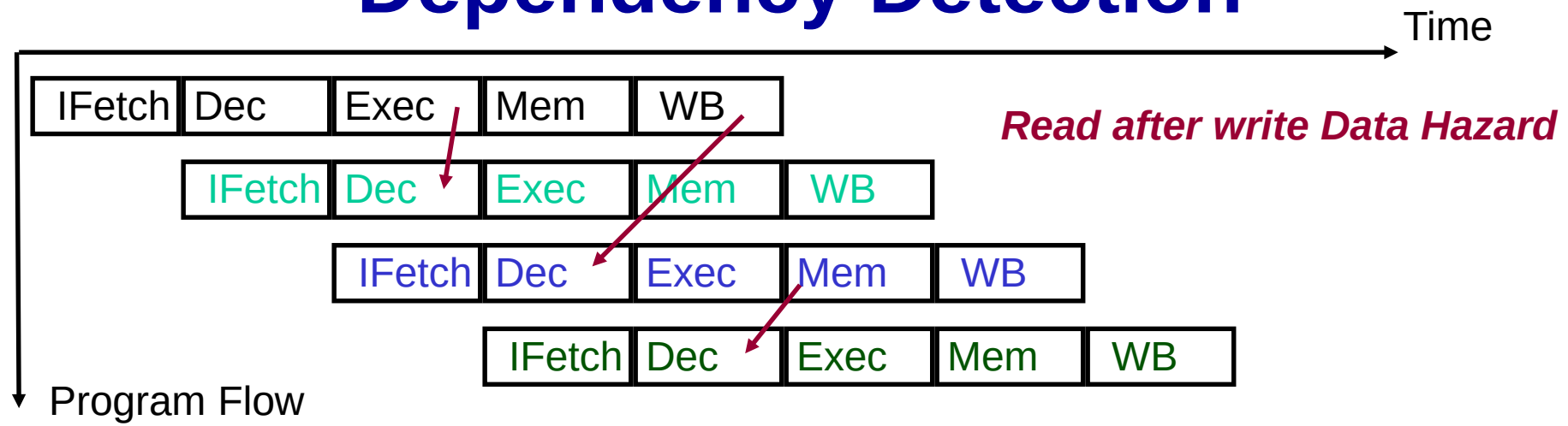


Data Hazards



- Generally caused by data dependence among instructions
- Can cause erroneous semantics if went undetected and avoided
- Raised when an instruction depends on result of a prior instruction still in the pipeline and attempts to use item before it is ready

Dependency Detection



- Data Hazards is caused by backward dependency at the DEC stage
- A hazardous register is to be updated at an earlier instruction at the EX, MEM or WB stages of the pipeline
- It is assumed that register writing takes precedence over reading if happened in the same clock cycle ⇒ WB stages will not cause data hazard

Data Hazard conditions:

1a. EX/MEM.Register-Rd = ID/EX.Register-Rs

1b. EX/MEM.Register-Rd = ID/EX.Register-Rt

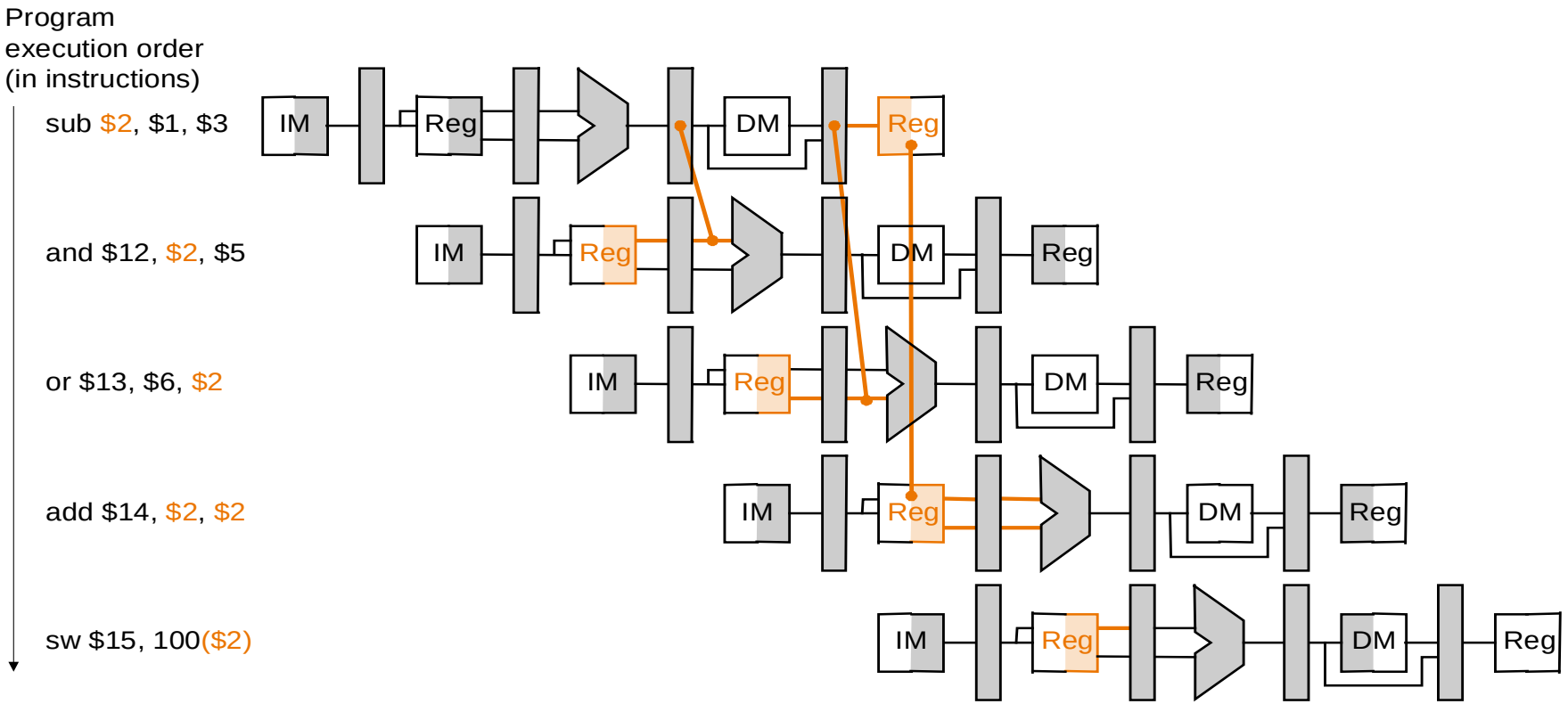
2a. MEM/WB.Register-Rd = ID/EX.Register-Rs

2b. MEM/WB.Register-Rd = ID/EX.Register-Rt

Data Forwarding

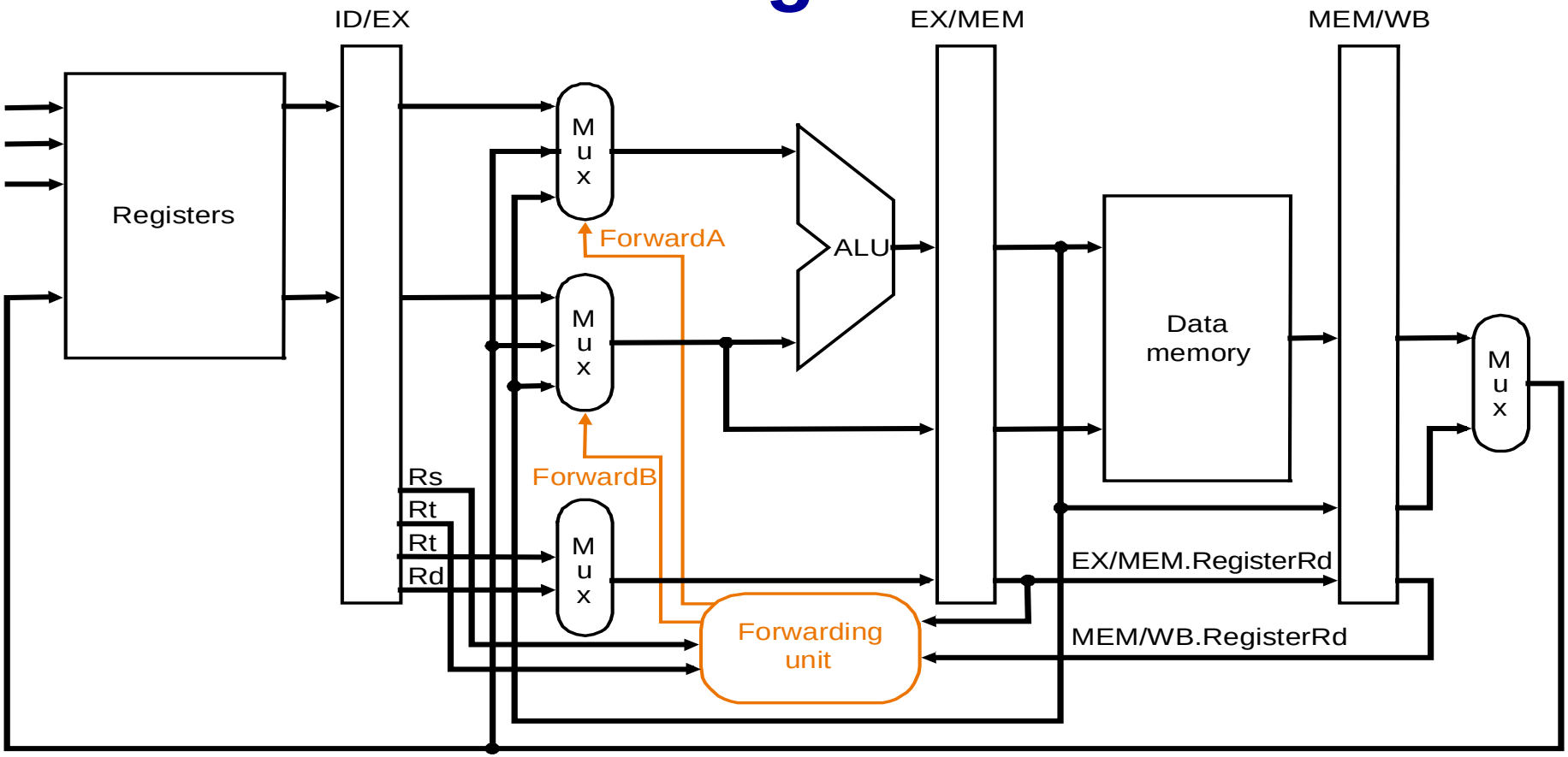
Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



- Detecting data hazard conditions allows using intermediate results
- Forward only when there is a WB stage (check the RegWrite signal)

Forwarding Hardware



EX Hazard:

if (EX/MEM.RegWrite & (EX/MEM.Register-Rd = ID/EX.Register-Rs)) ForwardA = EX/MEM

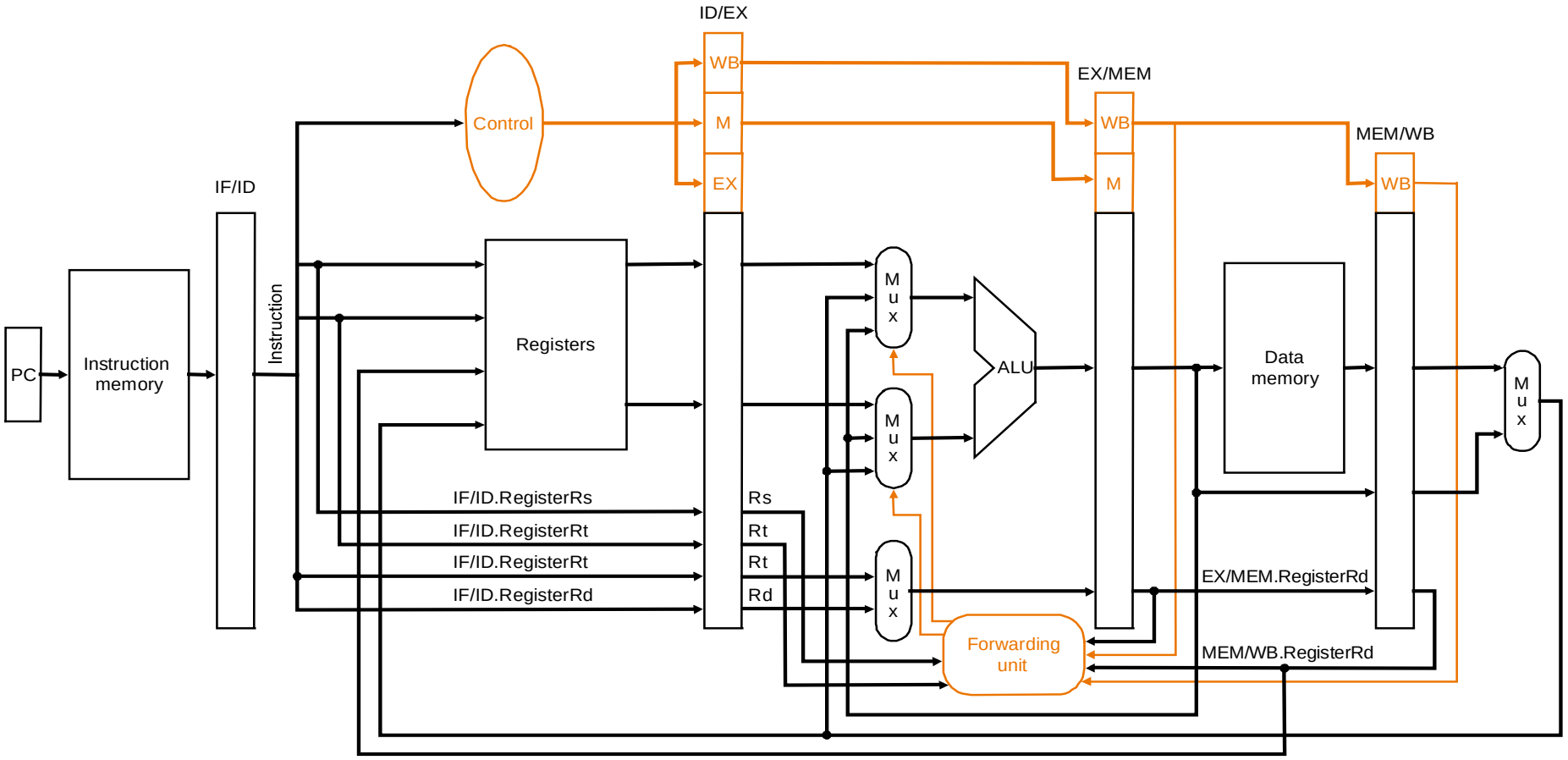
if (EX/MEM.RegWrite & (EX/MEM.Register-Rd = ID/EX.Register-Rt)) ForwardB = EX/MEM

MEM Hazard:

if (MEM/WB.RegWrite & (MEM/WB.Register-Rd = ID/EX.Register-Rs)) ForwardA = MEM/WB

if (MEM/WB.RegWrite & (MEM/WB.Register-Rd = ID/EX.Register-Rt)) ForwardB = MEM/WB

Forwarding Register of Successive Use



Example:

add $1, $1, $2;

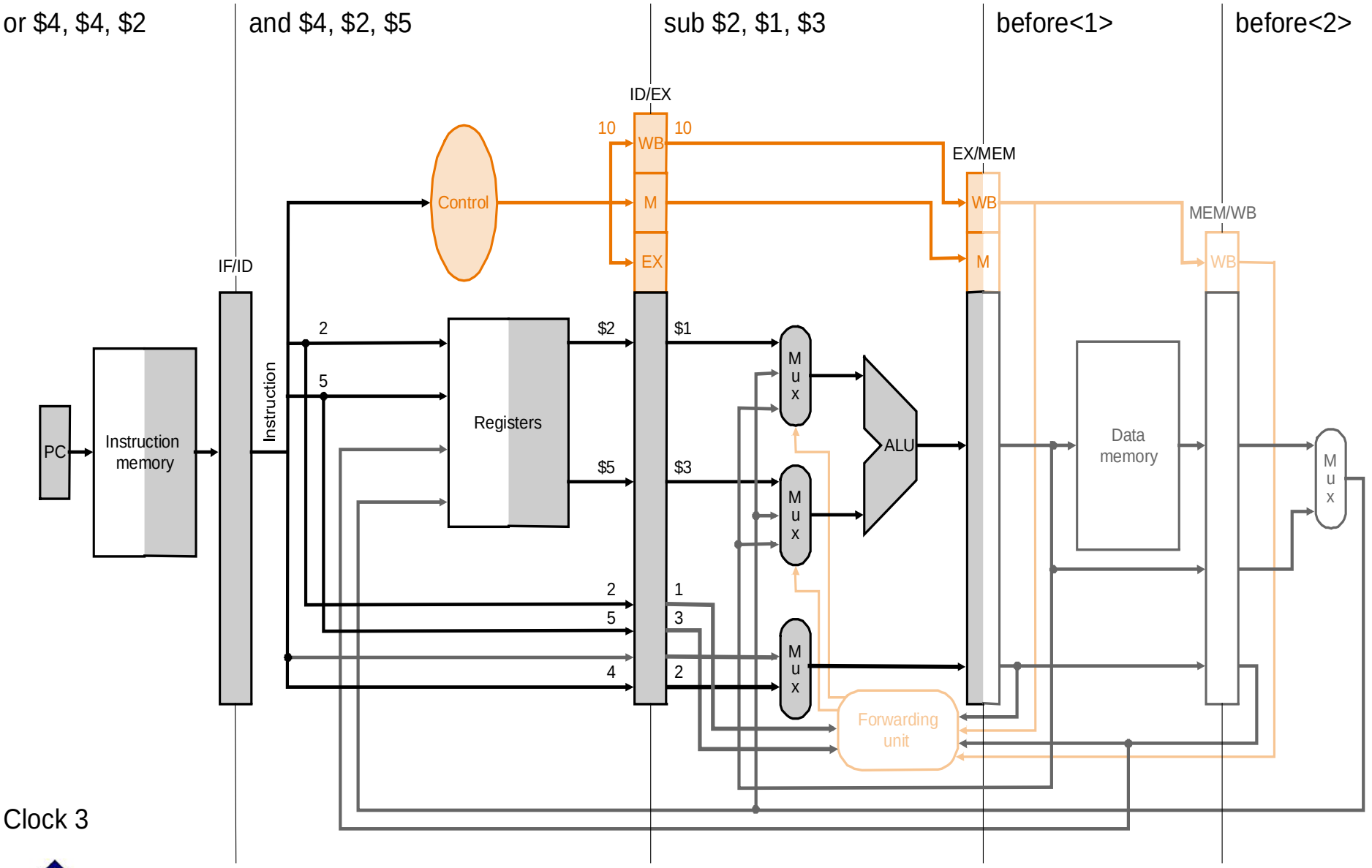
add $1, $1, $3;

add $1, $1, $4;

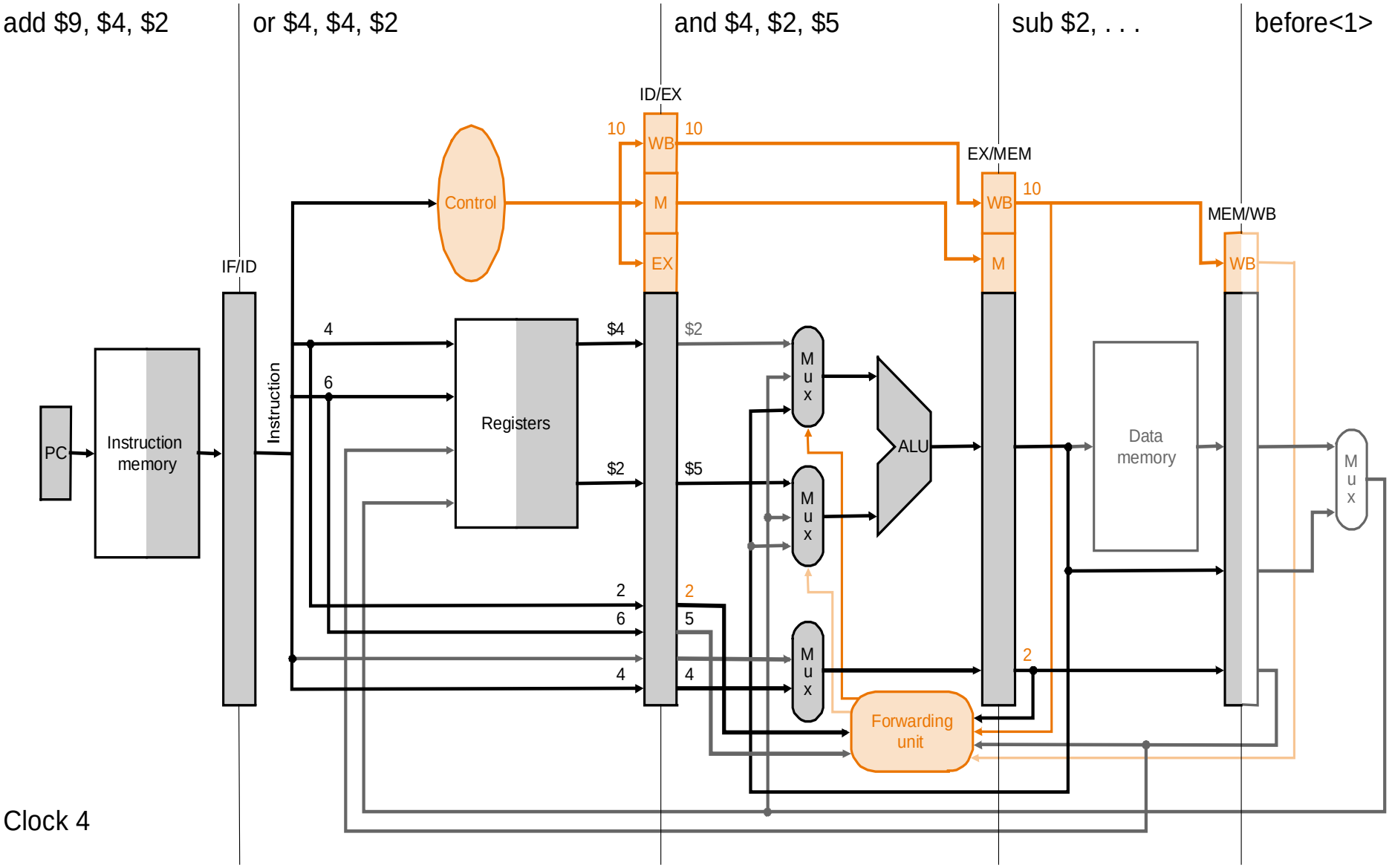
if (MEM/WB.RegWrite & (EX/MEM.Register-Rd ≠ ID/EX.Register-Rt) &
(MEM/WB.Register-Rd = ID/EX.Register-Rs)) ForwardA = MEM/WB

if (MEM/WB.RegWrite & (EX/MEM.Register-Rd ≠ ID/EX.Register-Rs) &
(MEM/WB.Register-Rd = ID/EX.Register-Rt)) ForwardB = MEM/WB

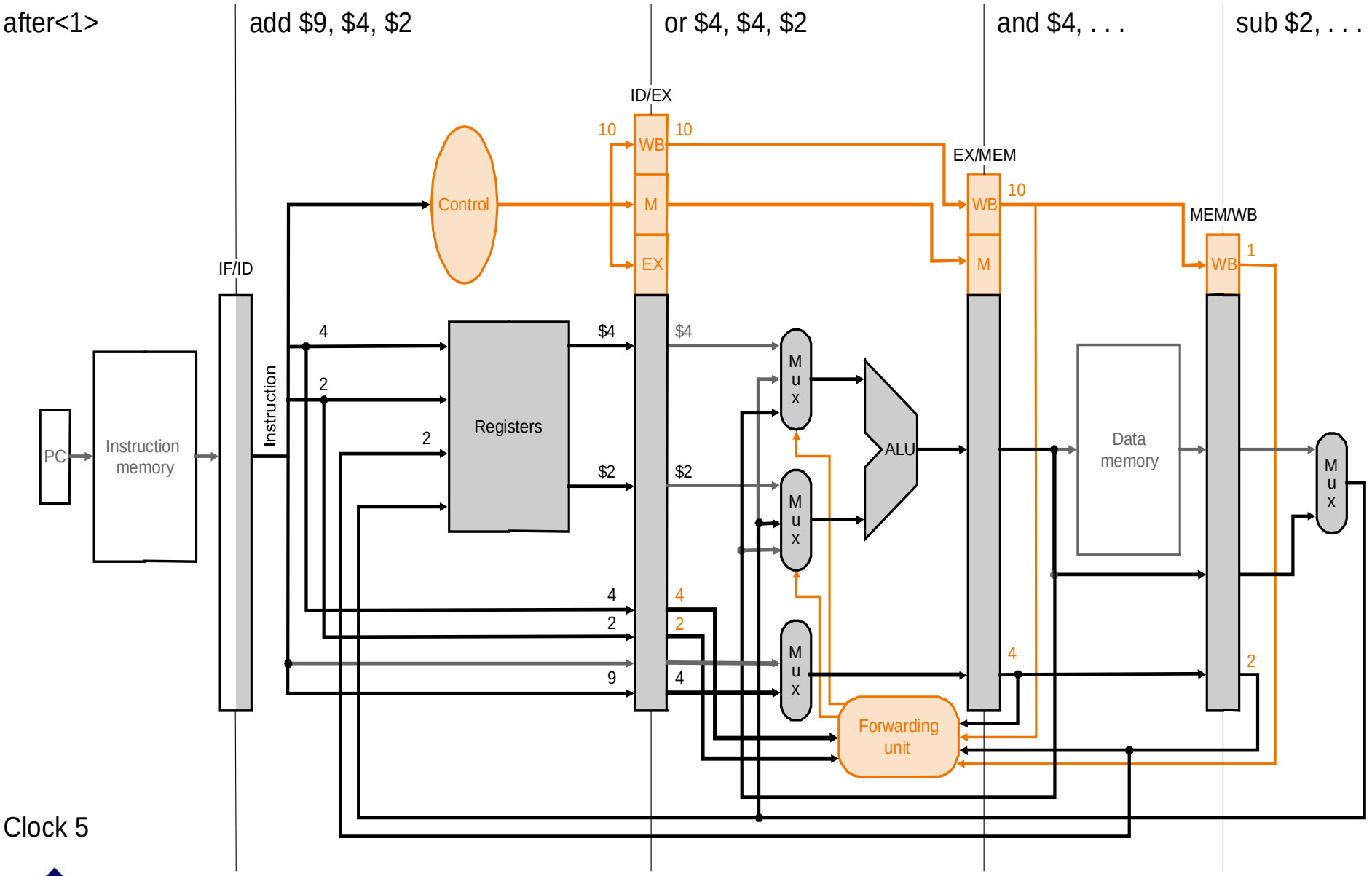
Data Forwarding Example



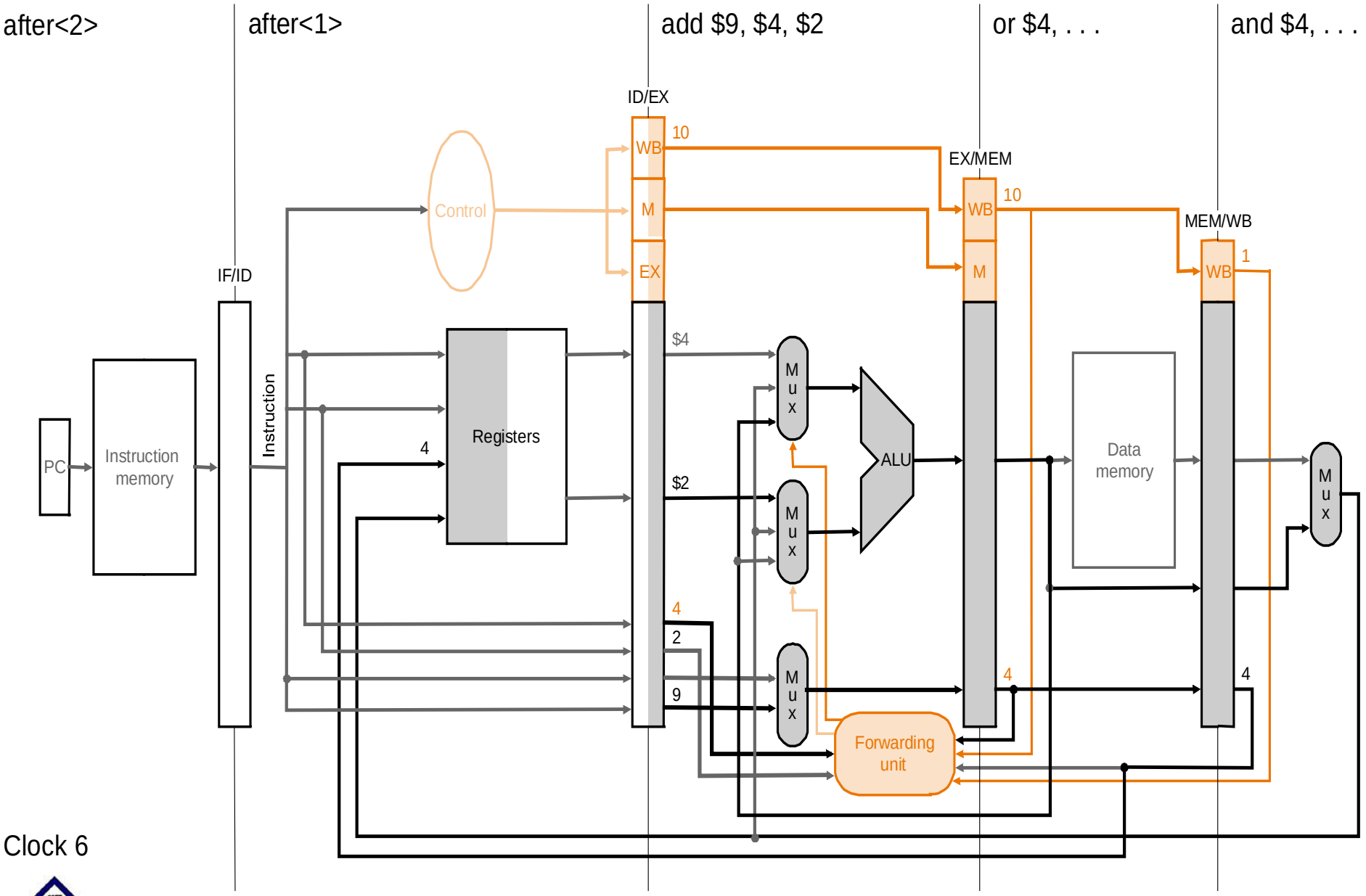
Example


add $9, $4, $2
or $4, $4, $2
and $4, $2, $5
sub $2, . . .
before<1>
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
10
10
10
10
PC
Instruction memory
Instruction
4
6
Registers
$4
$2
$2
$5
2
6
4
2
5
4
2
ALU
Mux
Data memory
Forwarding unit
Clock 4

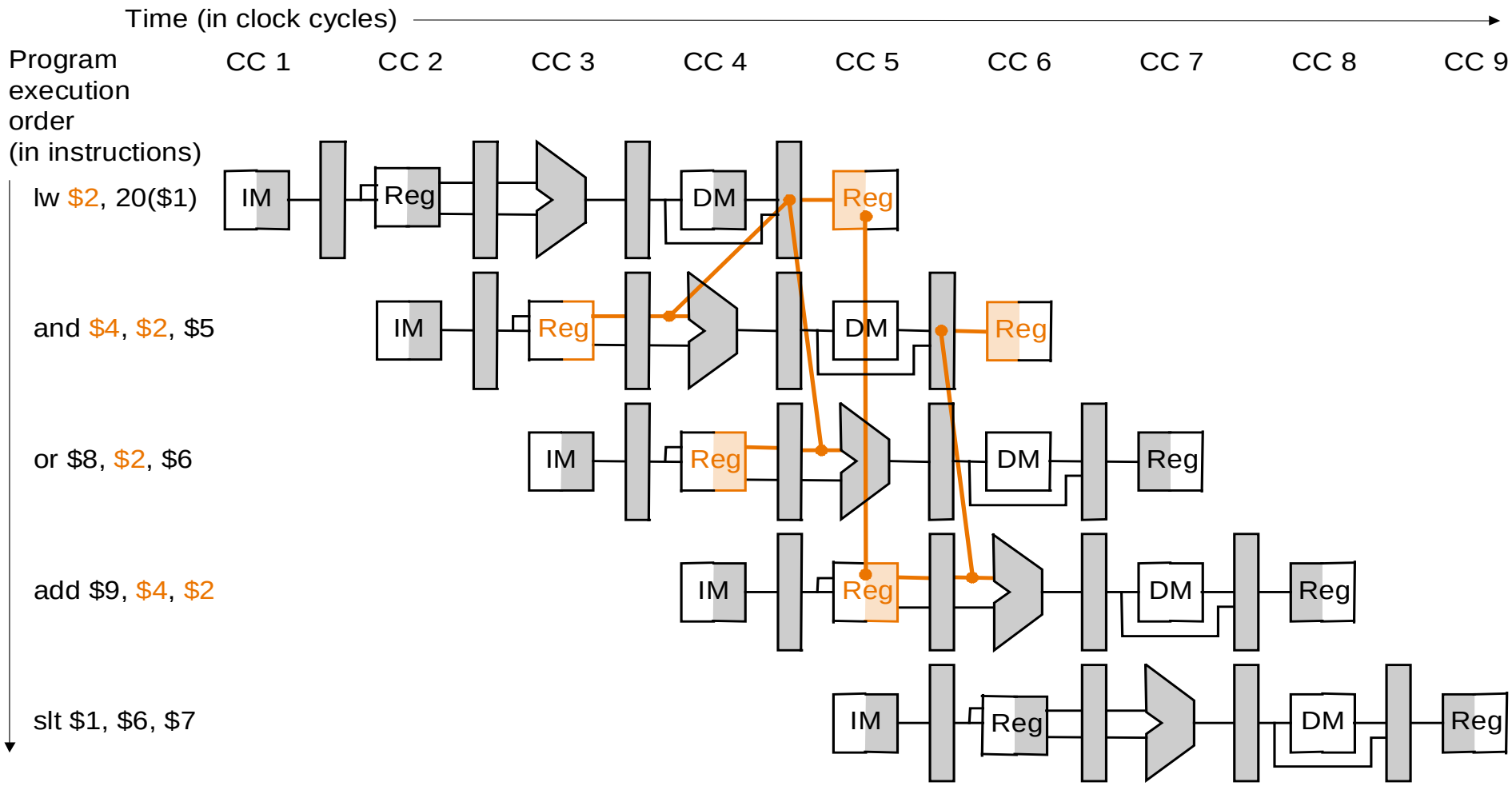
Example


after<1>
add $9, $4, $2
or $4, $4, $2
and $4, . . .
sub $2, . . .
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
10
10
10
1
PC
Instruction memory
Instruction
Registers
4
2
2
$4
$2
$4
$2
4
2
9
4
4
2
4
2
Mux
Mux
Mux
ALU
Data memory
Mux
Forwarding unit
Clock 5

Example

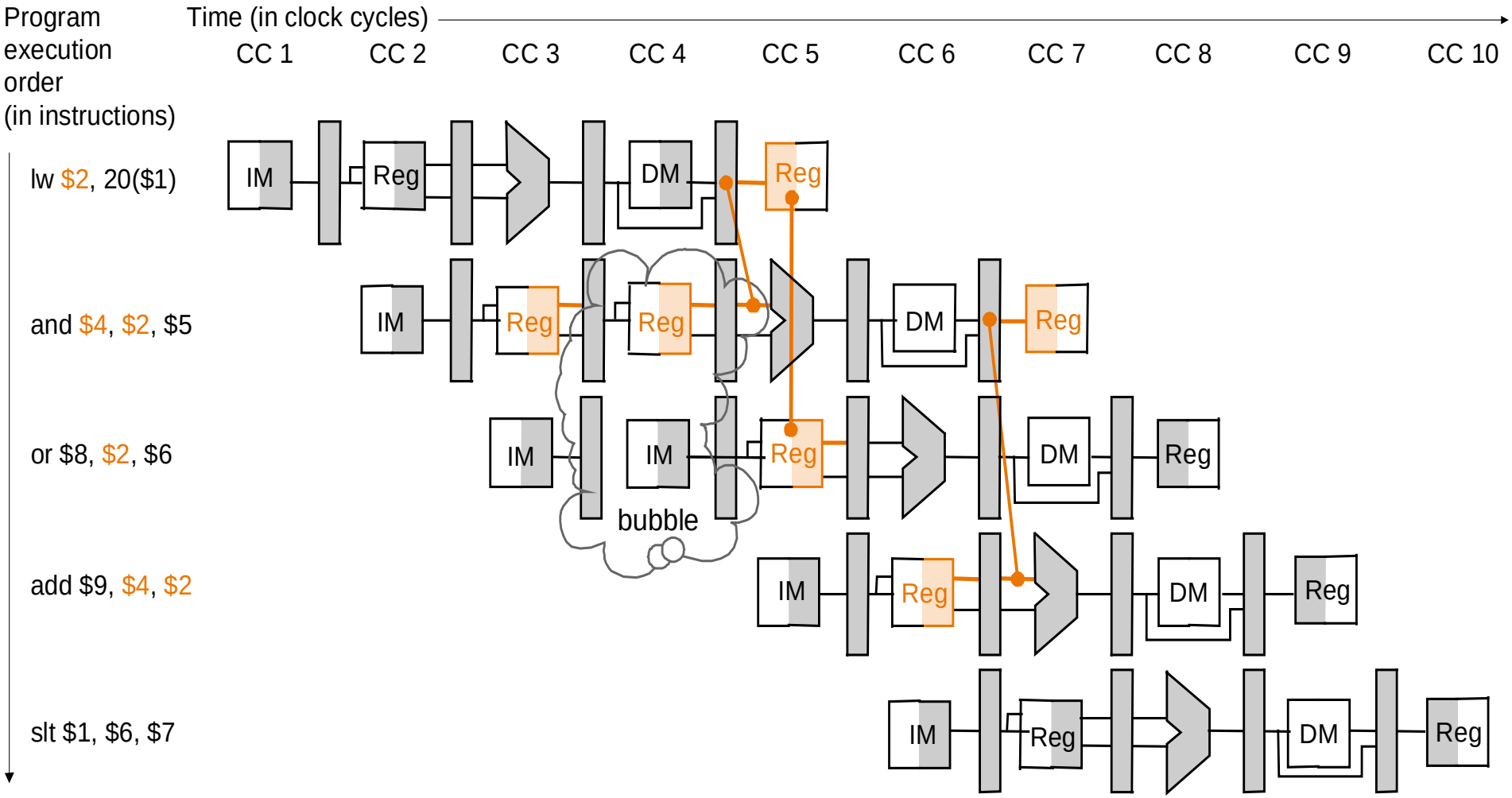

after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
ID/EX
EX/MEM
MEM/WB
WB
M
EX
10
Control
IF/ID
PC
Instruction memory
Instruction
Registers
4
$4
$2
4
2
9
Mux
ALU
Data memory
Forwarding unit
1
Clock 6

Data Hazard Caused by Load



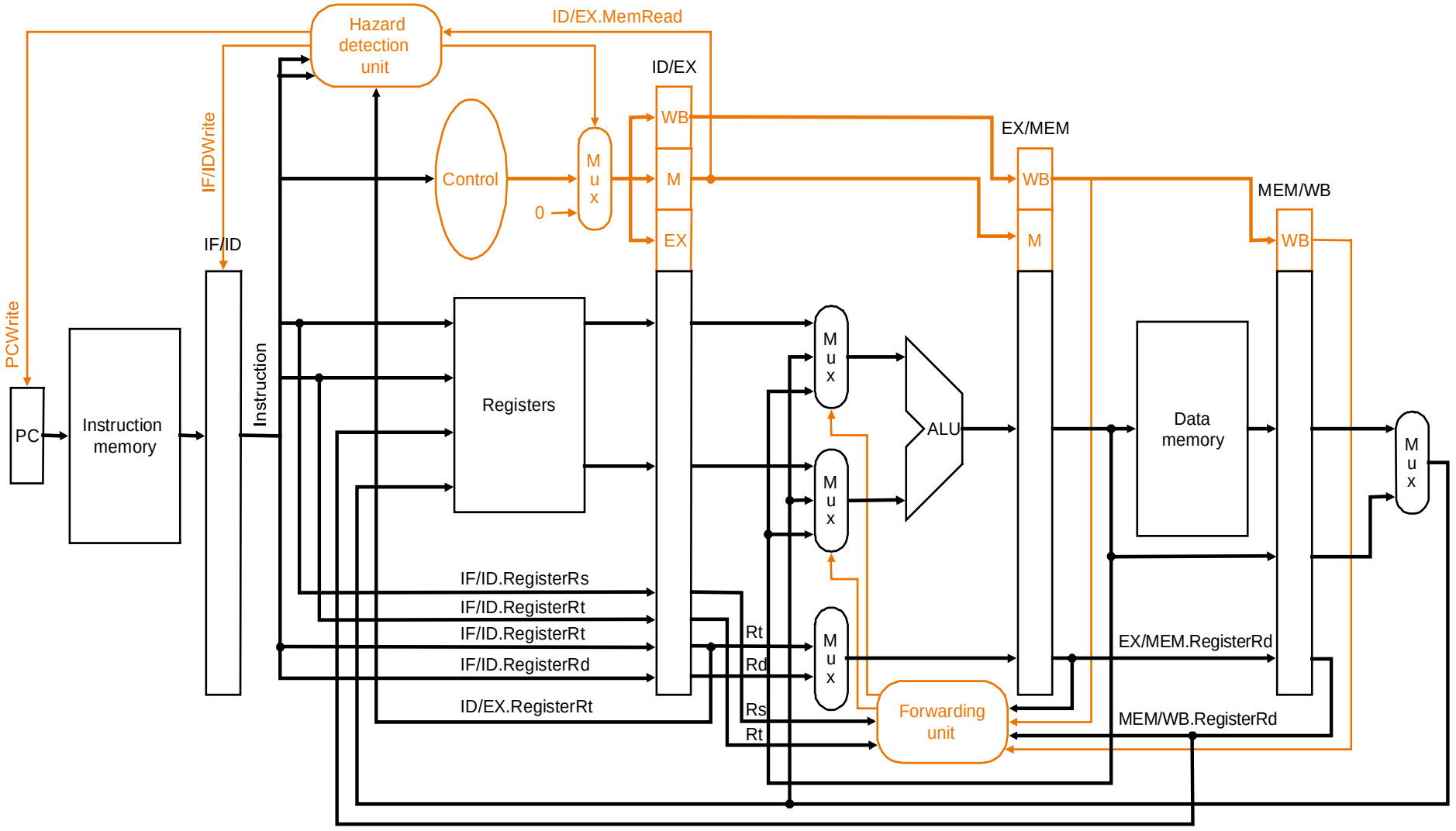
- Dependencies backward in time are hazards
- Cannot solve with forwarding
- Must delay/stall instruction dependent on loads

Hazard Detection Unit



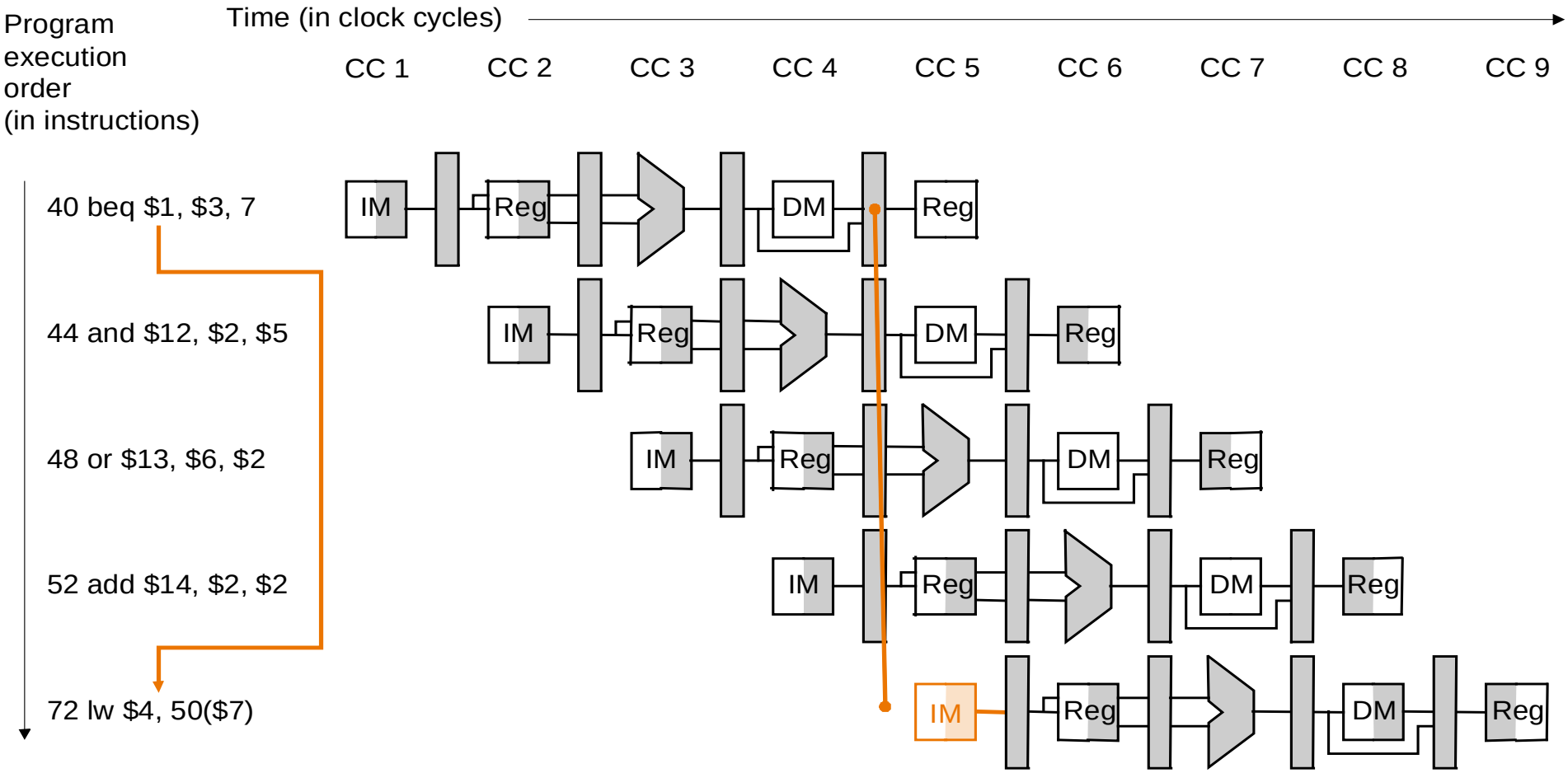
if (ID/EX.MemRead & ((ID/EX.Register-Rt = IF/ID.Register-Rs) | (ID/EX.Register-Rt = IF/ID.Register-Rt))) stall the pipeline

Supporting Pipeline Stall



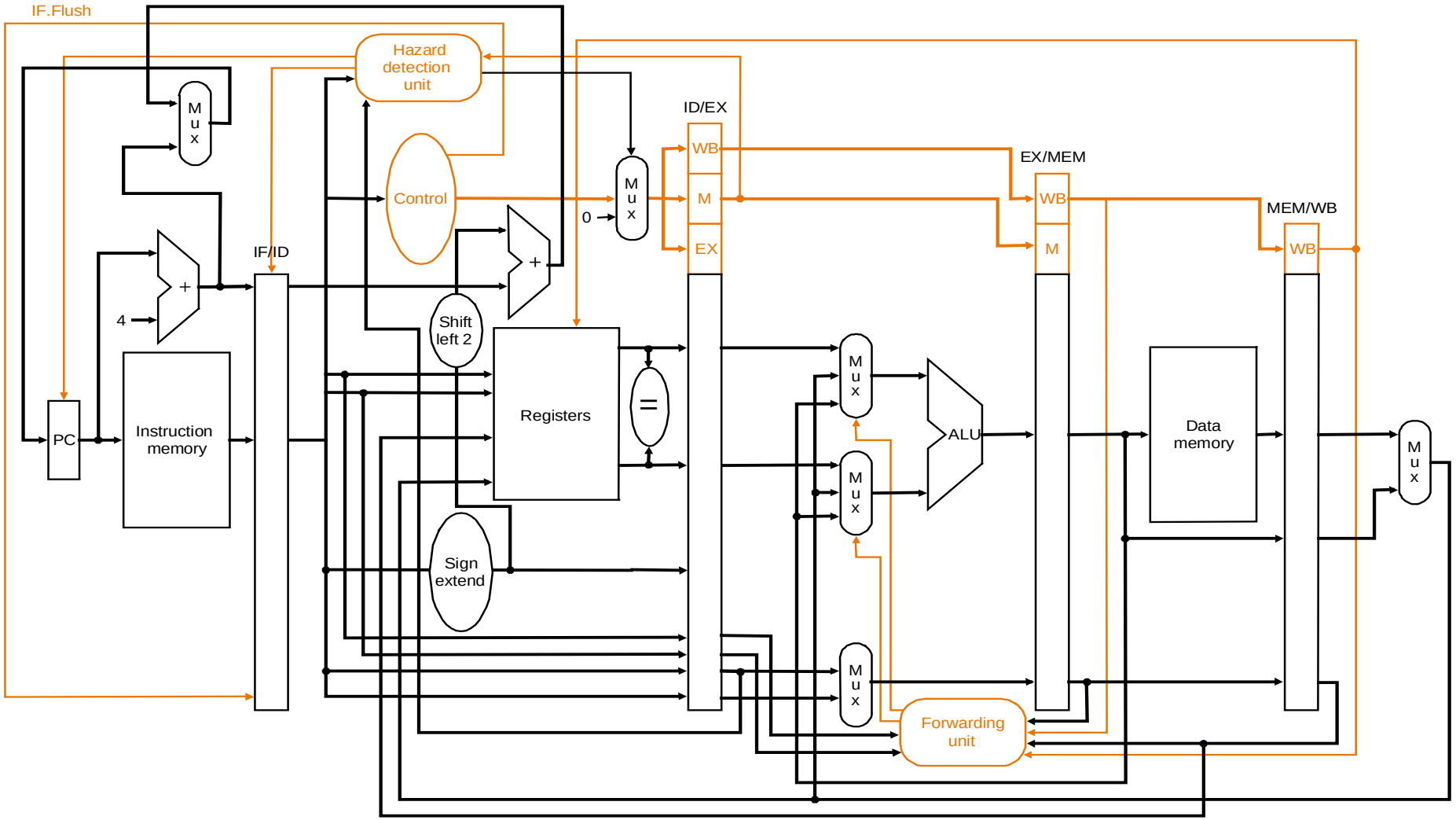
De-asserting all the control lines in EX, MEM and WB control fields of the ID/EX pipeline stationary register would stall the pipeline

Control Hazards



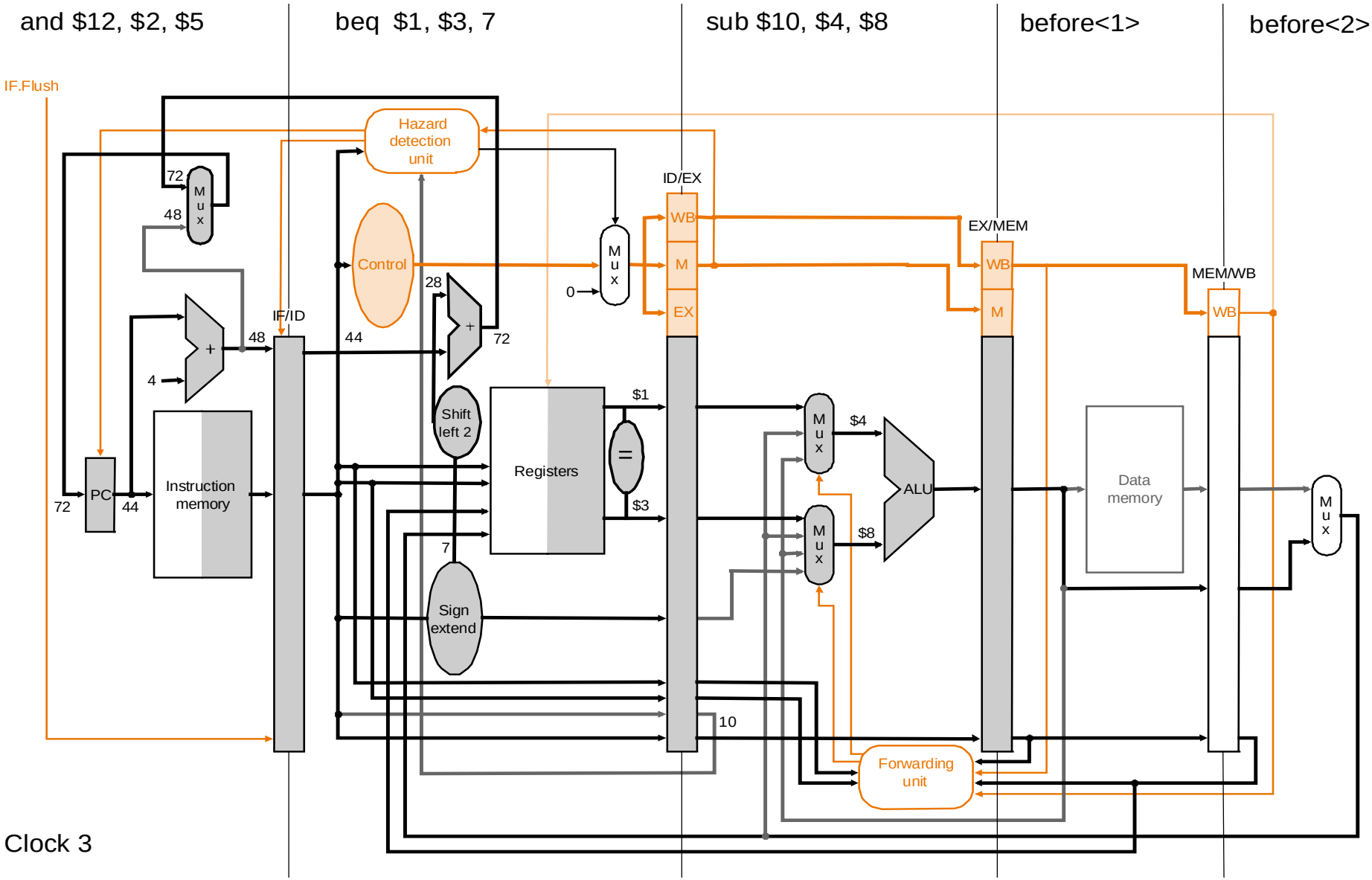
- Must be carefully detected and handled to avoid semantic error
- Can be solved by excessive pipeline stalls which is inefficient
- Two approaches: “delayed branching” and “branch prediction”

Supporting Conditional Branching

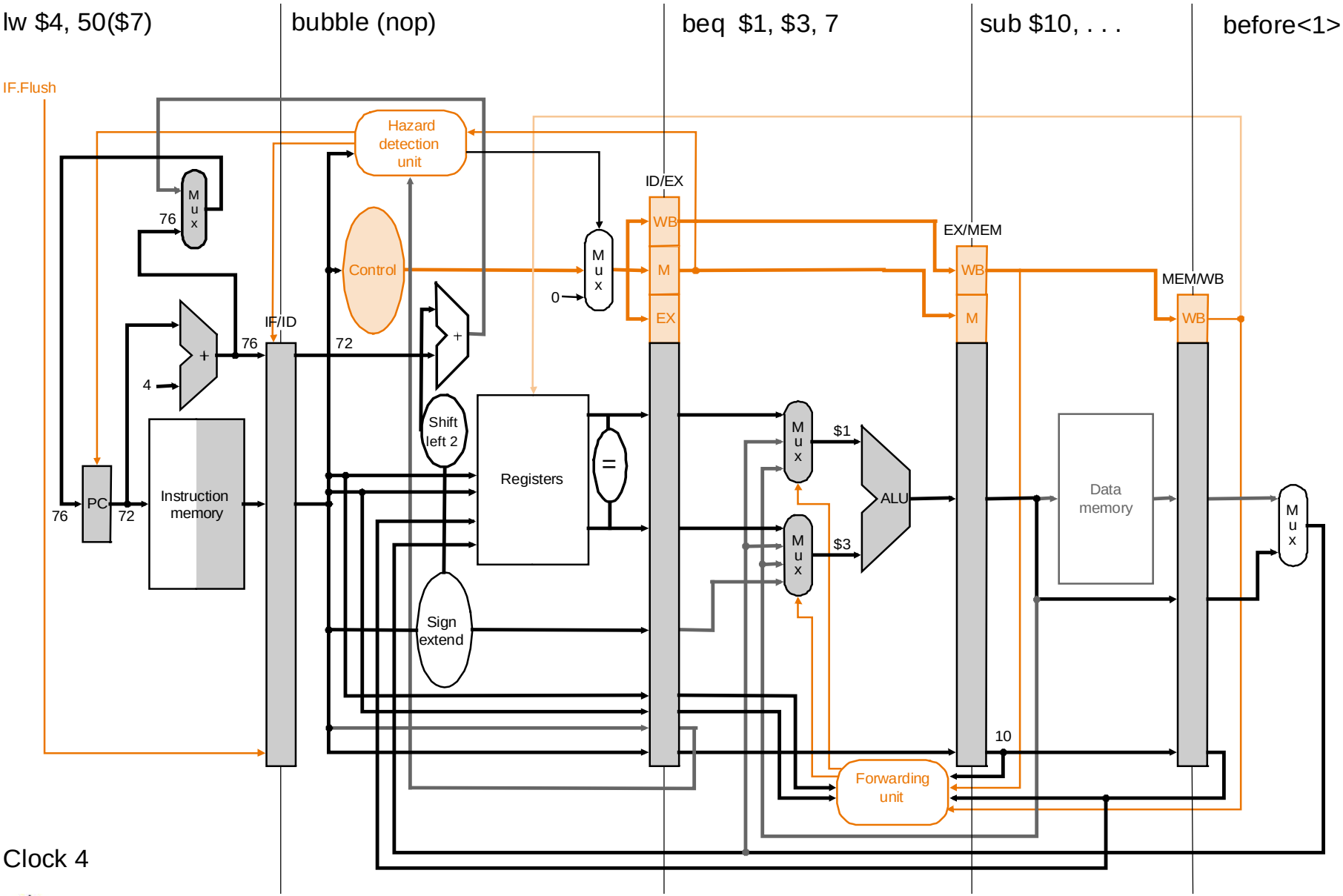


- Moves comparison to ID stage to minimize worst case pipeline flushing
- Interacts with data hazards hardware to ensure right register values
- Flushes the pipeline with zero control values and instruction register

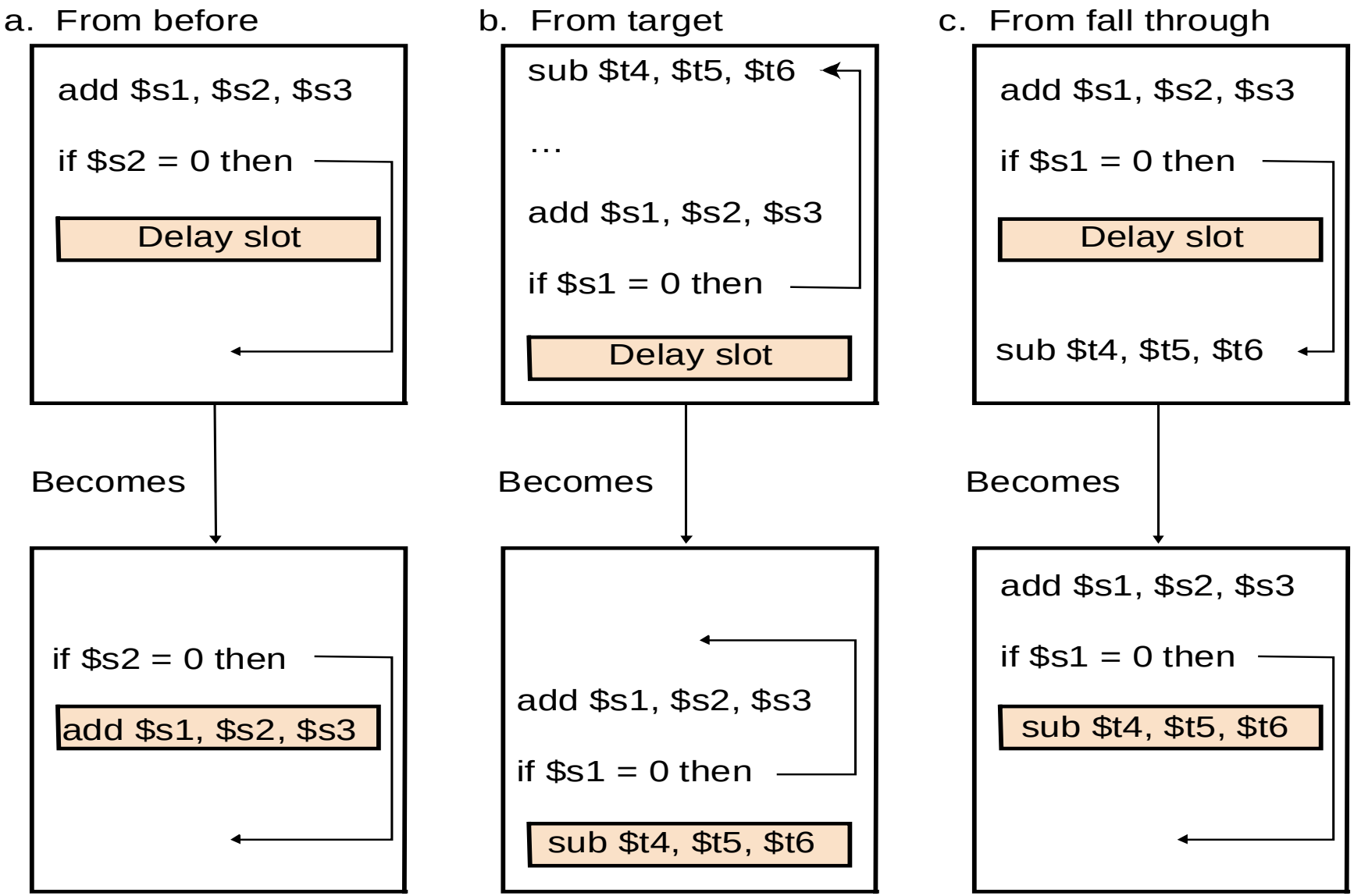
Conditional Branching Example



Example


lw $4, 50($7)
bubble (nop)
beq $1, $3, 7
sub $10, . . .
before<1>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
IF/ID
Mux
76
72
4
0
PC
Instruction memory
Shift left 2
Registers
Sign extend
ALU
$1
$3
Data memory
10
Forwarding unit
Clock 4

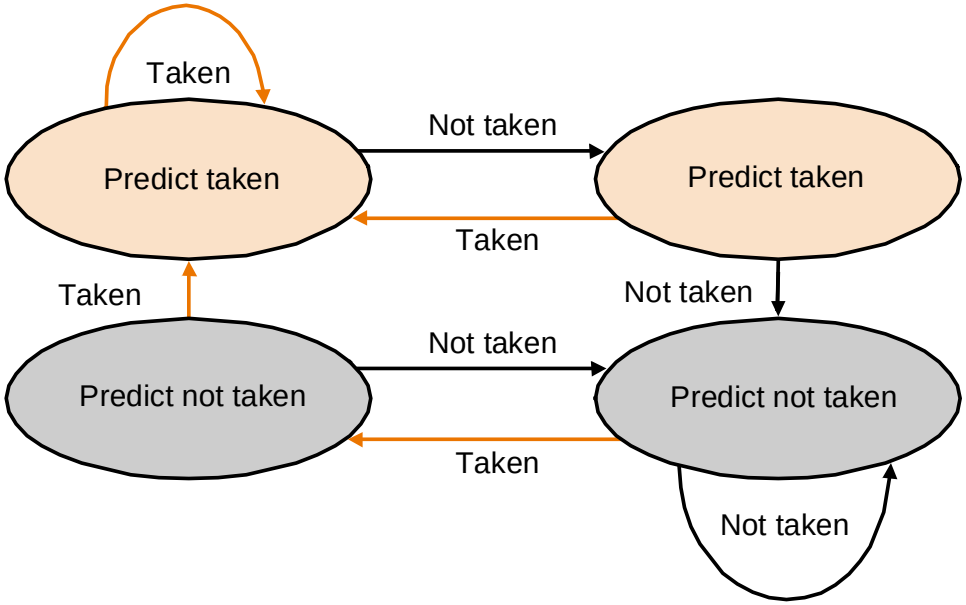
Compiler Optimization of Delays



Dynamic Branch Prediction

- Uses a history buffer to track branching instruction using their addresses
- Simplest approach uses one-bit per branch instruction to indicate the last experience
- Branching differently from prediction gets the history bit flipped
- One-bit buffer still fails to predict at least twice in case of a loop

A two-bit buffer better captures the history of the branch instruction



Exception/Interrupts

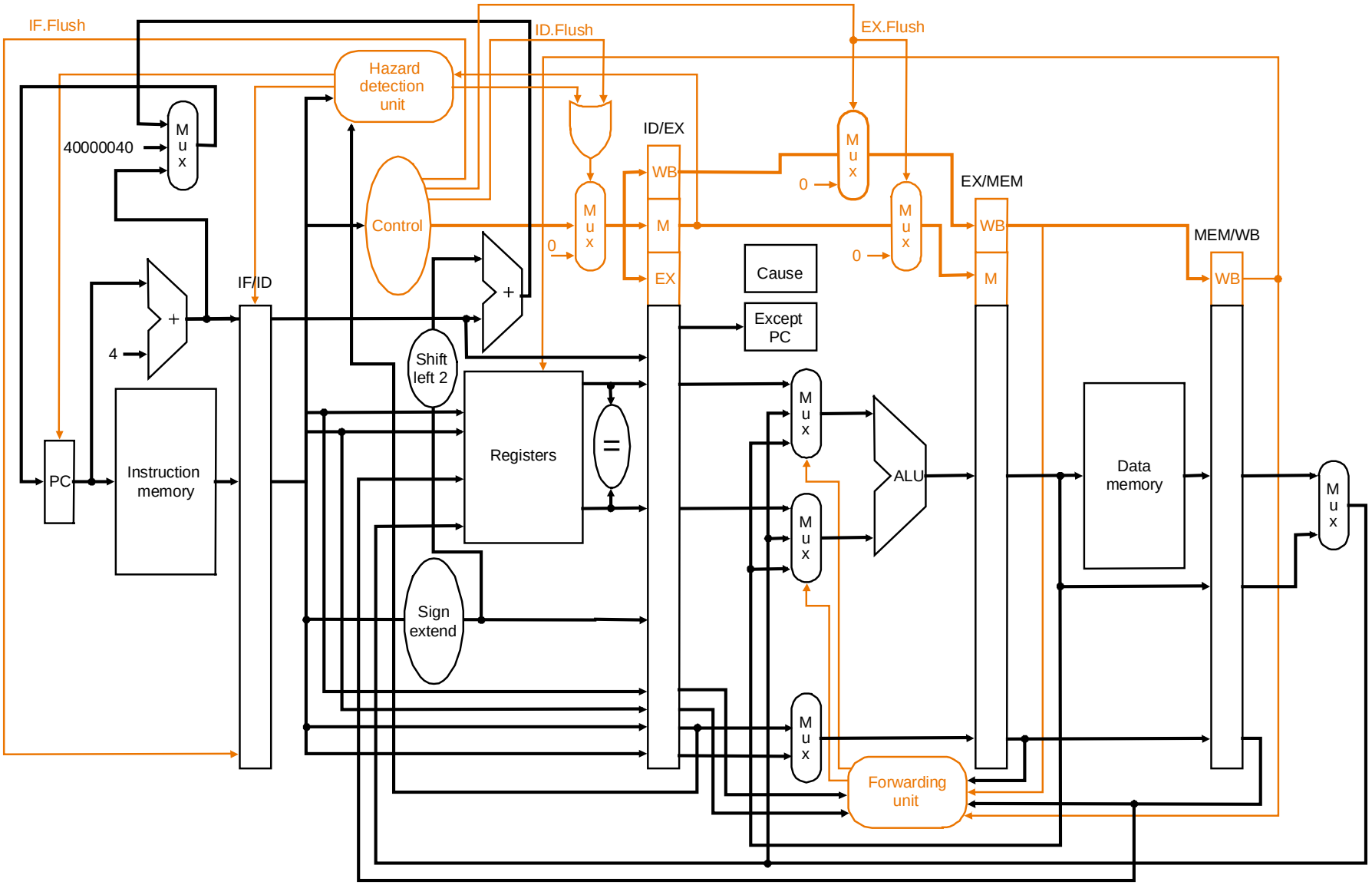
- 5 instructions executing in 5 stages pipeline
  - ➔ How to stop the pipeline?
  - ➔ Restart?
  - ➔ Who caused the interrupt?

Stage	*Problem interrupts occurring*
IF	Page fault on instruction fetch; memory-protection violation
ID	Undefined or illegal op-code
EX	Arithmetic exception
MEM	Page fault on data fetch; memory-protection violation; memory error

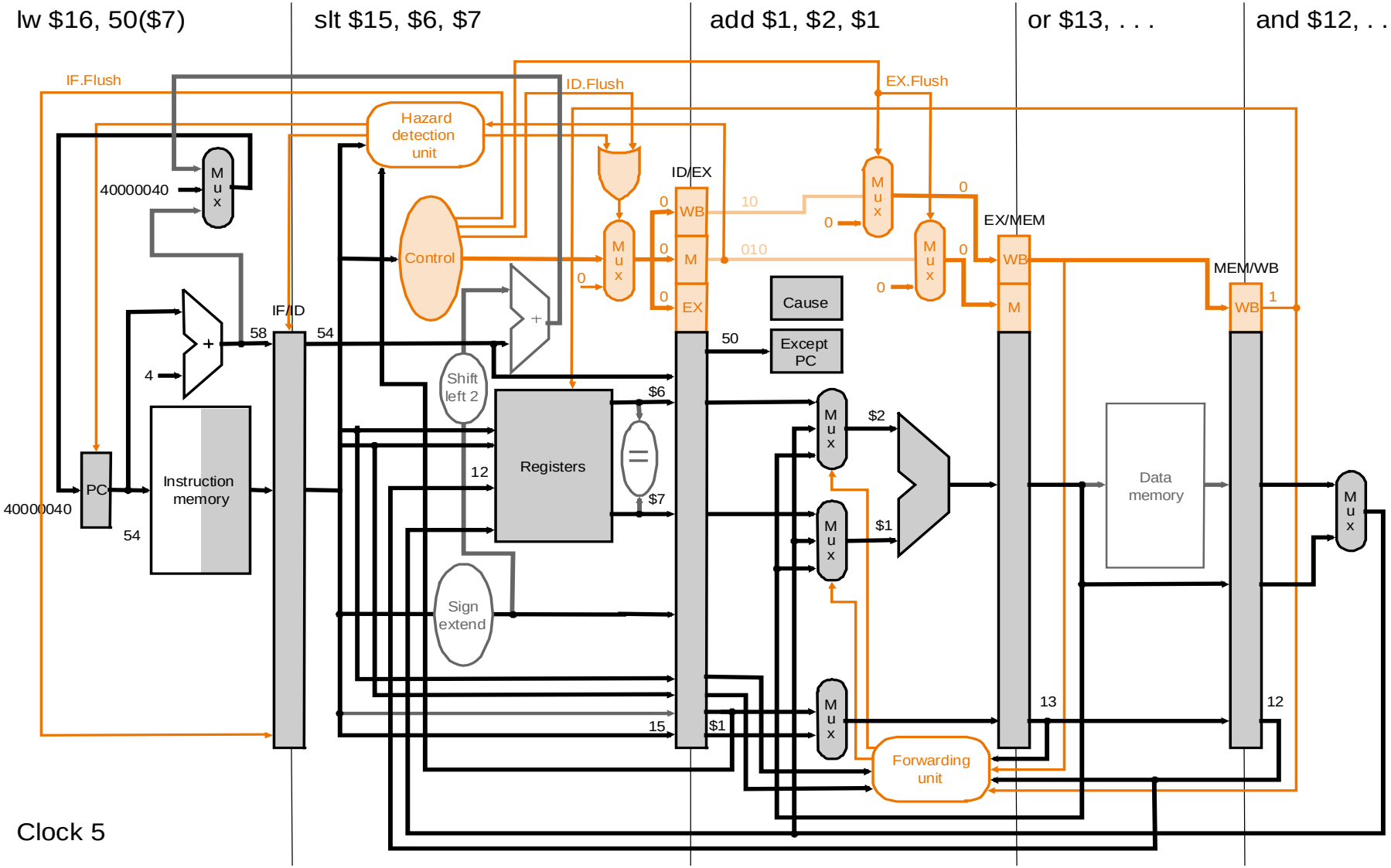
- Load with data page fault, Add with instruction page fault?
- *Solution*: interrupt vector/instruction, interrupt ASAP, restart everything incomplete

- External Interrupts:
  - ➔ Allow pipeline to drain
  - ➔ Load PC with interrupt address
- Faults (within instr., can be restarted)
  - ➔ Force trap instruction into IF
  - ➔ Disable writes till trap hits WB
  - ➔ Must save multiple PCs or PC + state

Supporting Exception Handling

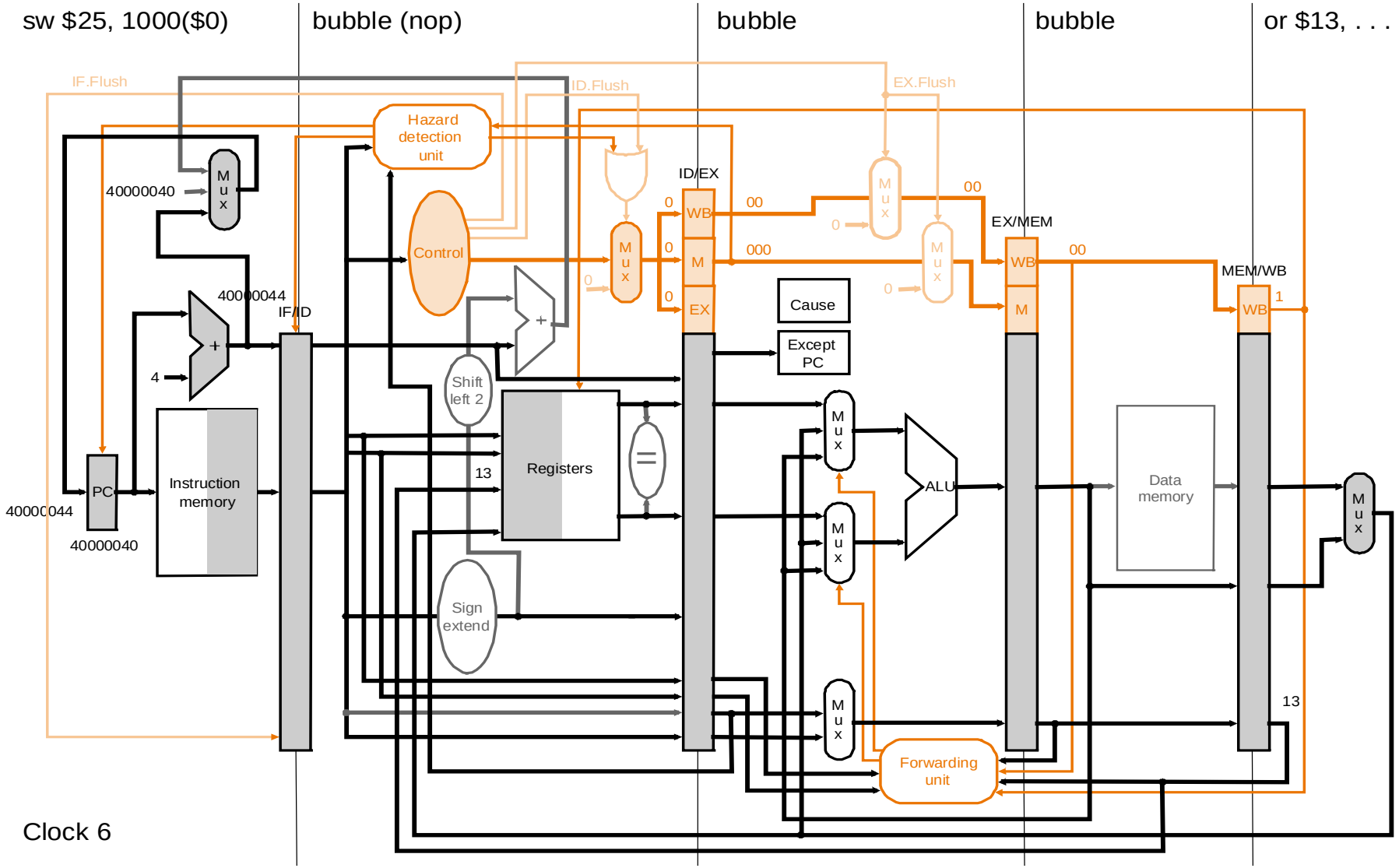


Example for Exception Handling



Add instruction causes overflow

Example



- Pipeline is drained after saving address of instruction
- First instruction of the handler is fetched

Conclusion

❑ *Summary*

➔ Pipeline hazard detection
- Data hazard
- Control hazard

➔ Handling hazard in the pipeline design
- Data forwarding and branch prediction
- Coordination between control and data hazard

➔ Supporting exceptions
- Pipeline flushing
- Invoking exception handling routines

❑ *Next Lecture*

➔ Pipelining floating point operations

➔ Instruction level parallelism

Read sections 4.7-4.9 in the 4th Ed or the 5th Ed. of the textbook