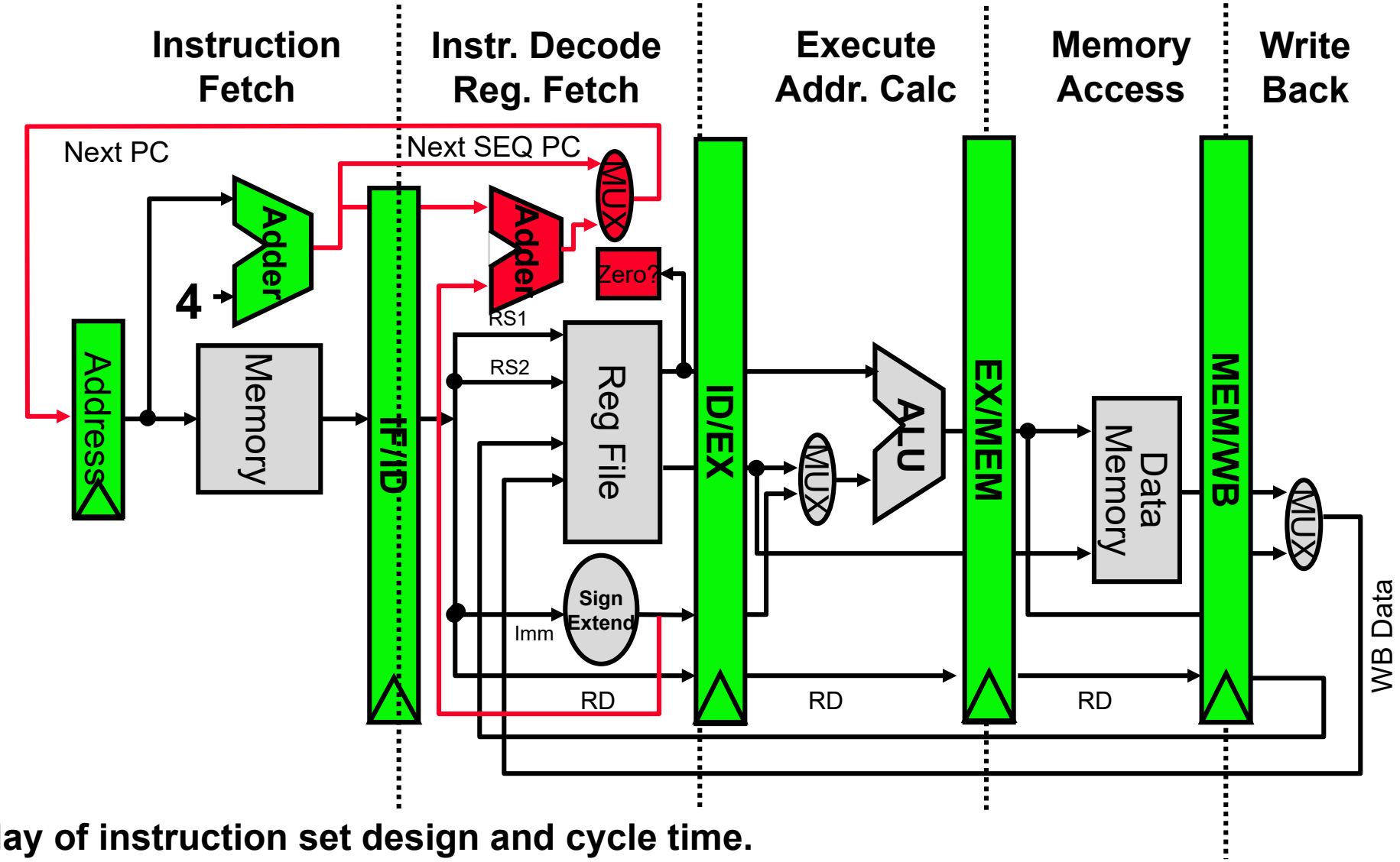
CPT_S 260 Intro to Computer Architecture
Lecture 36

Pipeline Datapath
April 13, 2022

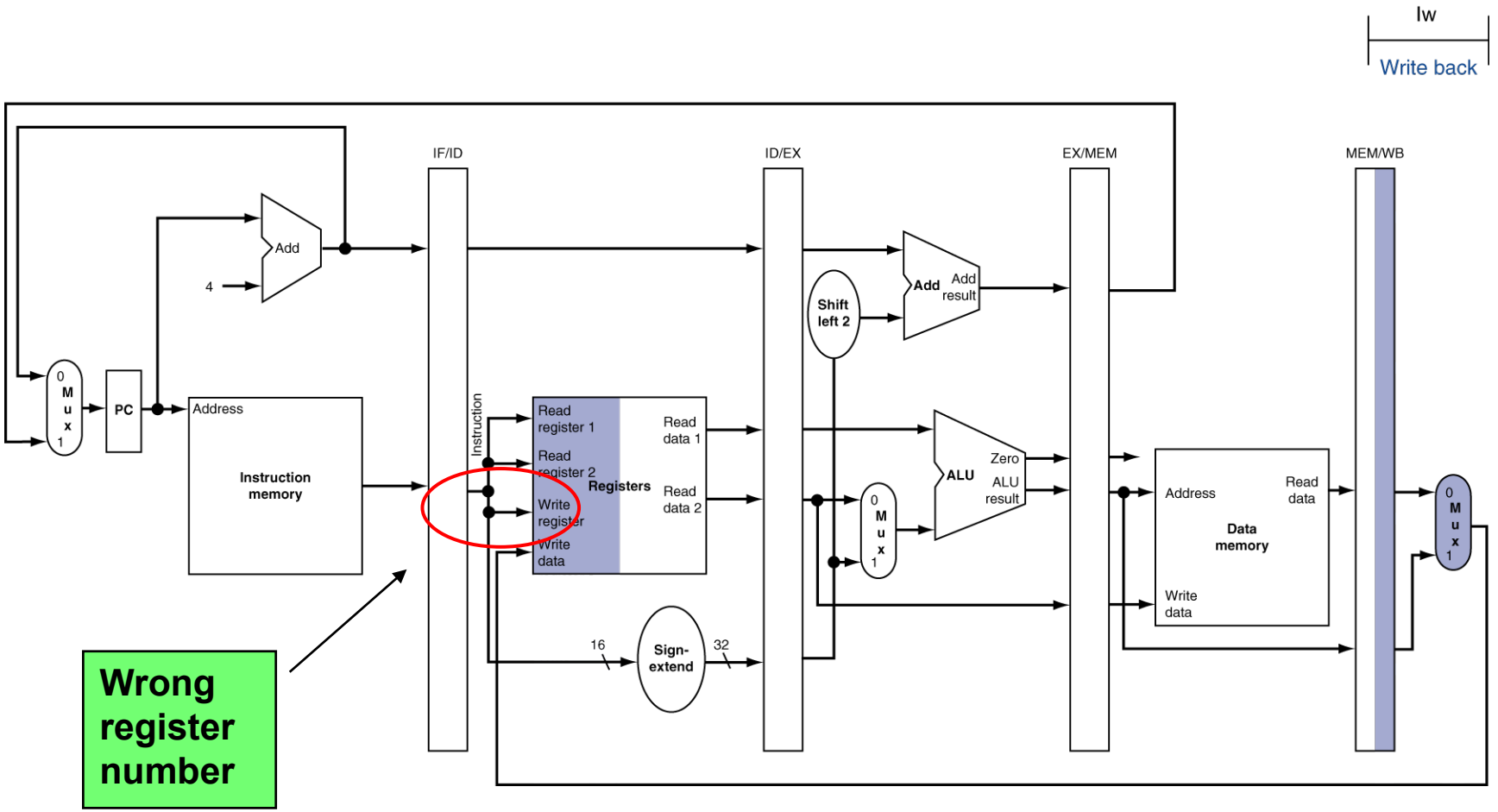
Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Pipelined MIPS Datapath

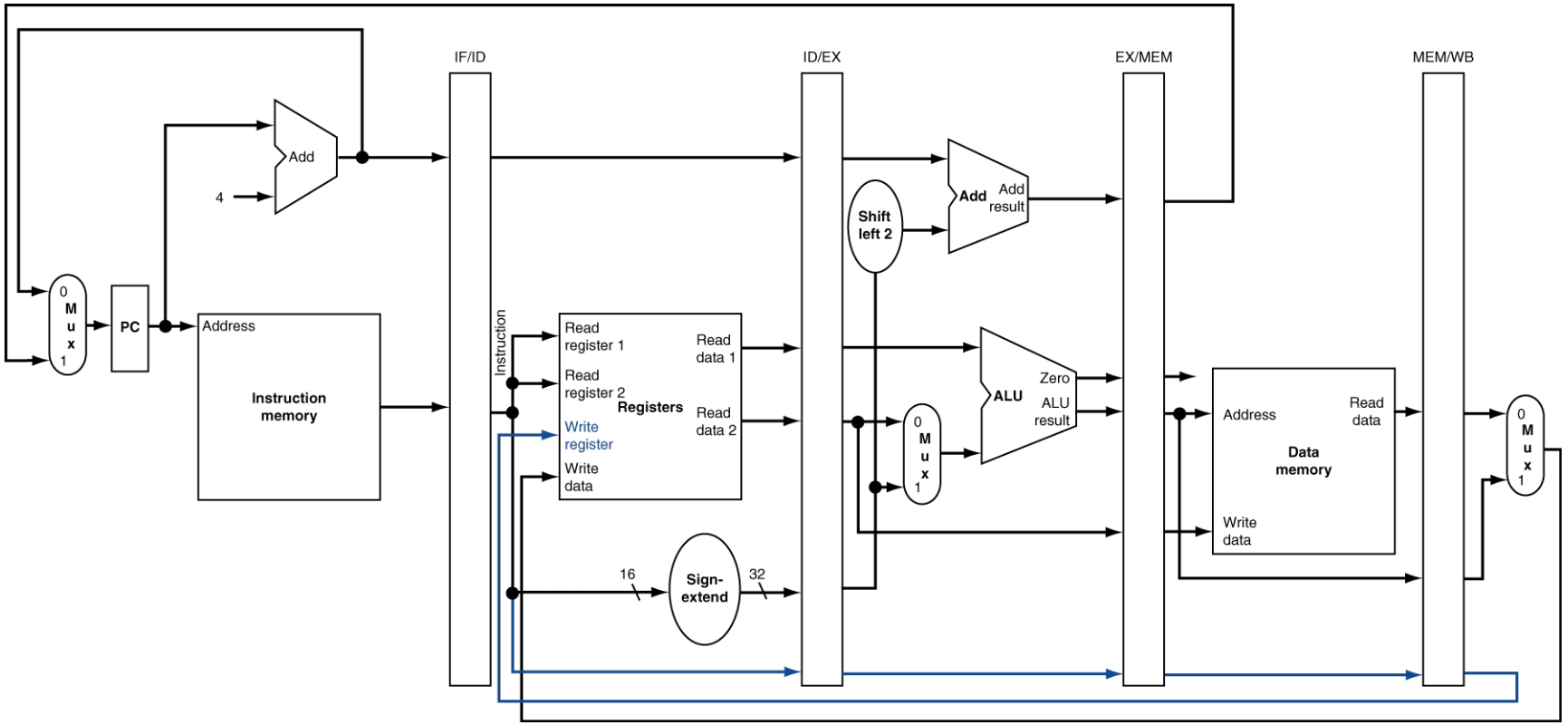


- Interplay of instruction set design and cycle time.

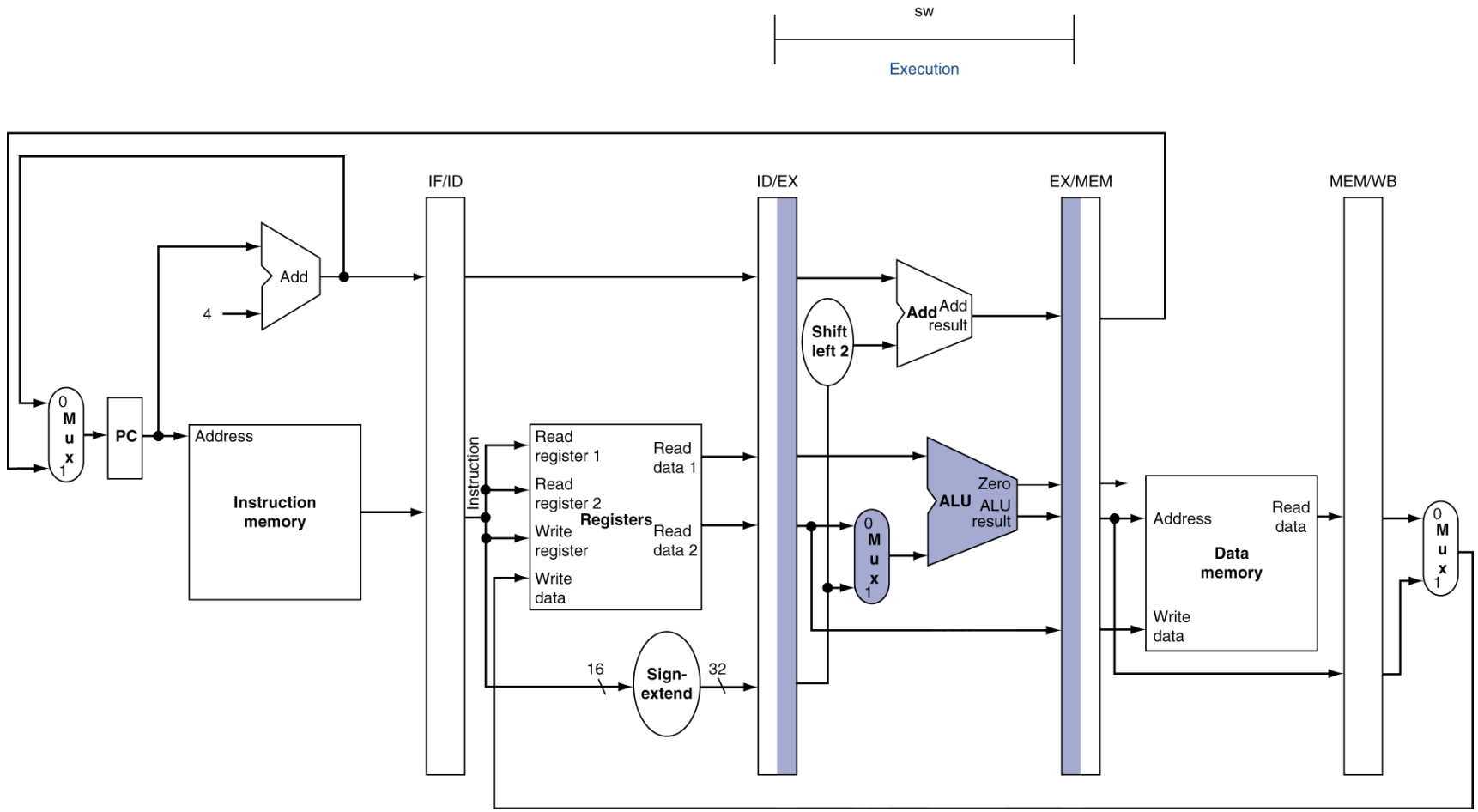
WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Address
Data memory
Read data
Write data
Wrong register number

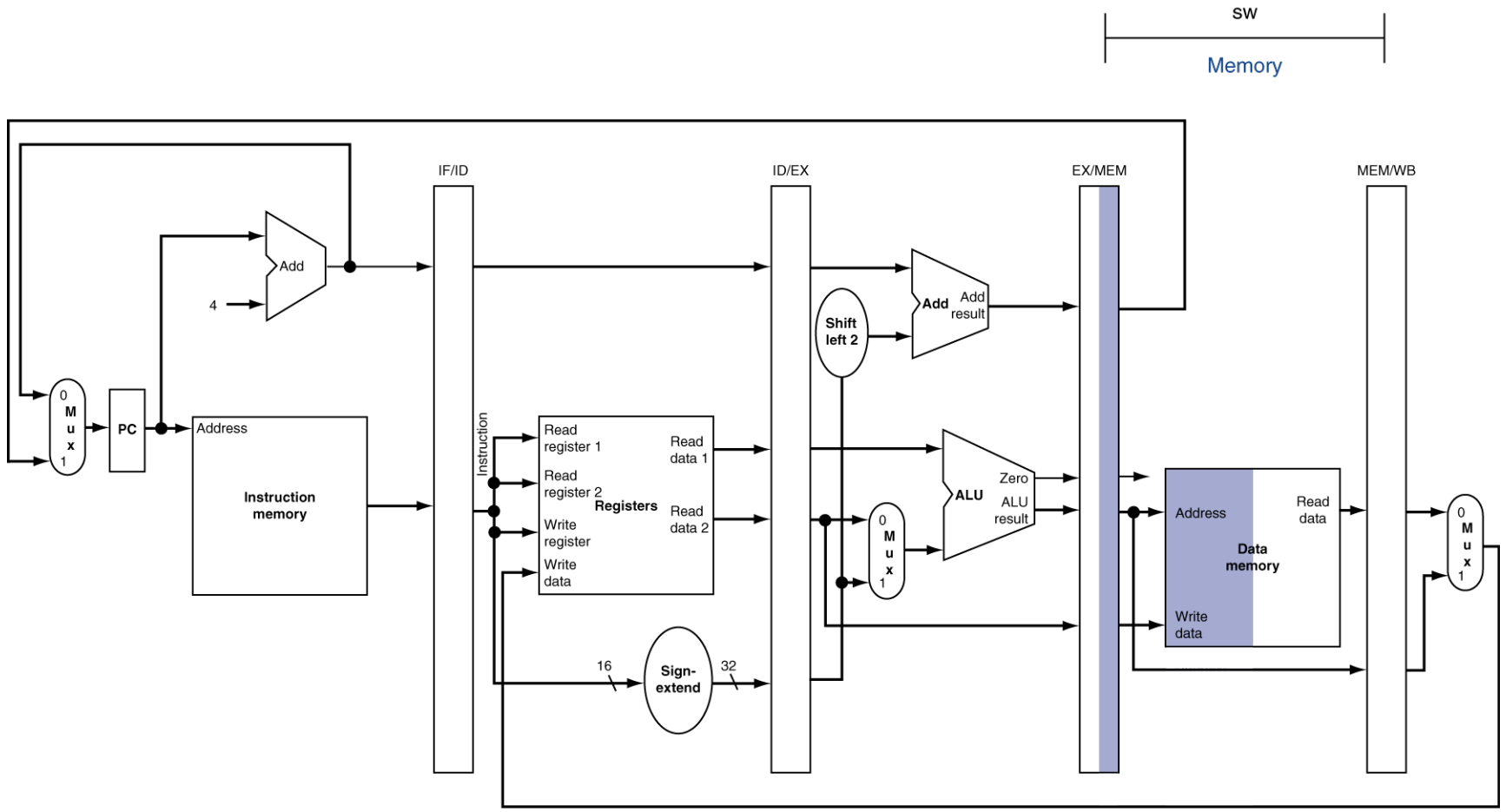
Corrected Datapath for Load



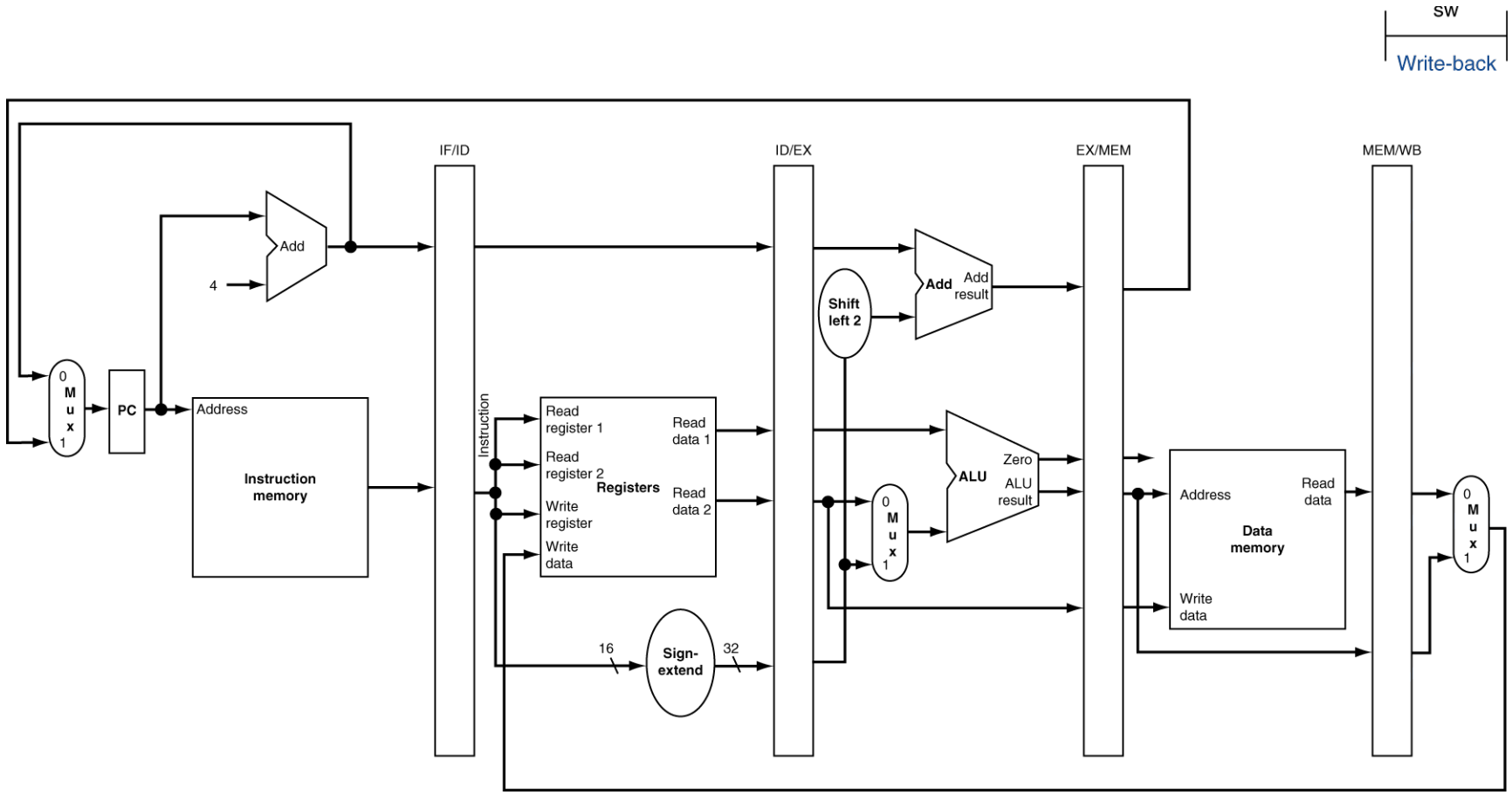
EX for Store


sw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

MEM for Store



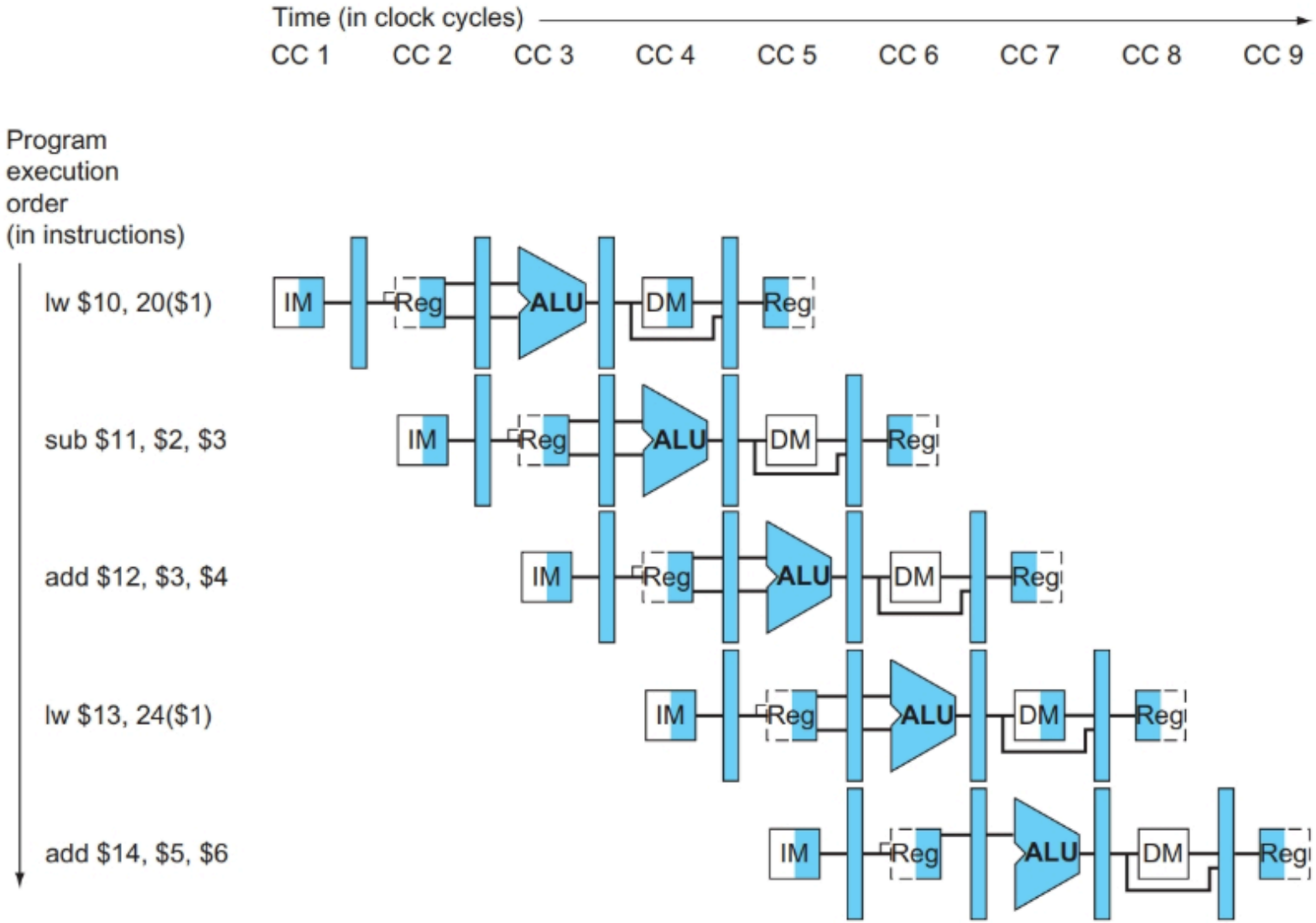
WB for Store


SW
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

Example

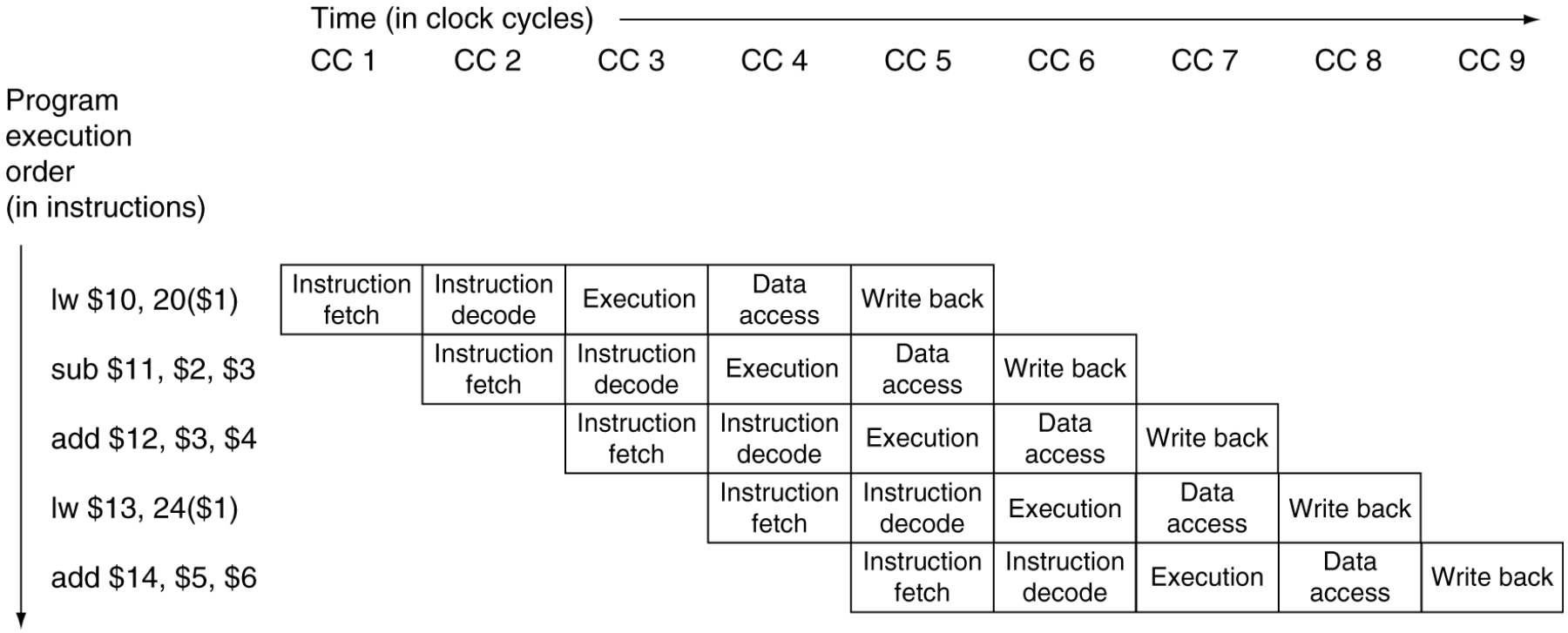
```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```

Multi-Cycle Pipeline Diagram



Multi-Cycle Pipeline Diagram

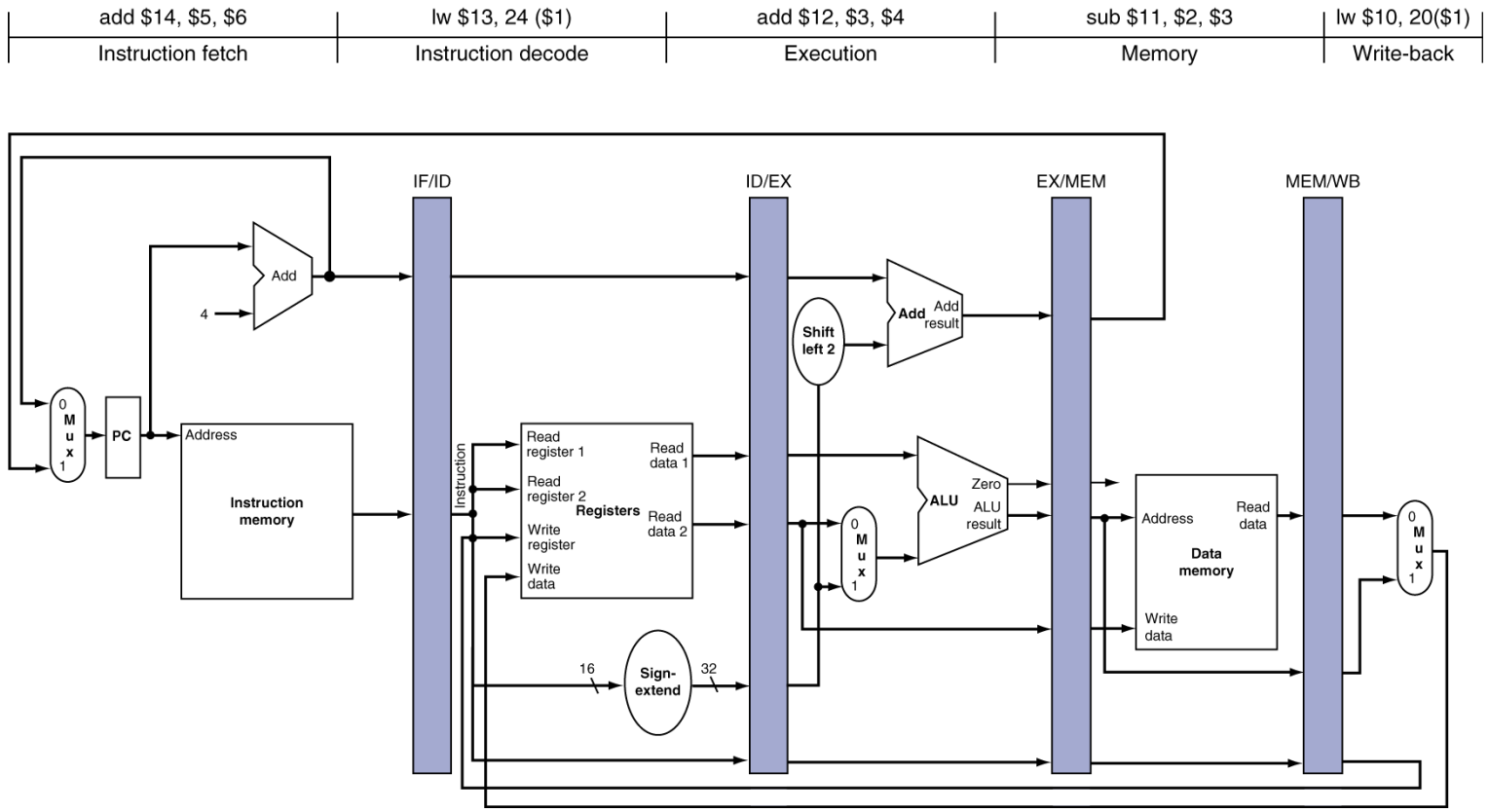
Traditional form



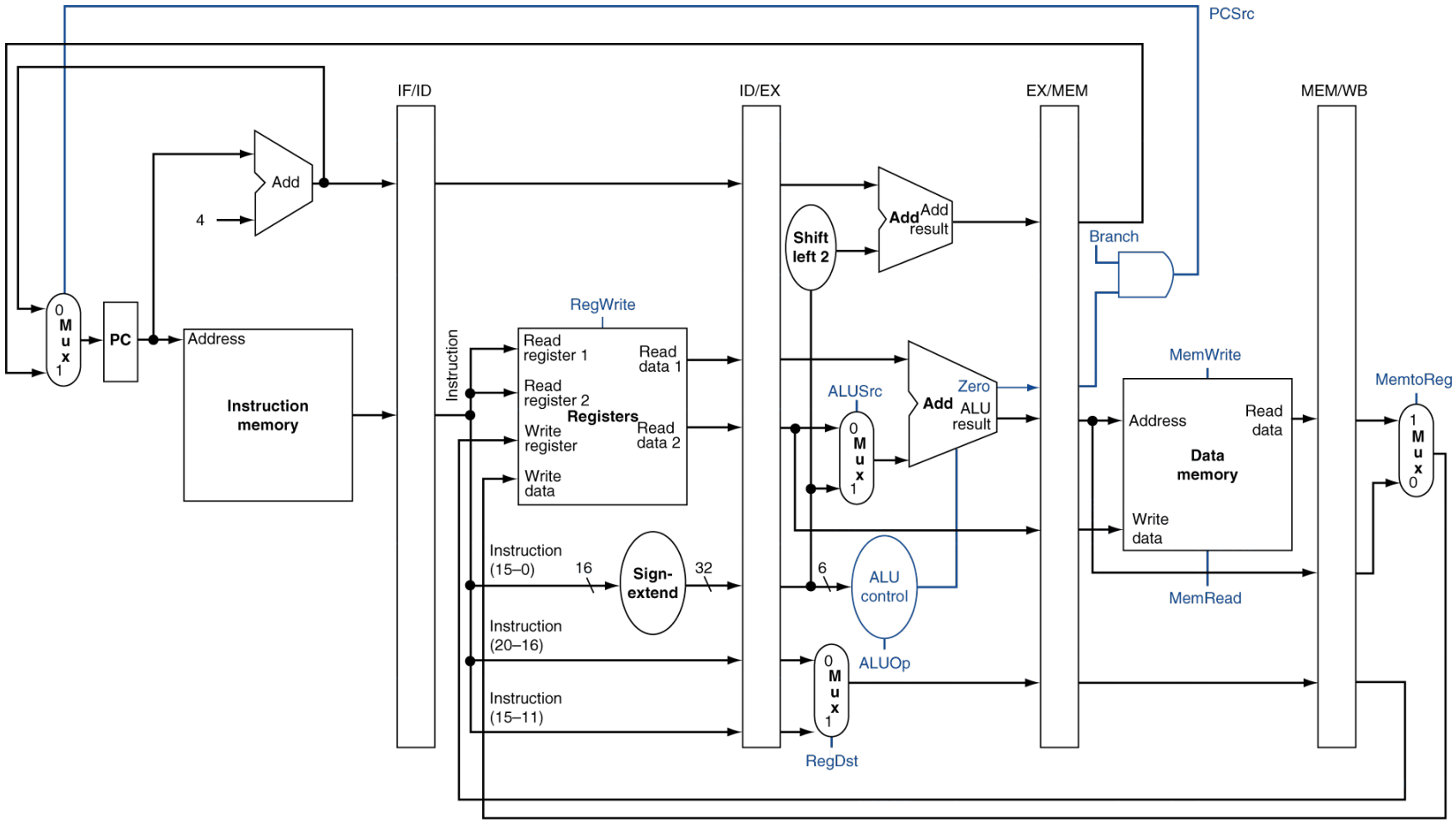
Single-Cycle Pipeline Diagram

- **State of pipeline in a given cycle (Clock cycle 5)**

```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```



Pipelined Control (Simplified)

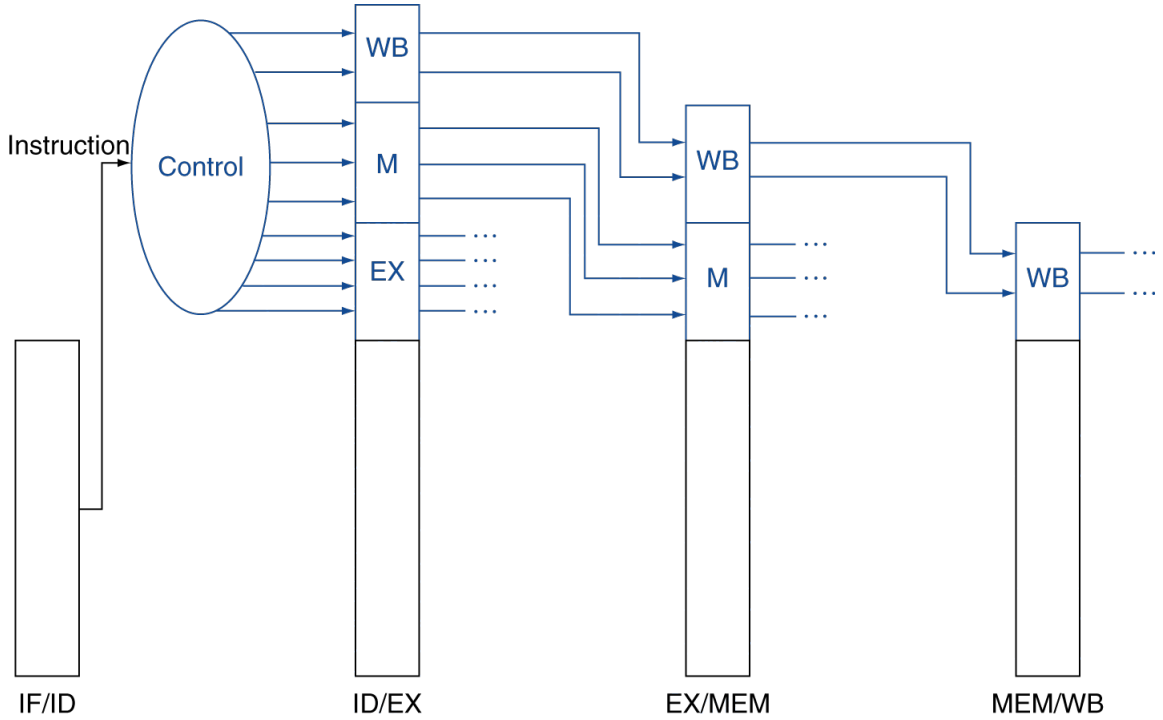


Control Signals

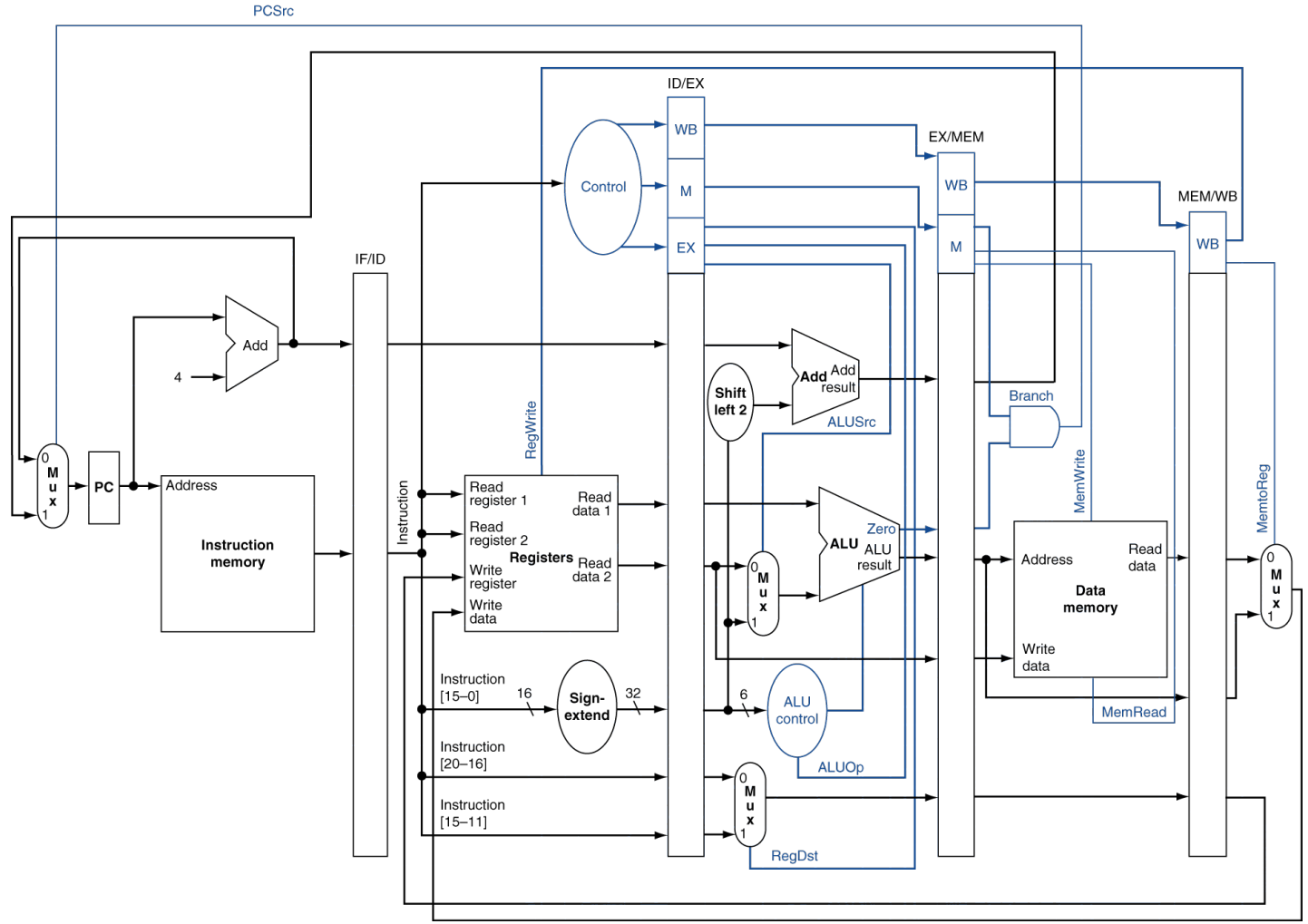
Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Pipelined Control

- **Control signals derived from instruction**
  - As in single-cycle implementation



Pipelined Control



Dealing with Hazards and Changes in Pipeline Datapath & Control

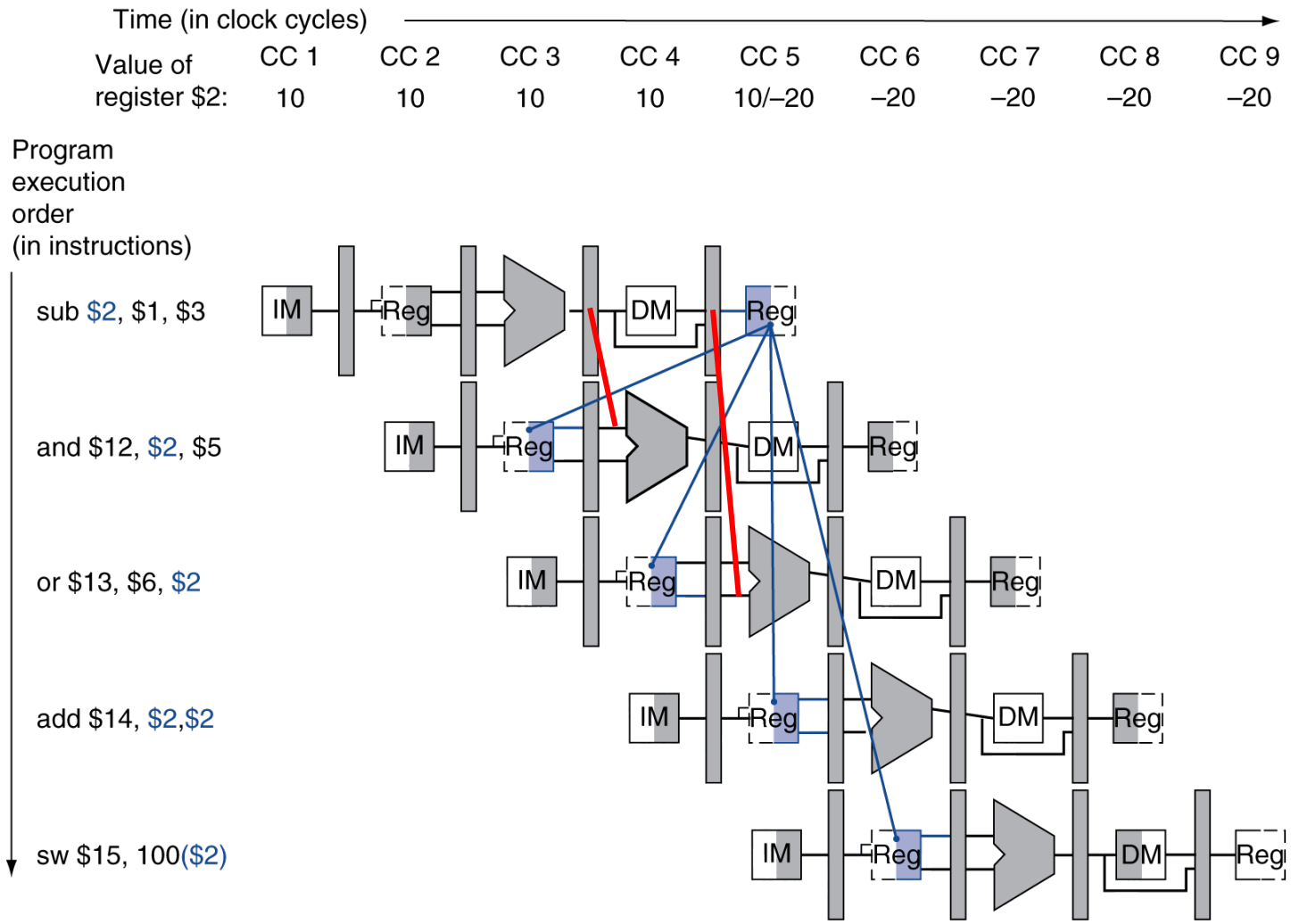
Data Hazards in ALU Instructions

- **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



Detecting the Need to Forward

- **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

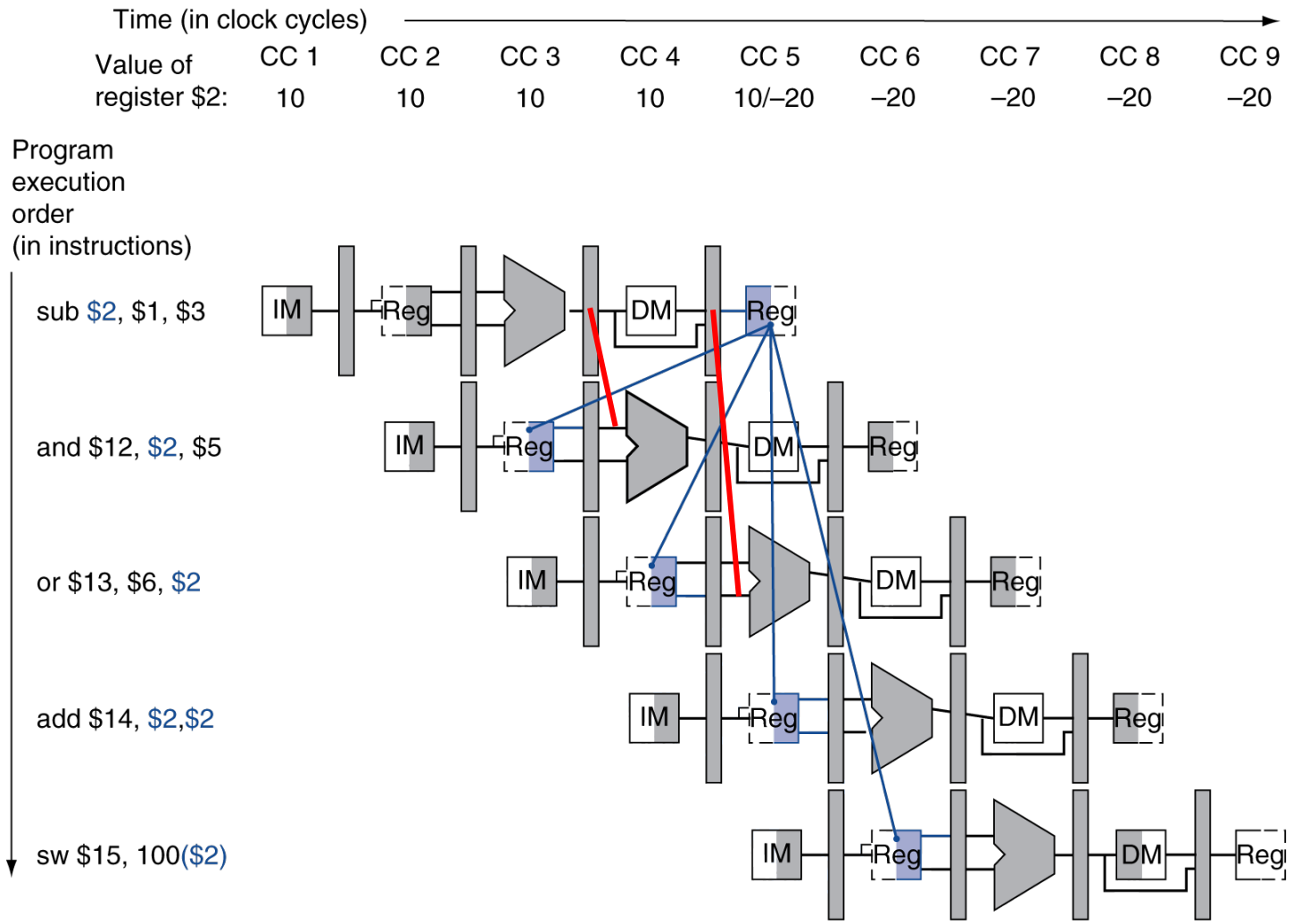
  Fwd from MEM/WB pipeline reg

Data Hazards in ALU Instructions

- **Consider this sequence:**

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

Dependencies & Forwarding



Detecting the Need to Forward

- **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

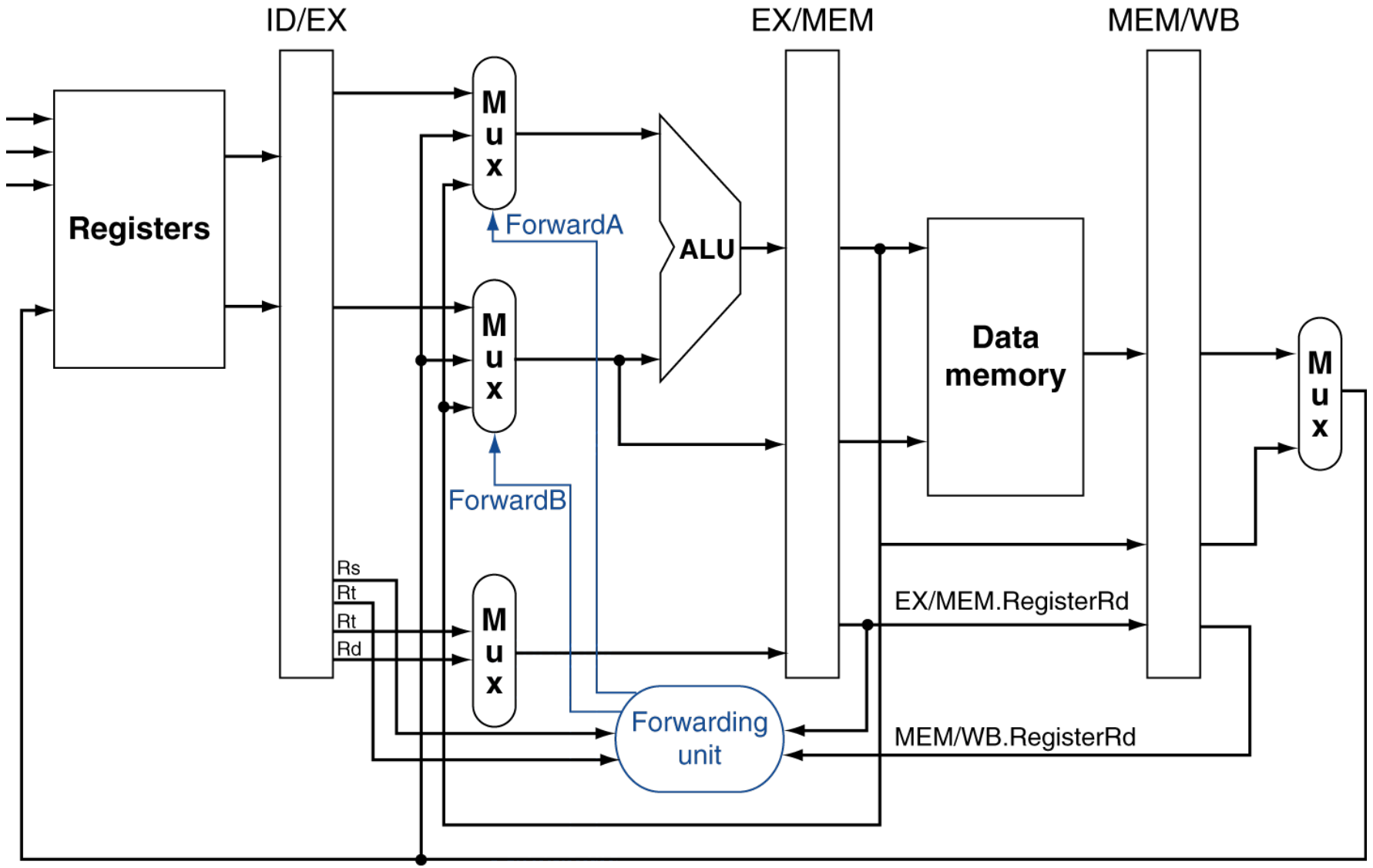
Fwd from EX/MEM pipeline reg

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

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.

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:**

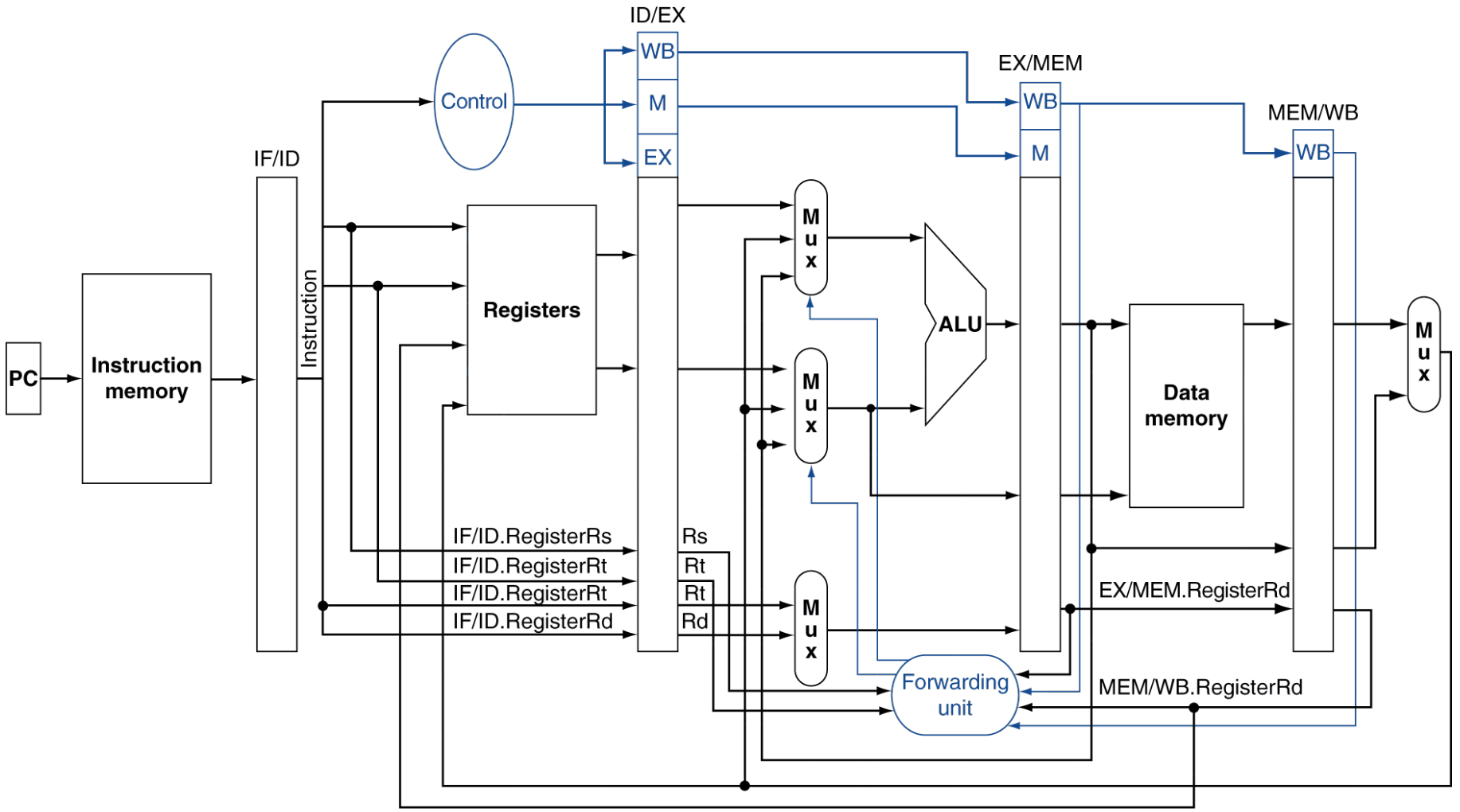
```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

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

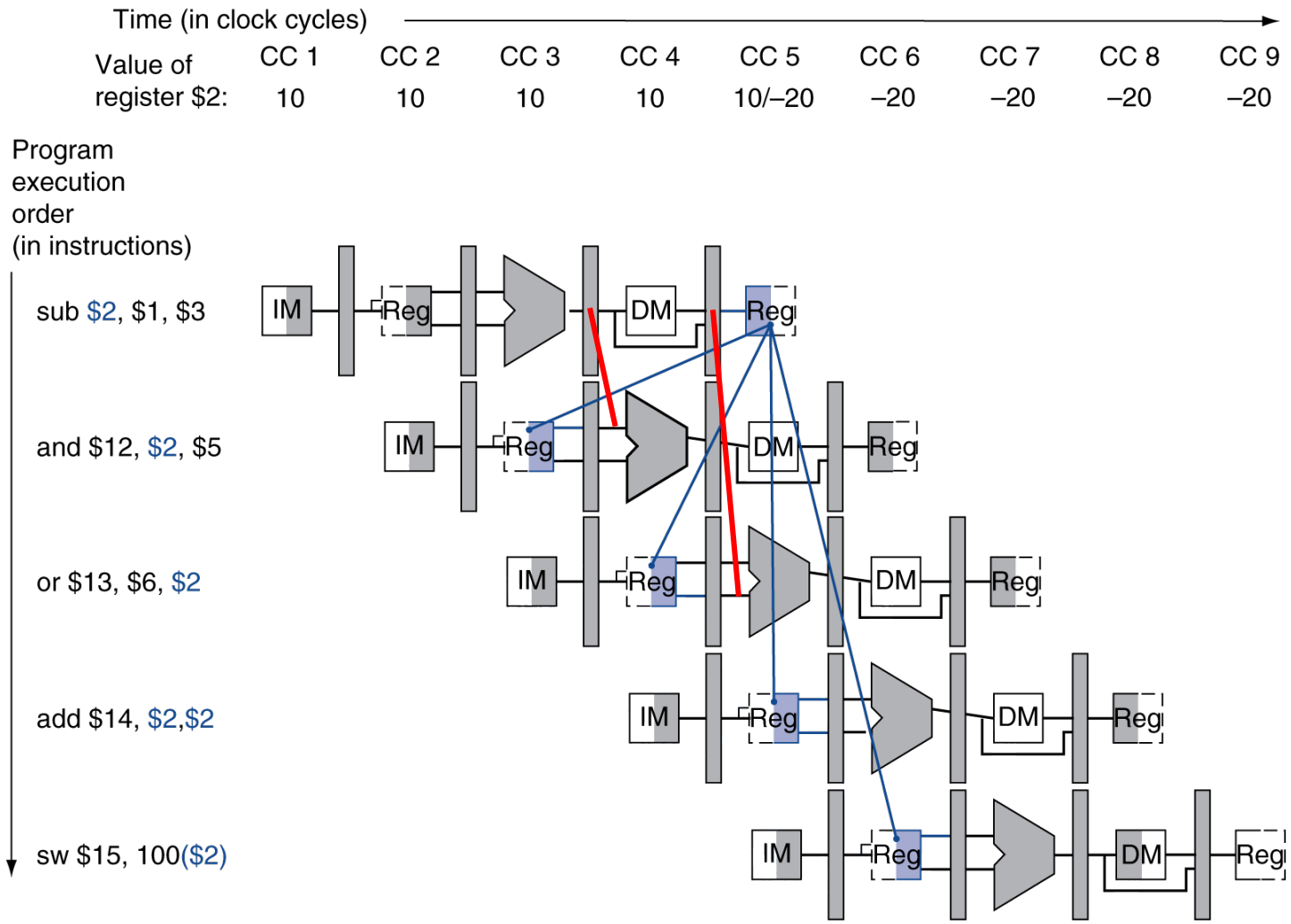
Revised Forwarding Condition

- **MEM hazard**
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 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

Datapath with Forwarding



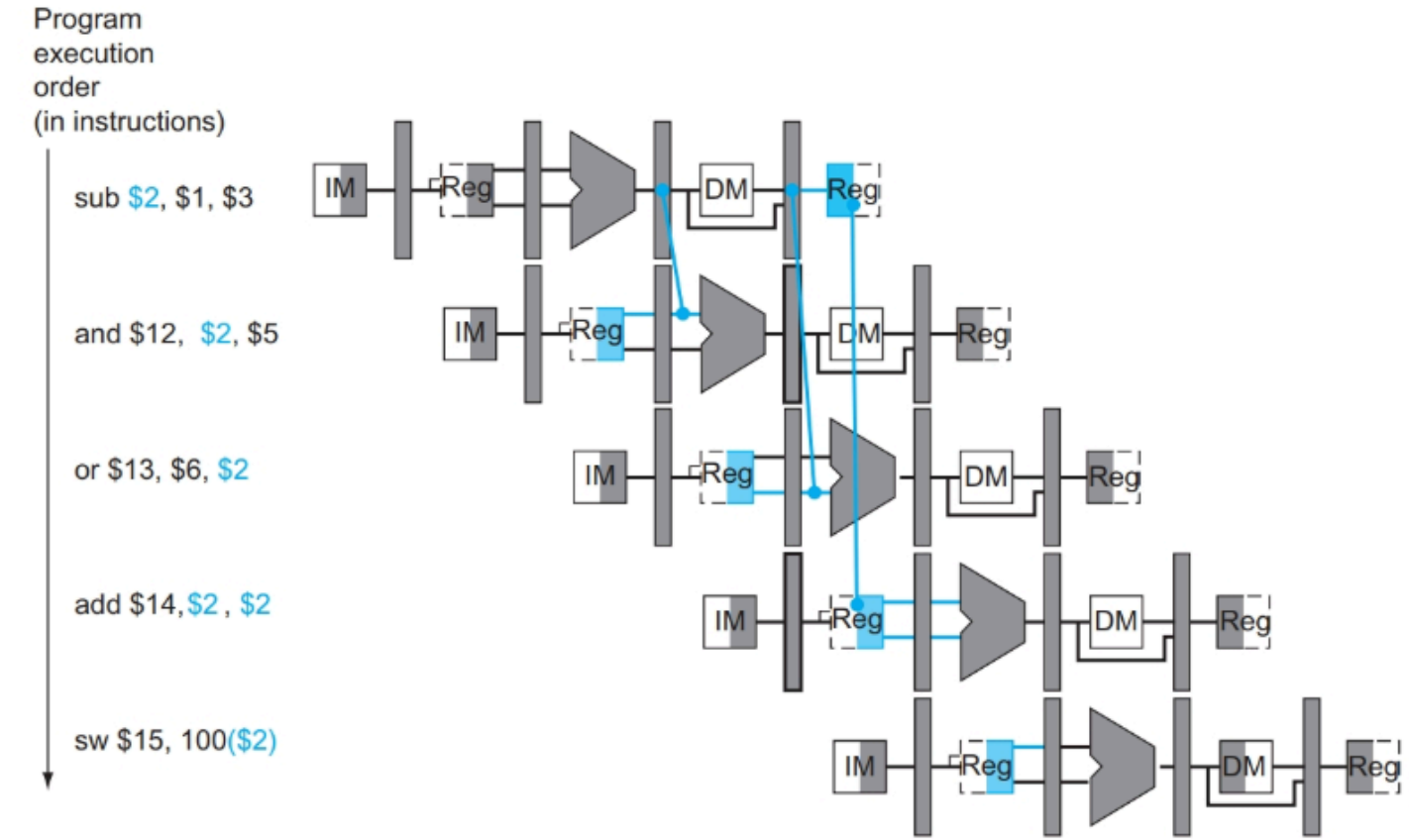
Dependencies & Forwarding



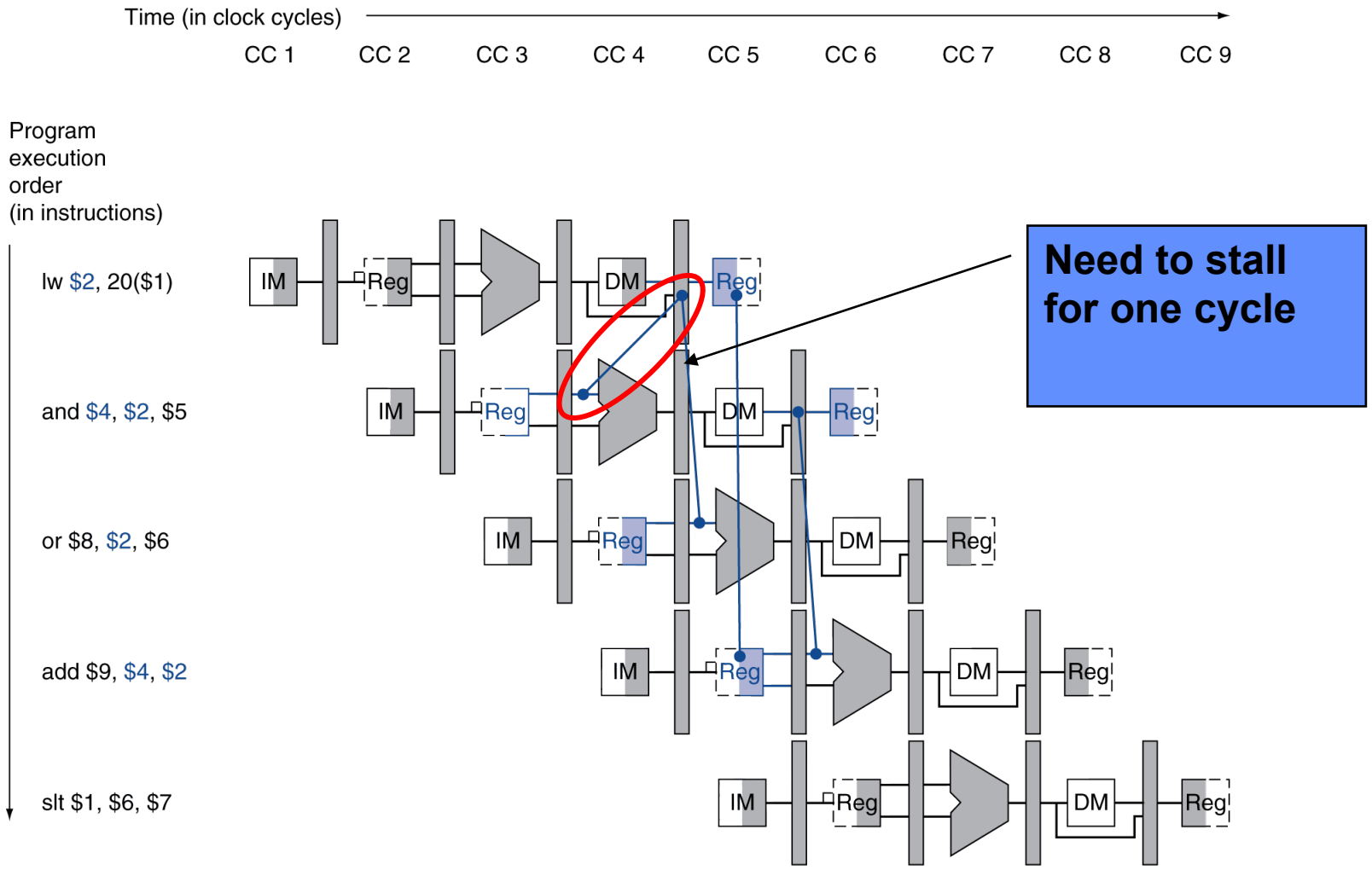
Forwarding Implemented

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



Load-Use Data Hazard



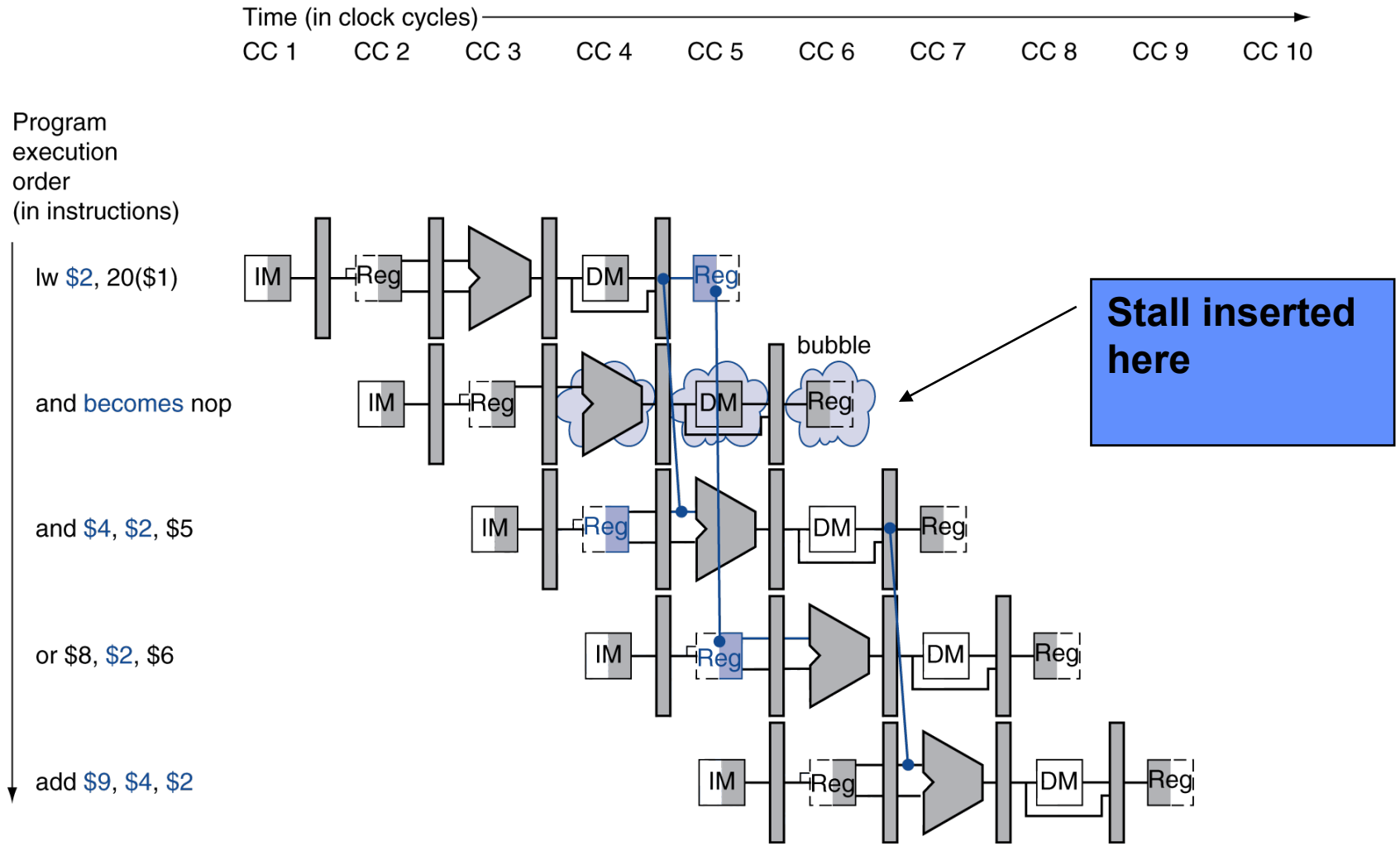
Load-Use Hazard Detection

- **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**

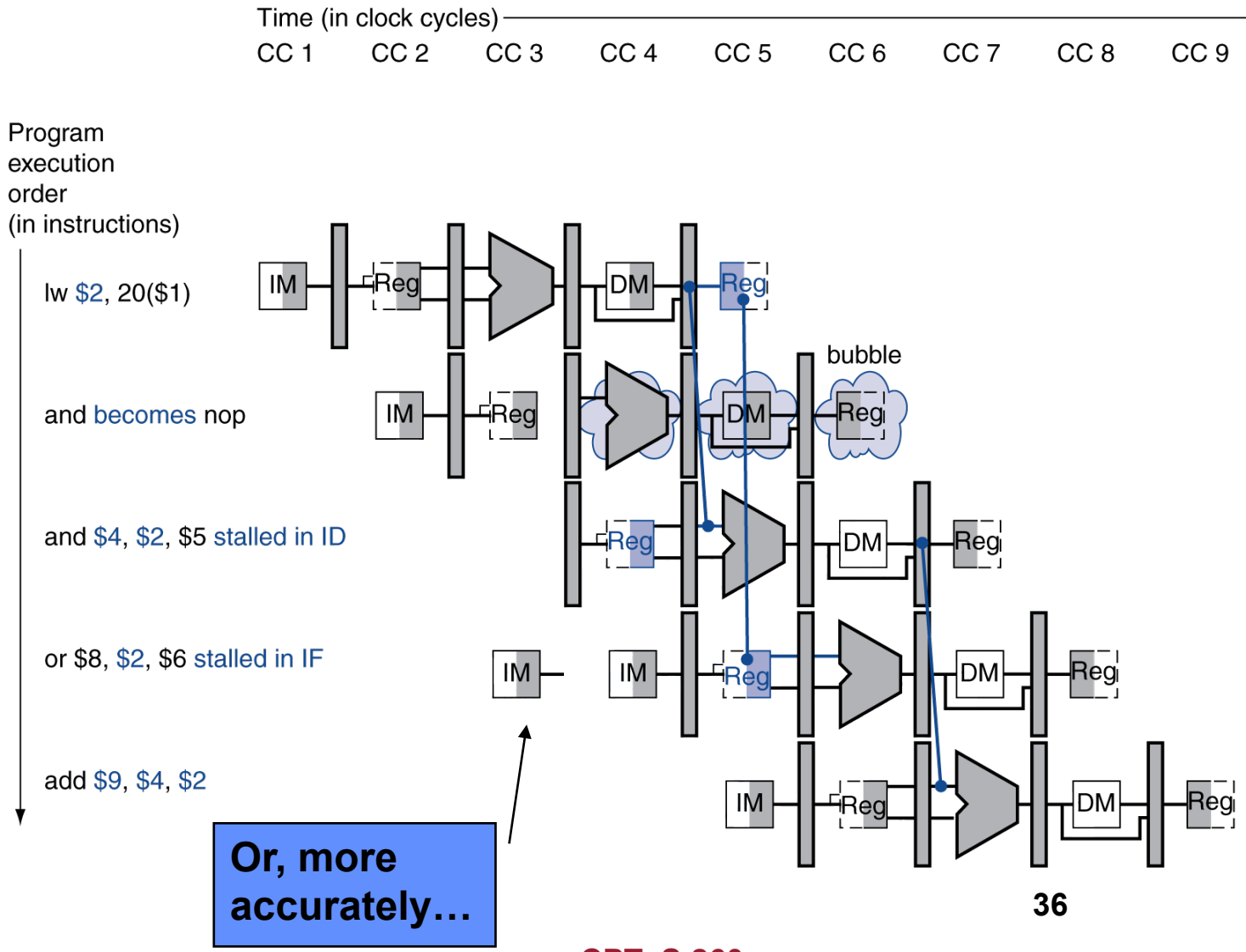
How to Stall the Pipeline

- **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 lw
    - » Can subsequently forward to EX stage

Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline



Datapath with Hazard Detection