Computer Organization and Architecture

Pipelining

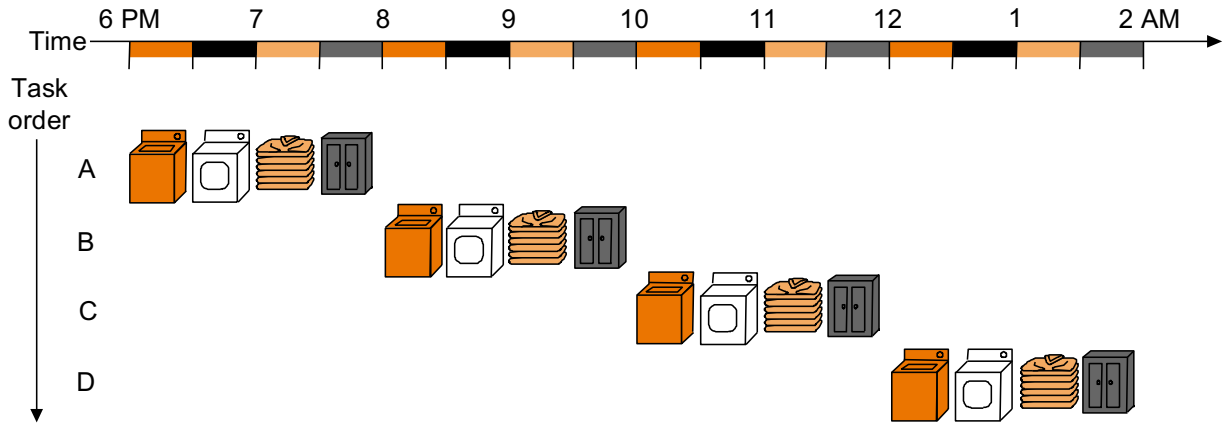
Jeongseob Ahn

Department of Software & Computer Engineering
Ajou University

Outline

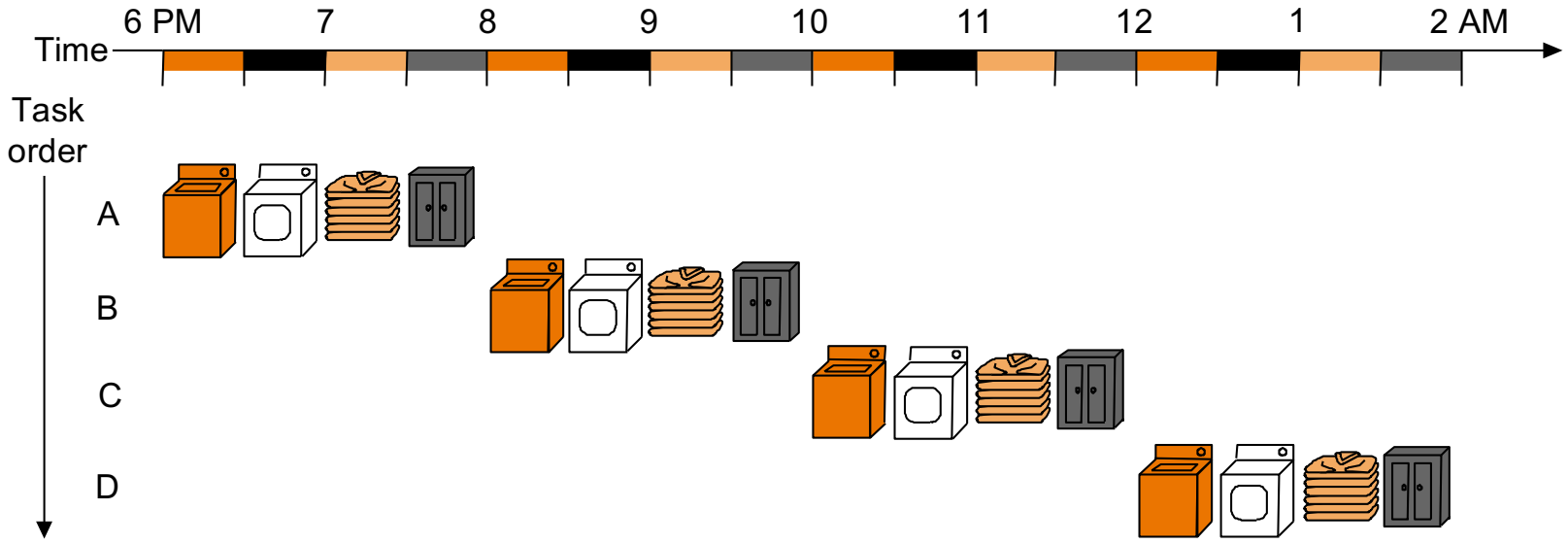
- Basic Pipelining
- Data Hazards
- Control Hazards

The Laundry Analogy



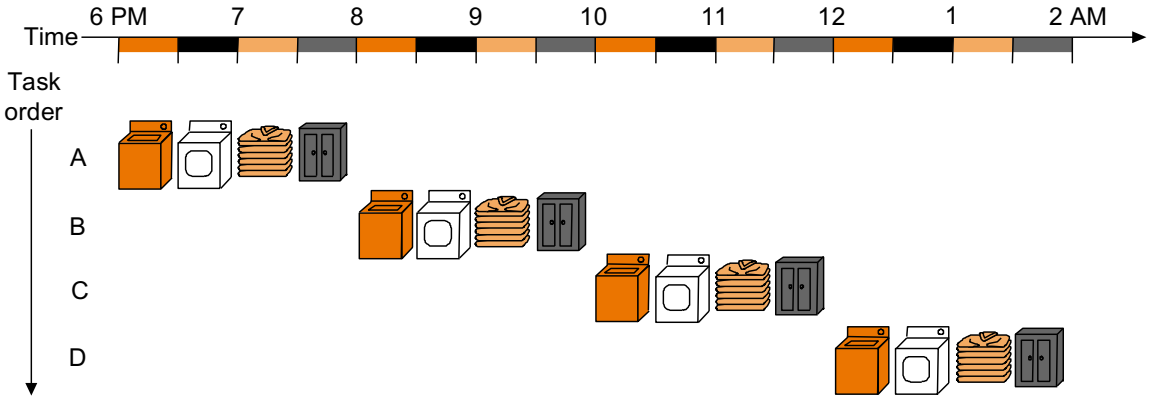
- Washer takes 30 minutes
- Dryer takes 30minutes
- Folder takes 30 minutes
- Mover takes 30 minutes

Sequential Laundry

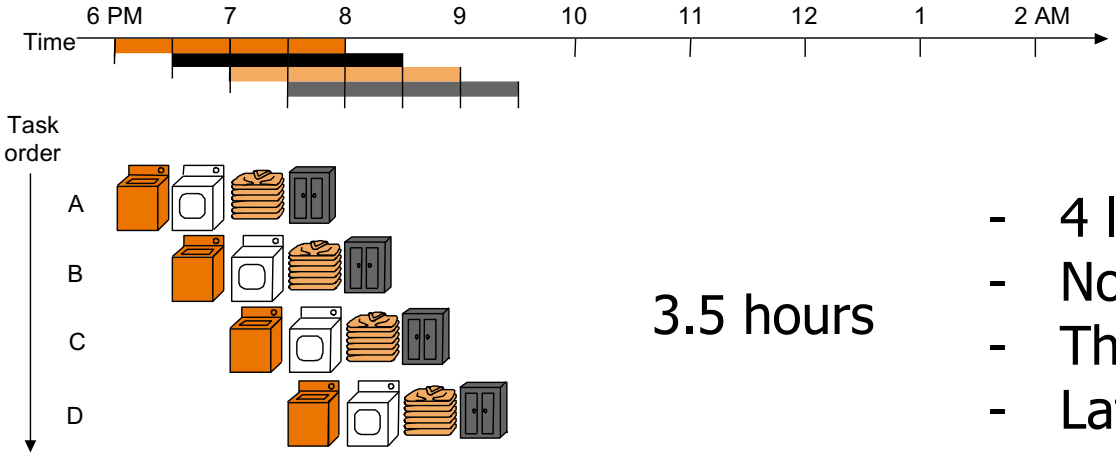


- Sequential laundry takes 8 hours for 4 loads
- But
  - Steps to do a load are sequentially dependent
  - No dependence between different loads
  - Different steps do not share resources

Pipelining Multiple Loads of Laundry



8 hours



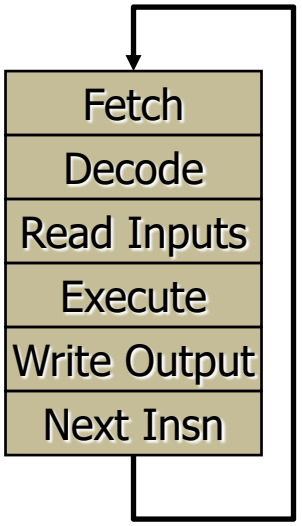
3.5 hours

- 4 loads of laundry in parallel
- No additional resources
- Throughput increased by 4
- Latency per load is the same

Pipelining

- Important performance technique
  - Improves instruction throughput, not instruction latency
- Latency vs. throughput: two views of performance
  - (1) at the program level and (2) at the instruction level
- Single instruction latency
  - Doesn't matter: programs comprised of billions of instructions
  - Difficult to reduce anyway
- Goal is to make programs, not individual instructions, go faster
  - Instruction throughput → program latency
  - Key: exploit inter-instruction parallelism

Recall: Simple MIPS Execution Model



- Five stages, one step per stage
  - IF: Instruction fetch from memory
  - ID: Instruction decode & register read
  - EX: Execute operation or calculate address
  - MEM: Access memory operand
  - WB: Write result back to register

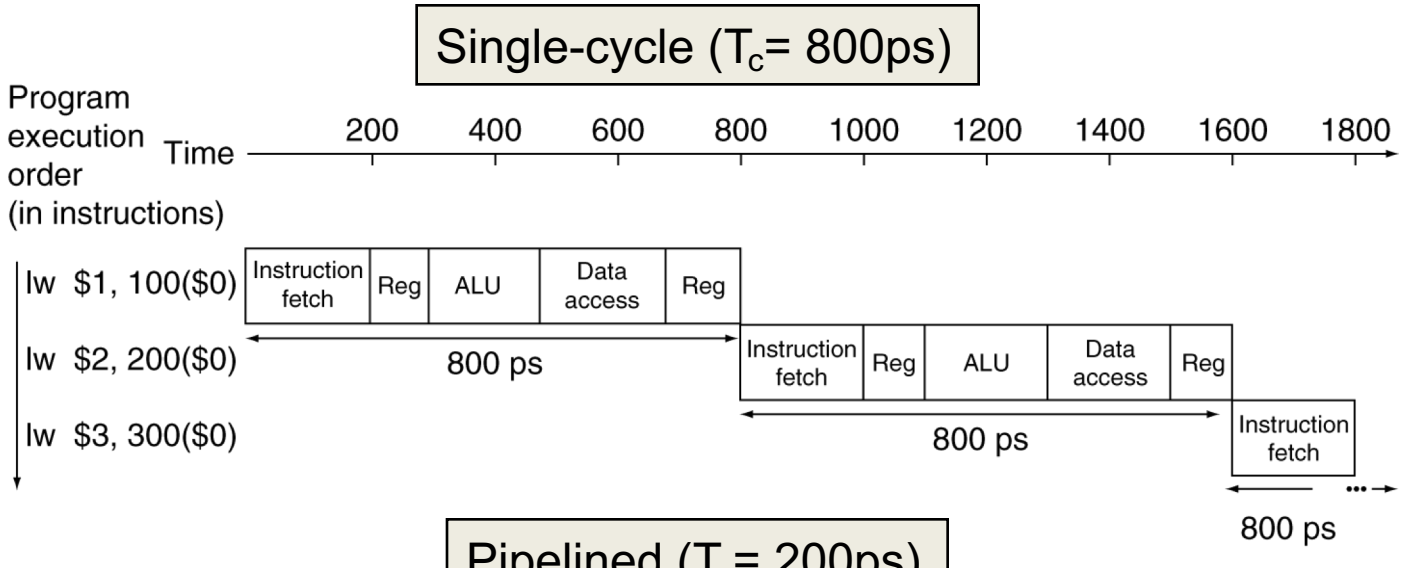
Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages

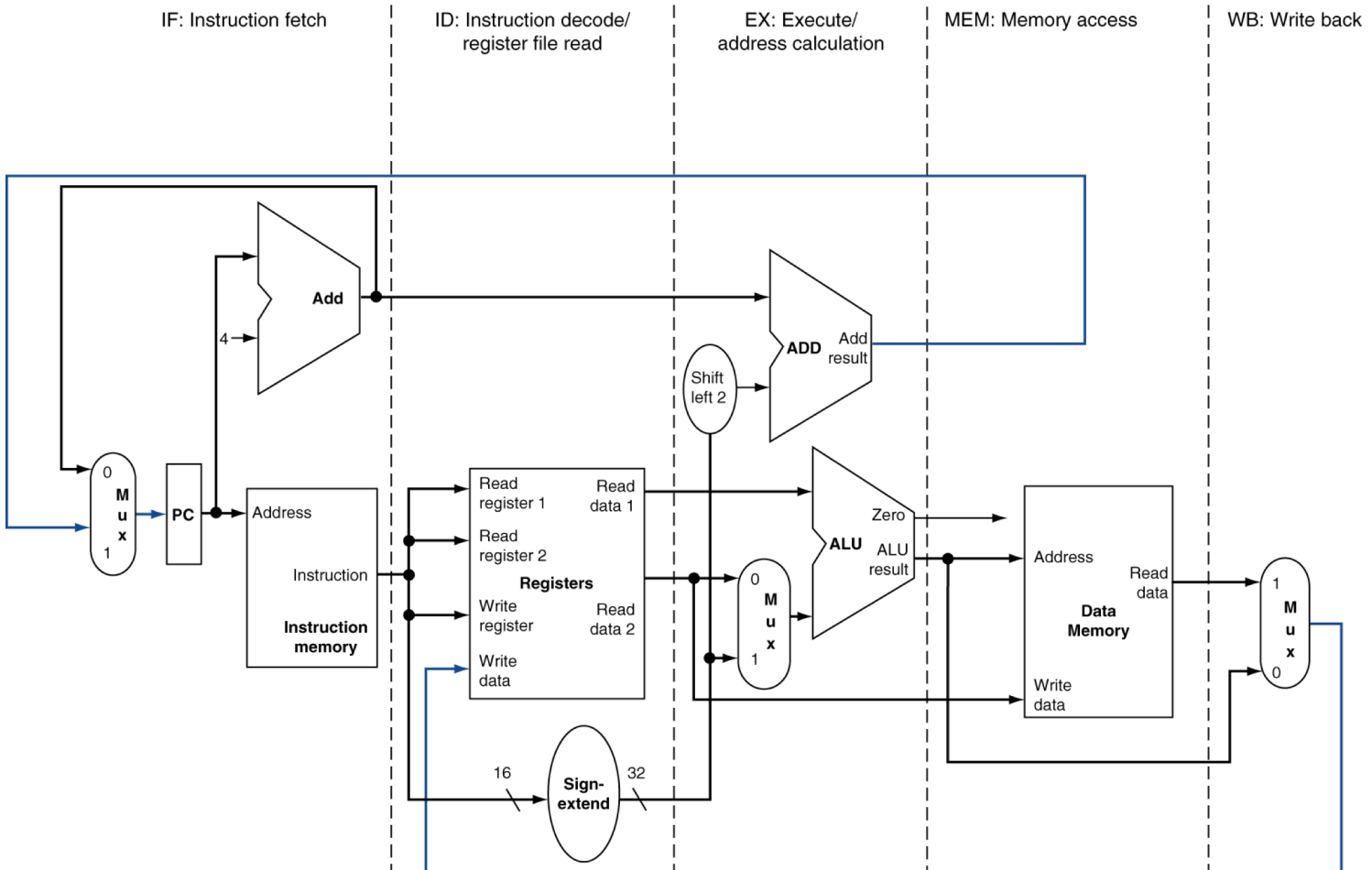
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance



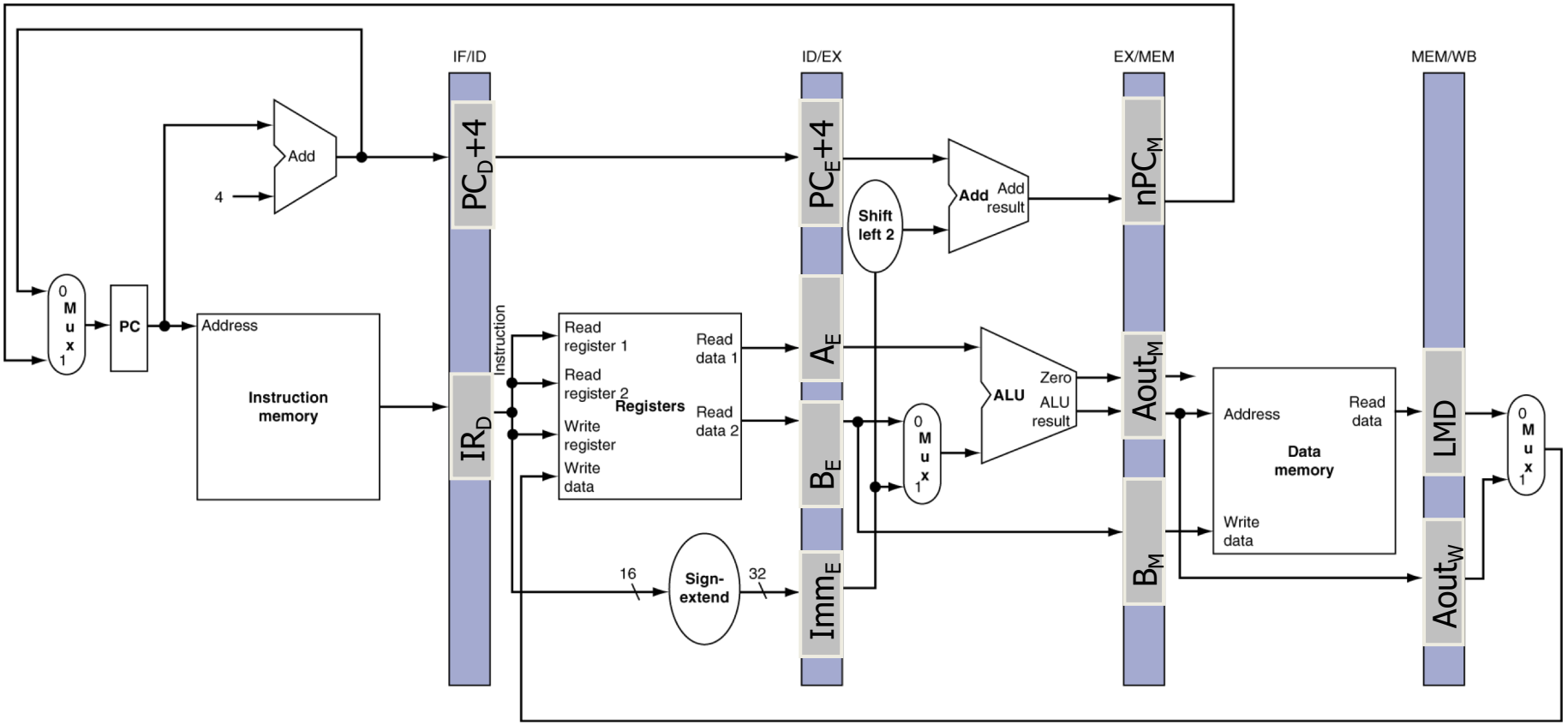
MIPS Pipelined Datapath



$T_{insn\text{-}mem}$ $T_{regfile}$ T_{ALU} $T_{data\text{-}mem}$ $T_{regfile}$

Pipeline registers

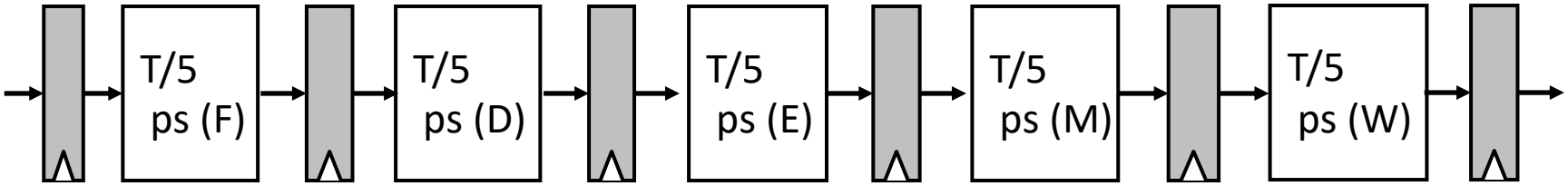
- Need registers between stages
  - To hold information produced in previous cycle



Ideal Pipelining

- Non-pipelined version

- 5-stage pipelined version



BW=~(5/T)

Throughput and Cost

- Register delay
  - BW = 1/(T+S) where S = Register delay

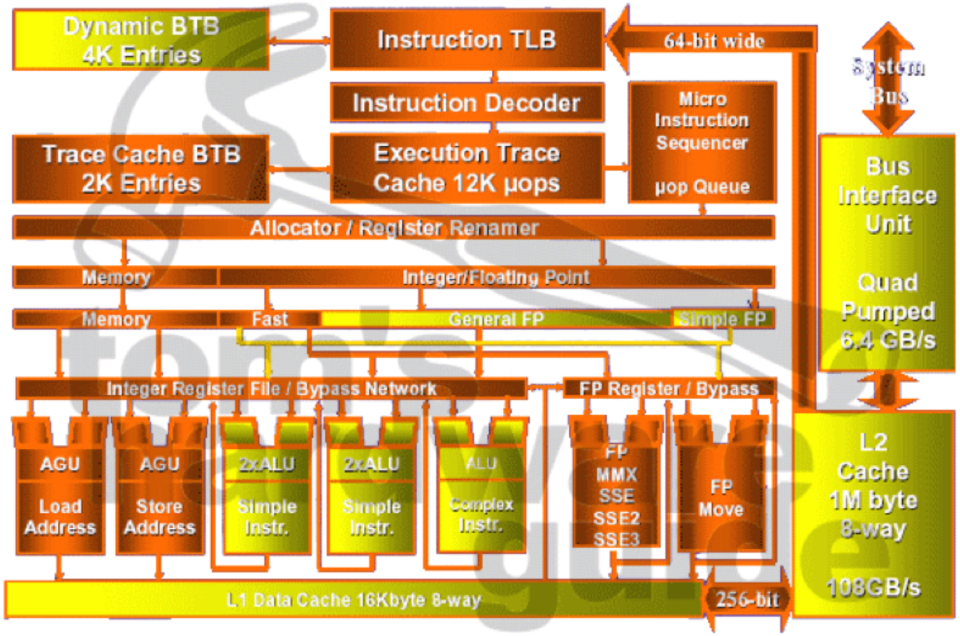
- Register cost
  - Cost = G+L where L = Register cost

Intel's New Weapon: Pentium 4 Prescott

by Patrick Schmid February 1, 2004 at 7:00 PM

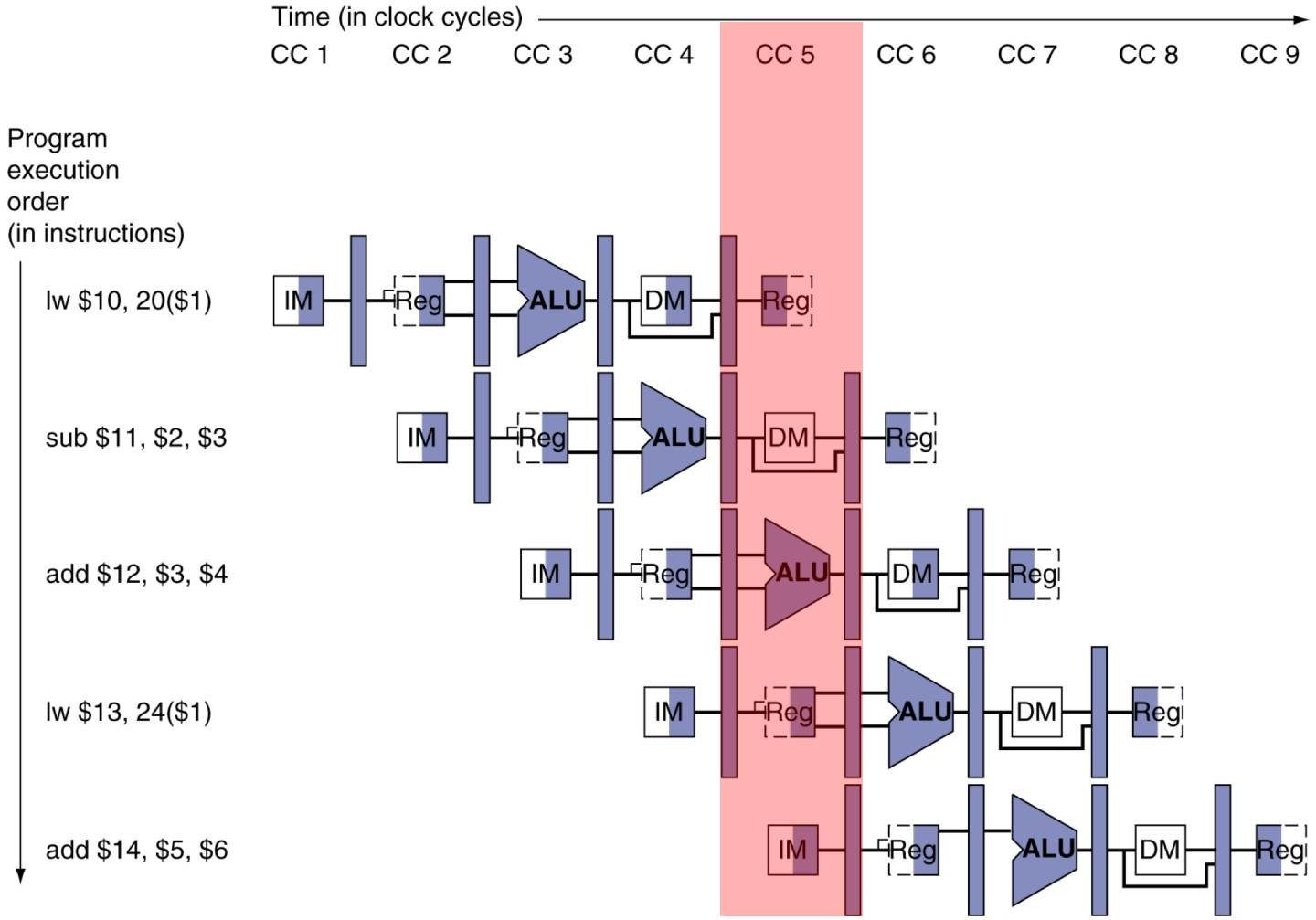
Page 5: **NetBurst Architecture: Now 31 Pipeline Stages**

NetBurst Architecture: Now 31 Pipeline Stages



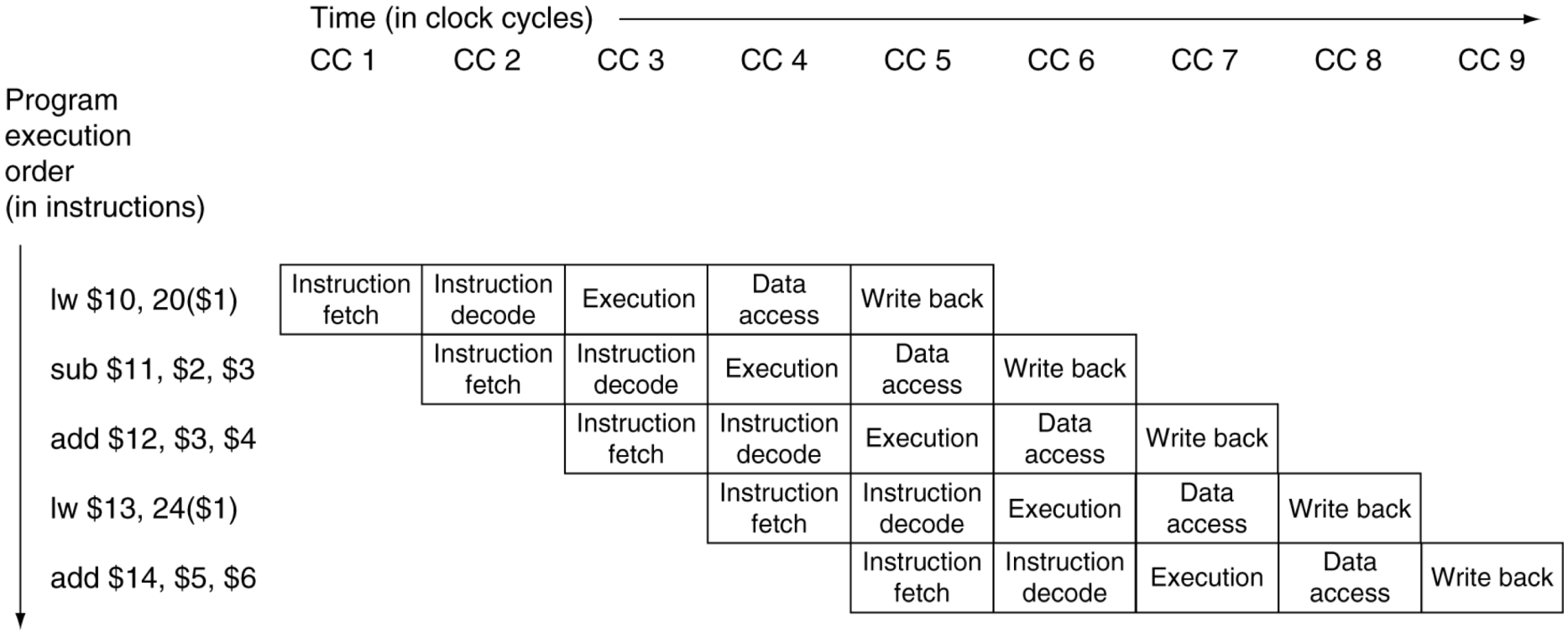
Multi-Cycle Pipeline Diagram

- Form showing resource usage



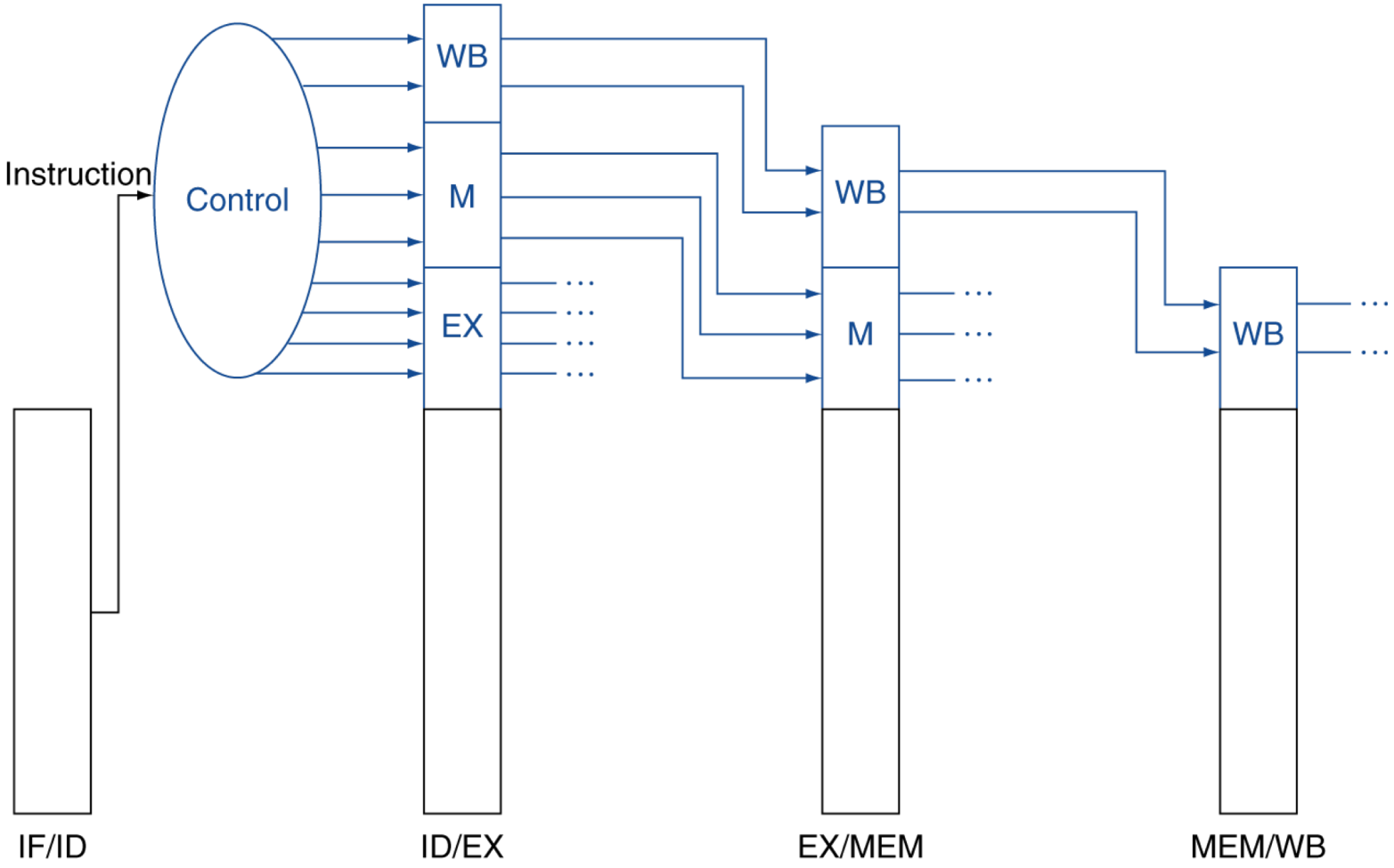
Multi-Cycle Pipeline Diagram

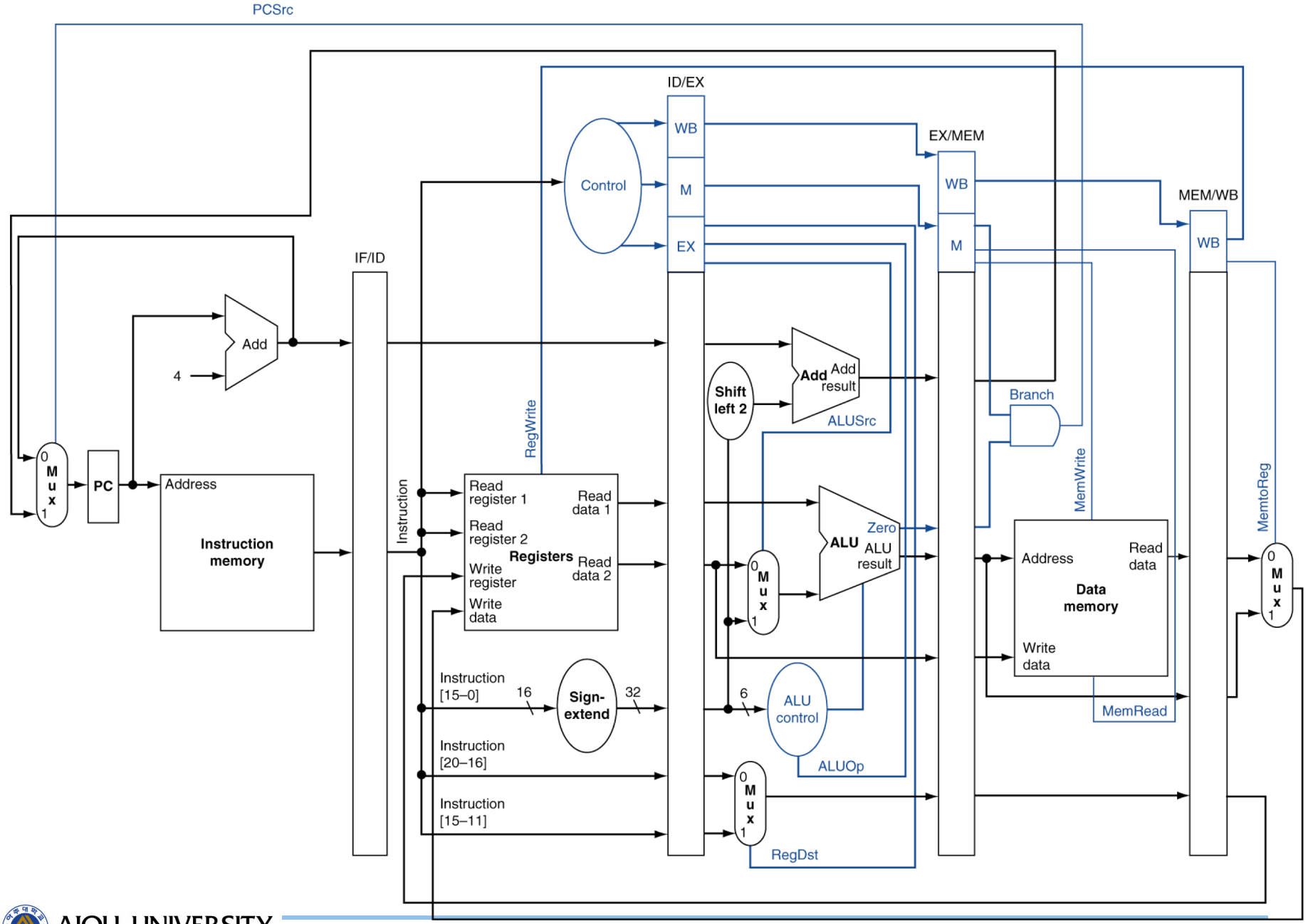
- Traditional form



Pipelined Control

- Control signals derived from instruction




PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Control
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALUSrc
Branch
MemWrite
MemtoReg
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15–0]
16
Sign-extend
32
6
ALU control
ALUOp
Instruction [20–16]
Instruction [15–11]
RegDst

Pipeline Hazards

- Situations that prevent starting the next instruction in the next cycle

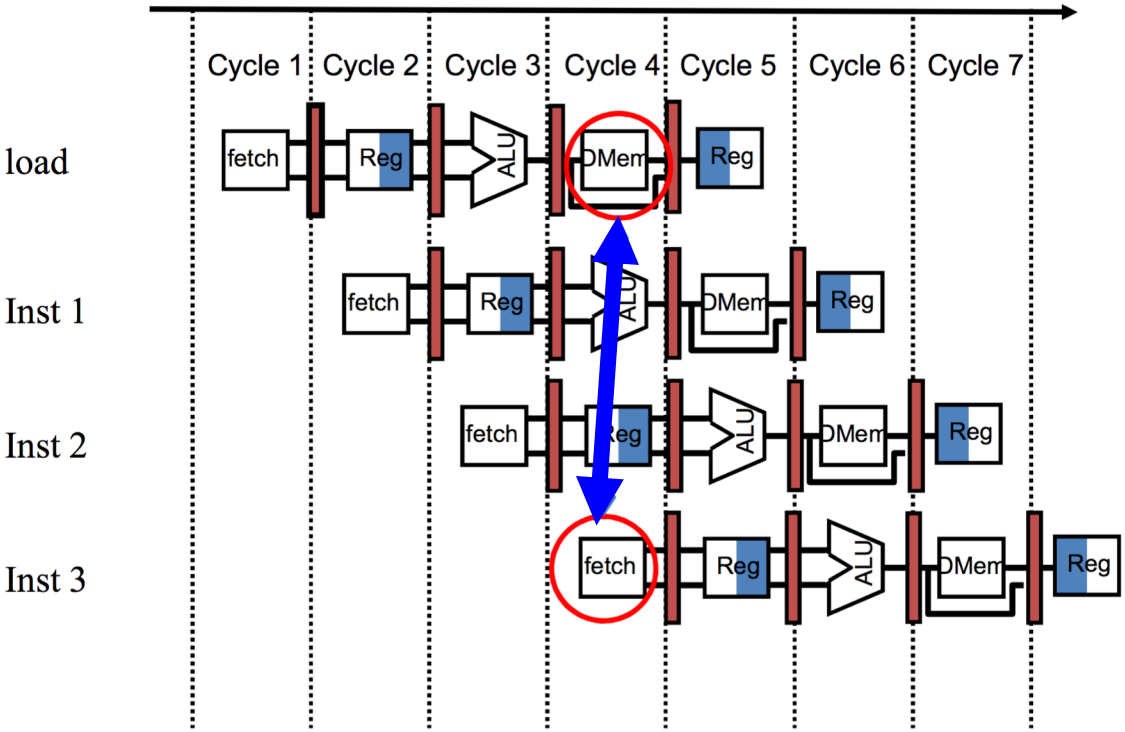
- *Structural hazards*
  - Conflict for use of a resource

- *Data hazards*
  - Need to wait for previous instruction to complete its data read/write

- *Control hazards*
  - Deciding on control action depends on previous instruction

- General solution to hazards: *pipeline stall*
  - Stop the younger instructions till hazard is resolved
  - Make Pipeline imperfect: loss of performance

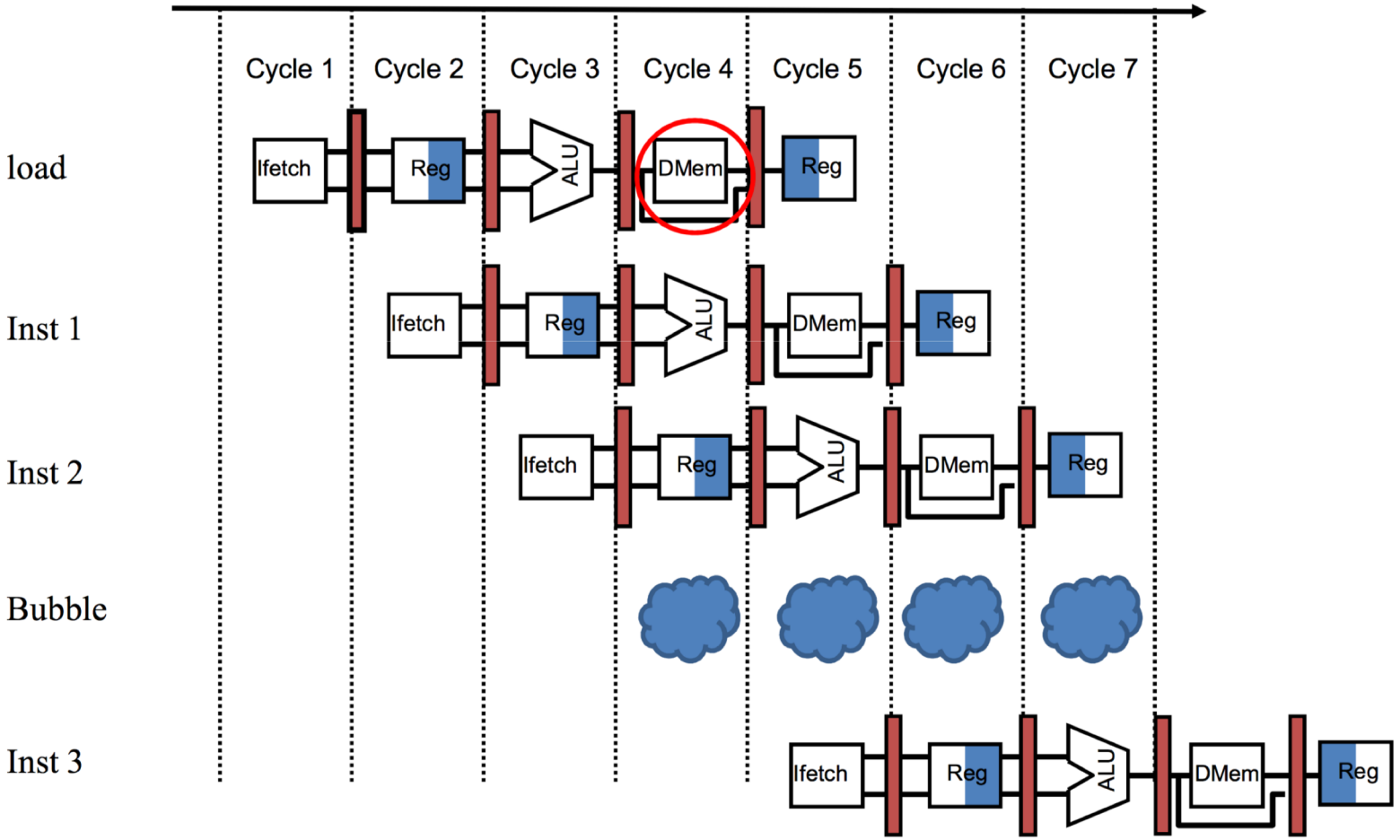
Example: Structural Hazard



- Suppose: single-ported memory for both instruction and data
- Solution: *stall pipeline* OR *add one more port*

Example: Structural Hazard

- Stalling pipe to avoid conflict access to memory



Data Hazards in ALU Instructions

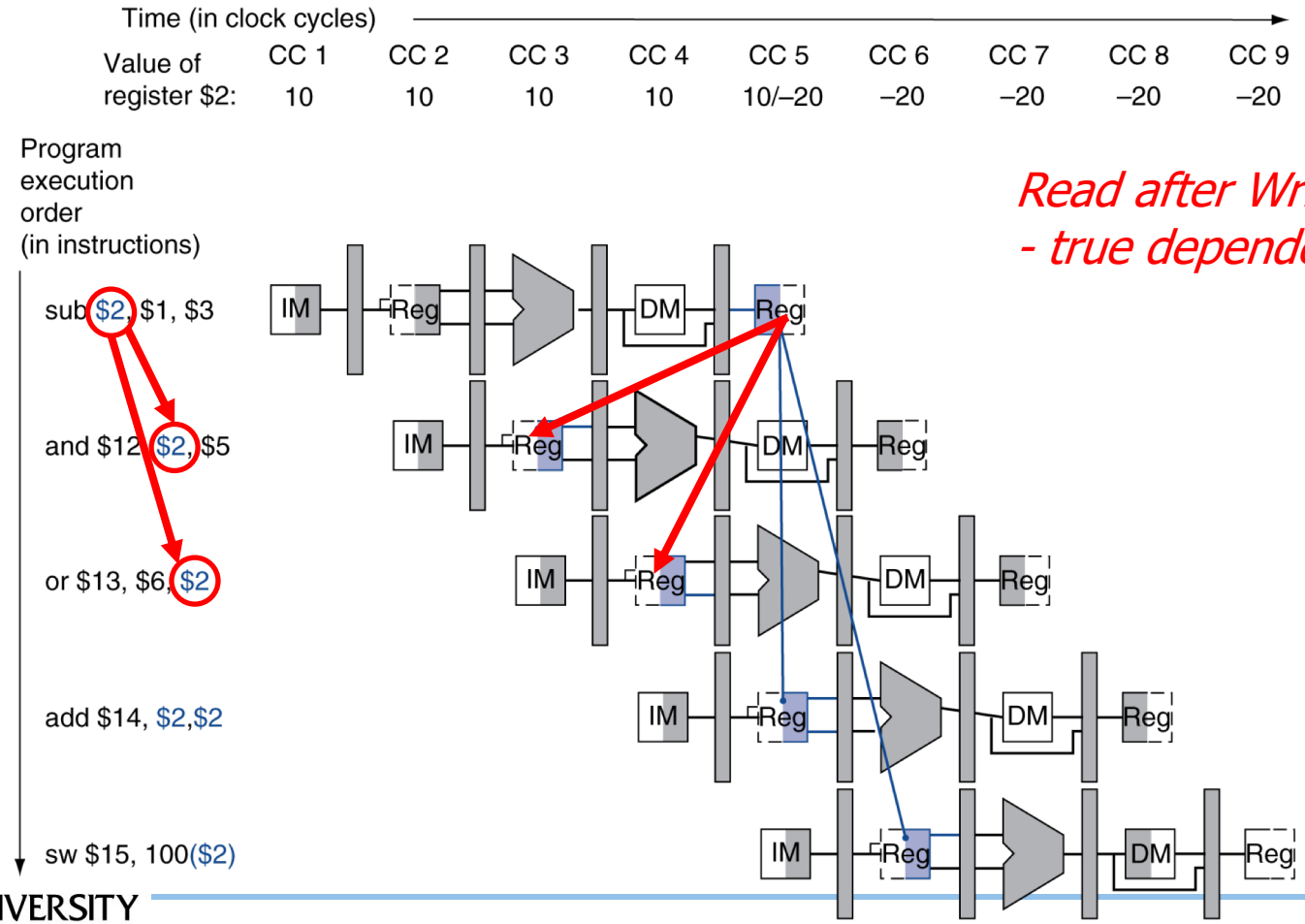
- Consider this sequence:

```
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?

Data Hazard: RAW

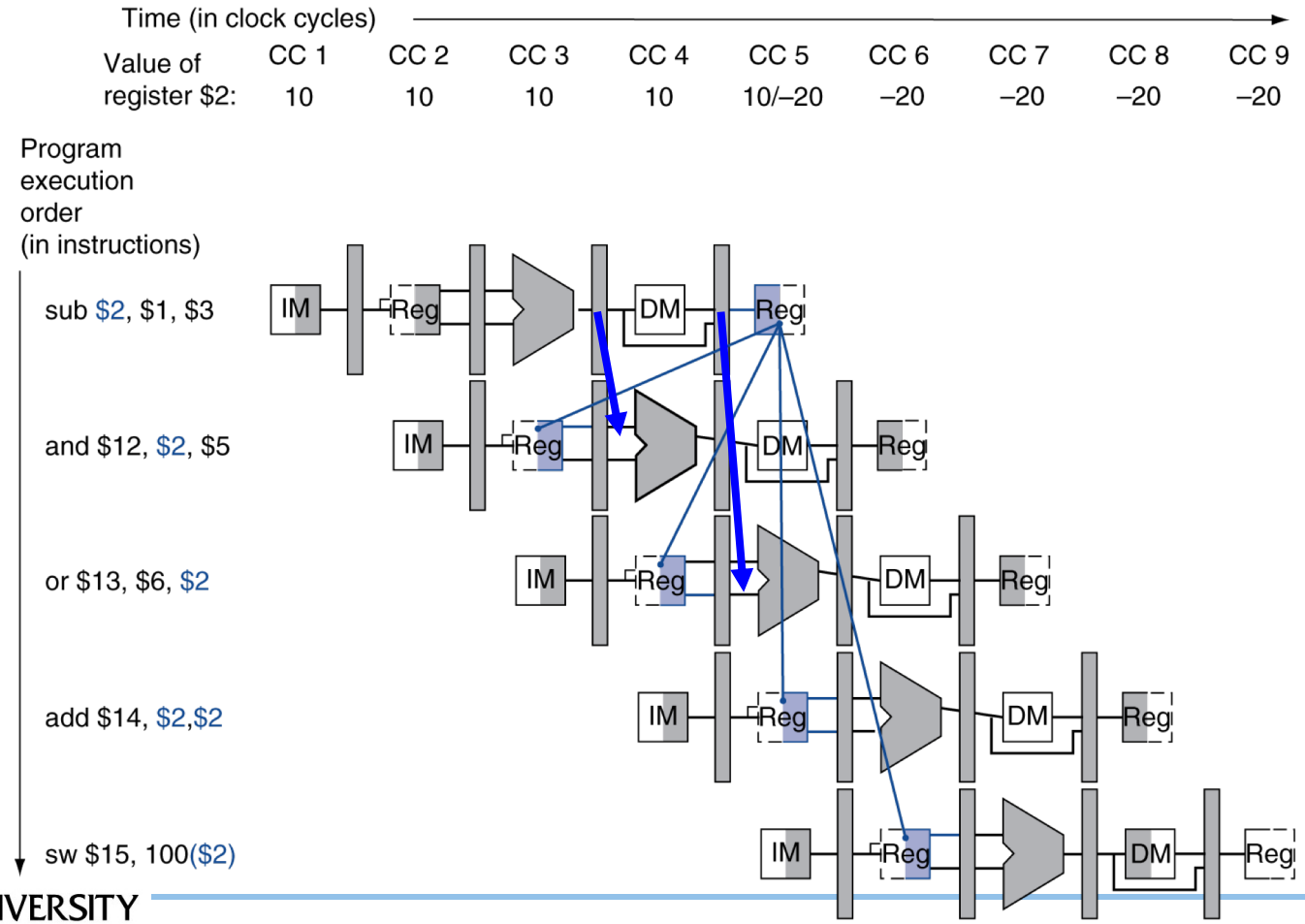
- An instruction depends on completion of data access by previous instruction



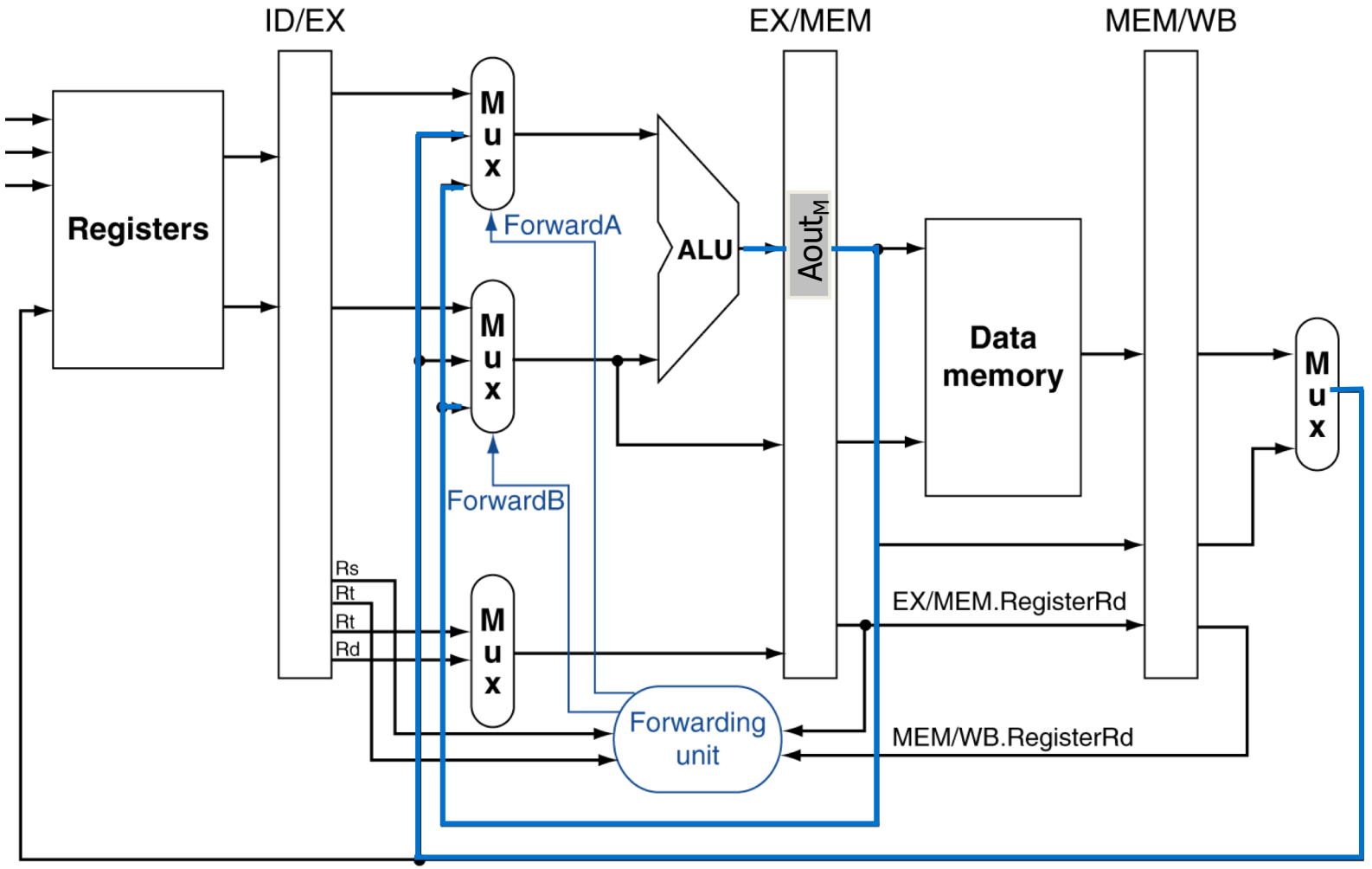
Read after Write (RAW)
- true dependence

Solution: Bypassing (or Forwarding)

- The result of `sub` instruction can be known at the EX/MEM stage and MEM/WB stage

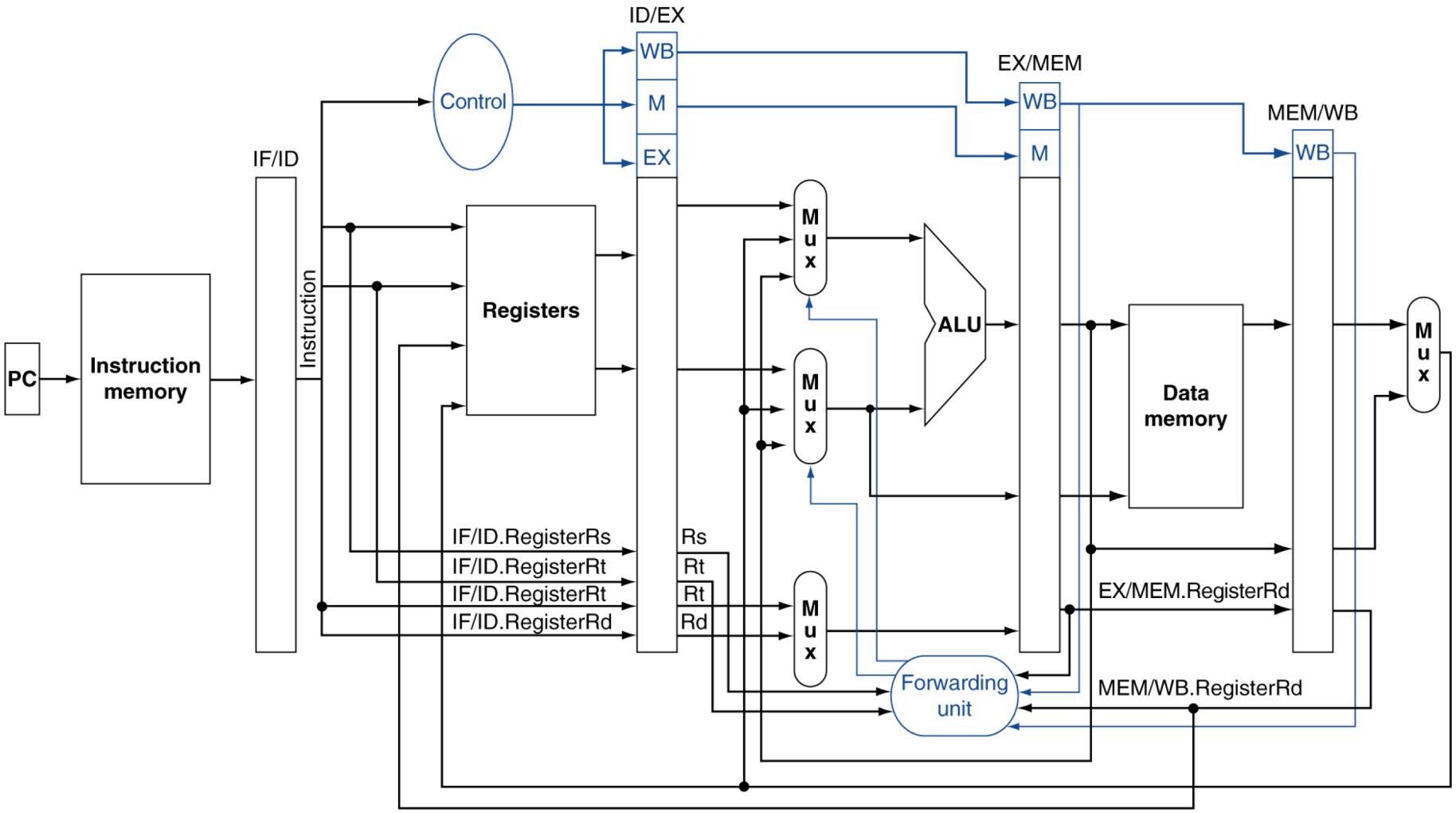


Forwarding Paths



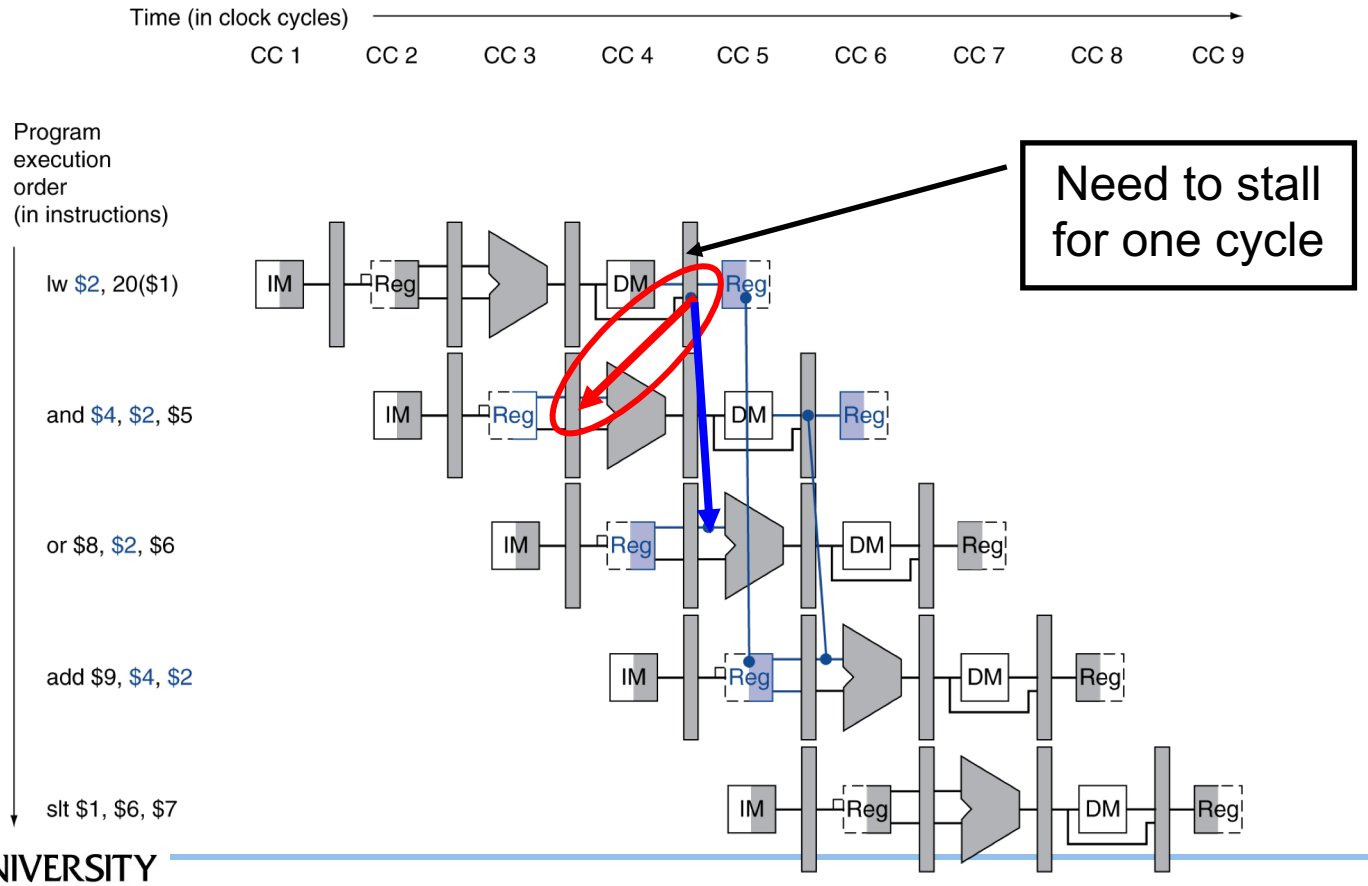
b. With forwarding

Datapath with Forwarding

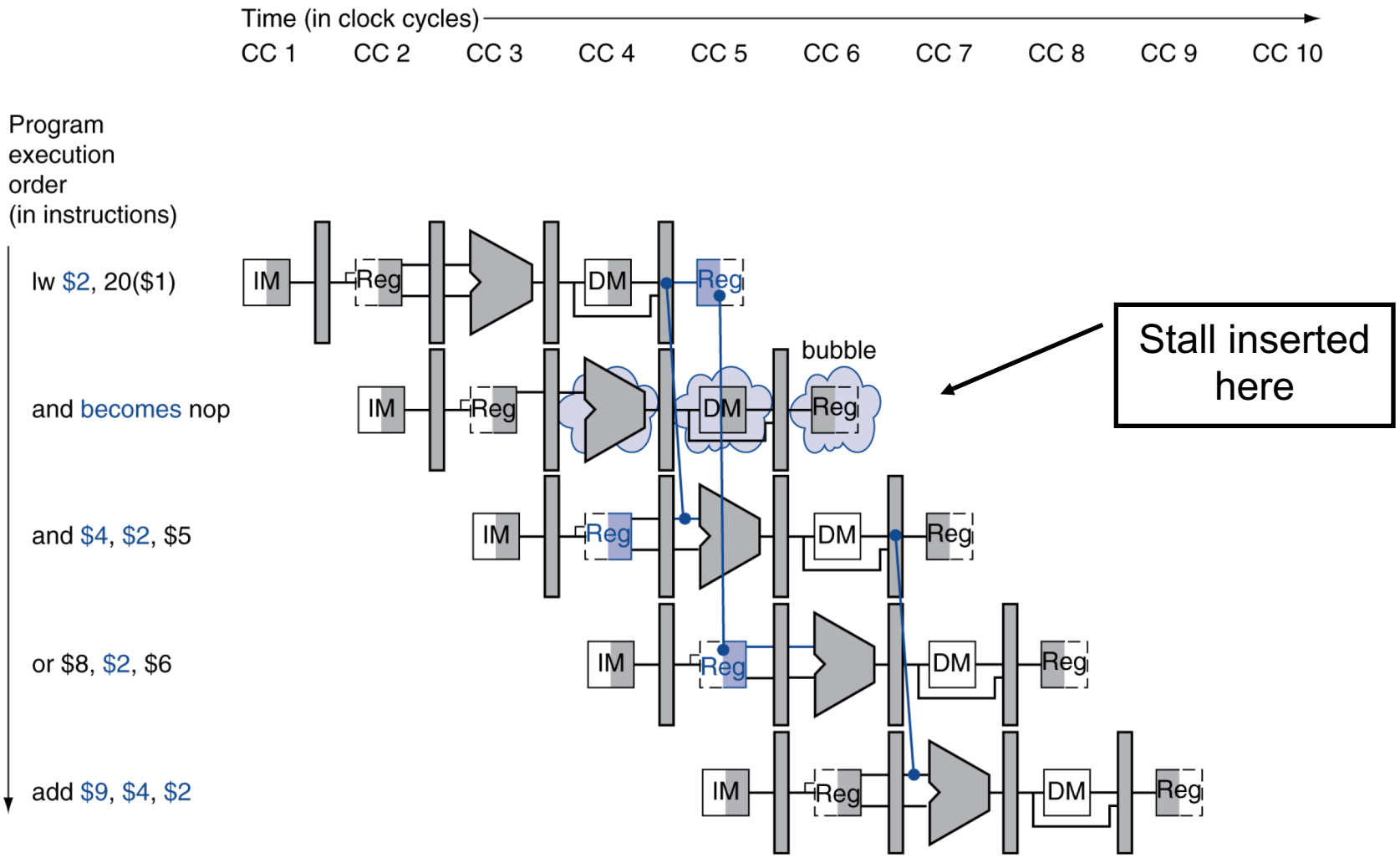


Data Hazard: Load-Use

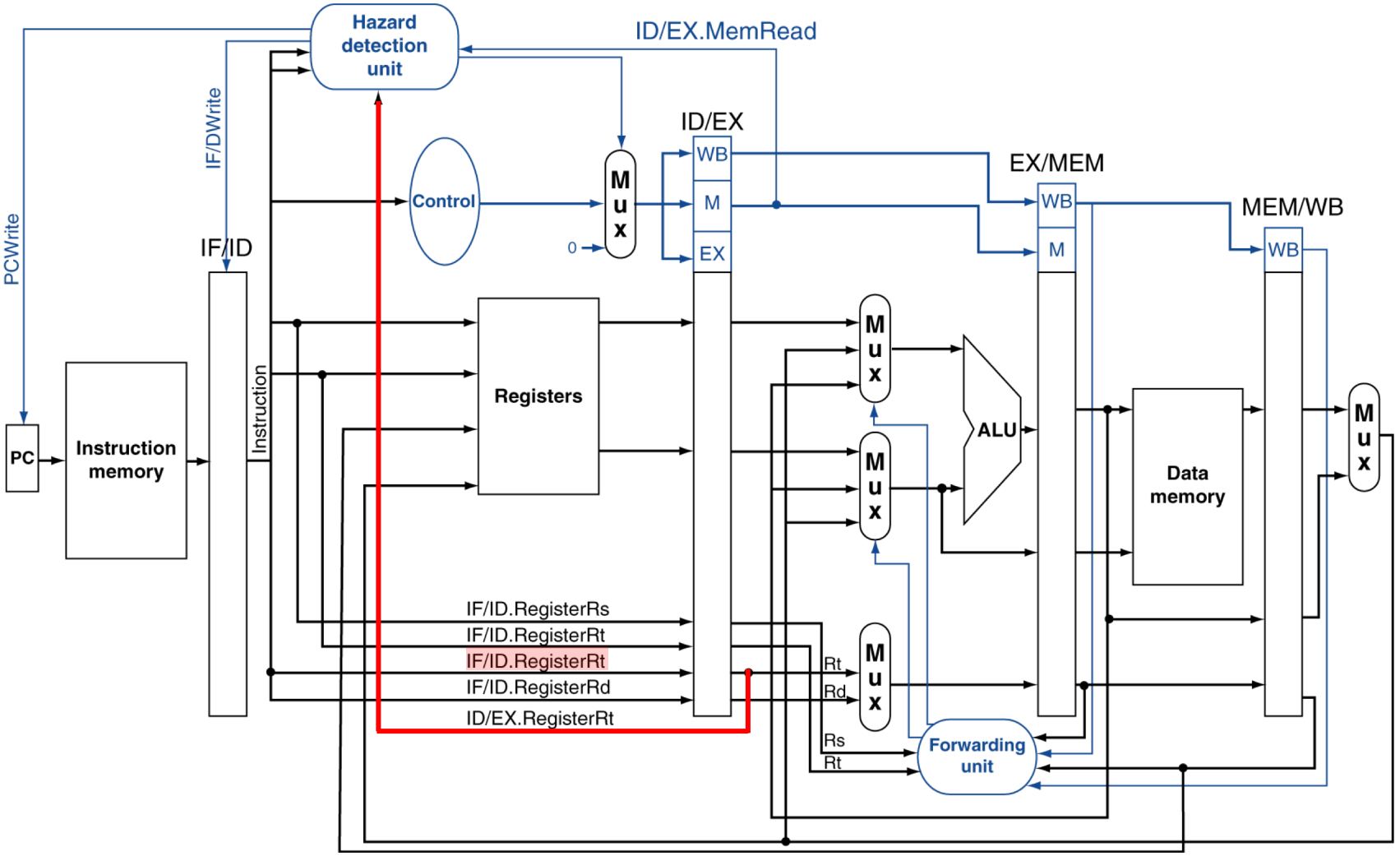
- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - In `lw` instruction, the data is prepared after `MEM` stage



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
  - Require knowledge of the pipeline structure

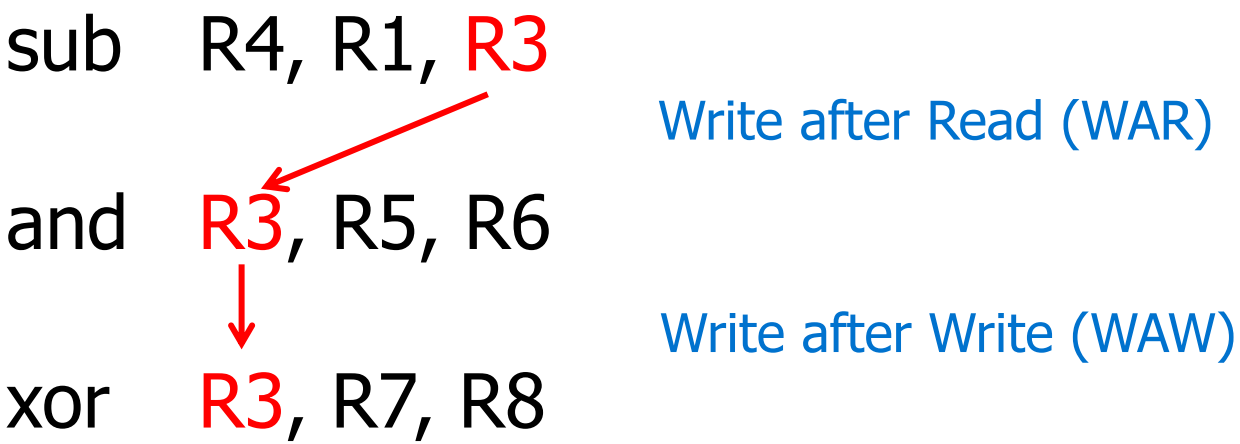
Can Software Do Something?

- Reorder code to avoid use of load result in the next instruction
- C code

  A = B + E;

  C = B + F;

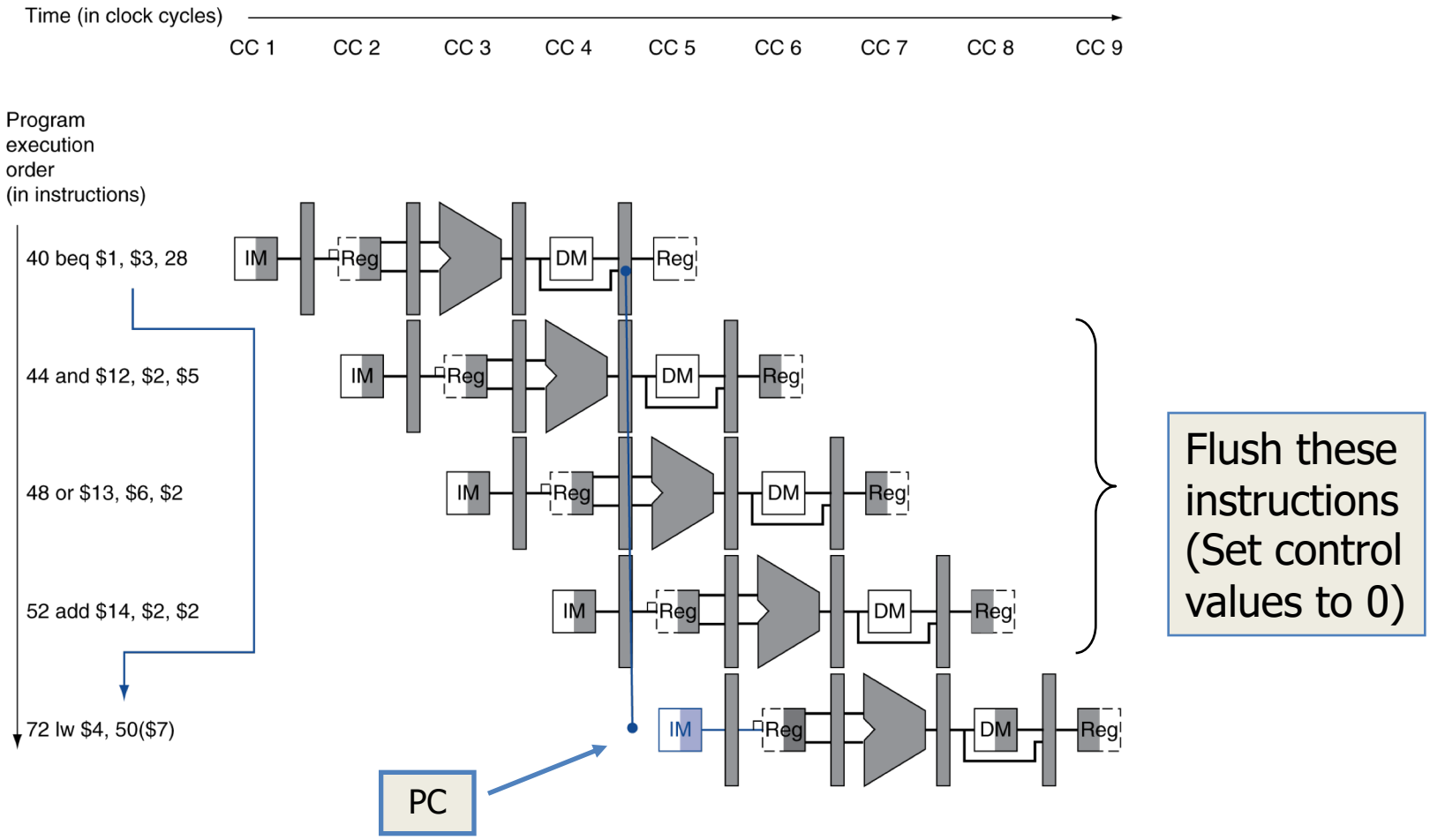
Other Data Hazards



- When can WAR and WAW cause performance loss?

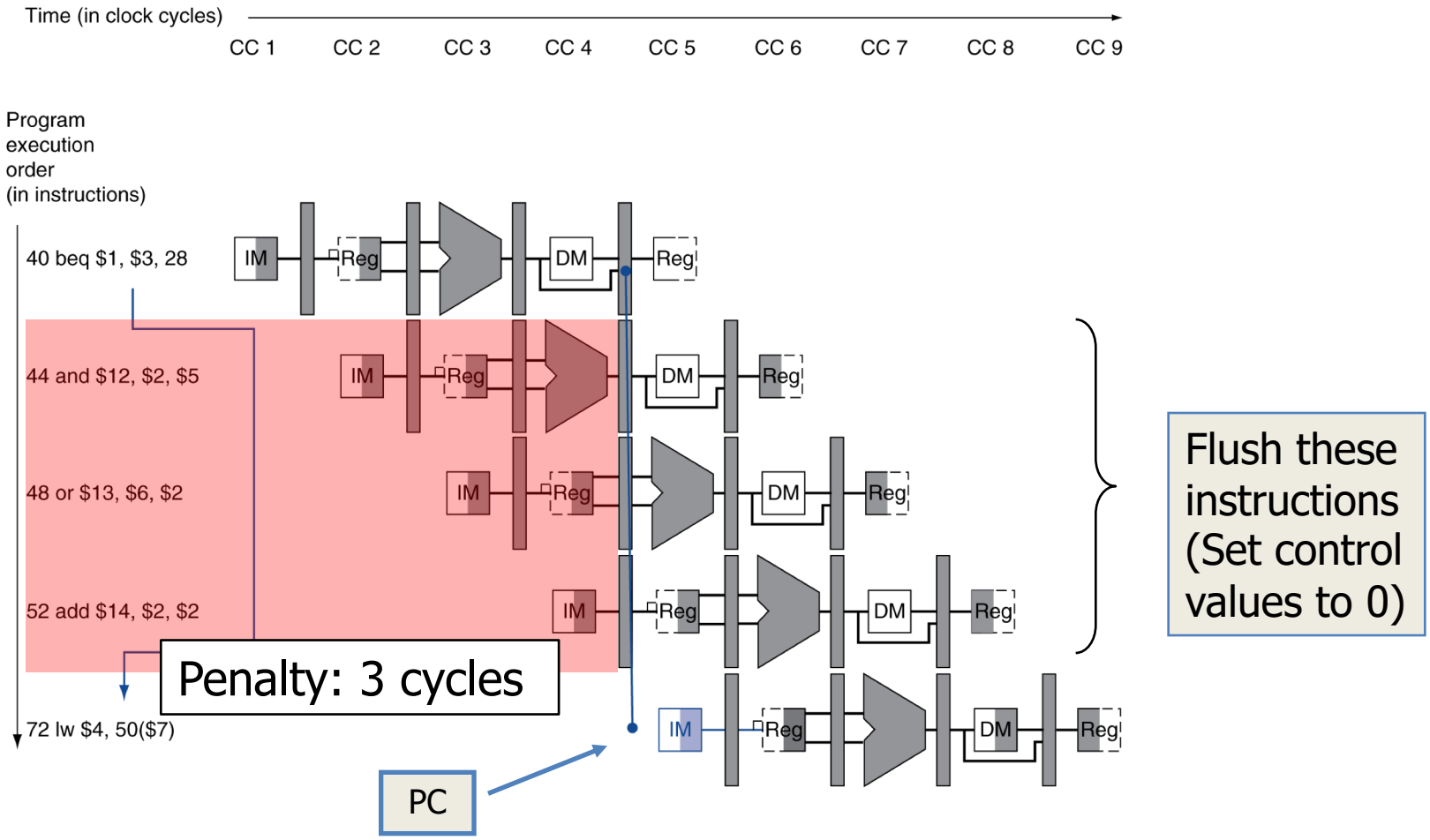
Control Hazard

- If branch outcome determined in MEM



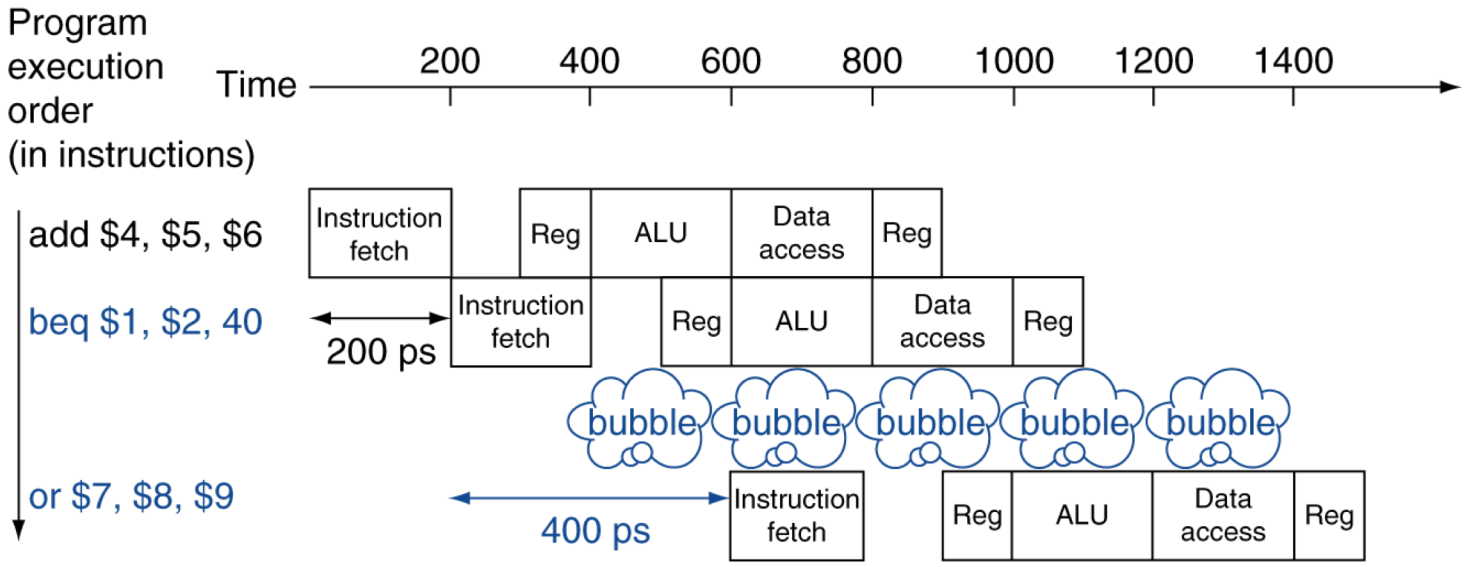
Control Hazard

- If branch outcome determined in MEM

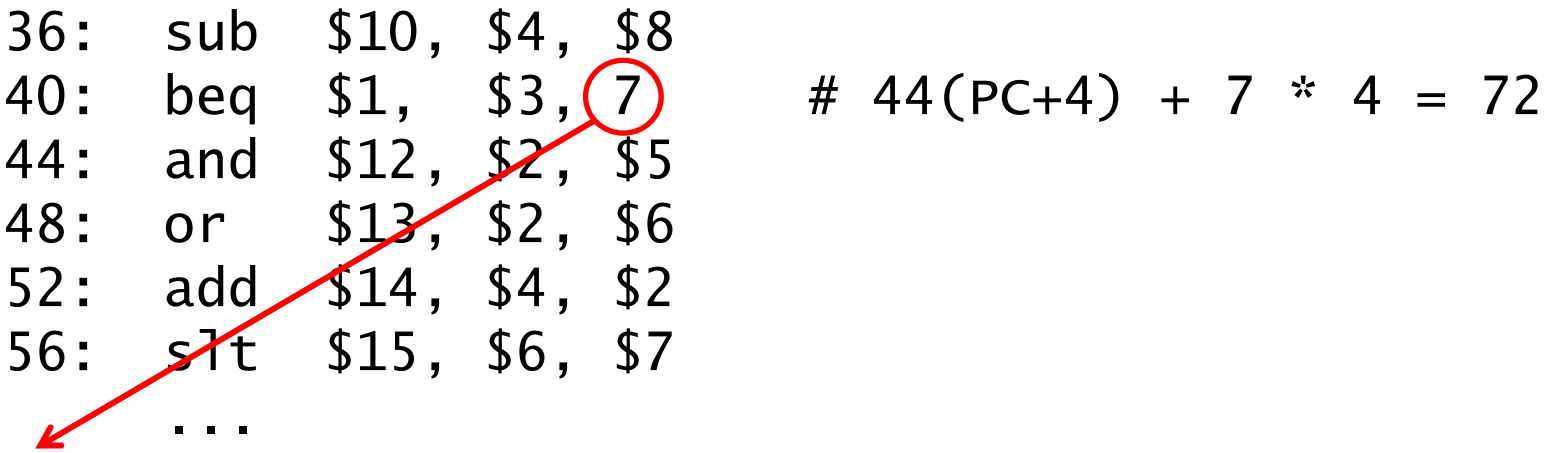


A Solution in MIPS

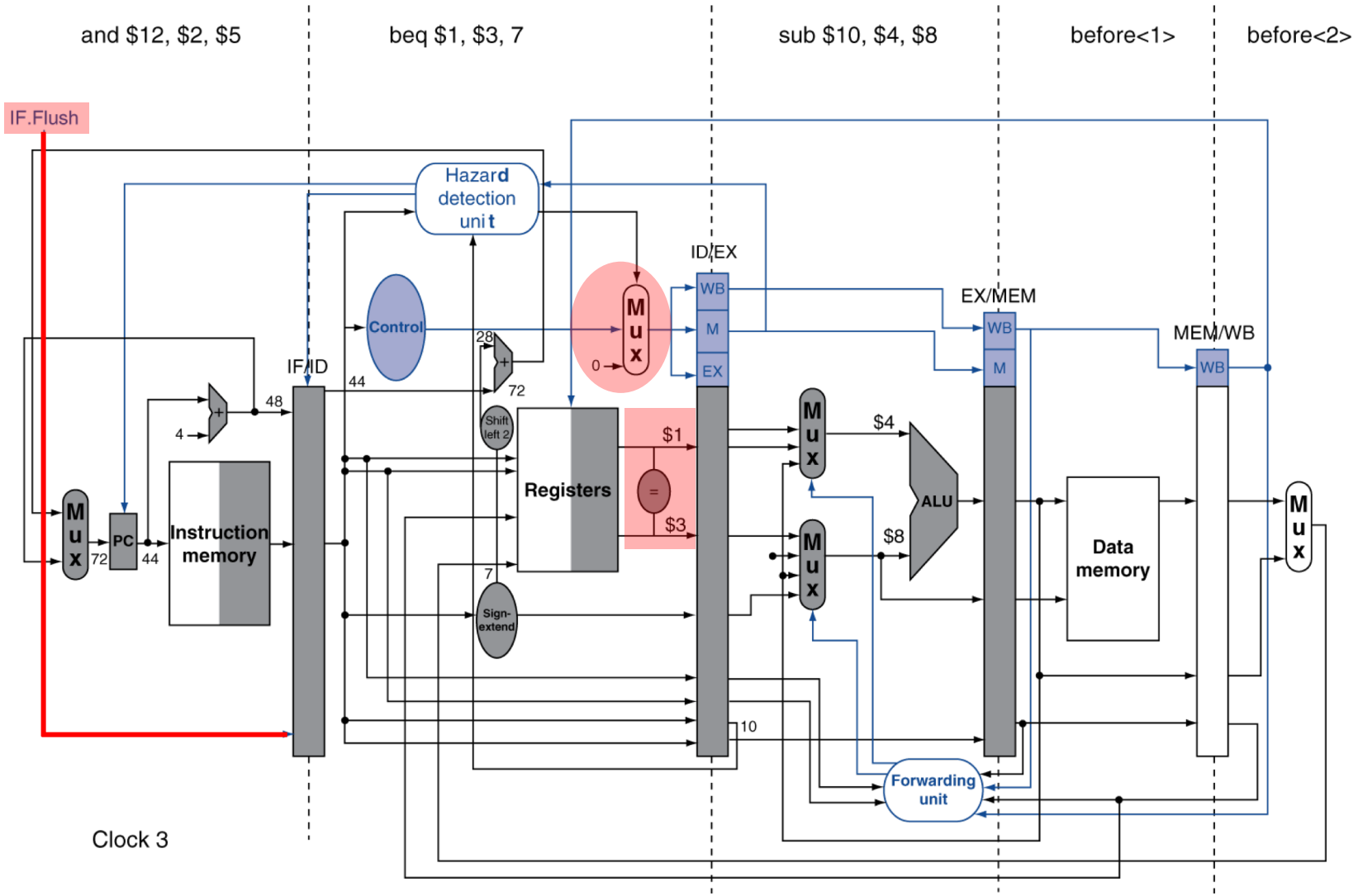
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty



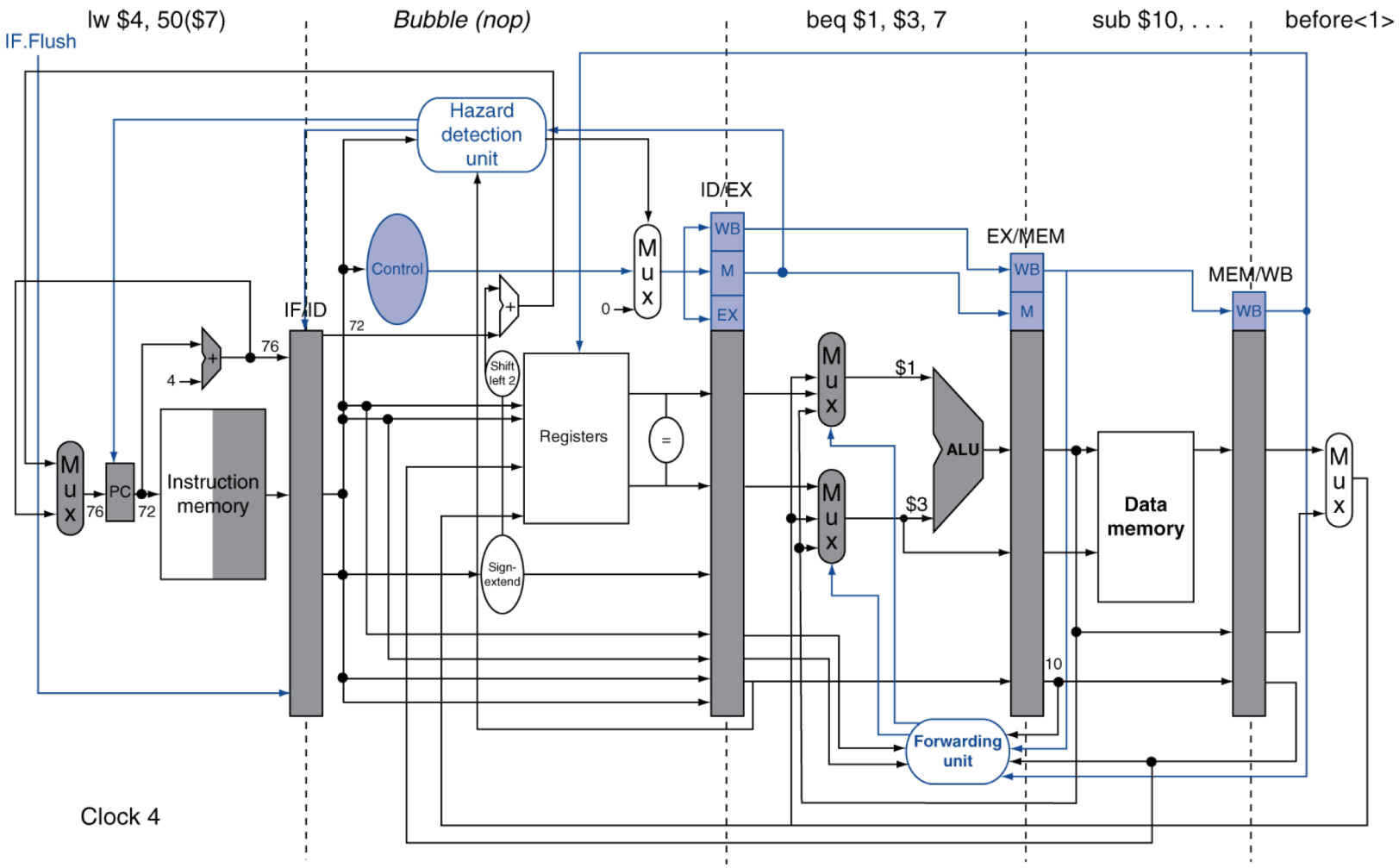
Example: Branch Taken



Example: Branch Taken

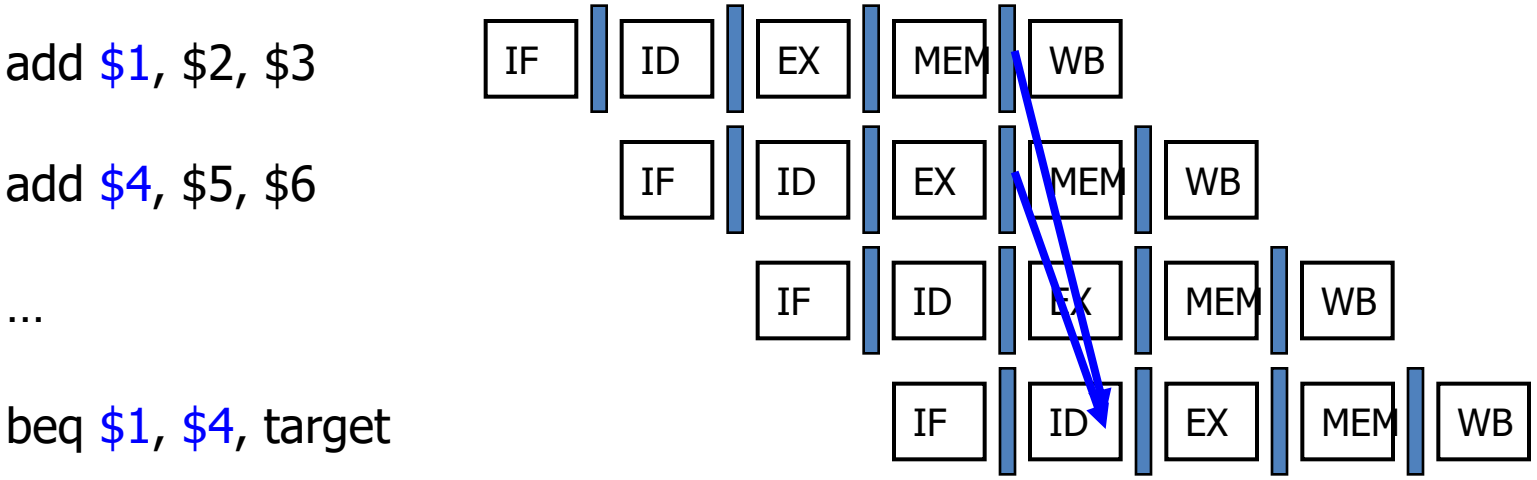


Example: Branch Taken



Data Hazards for Branches

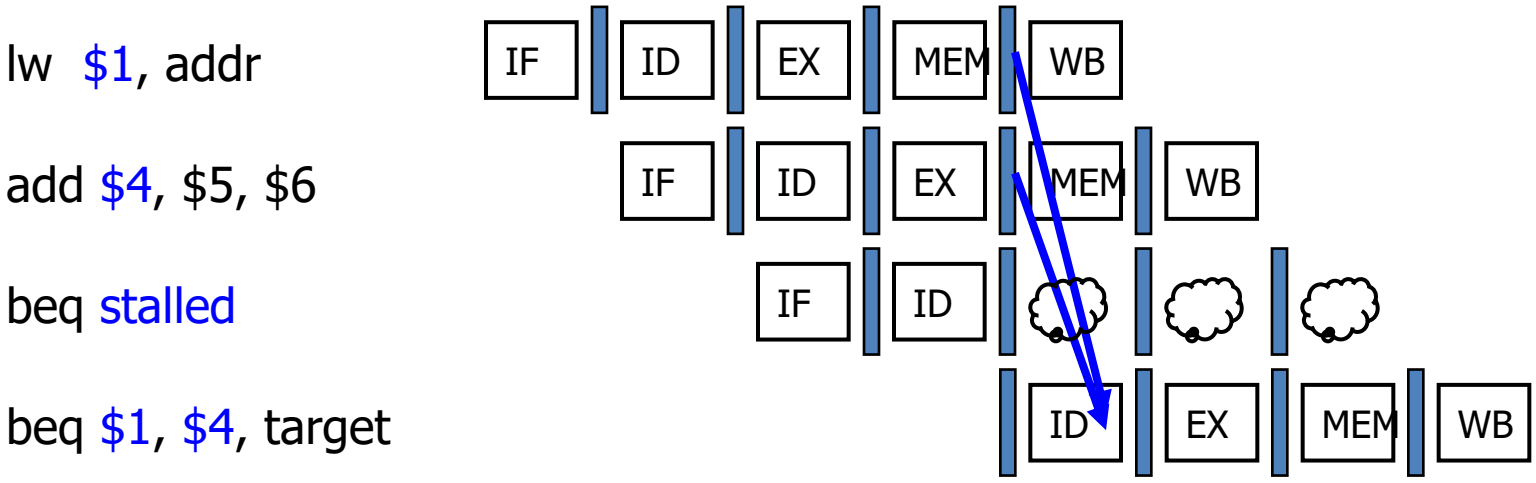
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction



- Can resolve using forwarding

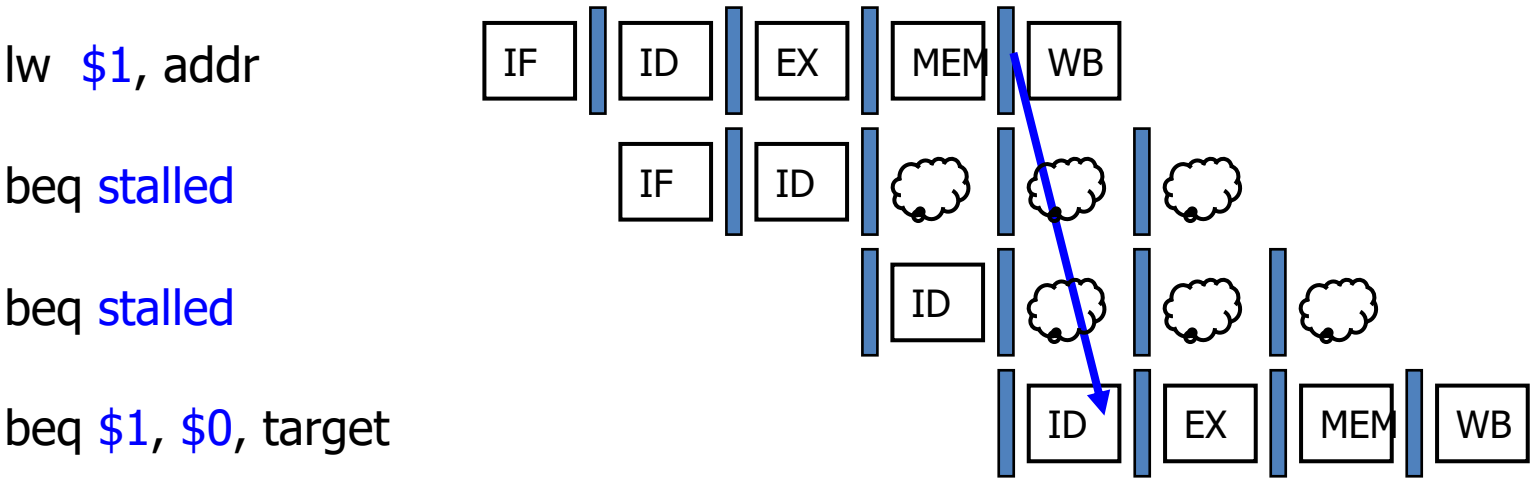
Data Hazards for Branches

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



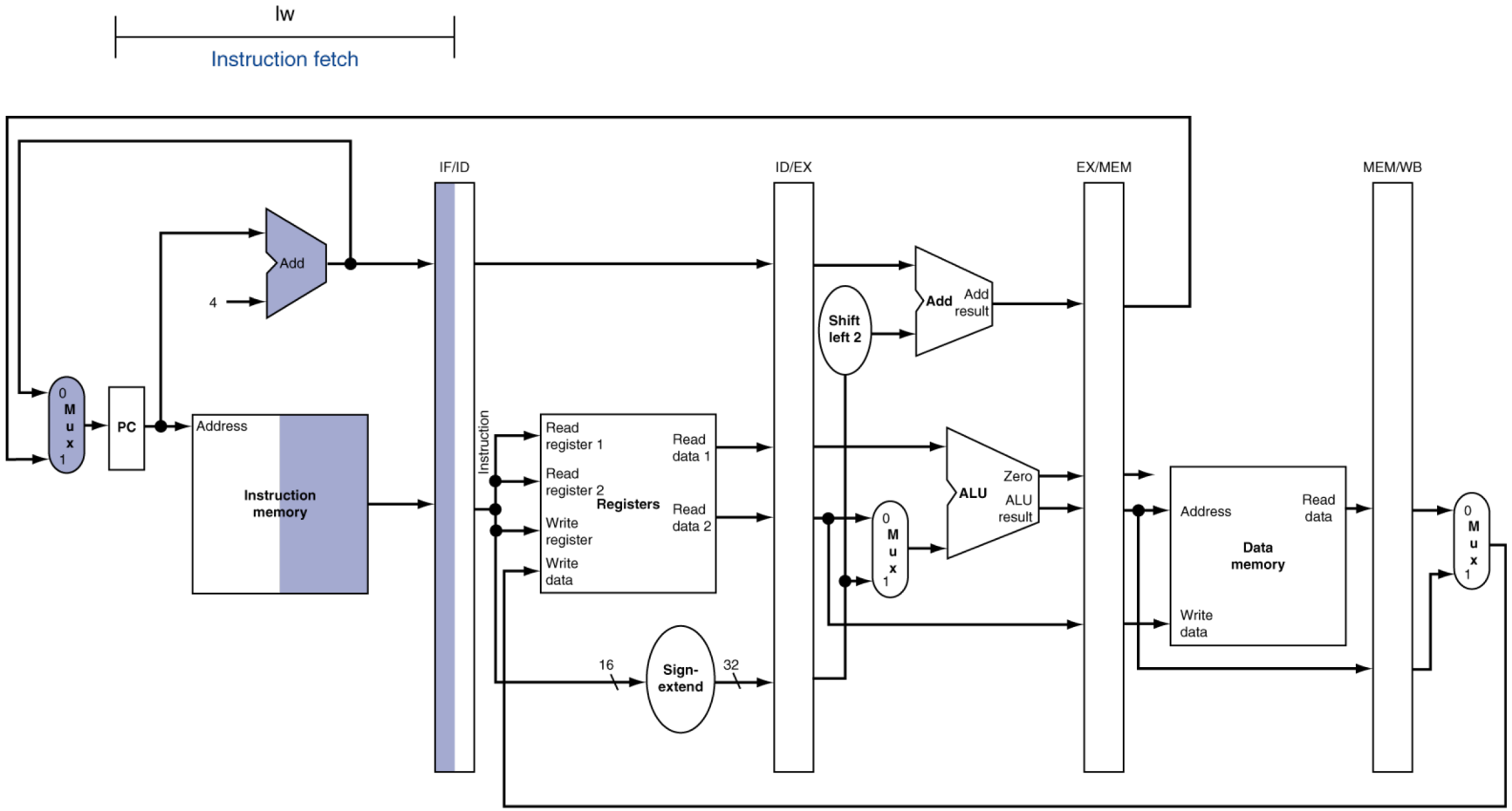
Data Hazards for Branches

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

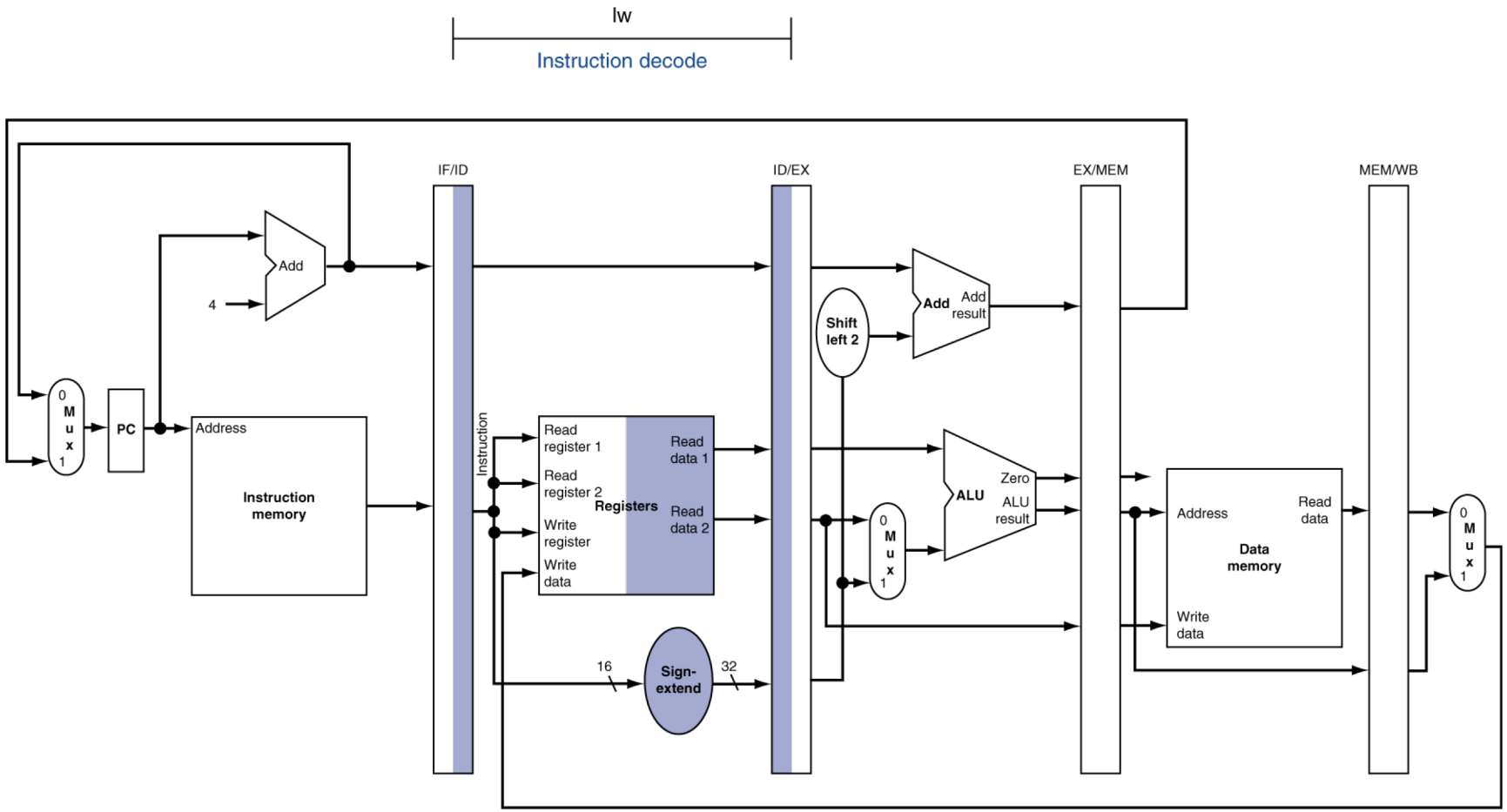


Extra slides

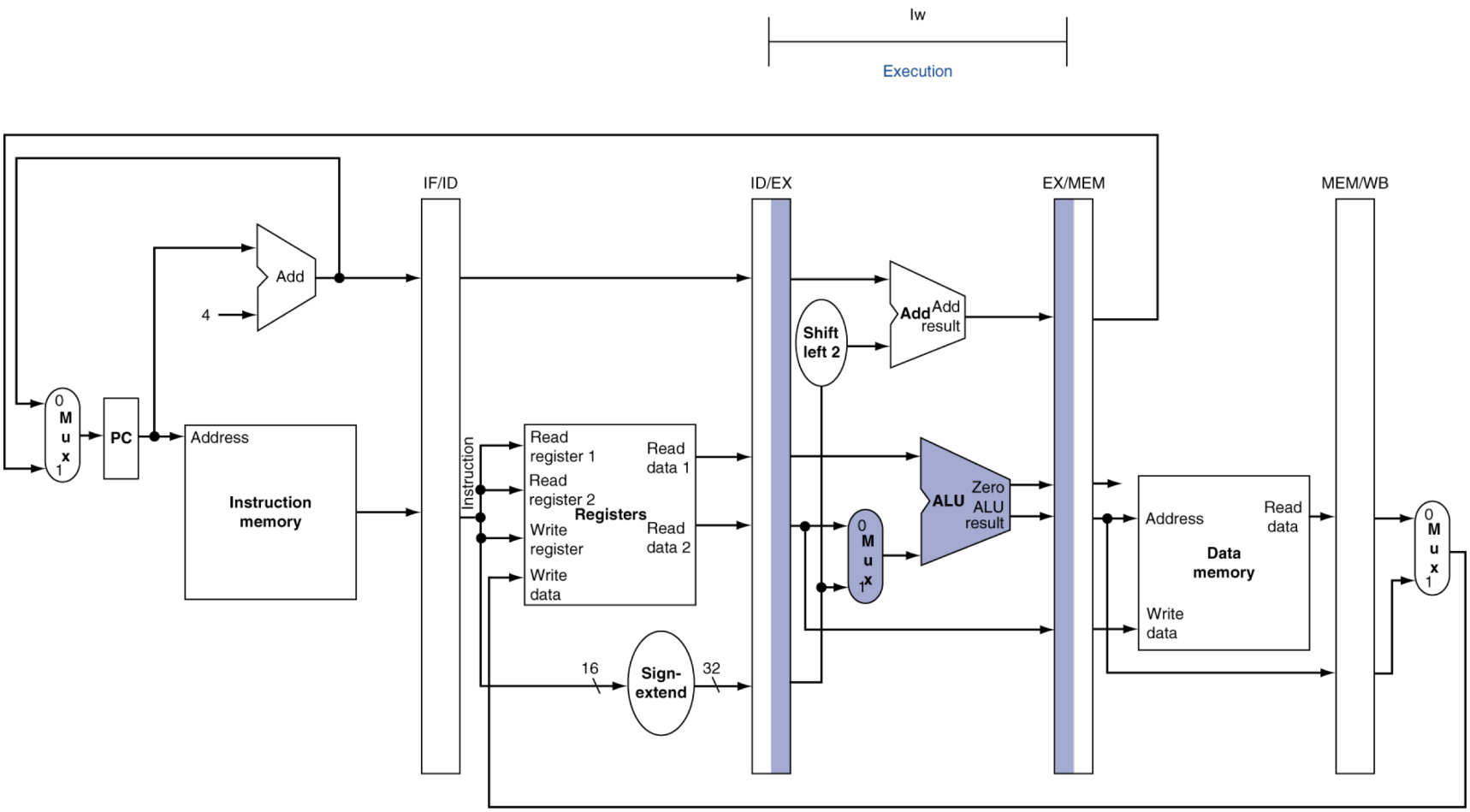
IF for Load, Store, …



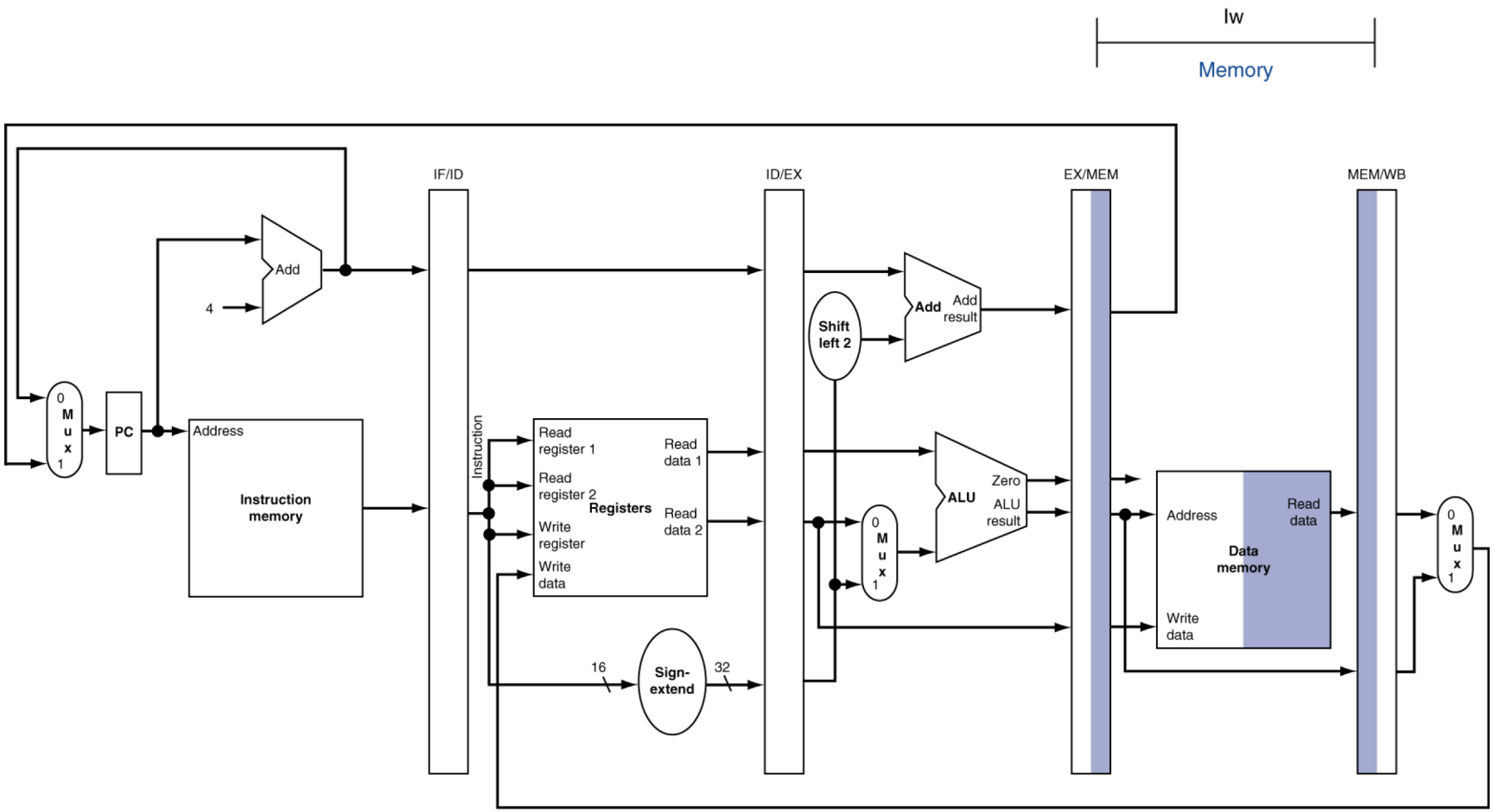
ID for Load, Store, …



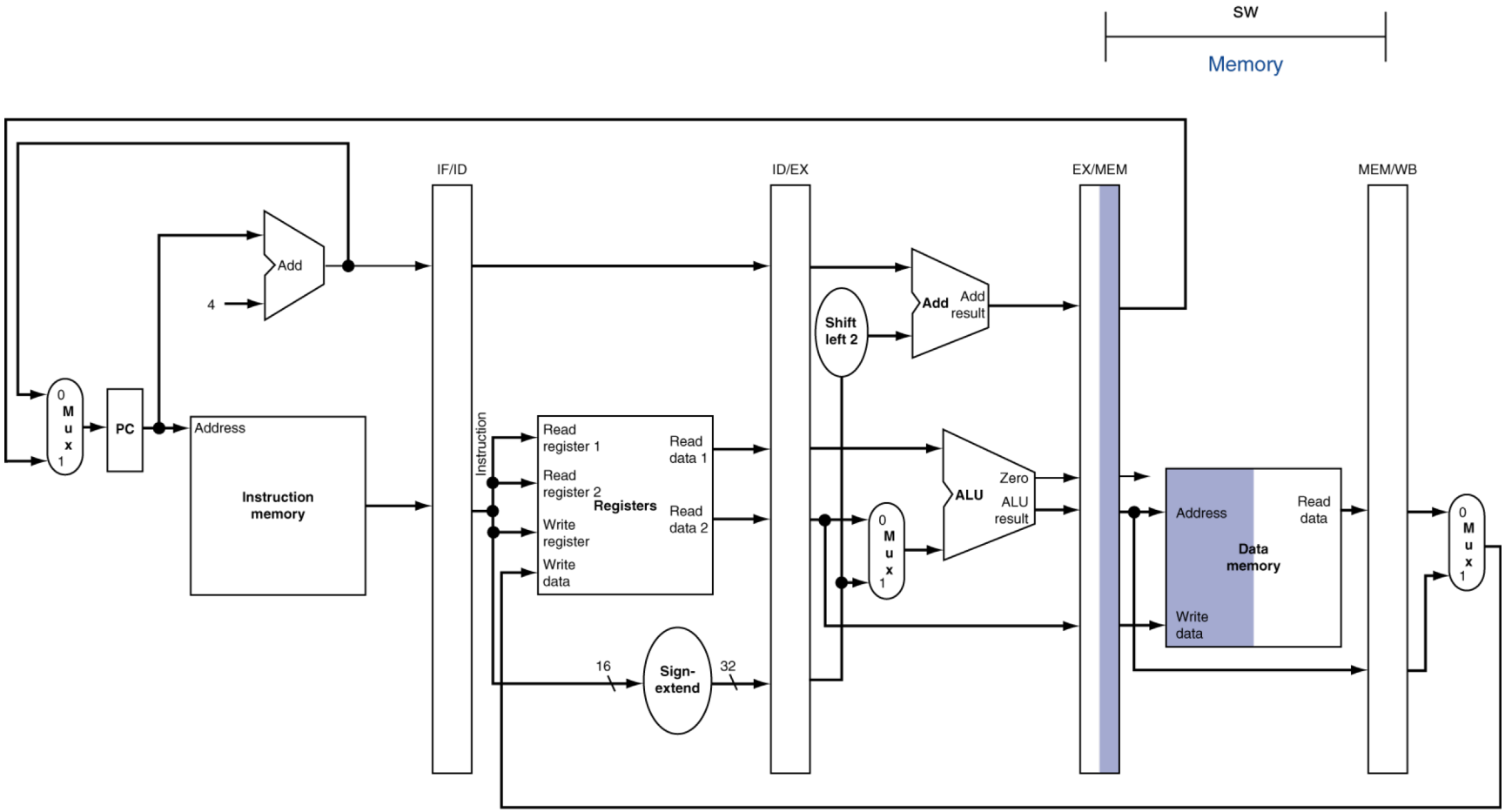
EX for Load


lw
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
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

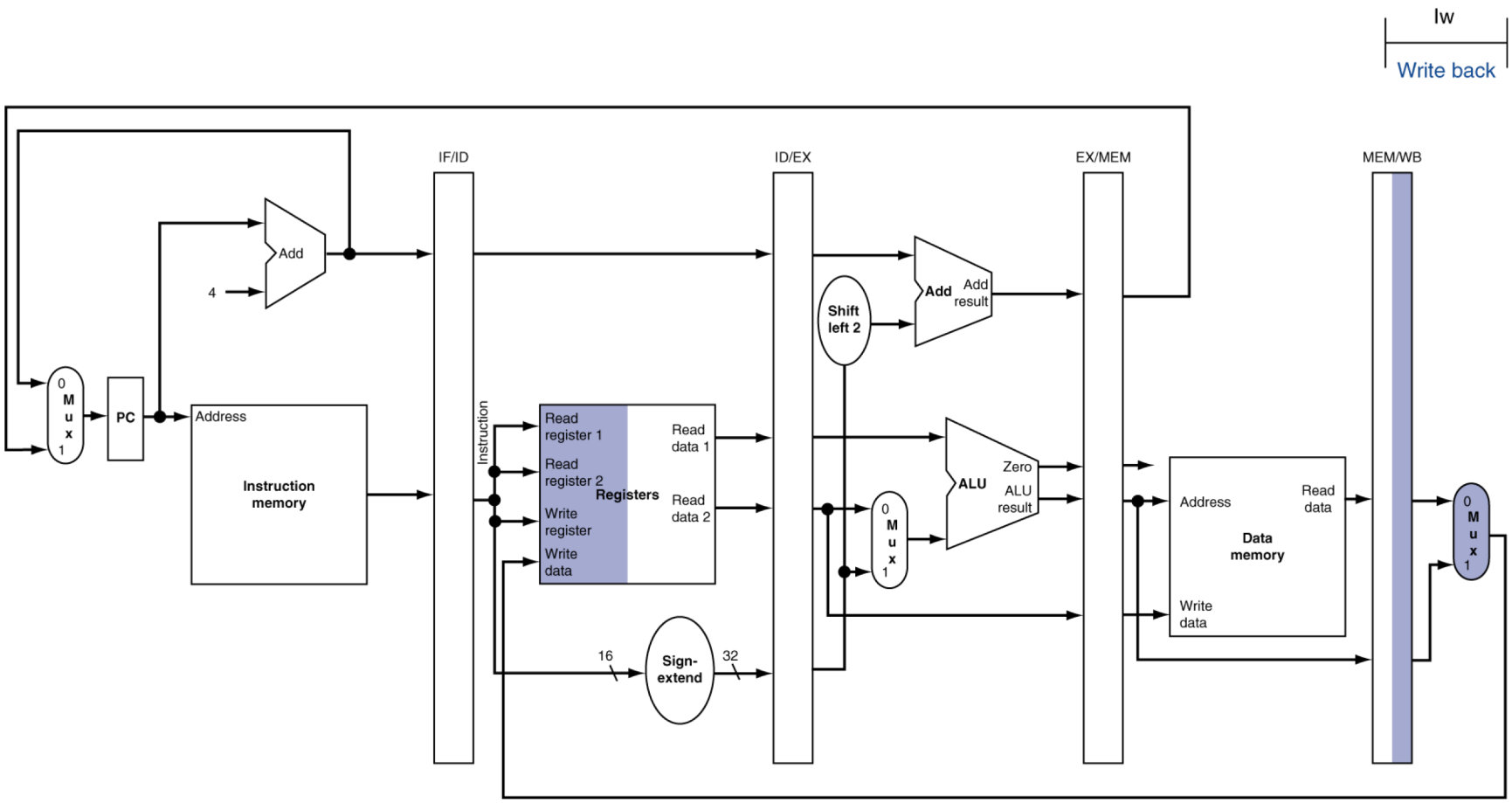
MEM for Load


lw
Memory
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

MEM for Store



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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data