ECE/CS 472/572 Computer Architecture: Pipeline Advanced

Prof. Lizhong Chen

Spring 2019

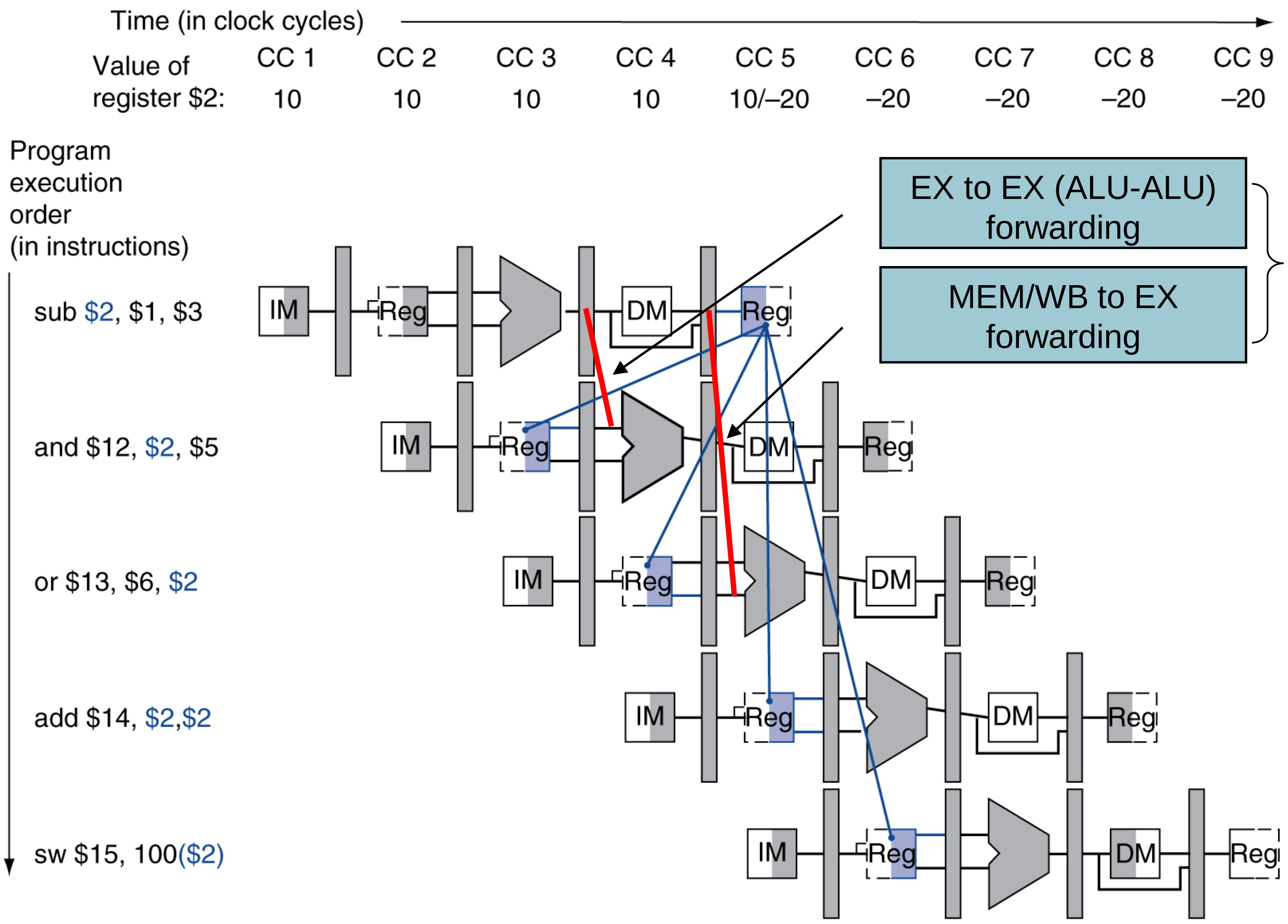
Data Hazards in ALU Instructions

- 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



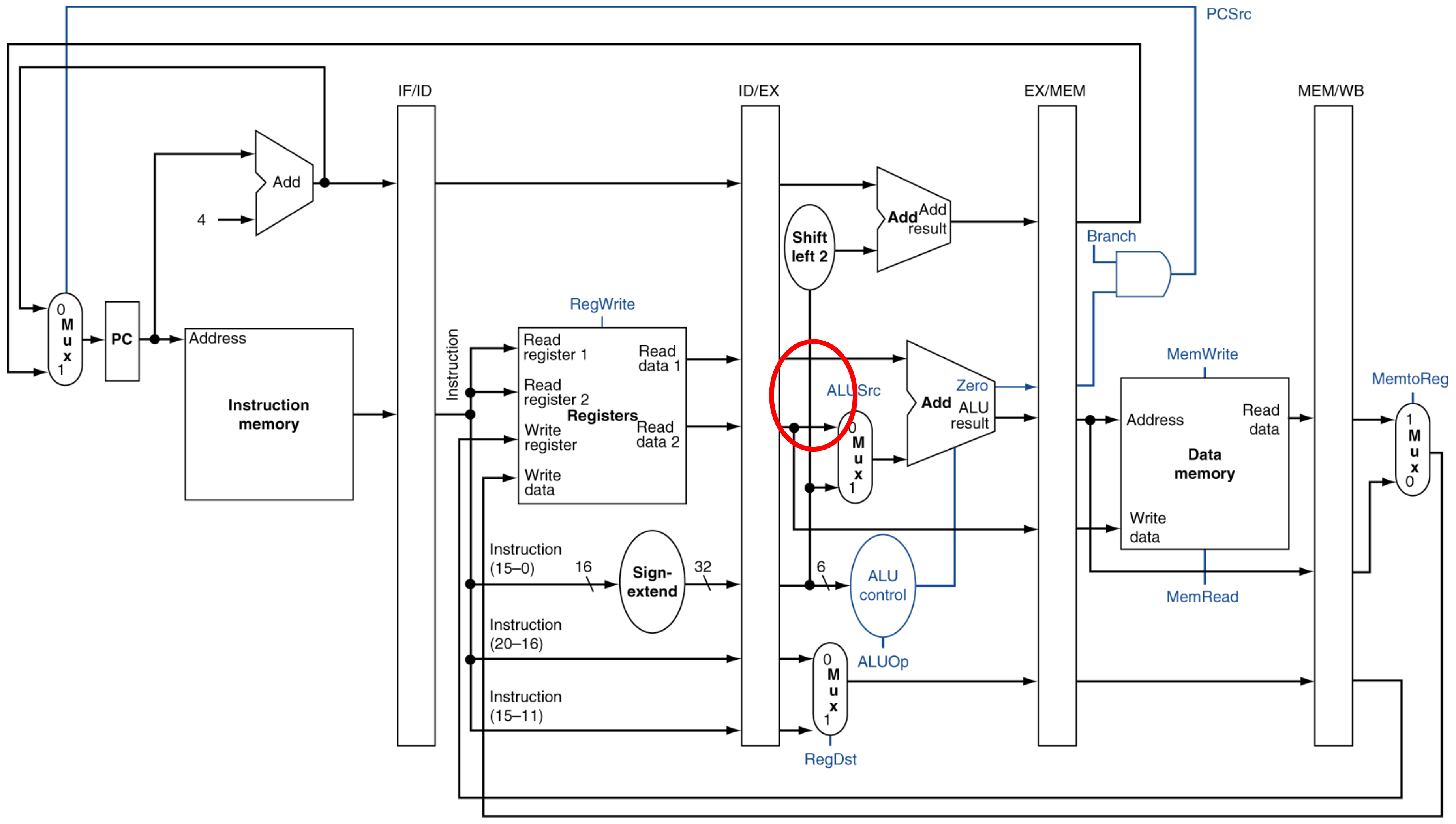
Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Value of register $2: 10 10 10 10 10/–20 –20 –20 –20 –20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2,$2
sw $15, 100($2)
IM
Reg
DM

EX to EX (ALU-ALU) forwarding

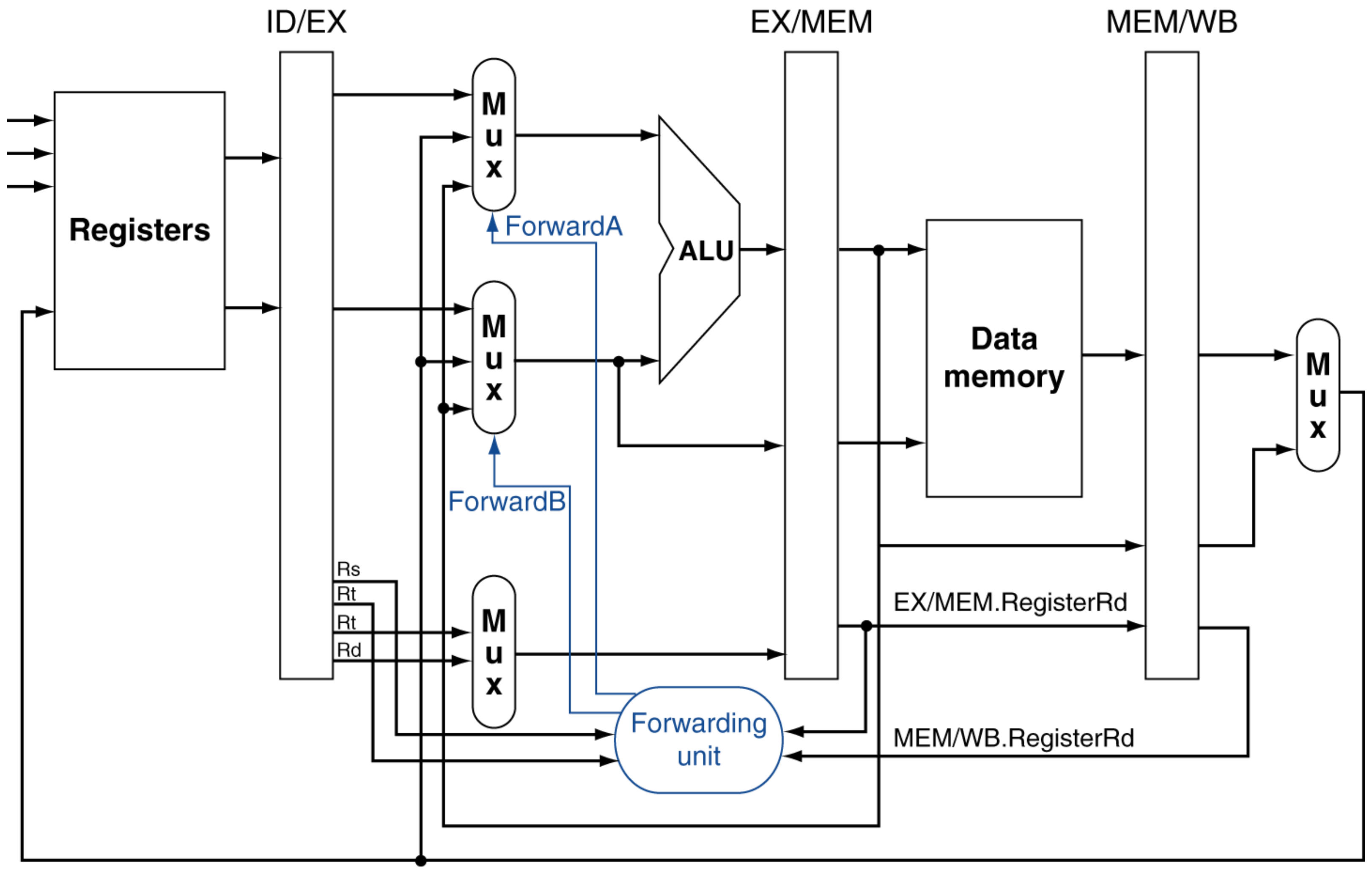
MEM/WB to EX forwarding

Full forwarding

Pipelined (Simplified)



Forwarding Paths



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

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- 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

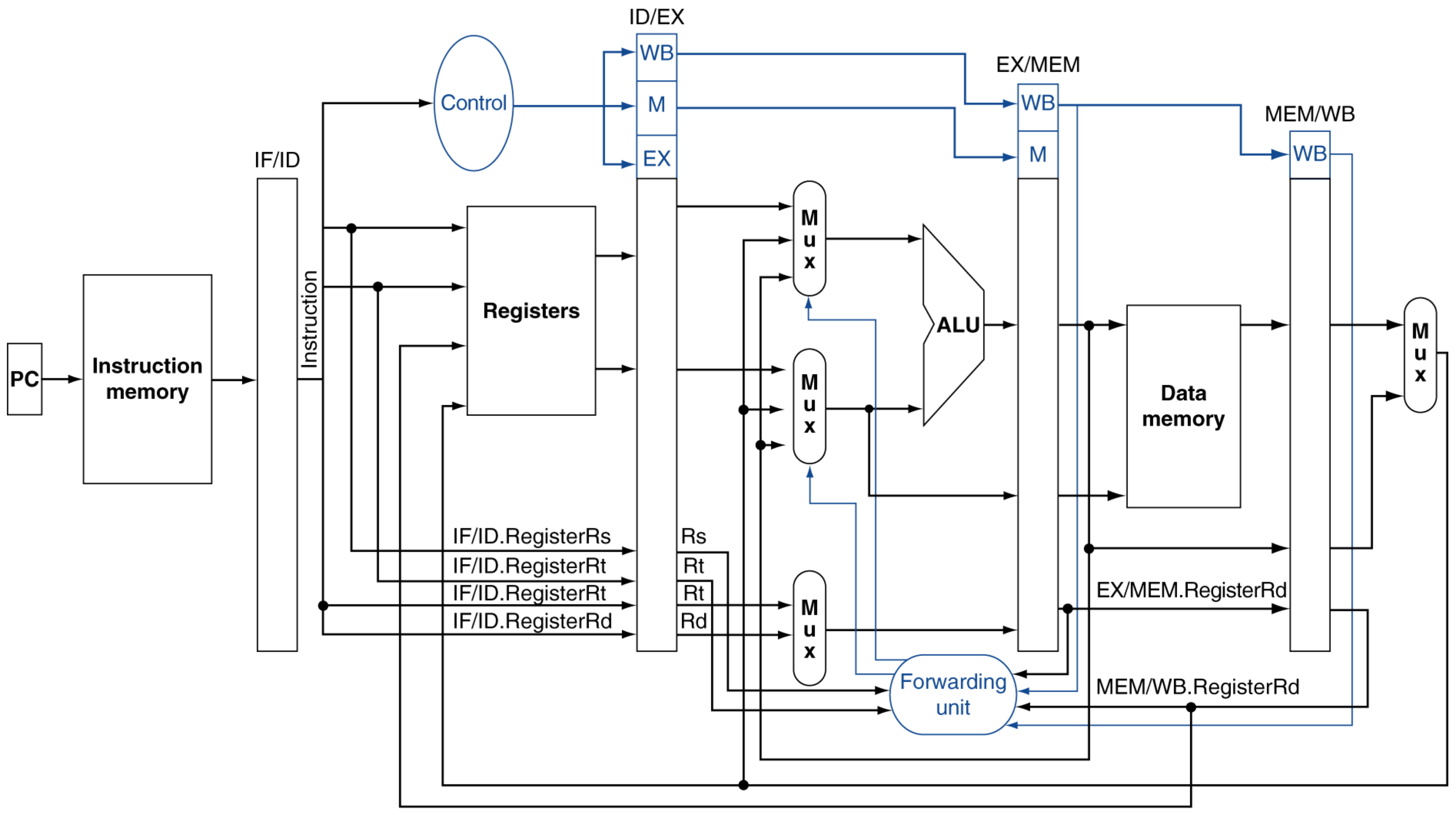
Detecting the Need to Forward

- 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 $0
    - The value of $0 is always 0 when fetched from Reg
    - Previous instruction could set it to non-zero
        - SUB $0, $1, $2
        - ADD $4, $0, $3
    - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

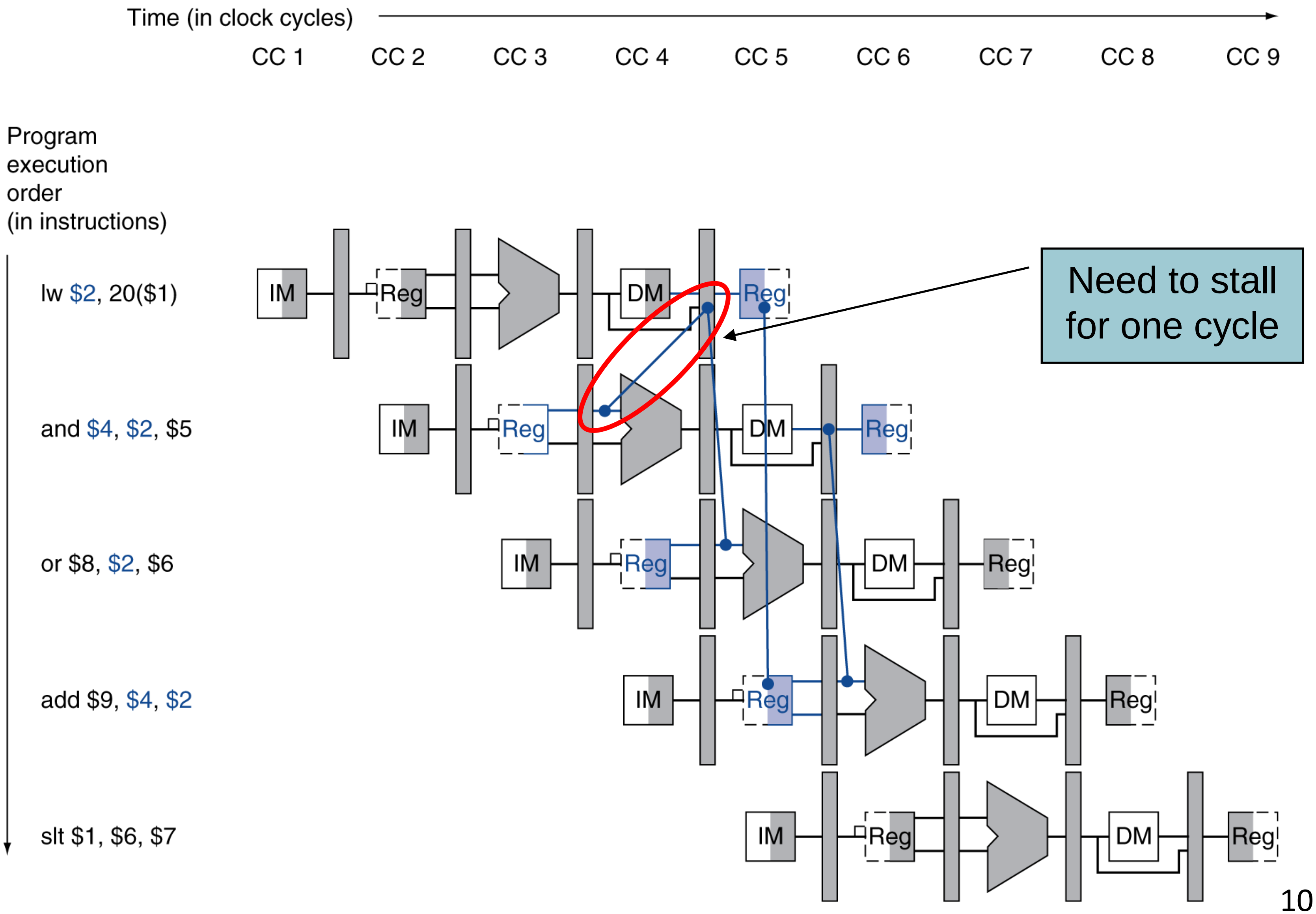
Forwarding Conditions

- 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

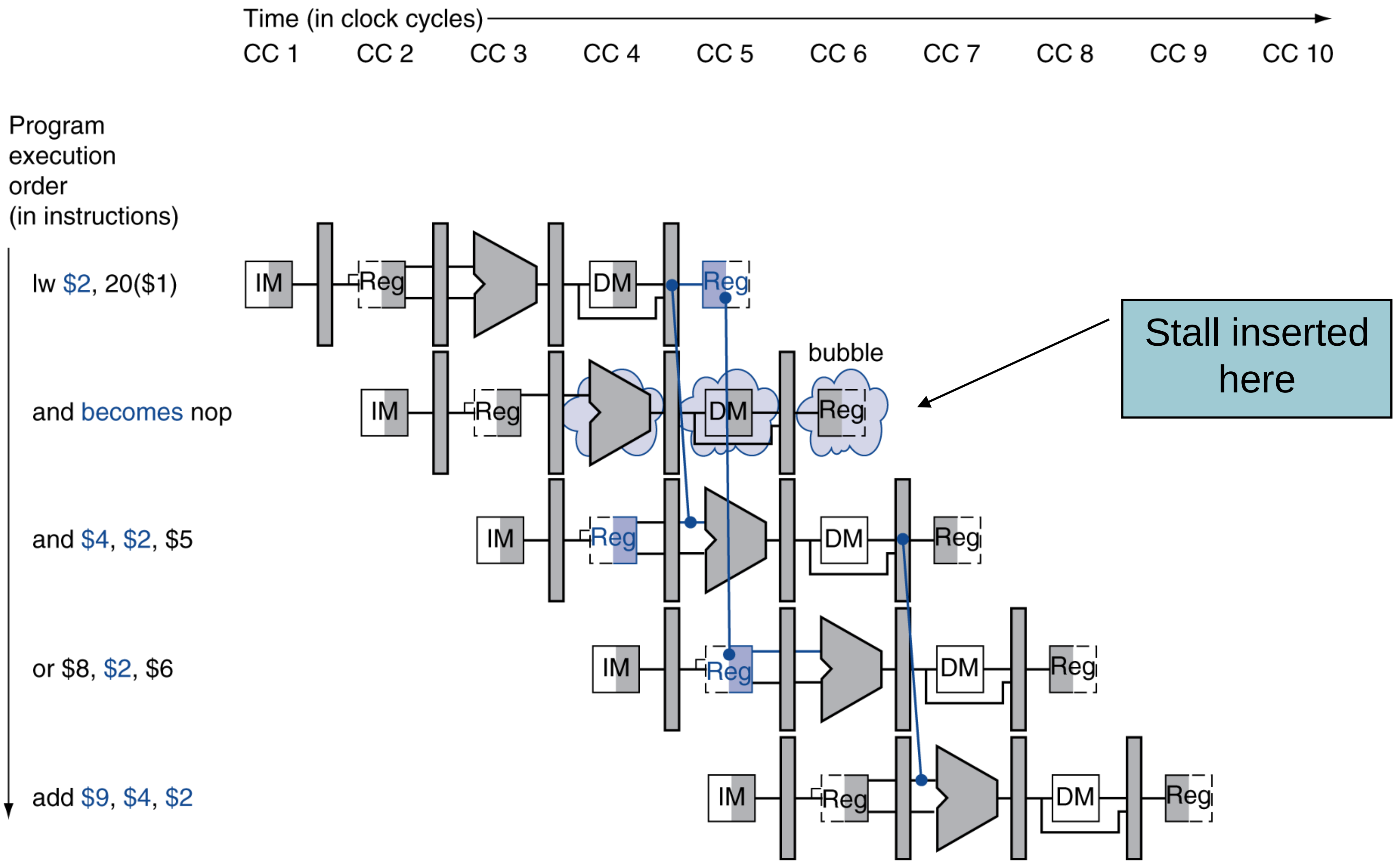
Datapath with Forwarding



Load-Use Data Hazard



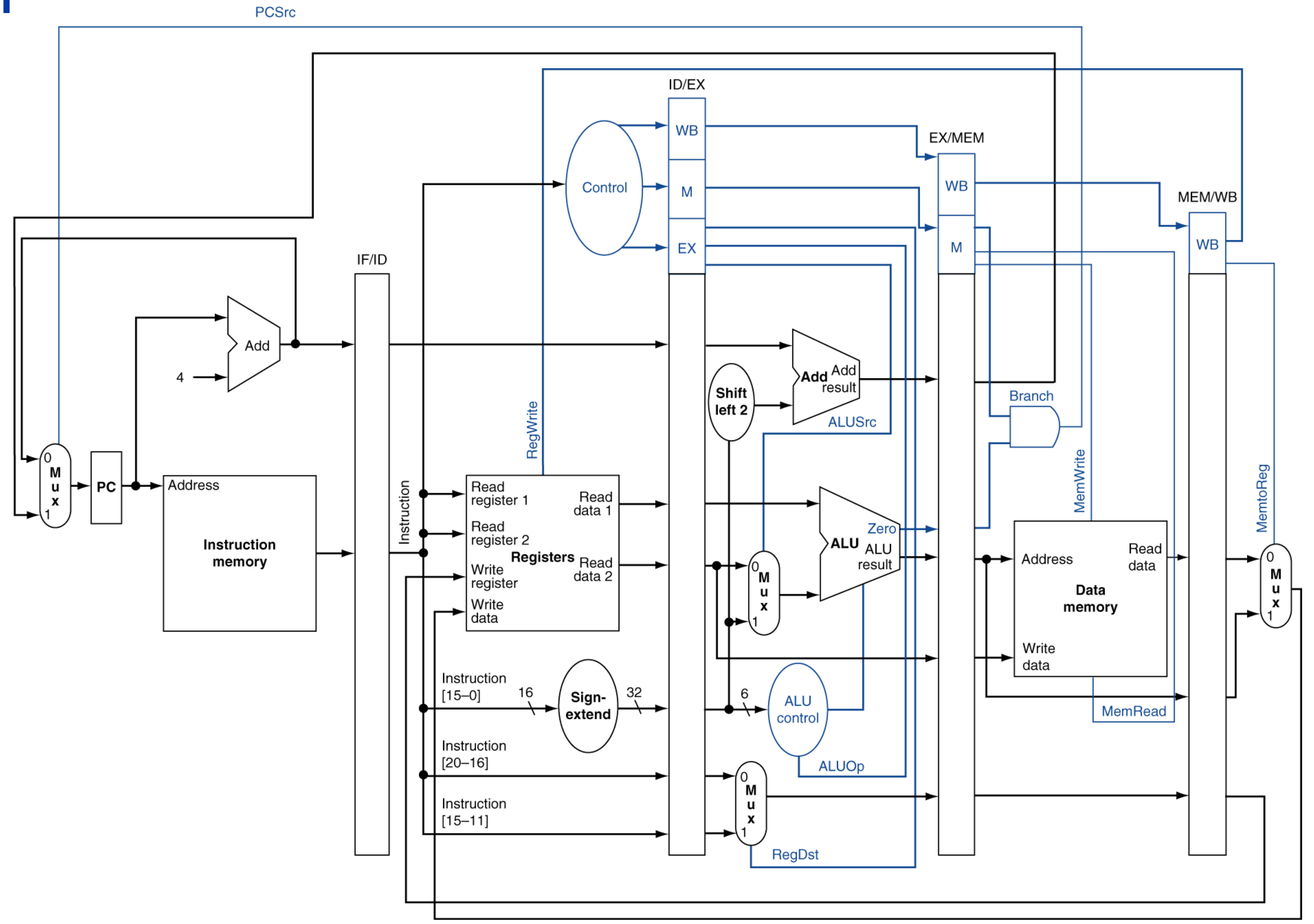
Stall/Bubble in the Pipeline



Load-Use Hazard Detection

- Check when 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 insert bubble?

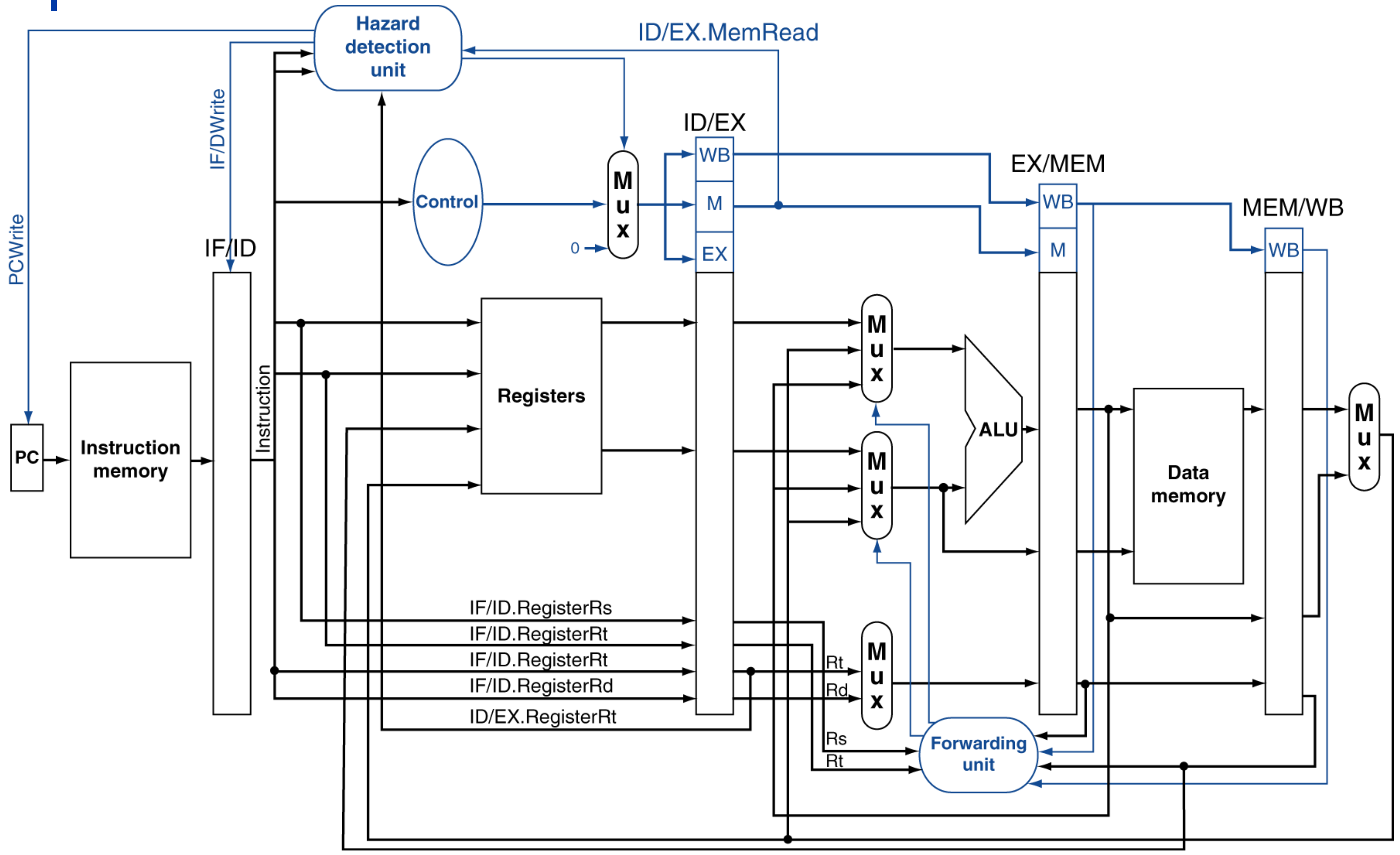
Pipelined Datapath and Control



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
  - Same 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

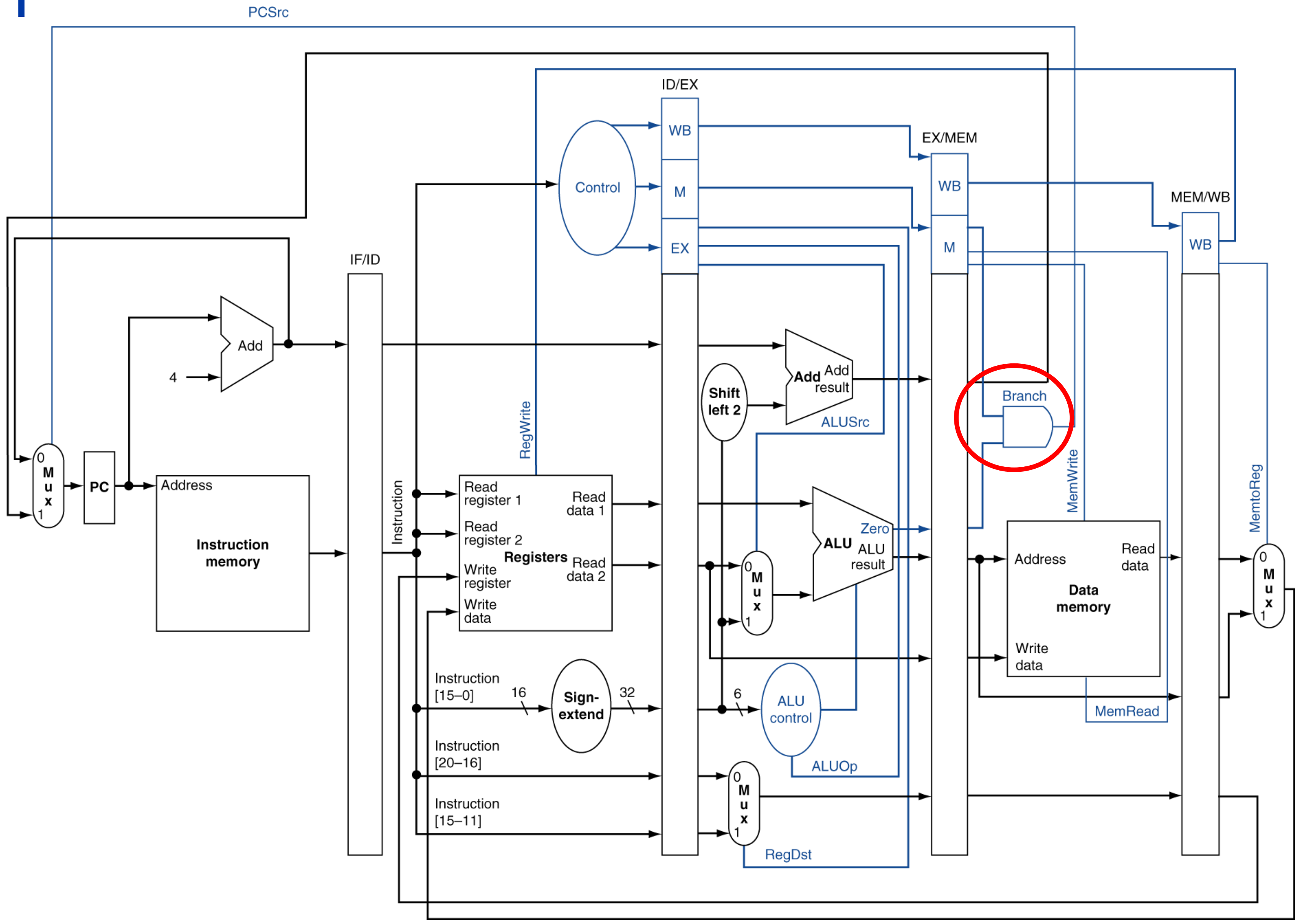
Datapath with Hazard Detection



Stalls and Performance

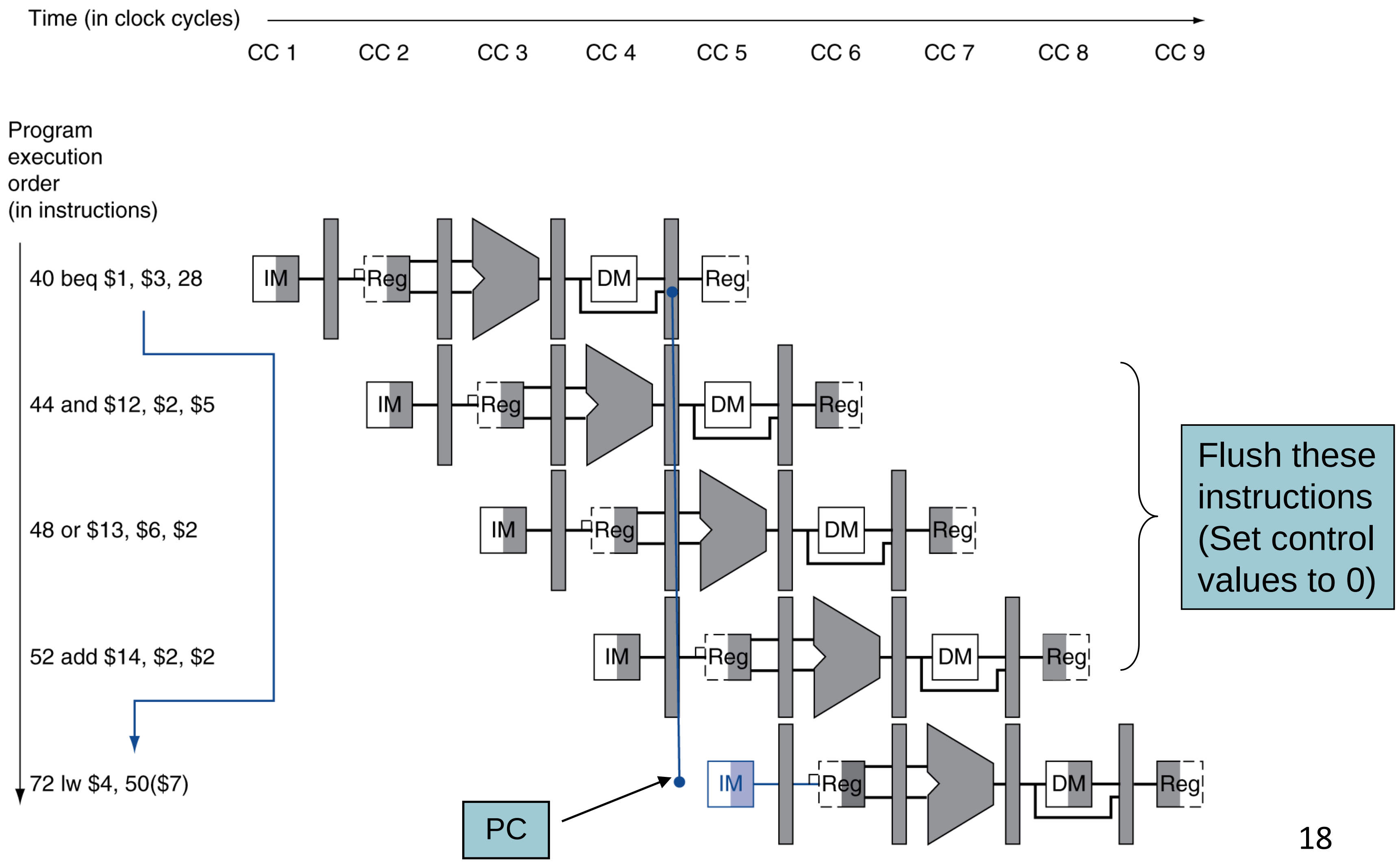
- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid some hazards and stalls
  - Requires knowledge of the pipeline structure
  - Limited effectiveness

Pipelined Datapath and Control



Branch Hazards

- If branch outcome is determined in MEM

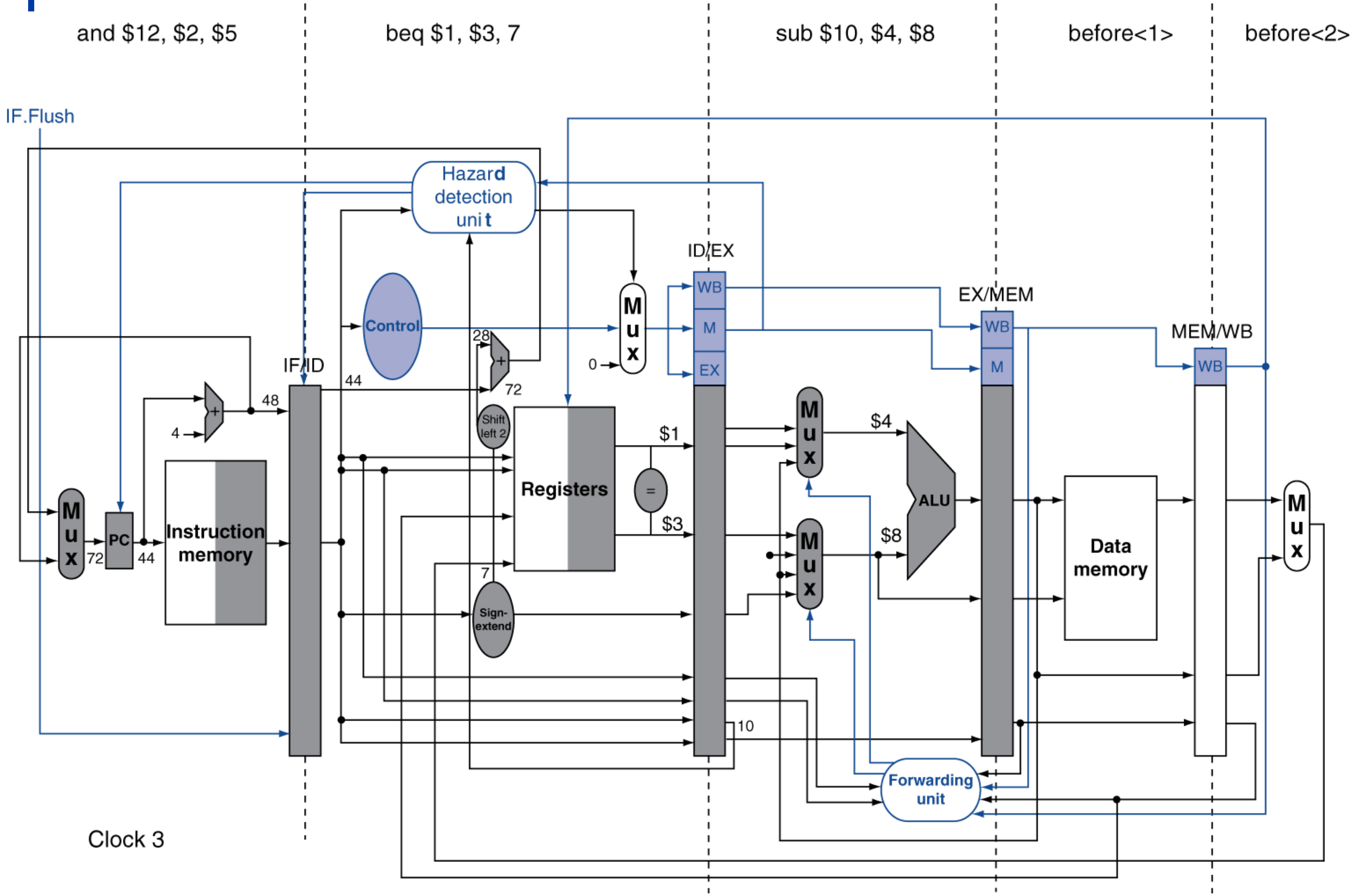


Reducing Branch Delay

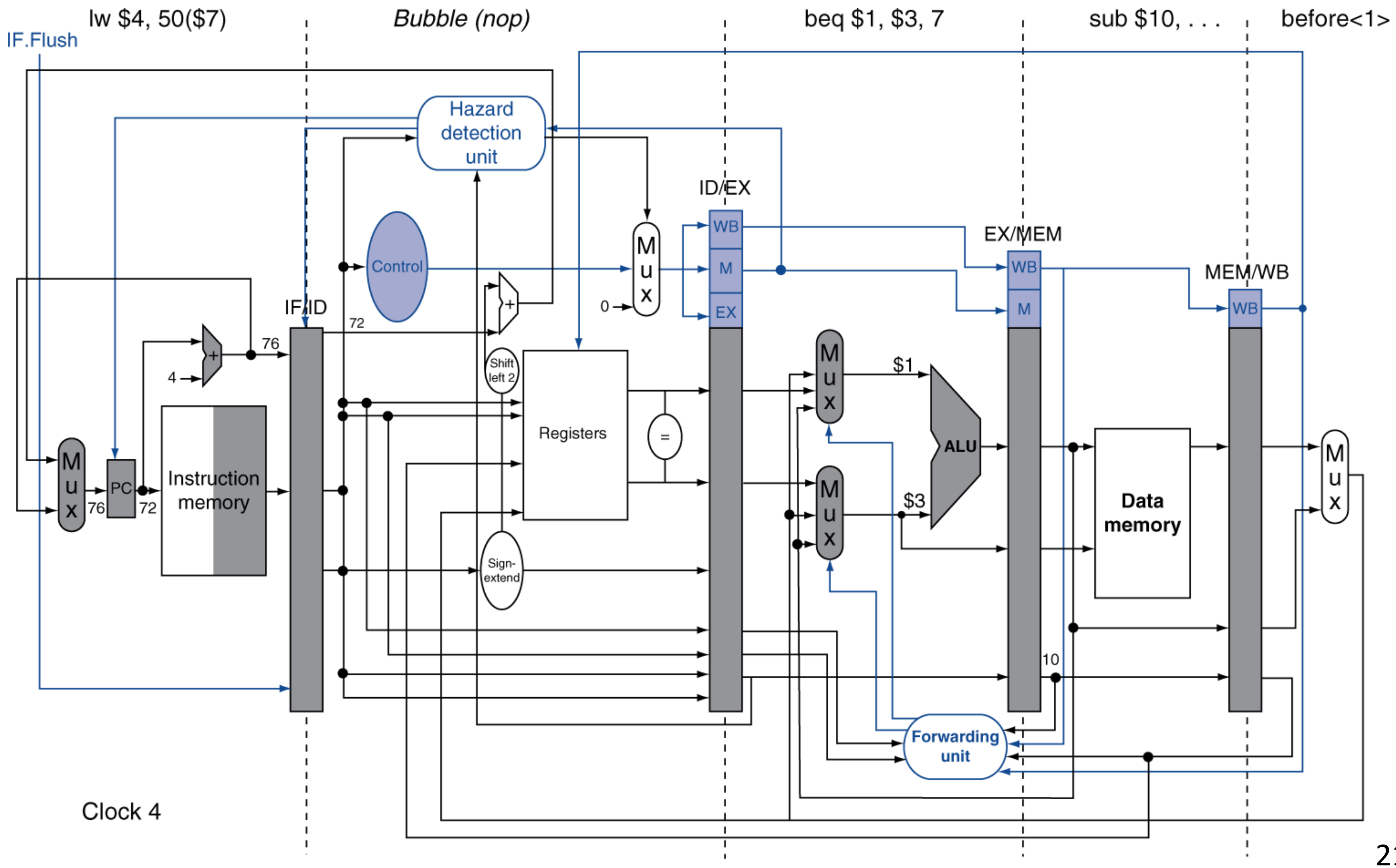
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

Example: Branch Taken



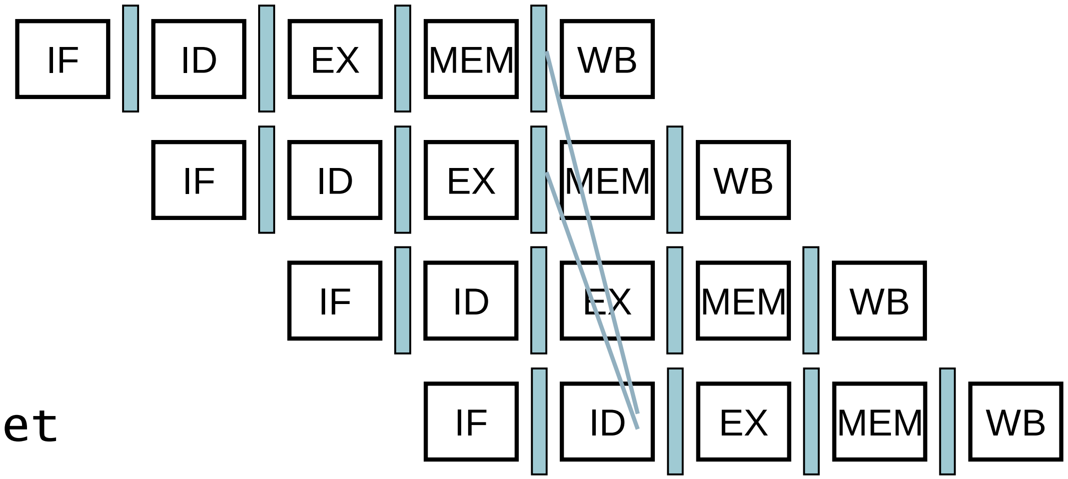
Example: Branch Taken



Data Hazards for Branches

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

```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```

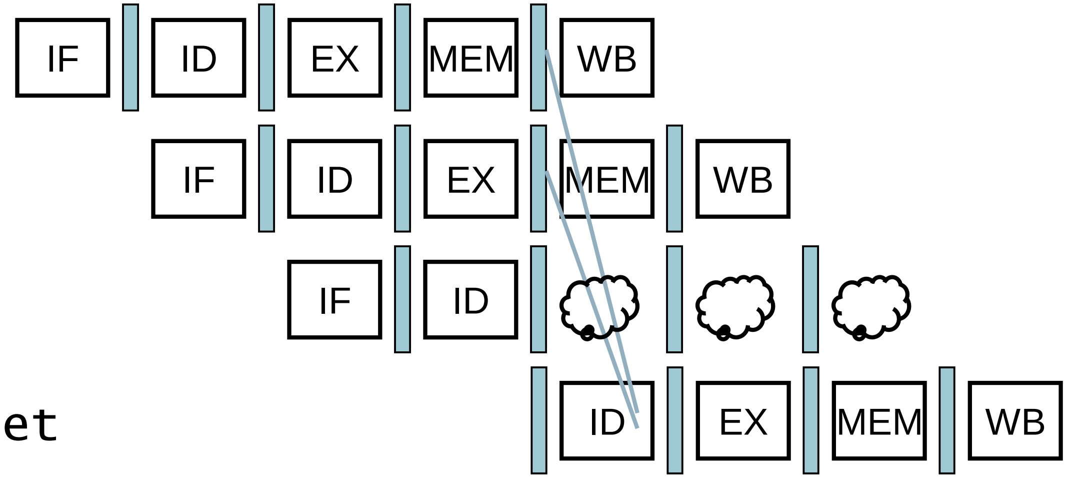


- Can resolve using forwarding

Data Hazards for Branches

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

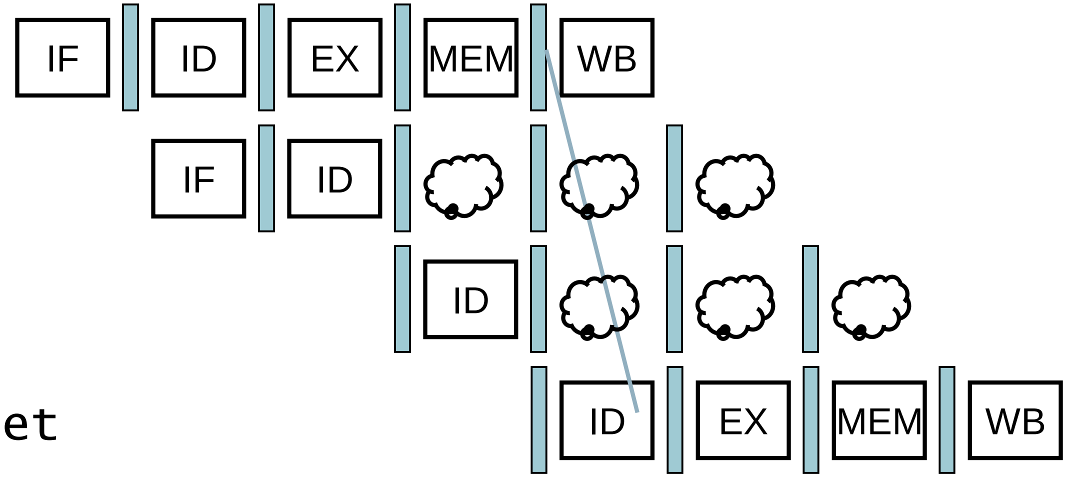
```
lw  $1, addr

add $4, $5, $6

beq stalled

beq $1, $4, target
```



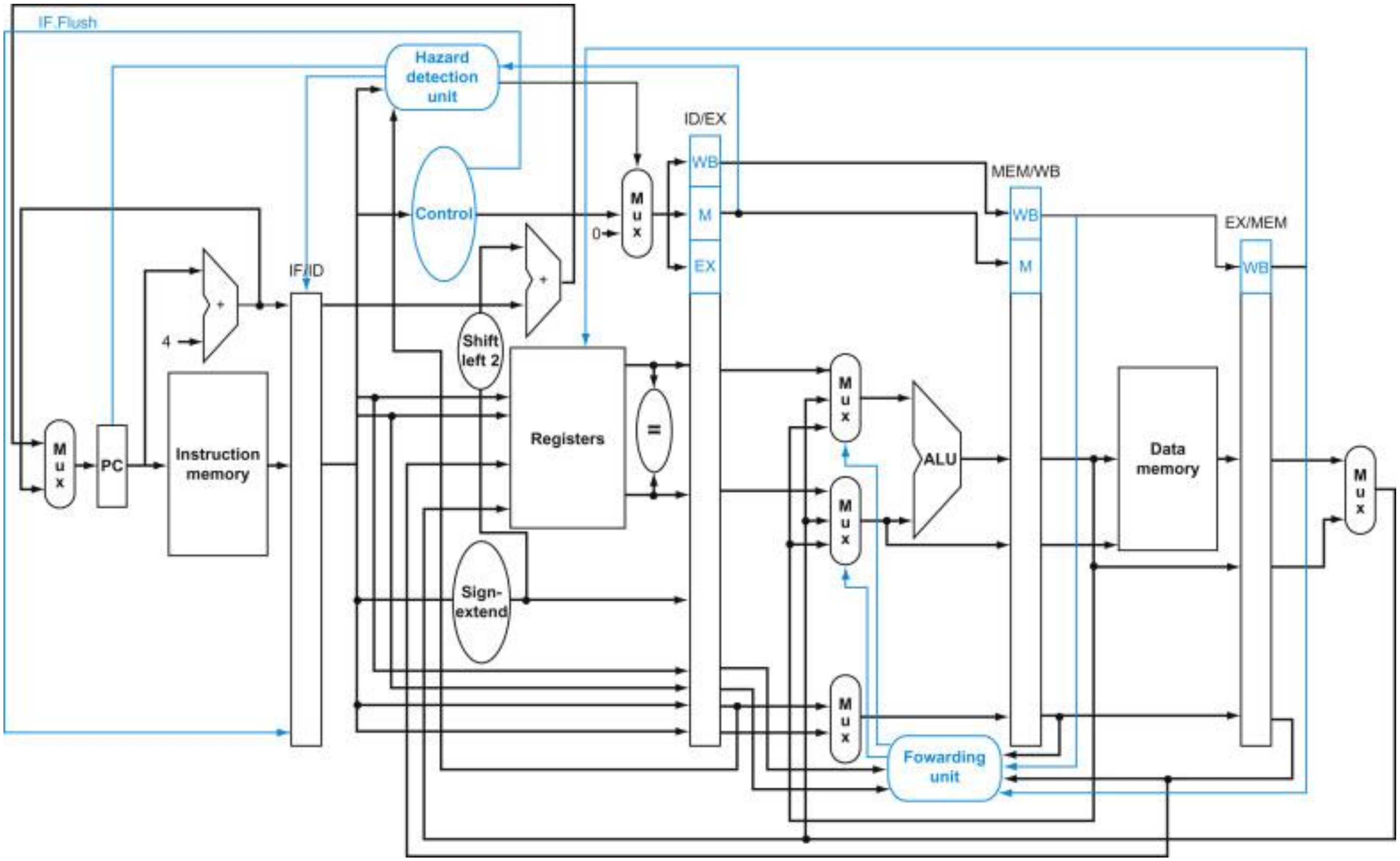
Data Hazards for Branches

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

```
lw  $1, addr

beq stalled

beq stalled

beq $1, $0, target
```



Final datapath and control (simplified)


IF.Flush
Hazard detection unit
ID/EX
WB
M
EX
Control
M u x
0
MEM/WB
WB
M
EX/MEM
WB
IF/ID
+
4
+
Shift left 2
Registers
=
M u x
PC
Instruction memory
M u x
ALU
M u x
Data memory
M u x
Sign-extend
M u x
Fowarding unit