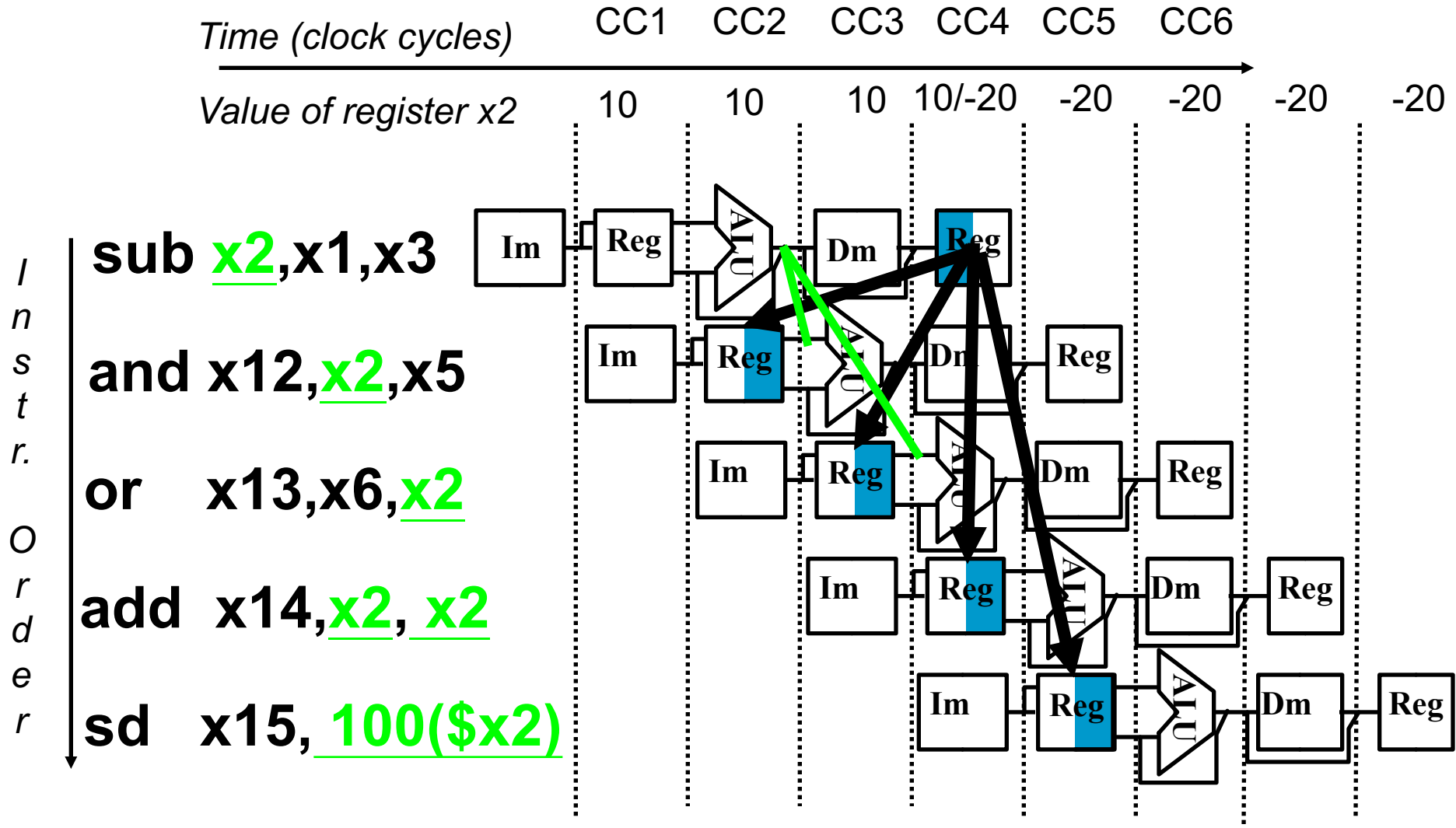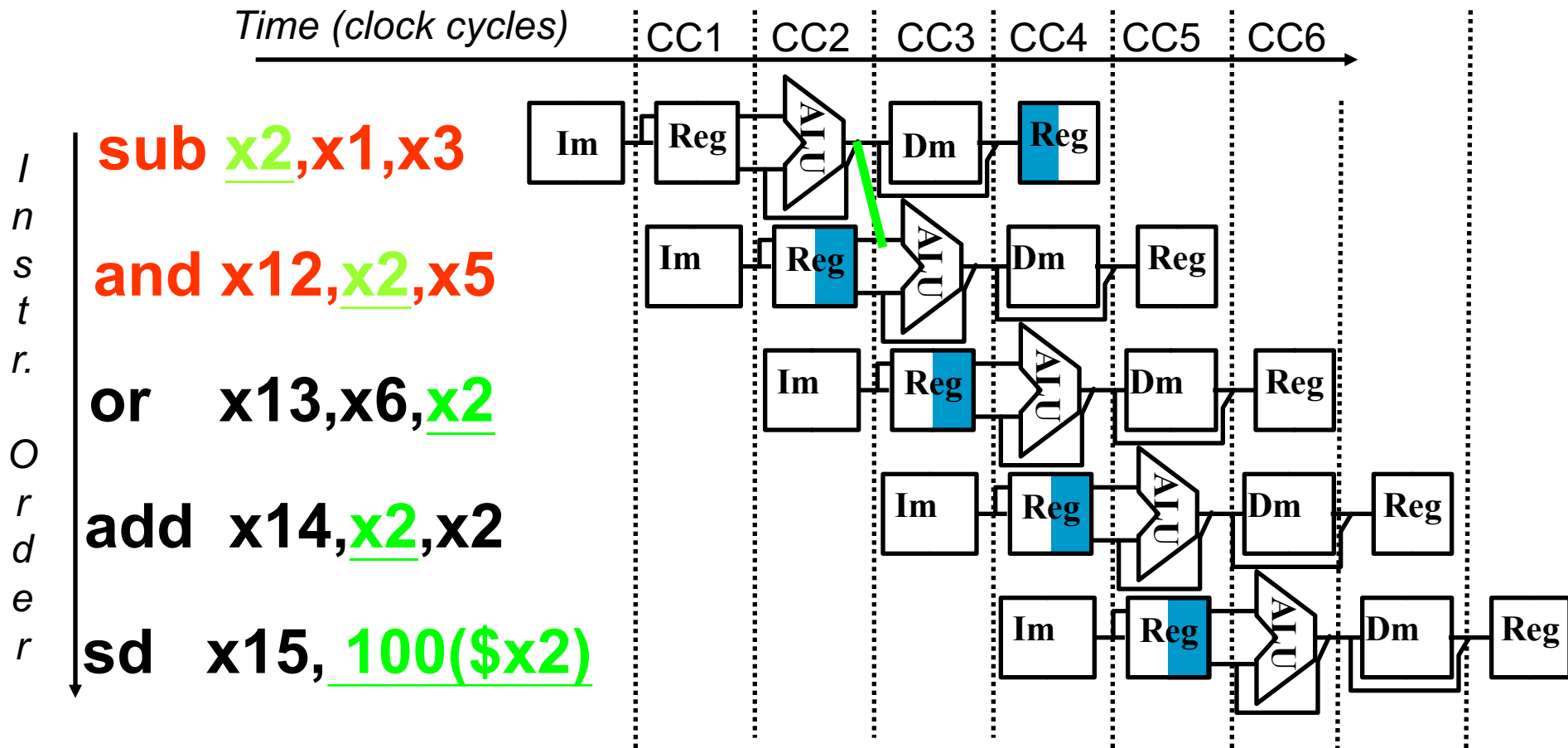# Pipeline II
Data Hazards and Forwarding

1. How to implement "data forwarding"?
2. How to detect load-use hazard? How to stall pipeline?
3. Exception
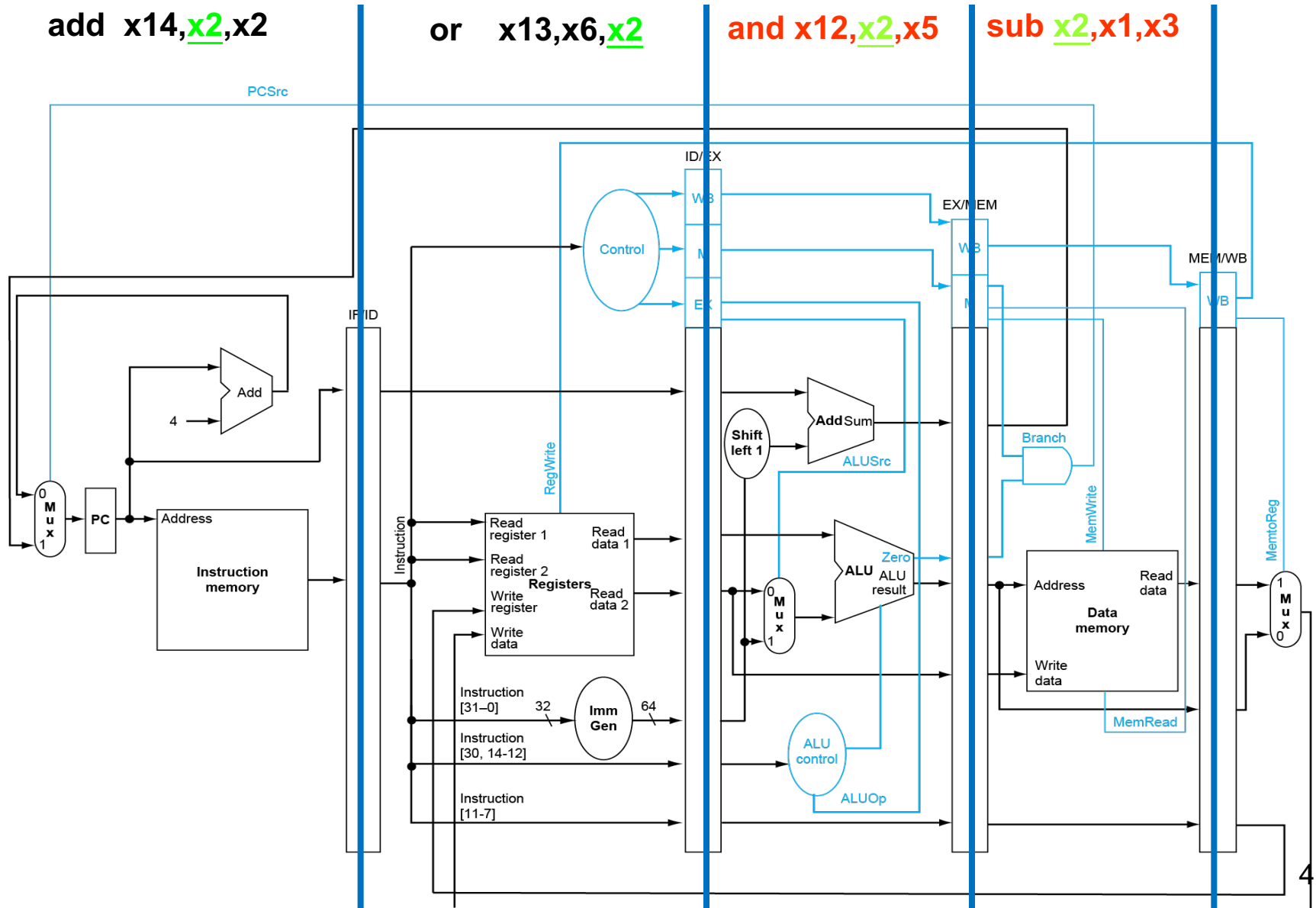
Data Dependence Detection & Forwarding

Time (clock cycles)

	CC1	CC2	CC3	CC4	CC5	CC6		
Value of register x2	10	10	10	10/-20	-20	-20	-20	-20

sub x2,x1,x3

and x12,x2,x5

or   x13,x6,x2

add x14,x2, x2

sd   x15, 100($x2)

Instr. Order

How to detect dependency between (sub, and)?

Time (clock cycles) — CC1 CC2 CC3 CC4 CC5 CC6

Instr. Order

sub x2,x1,x3

and x12,x2,x5
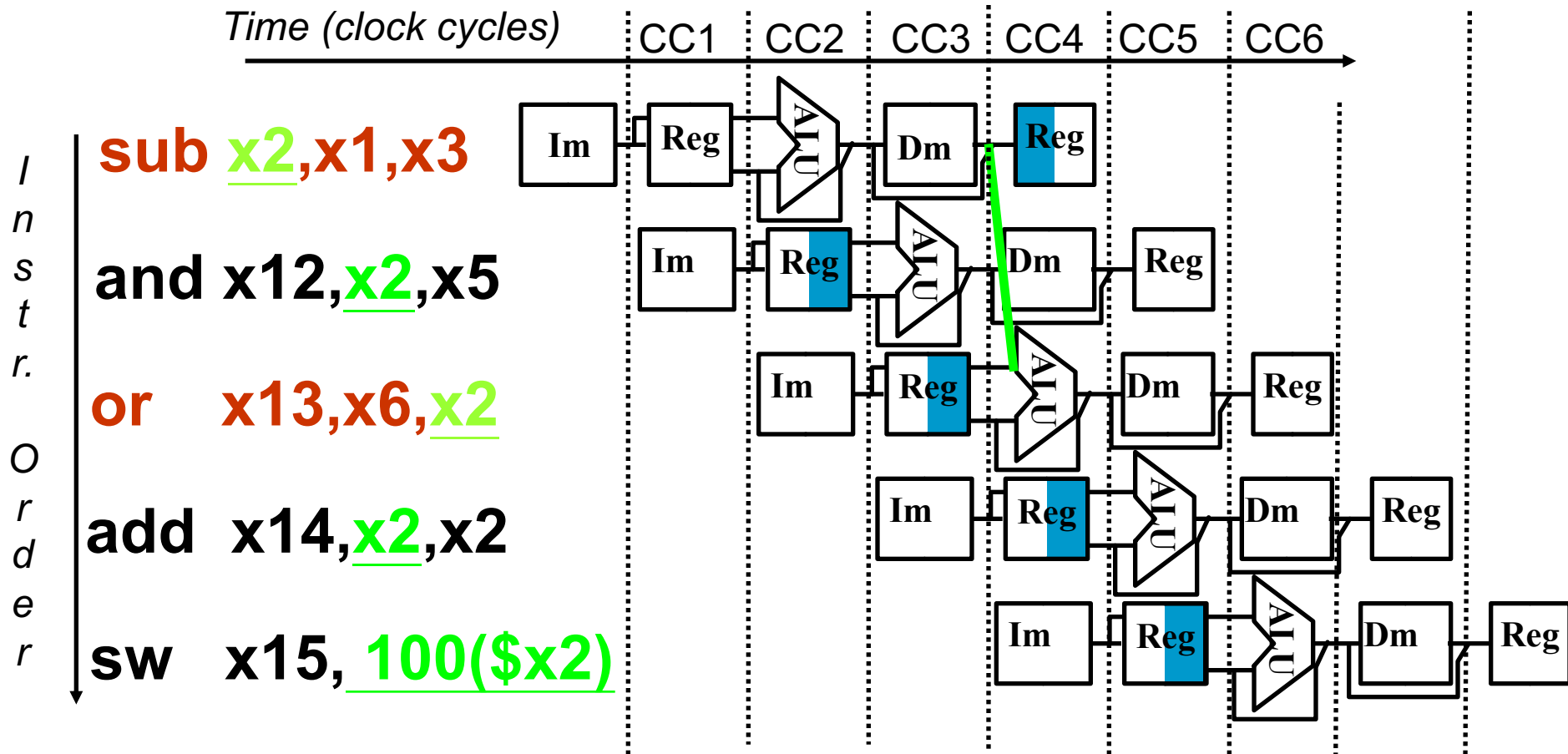
or   x13,x6,x2

add x14,x2,x2

sd   x15, 100($x2)

1a: EX/MEM. RegisterRd == ID/EX.RegisterR1
1b: EX/MEM. RegisterRd == ID/EX.RegisterR2

3

How to detect dependency between (sub, and)?

add x14,x2,x2   or   x13,x6,x2   and x12,x2,x5   sub x2,x1,x3

4

How to detect dependency between (sub, or)?

Time (clock cycles)

CC1 CC2 CC3 CC4 CC5 CC6

Instr. Order

sub x2,x1,x3

and x12,x2,x5

or   x13,x6,x2

add x14,x2,x2

sw   x15, 100($x2)

2a: MEM/WB.RegisterRd = = ID/EX.RegisterR1

2b: MEM/WB.RegisterRd = = ID/EX.RegisterR2

How to detect dependency between (sub, or)?

sw   x15, 100($x2)   add x14,x2,x2   or   x13,x6,x2   and x12,x2,x5   sub x2,x1,x3
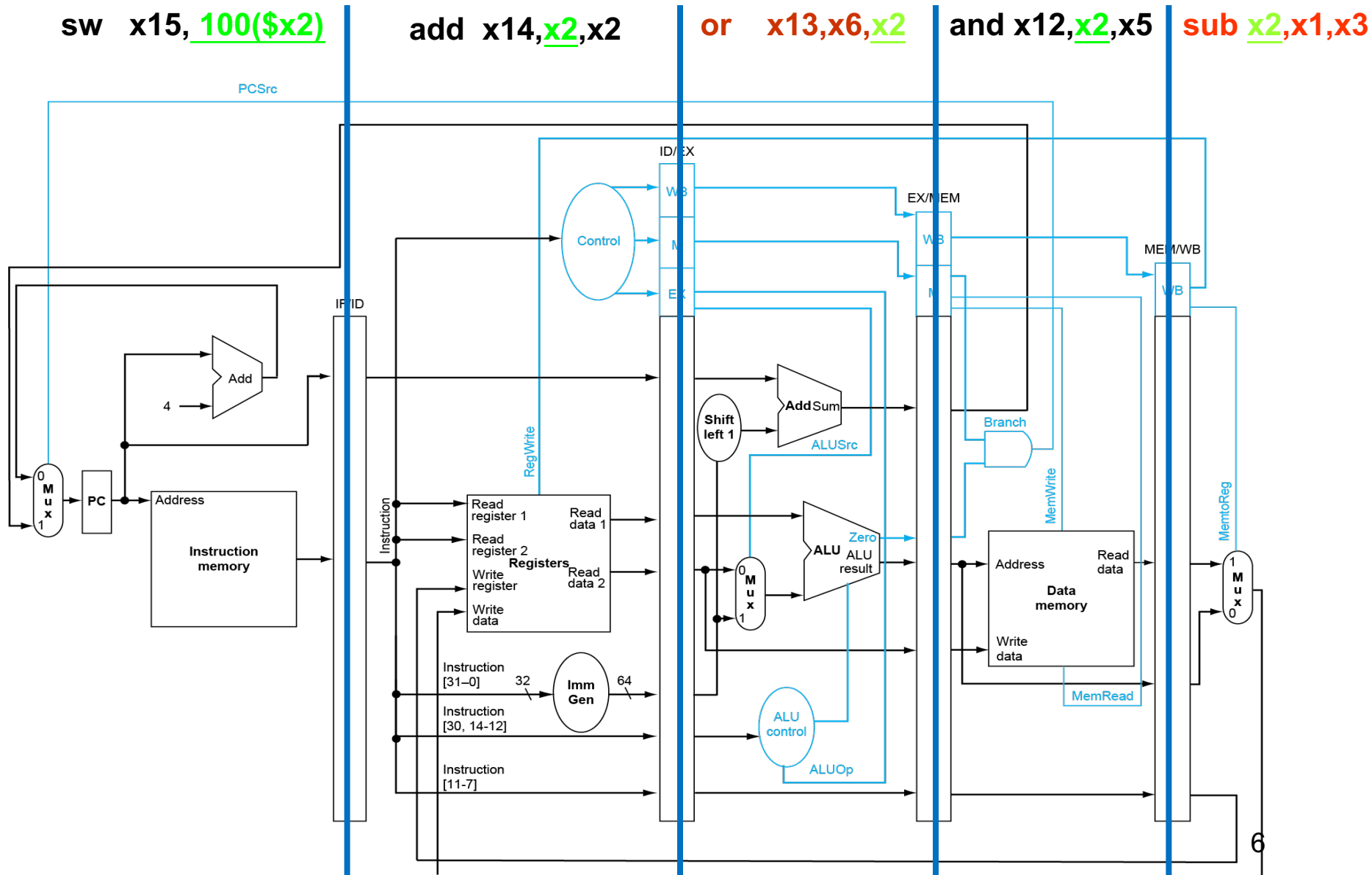
Data Dependence Detection (cont.)

■ Hazard conditions:

– 1a: EX/MEM. RegisterRd = ID/EX.RegisterR1   } EX hazard
– 1b: EX/MEM. RegisterRd = ID/EX.RegisterR2
– 2a: MEM/WB.RegisterRd = ID/EX.RegisterR1   } MEM hazard
– 2b: MEM/WB.RegisterRd = ID/EX.RegisterR2
– RegWrite signal of WB Control field
  • EX/MEM.RegWrite, MEM/WB.RegWrite
– EX/MEM.RegisterRd ≠ $0
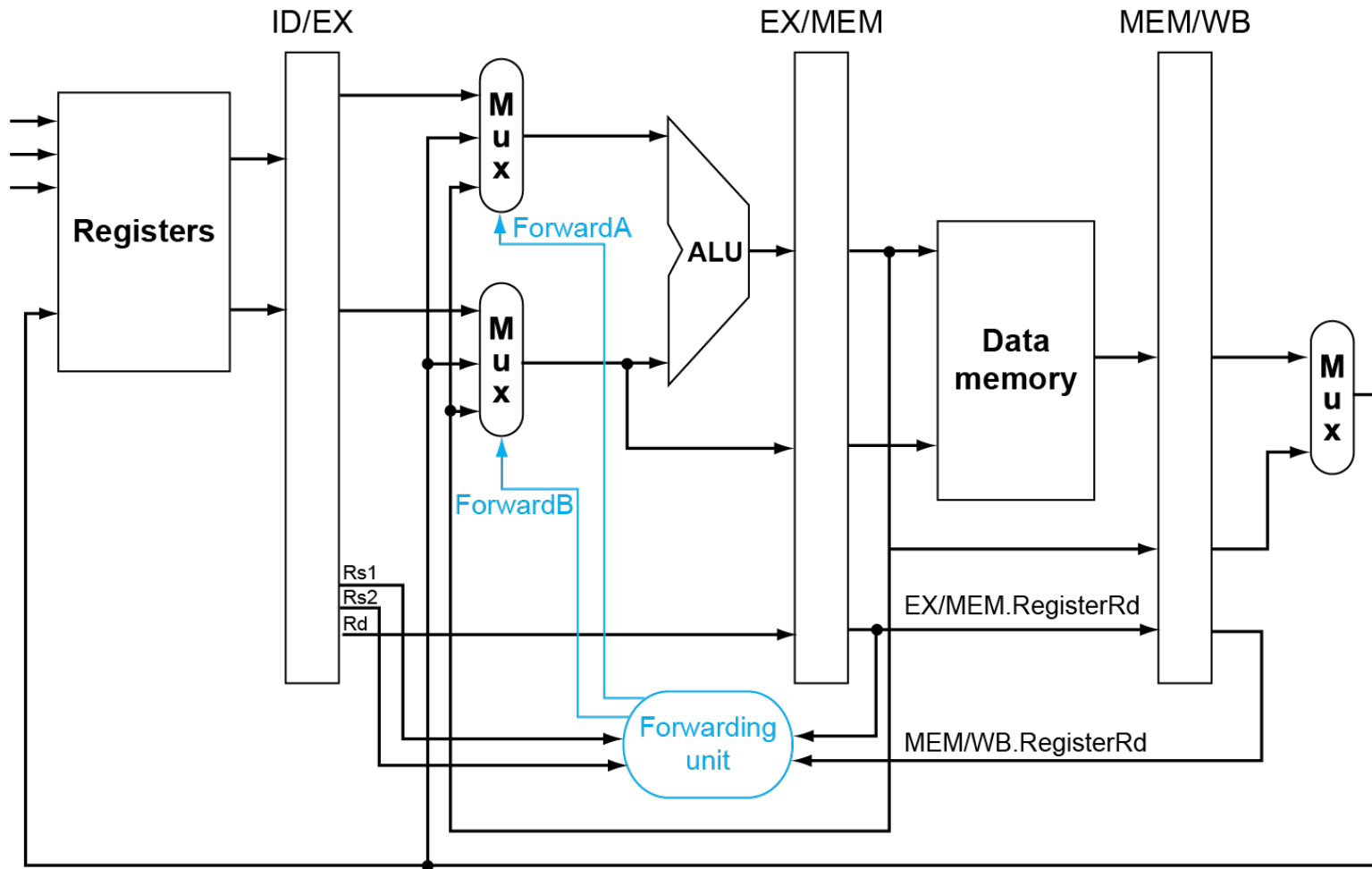– MEM/WB.RegisterRd ≠ $0

How to forward data?

Resolving Hazards by Forwarding

- Use the value in pipeline registers rather than waiting for the WB stage to write the register file.
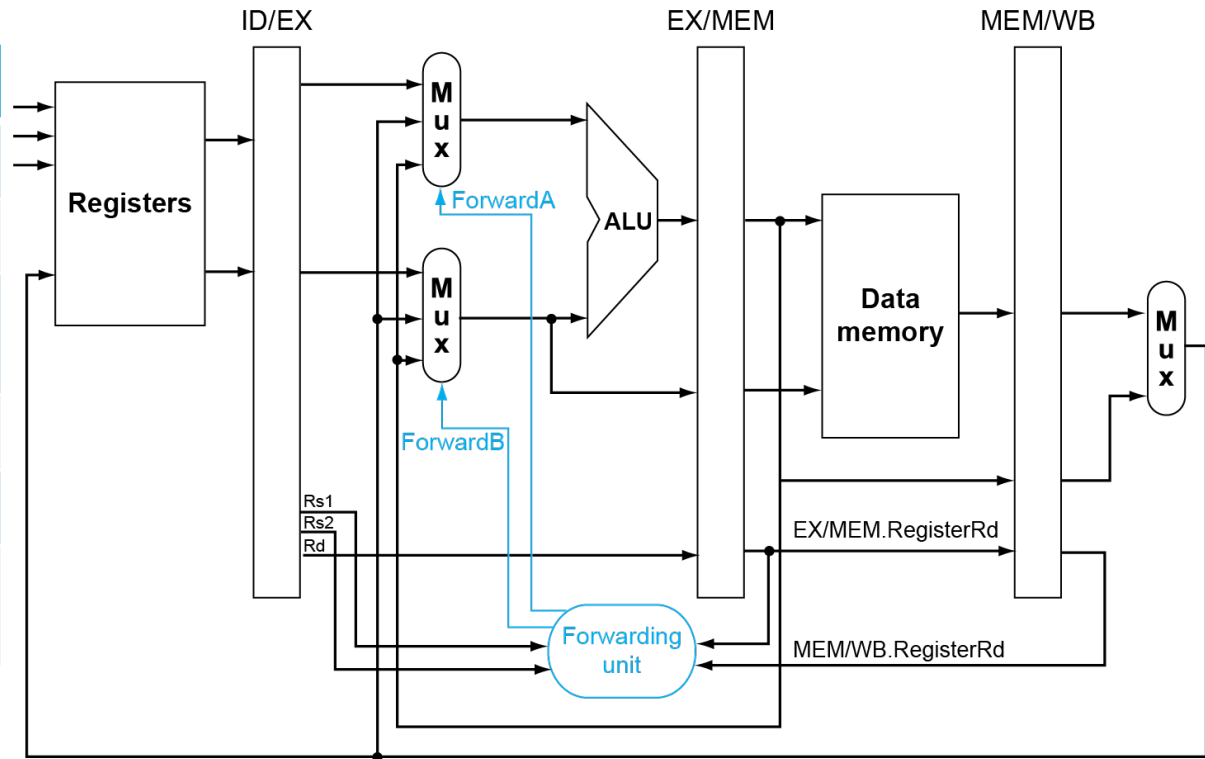 - EX/MEM.Aluout
  - MEM/WB.Aluout
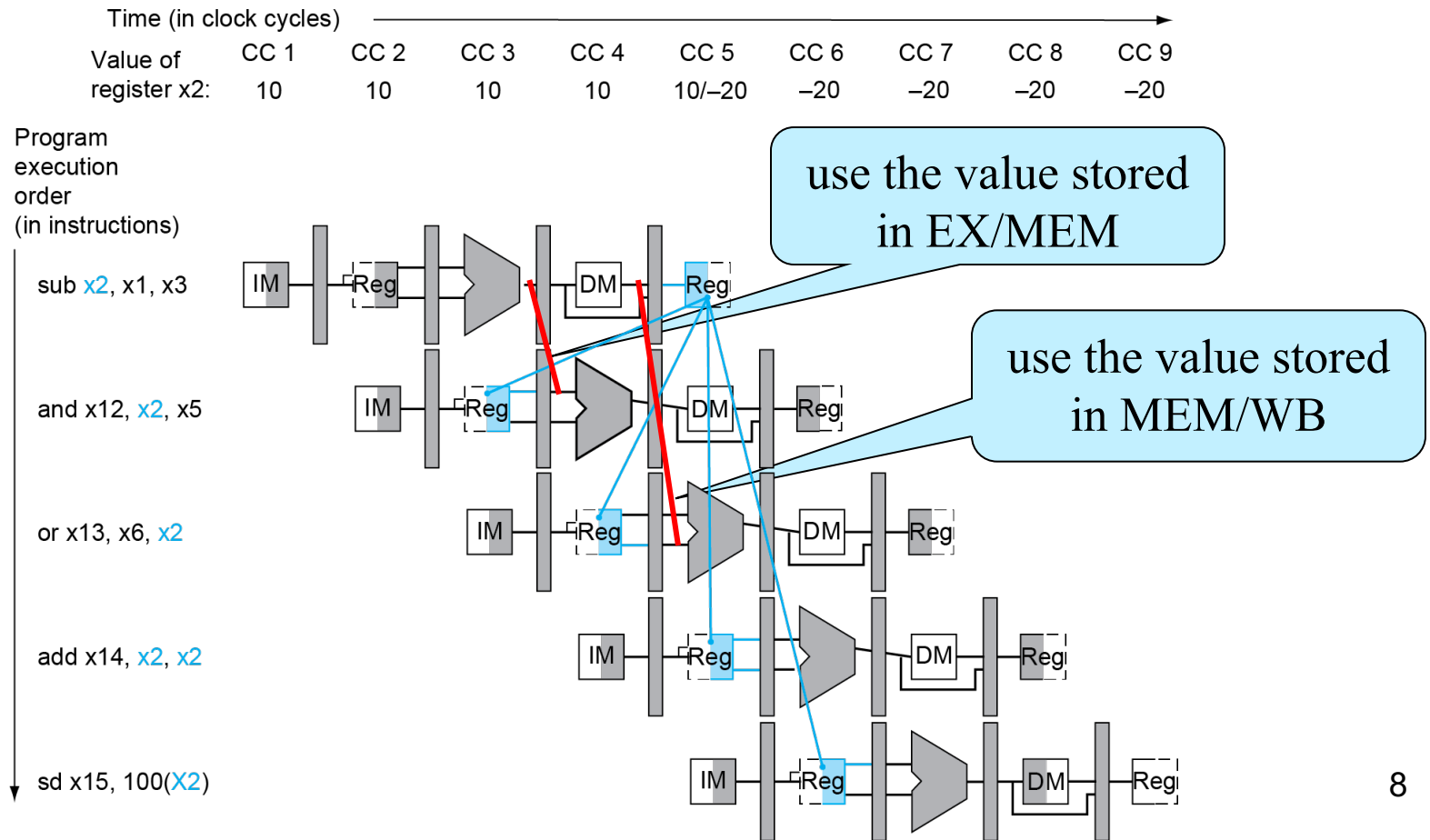
Forwarding Logic

- Forwarding: input to ALU from any pipe reg.
  - Add multiplexors to ALU input
  - Forwarding Control will be in EX

Forwarding Control

Mux control	Source
ForwardA = 00	ID/EX
ForwardA = 10	EX/MEM
ForwardA = 01	MEM/WB
ForwardB = 00	ID/EX
ForwardB = 10	EX/MEM
ForwardB = 01	MEM/WB

Forwarding Control

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.

1. EX hazard

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd=ID/EX.RegisterR1))
ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
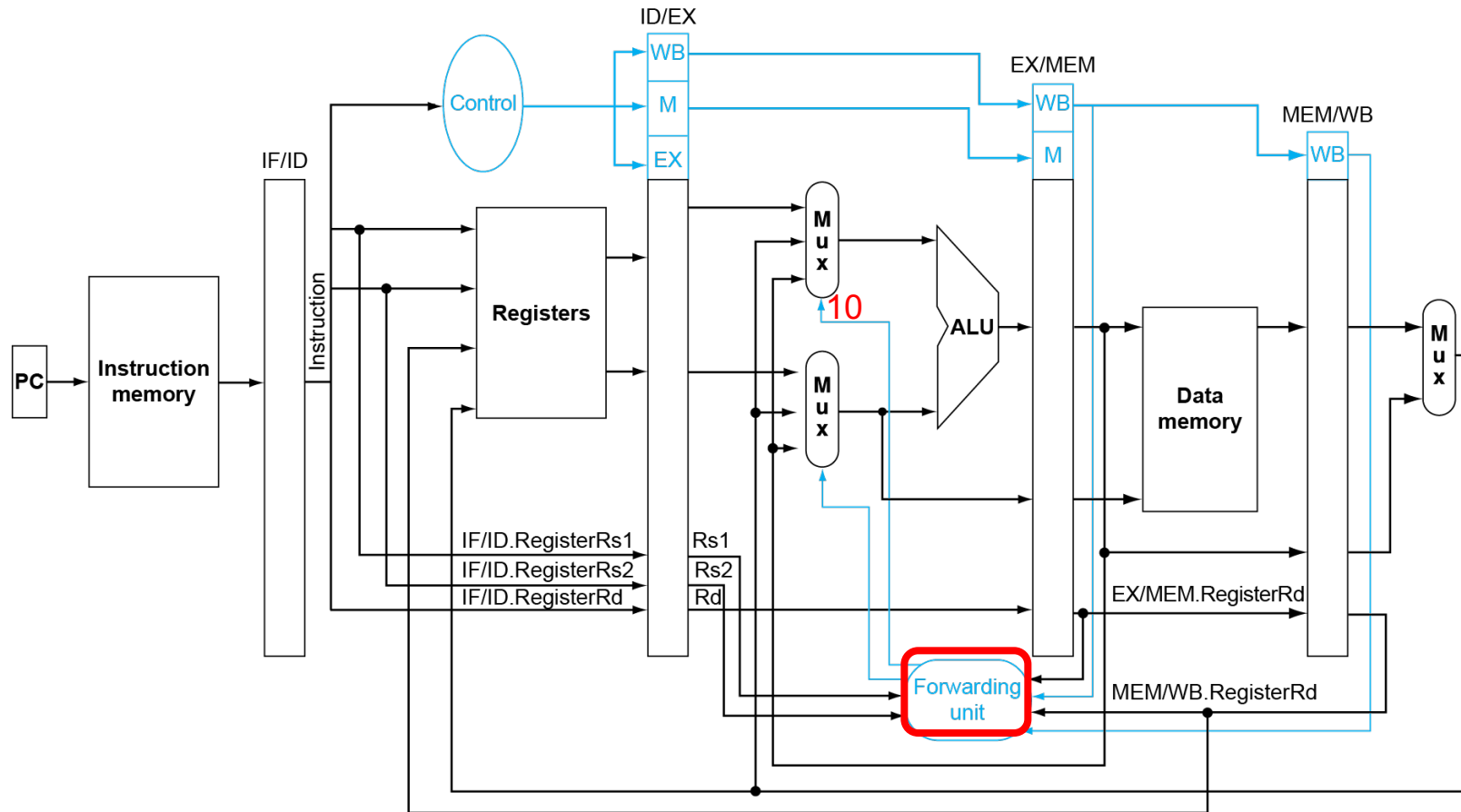and (EX/MEM.RegisterRd=ID/Ex.RegisterR2))
ForwardB = 10

2. MEM hazard

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd=ID/Ex.RegisterR1))
ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd=ID/Ex.RegisterR2))
ForwardB = 01

11

or   x13,x6,x2   and x12,x2,x5   sub x2,x1,x3
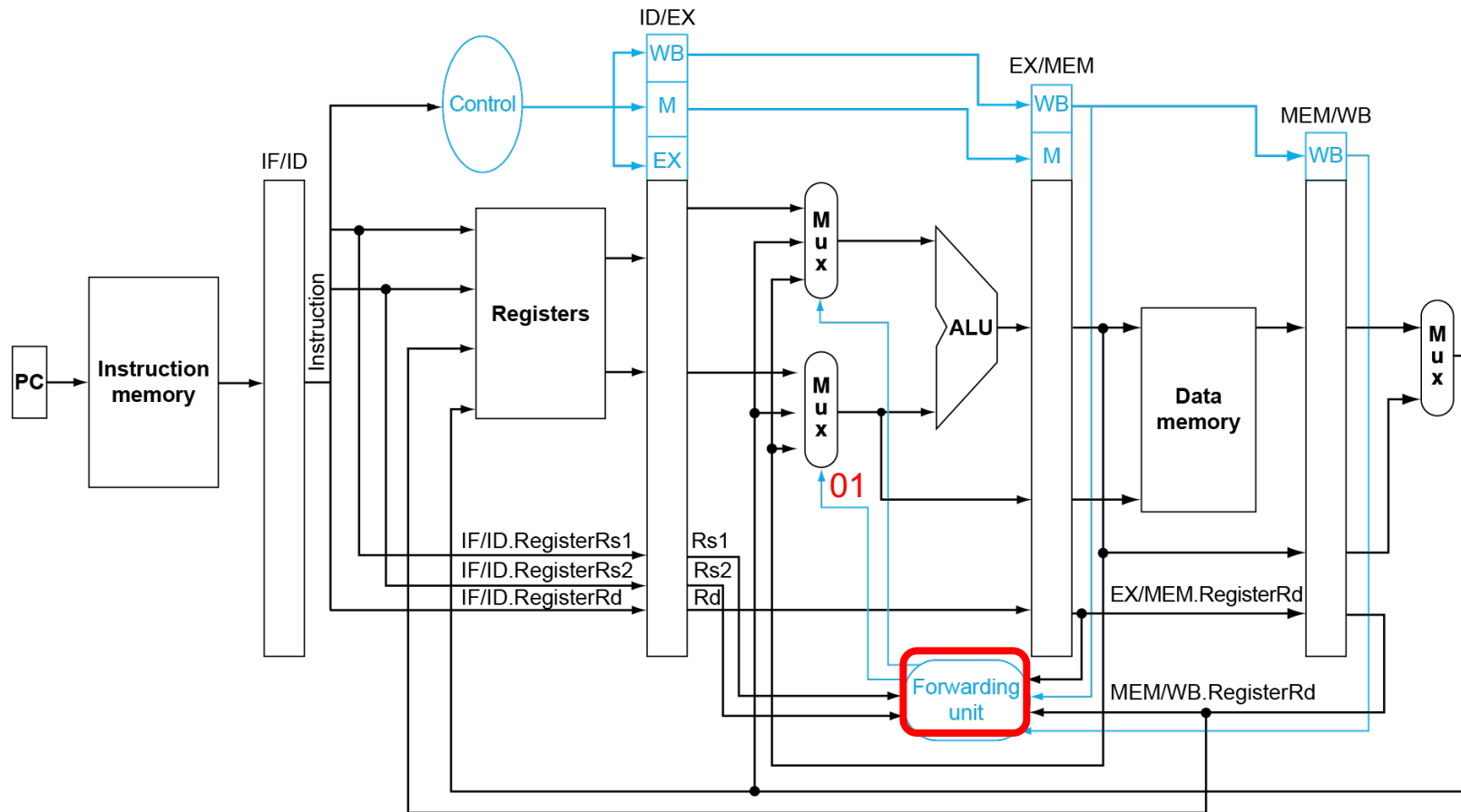

1. EX hazard
    if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd=ID/EX.RegisterR1))
    ForwardA = 10

sw   x15, 100($x2)   add x14,x2,x2   or   x13,x6,x2   and x12,x2,x5   sub x2,x1,x3

2. MEM hazard
   if (MEM/WB.RegWrite
   and (MEM/WB.RegisterRd ≠ 0)
   and (MEM/WB.RegisterRd=ID/Ex.RegisterR2))
   ForwardB = 01

Forwarding Control (cont.)

inst1   `add $1,$1,$2;`   `IF ID EX  MEM WB`

inst2   `add $1,$1,$3;`   `IF ID  EX  MEM  WB`

inst3   `add $1,$1,$4;`   `IF ID  EX  MEM WB`

   ……

   => Which instruction should forward its results to instruction 3?

MEM hazard condition becomes
   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.RegisterRs1))
   and (MEM/WB.RegRd=ID/Ex.RegisterRs1)) ForwardA = 01

   if (MEM/WB.RegWrite
   and (MEM/WB.RegRd ≠ 0)
   and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
       and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
   and (MEM/WB.RegRd=ID/Ex.RegisterRt))   ForwardB = 01

Example

- Show how forwarding works with this instruction sequence (with dependencies highlighted):

  sub $2, $1, $3

  and $4, $2, $5

  or $4, $4, $2
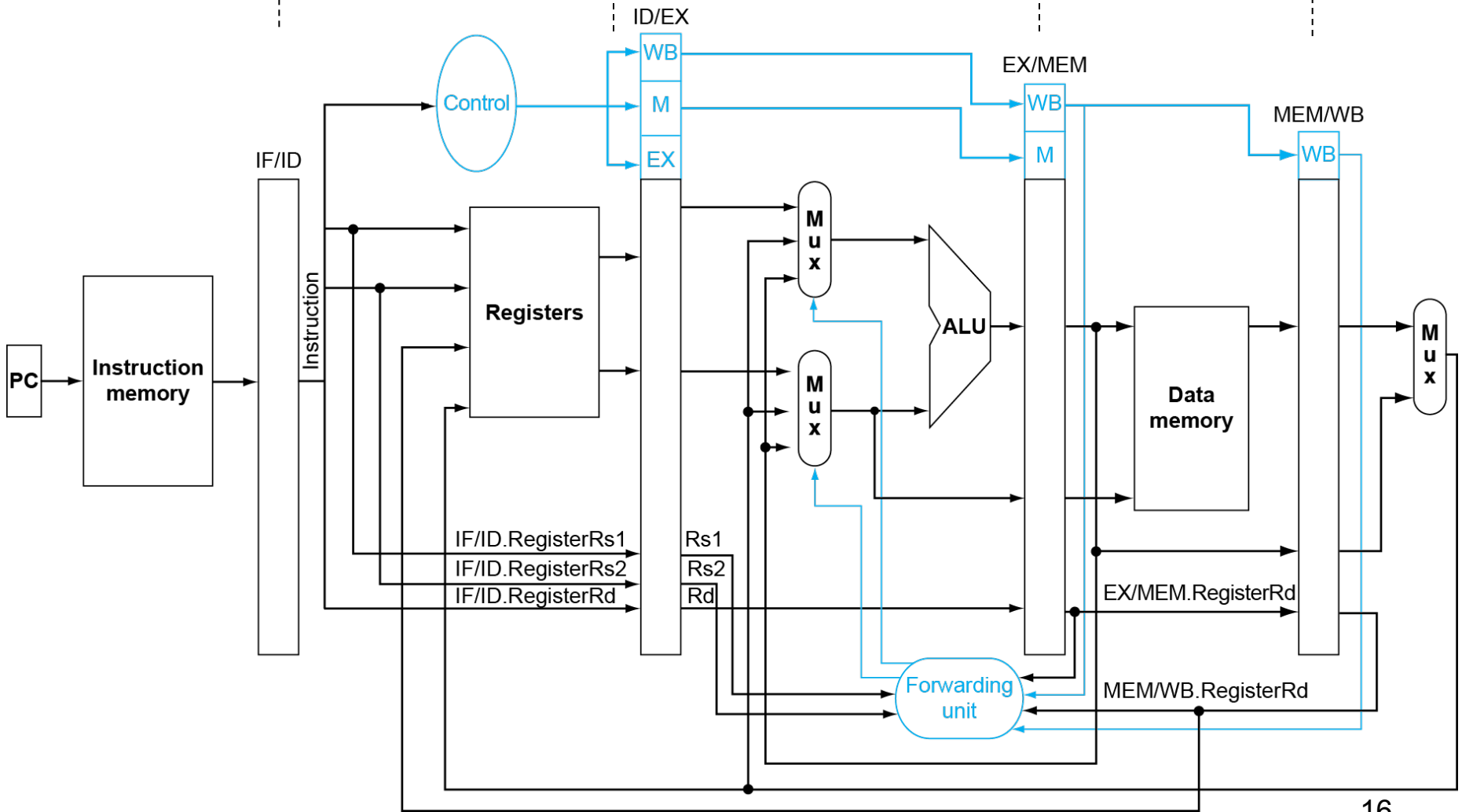
  add $9, $4, $2
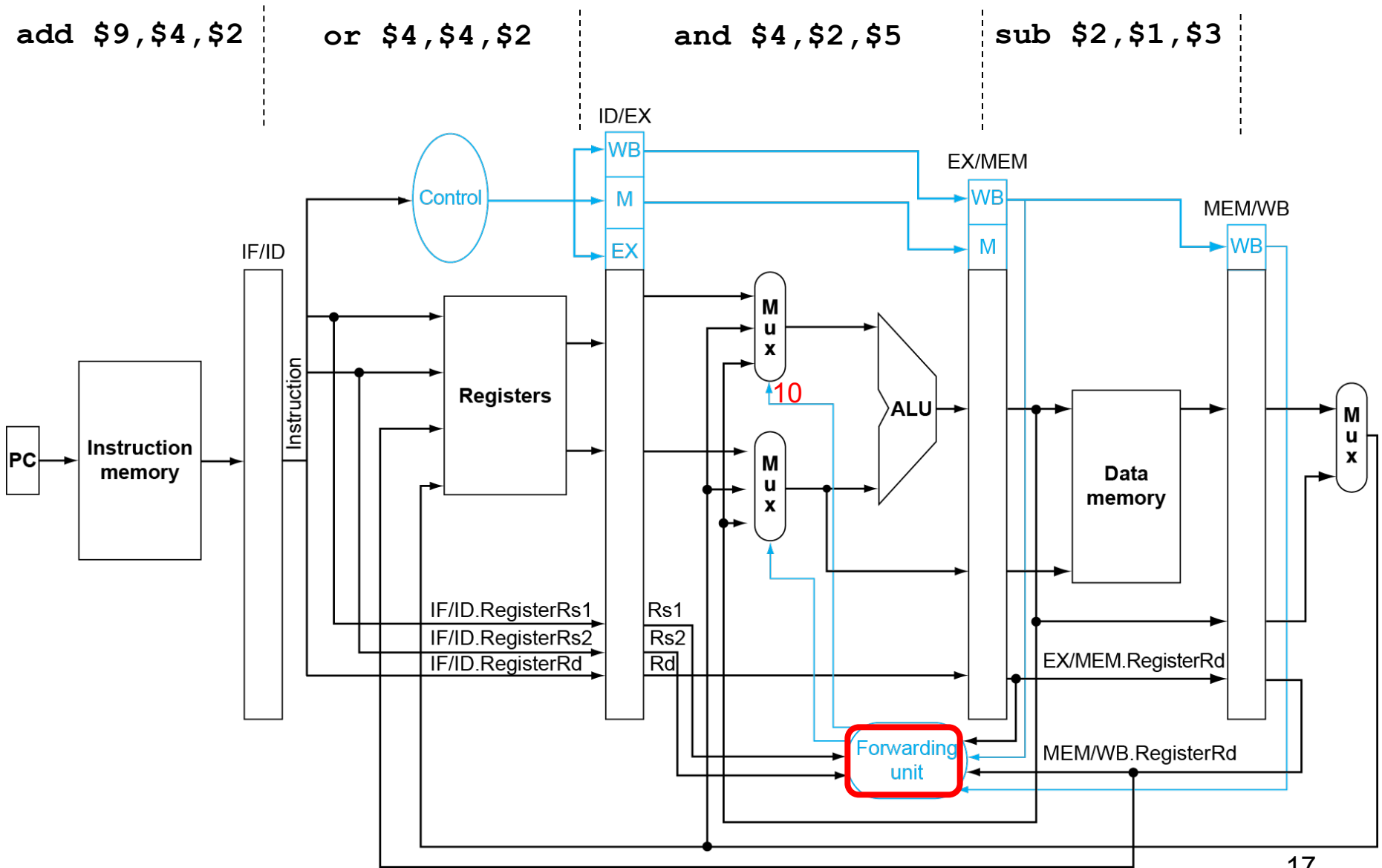
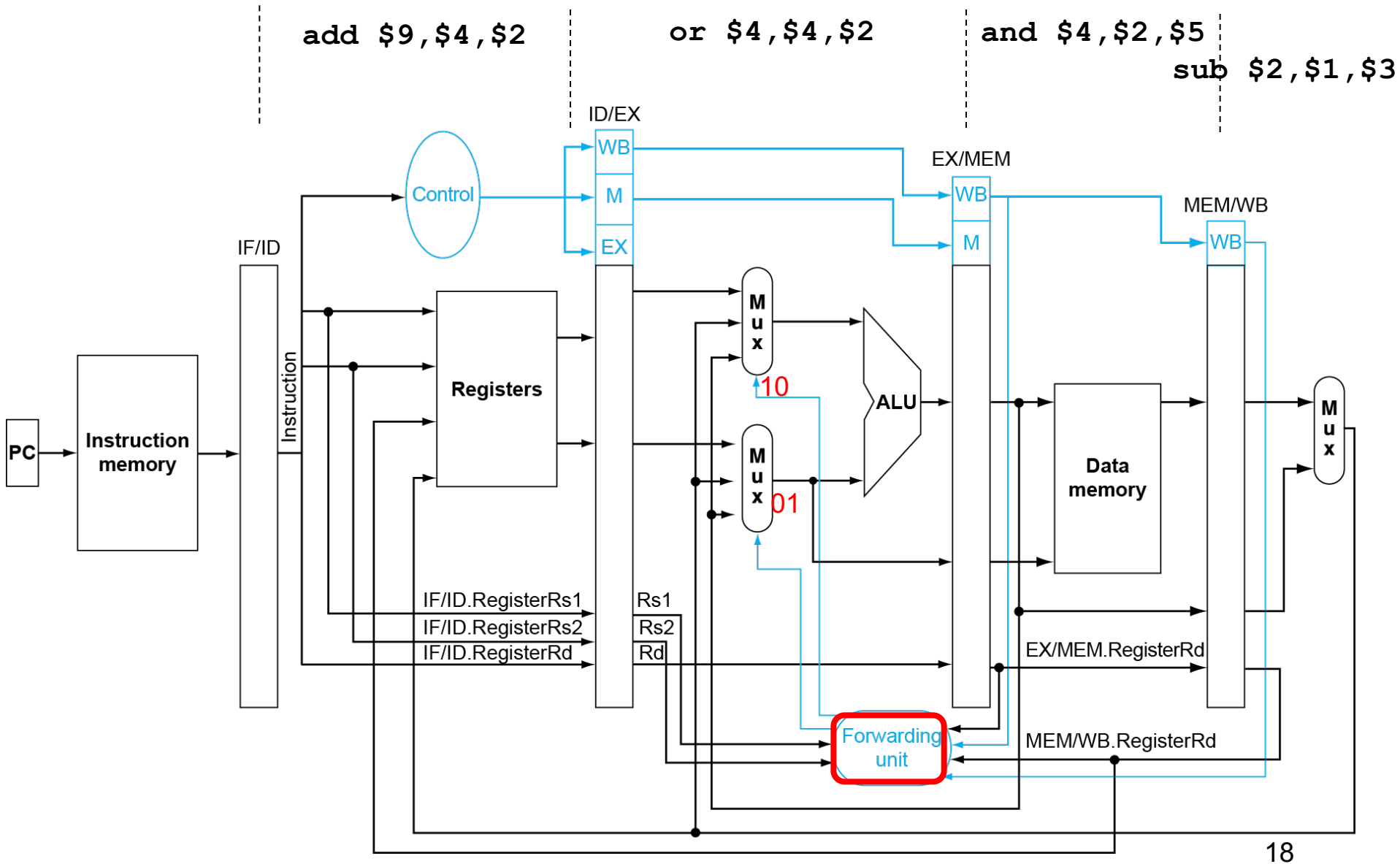add $9,$4,$2   or $4,$4,$2   and $4,$2,$5   sub $2,$1,$3

Cycle 5

add $9,$4,$2   or $4,$4,$2   and $4,$2,$5
                                                    sub $2,$1,$3

Cycle 6

add $9,$4,$2   or $4,$4,$2   and ...
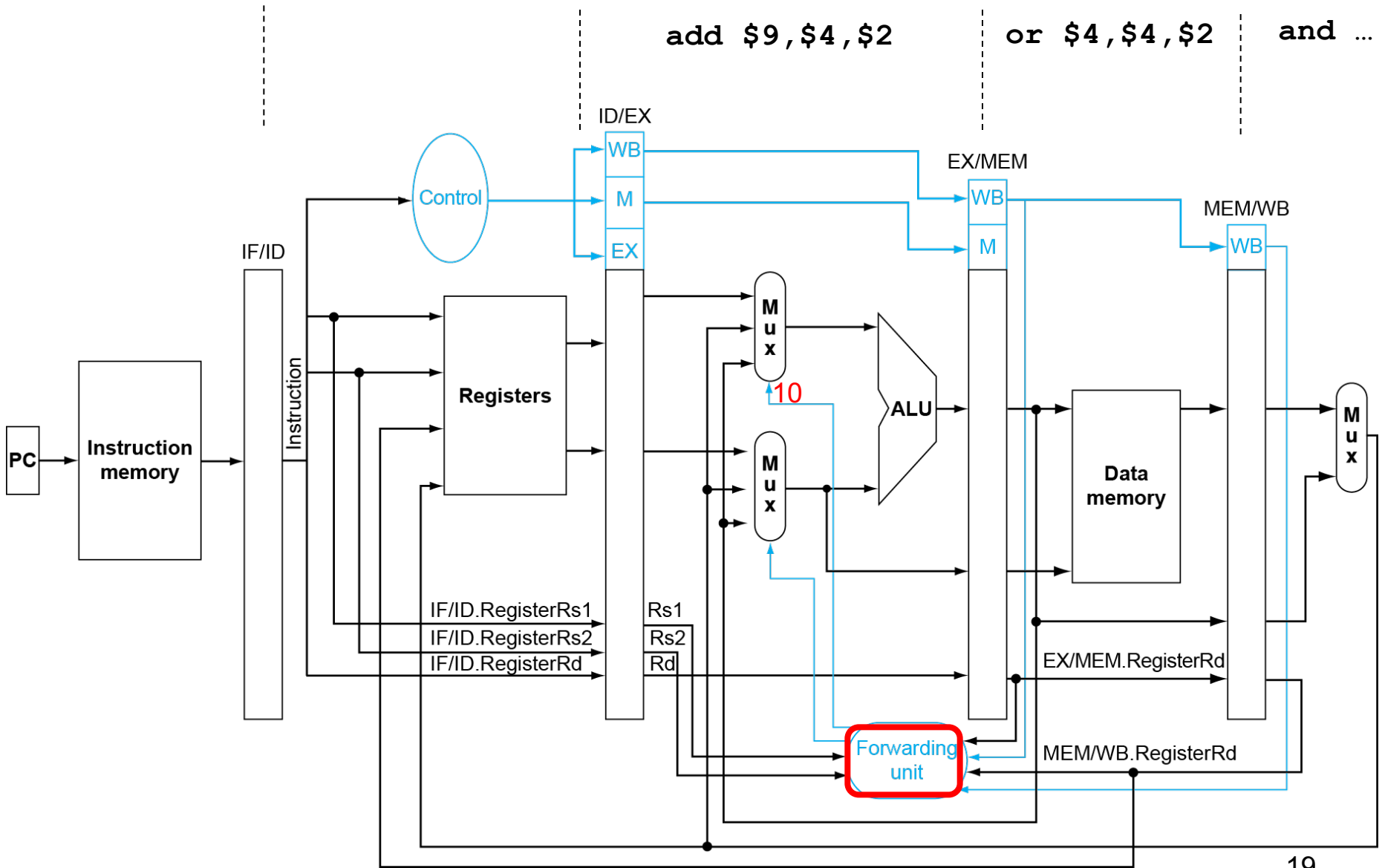
Can't always forward

- Load can still cause a hazard:
  - an instruction tries to read a register following a load instruction that writes to the same register.
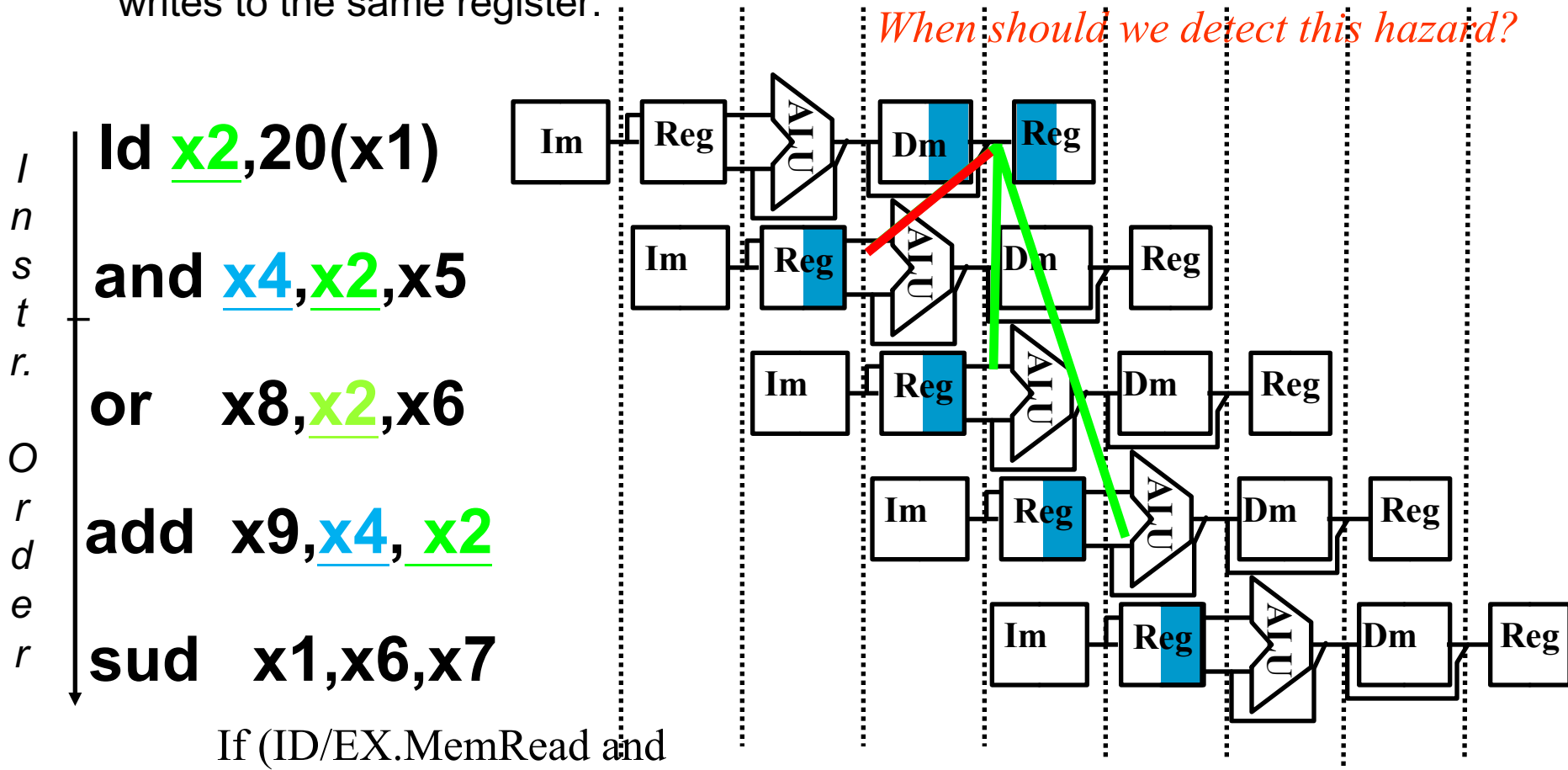
When should we detect this hazard?

Instr. Order

ld x2,20(x1)

and x4,x2,x5

or x8,x2,x6

add x9,x4, x2

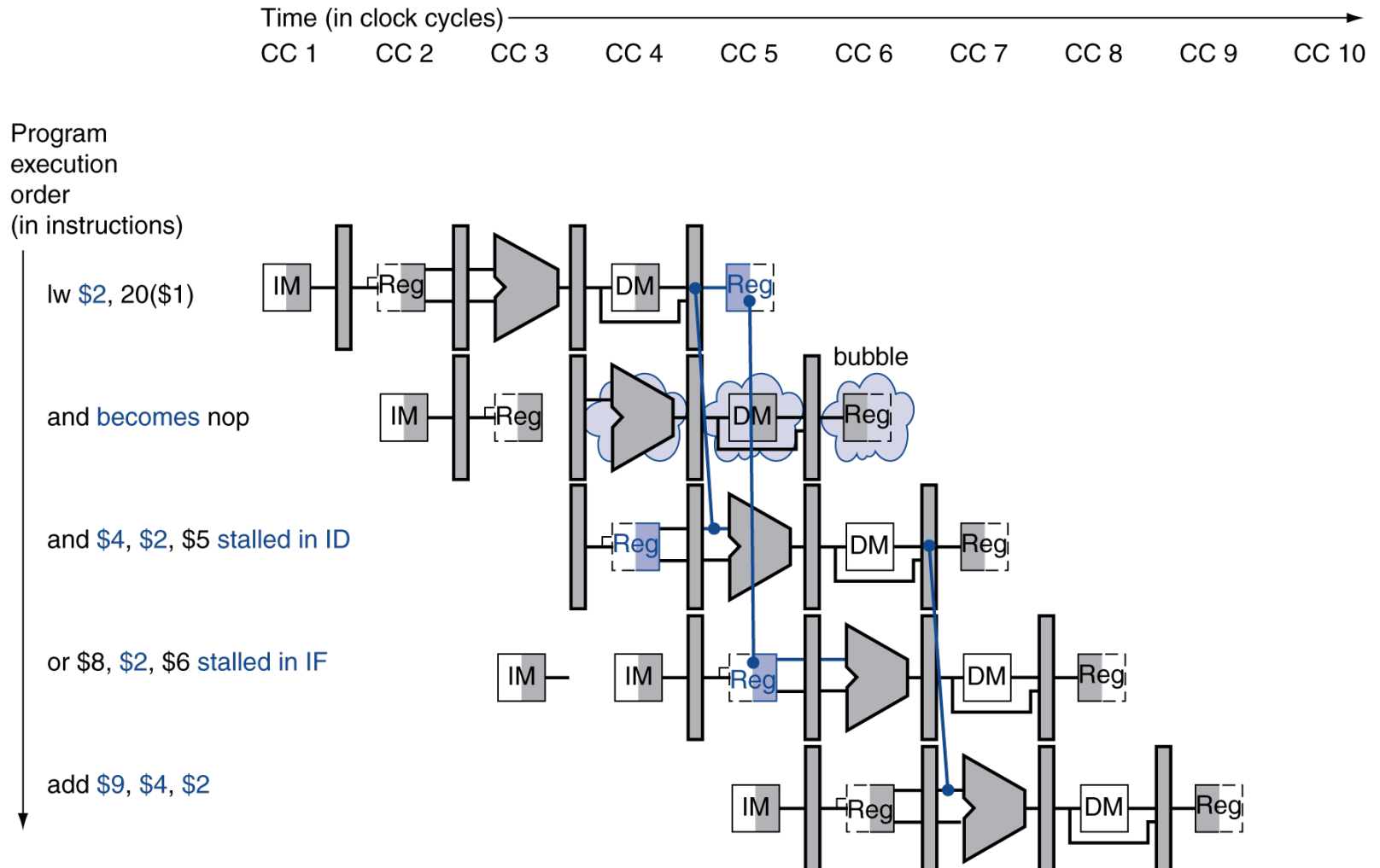**sud x1,x6,x7**

If (ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
    stall the pipeline

Hazard Detection and Stall

- If (ID/EX.MemRead and

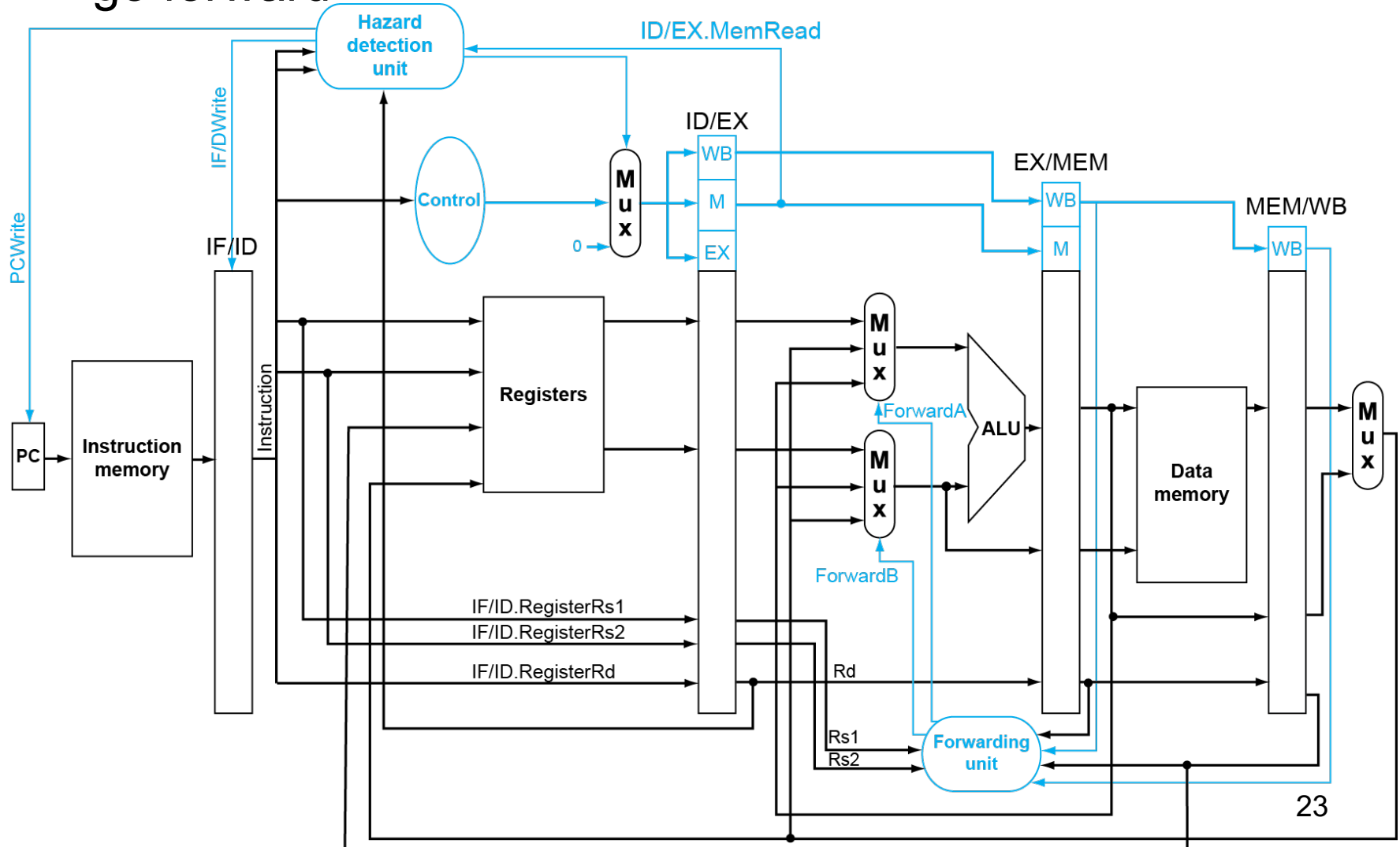 ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or

  (ID/EX.RegisterRd = IF/ID.REgisterRs2)))

  stall the pipeline

- Stall the pipeline
  - Preventing instructions in the IF and ID stages from making progress
    - Preserve the PC and IF/ID pipeline registers
  - We need to do nothing in EX at CC4, MEM at CC5 , WB at CC6
    - and become a nop (deasserting all control signals in the EX, MEM and WB stage)

Stall/Bubble in the Pipeline

Time (in clock cycles)

CC 1 | CC 2 | CC 3 | CC 4 | CC 5 | CC 6 | CC 7 | CC 8 | CC 9 | CC 10

Program execution order (in instructions)

lw $2, 20($1)

and becomes nop

and $4, $2, $5 stalled in ID

or $8, $2, $6 stalled in IF

add $9, $4, $2

bubble

- **Stall by letting an instruction that won't write anything go forward**
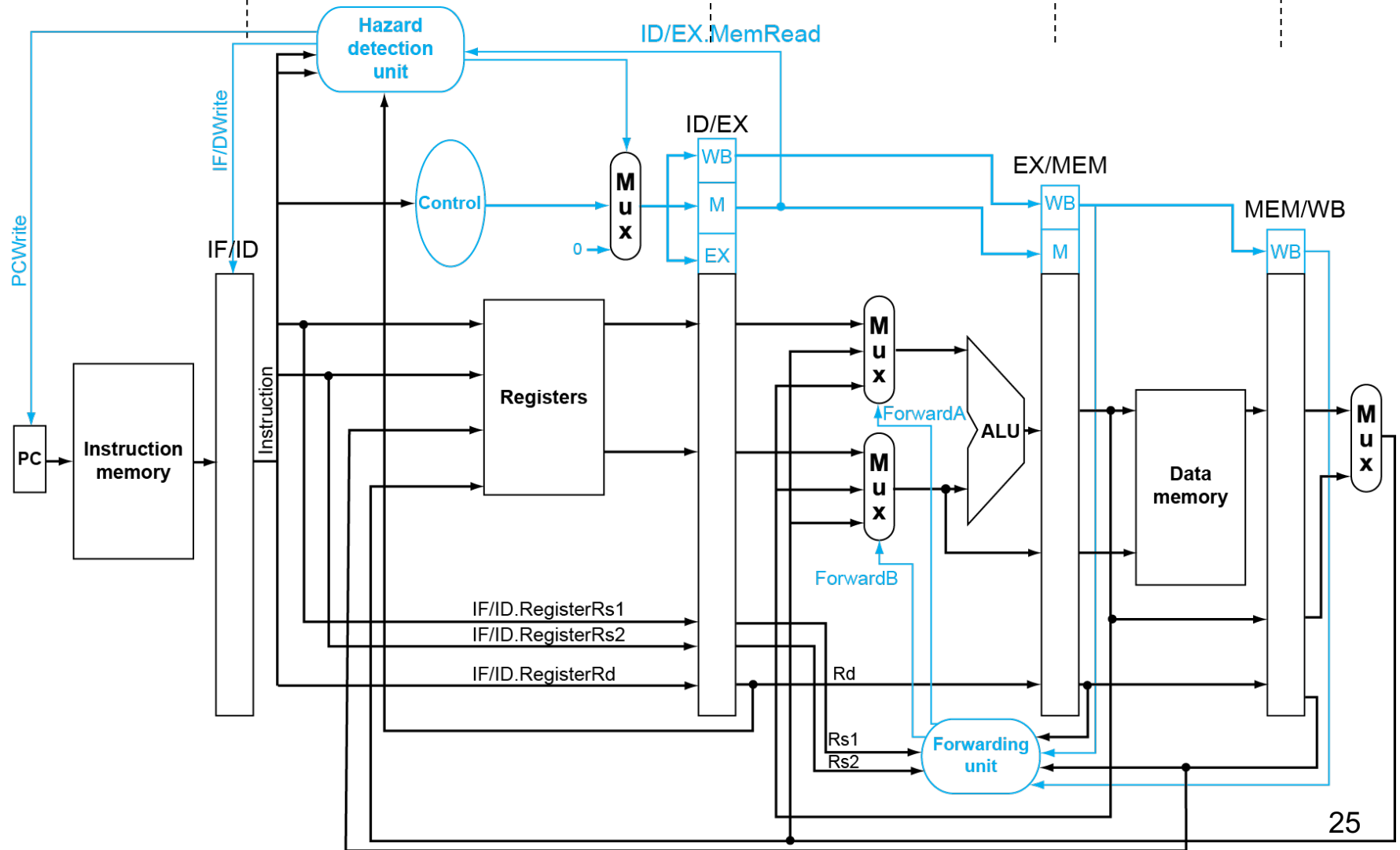
- Show how hazard detection unit works with this instruction sequence (with dependencies highlighted):

ld $2, 20($1)
and $4, $2, $5   load-use data hazard
or $4, $4, $2   Forwarding
add $9, $4, $2

`and $4,$2,$5`   `ld $2,20($1)`

25

or $4,$4,$2   and $4,$2,$5   ld $2,20($1)

Hazard detection unit

ID/EX.MemRead

IF/DWrite

PCWrite

Control

ID/EX
WB
M
EX

EX/MEM
WB
M

MEM/WB
WB

IF/ID

PC

Instruction memory

Instruction

Registers

Mux

Mux

ForwardA

ALU

Mux

ForwardB

Data memory

Mux

IF/ID.RegisterRs1
IF/ID.RegisterRs2
IF/ID.RegisterRd

Rd

Rs1
Rs2

Forwarding unit

26

or $4,$4,$2   and $4,$2,$5   **bubble**   ld $2,20($1)

27

add $9,$4,$2   or $4,$4,$2   and $4,$2,$5   **bubble**   ld $2,20($

28

Cycle 6

add $9,$4,$2   or $4,$4,$2   and $4,$2,$5 bubble

add $9,$4,$2   or $4,$4,$2   and $4,$2

30

Control Hazard Solution

1. How to resolve branch in the decode stage?
2. How to flush pipeline?

Control Hazard Solutions

■ Stall: wait until decision is clear

Time (clock cycles)

Instr. Order

Add
Mem — Reg — ALU — Mem — Reg

Beq
Mem — Reg — ALU — Mem — Reg

Load
stall stall Mem — Reg — ALU — Mem — Reg
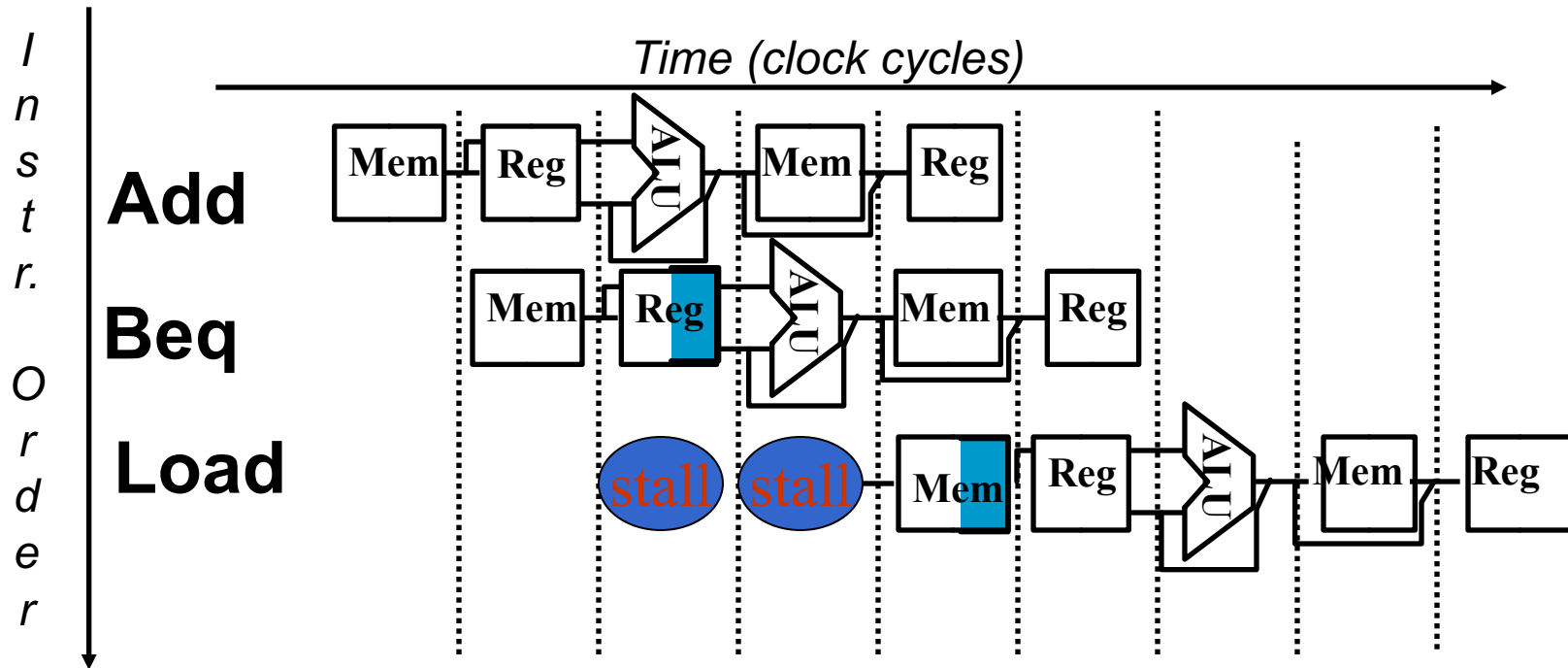
■ Impact: 3 clock cycles per branch instruction
=> slow

Control Hazard Solution (1): Reducing the Delay of Branches
(Example: BEQZ, BNEZ)

Instruction Fetch — **Instr. Decode Reg. Fetch** — **Execute Addr. Calc** — **Memory Access** — **Write Back**

Next PC

Next SEQ PC

MUX

Adder

4

Address

Memory

IF/ID

RS1

RS2

Reg File

Sign Extend

Imm

Zero?

ID/EX

MUX

ALU

EX/MEM

Data Memory

MEM/WB

MUX

WB Data

RD — RD — RD
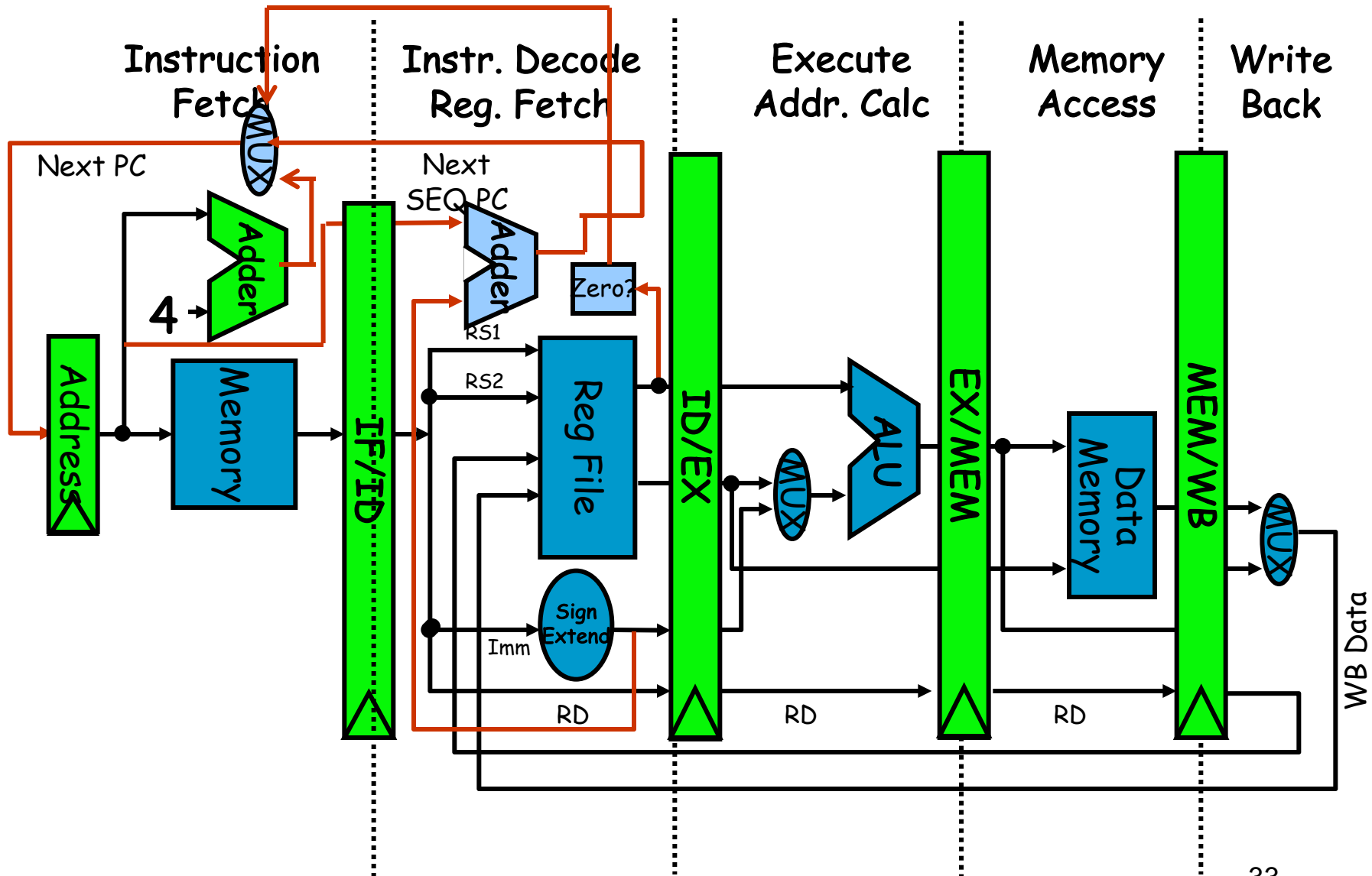
Data Hazards for Branches

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

add $1, $2, $3   [IF] [ID] [EX] [MEM] [WB]

add $4, $5, $6   [IF] [ID] [EX] [MEM] [WB]

…   [IF] [ID] [EX] [MEM] [WB]

beq $1, $4, target   [IF] [ID] [EX] [MEM] [WB]
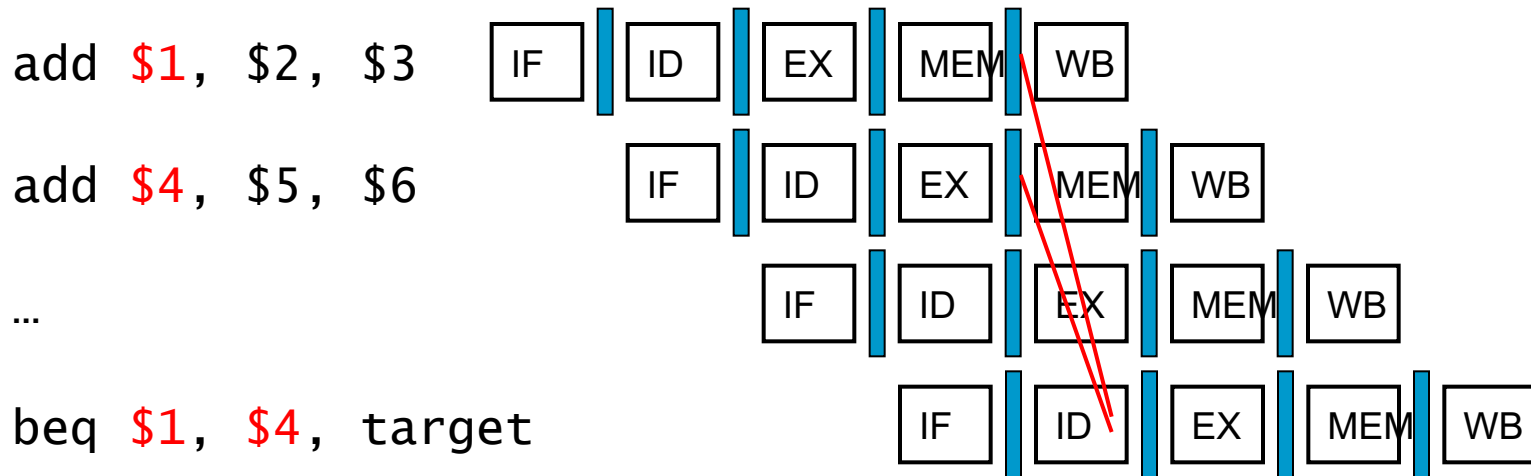
Can resolve using forwarding

Data Hazards for Branches

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

```
ld   $1, addr       | IF | ID | EX | MEM | WB |

add  $4, $5, $6          | IF | ID | EX | MEM | WB |

beq  stalled                 | IF | ID |  ☁  ☁  ☁

beq  $1, $4, target                    | ID | EX | MEM | WB |
```
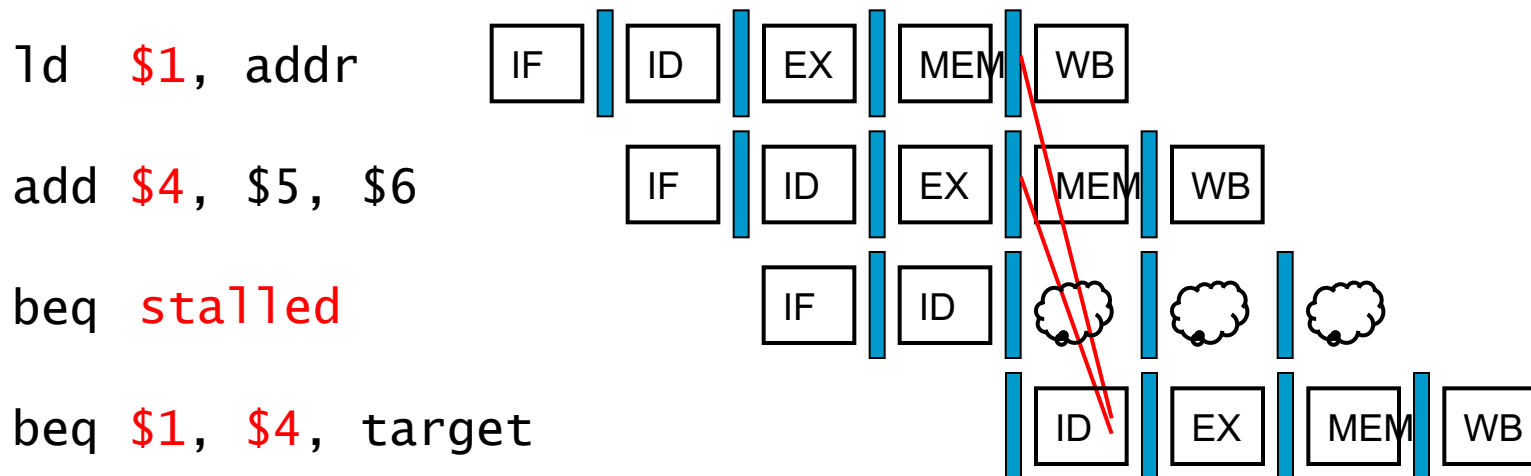
Data Hazards for Branches

- **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`

Control Hazard Solutions (1)

Time (clock cycles)
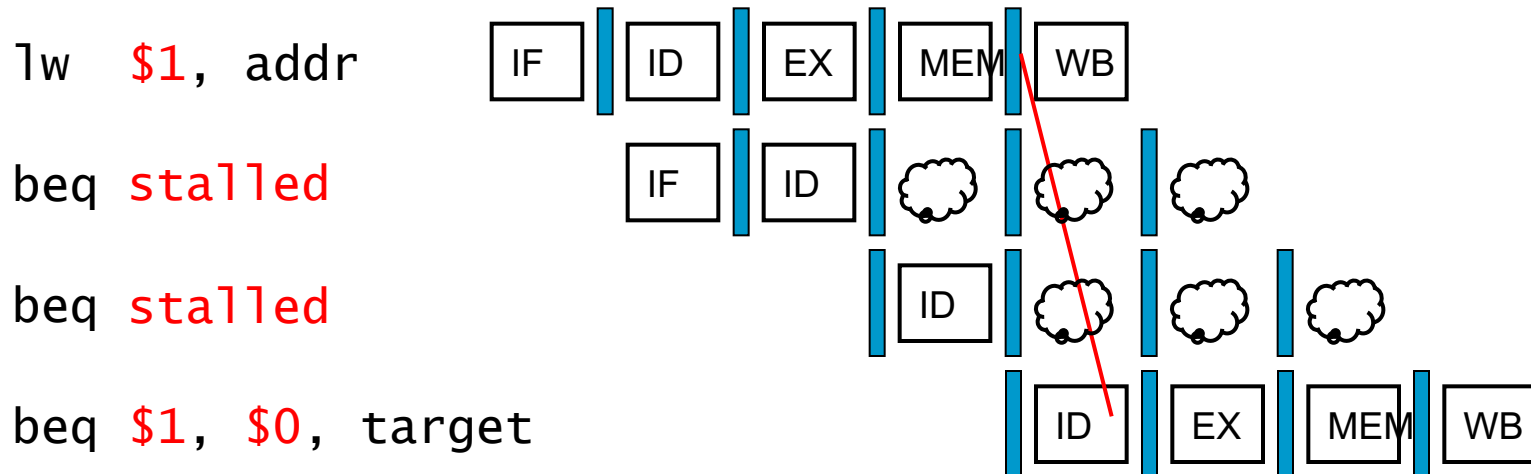
Instr. Order

Add — Mem Reg ALU Mem Reg

Beq — Mem Reg ALU Mem Reg

Load — stall Mem Reg ALU Mem Reg
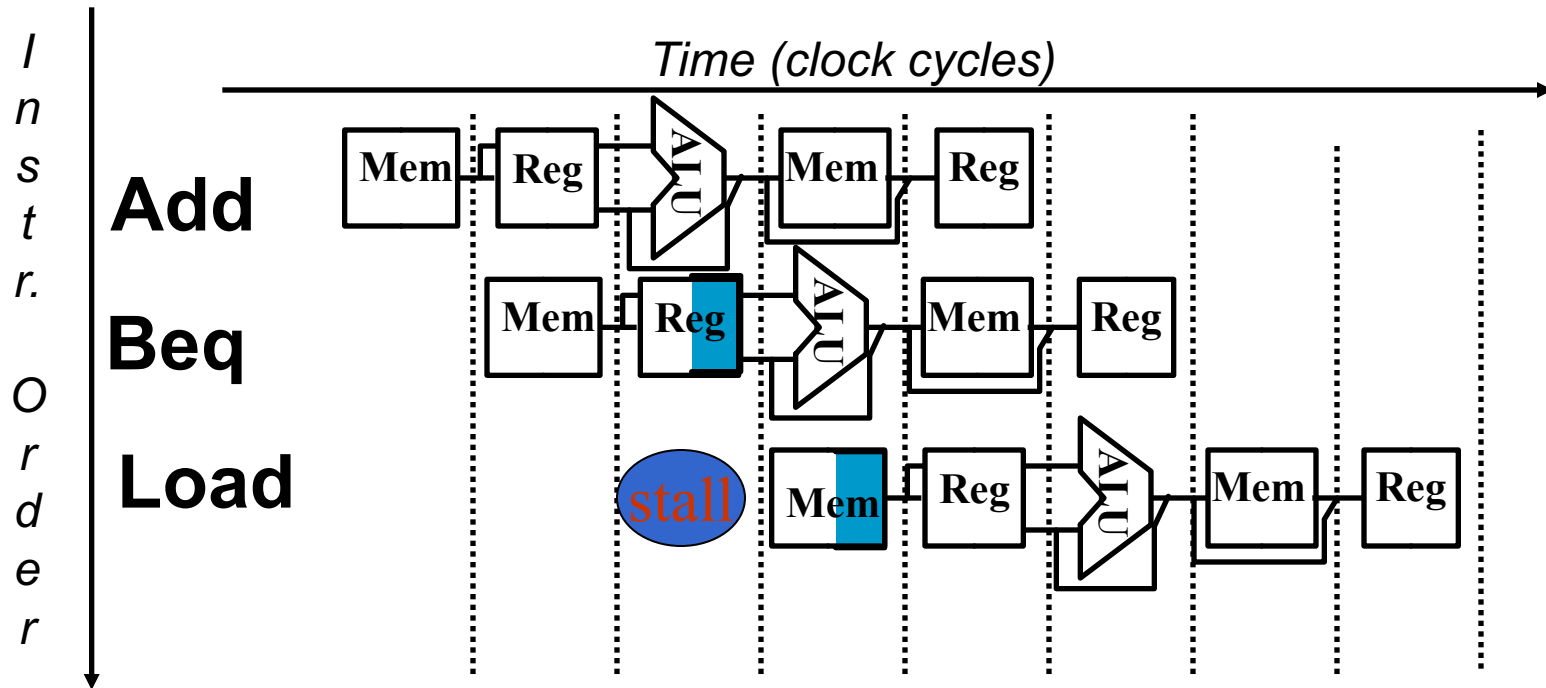
■ Impact: 2 clock cycles per branch instruction
=> slow

Control Hazard Solutions (2)

- Predict: guess one direction then back up if wrong
  - Predict not taken

Instr. Order

Time (clock cycles)

Add — Mem | Reg | ALU | Mem | Reg

Beq — Mem | Reg | ALU | Mem | Reg

Load — Mem | Reg | ALU | Mem | Reg
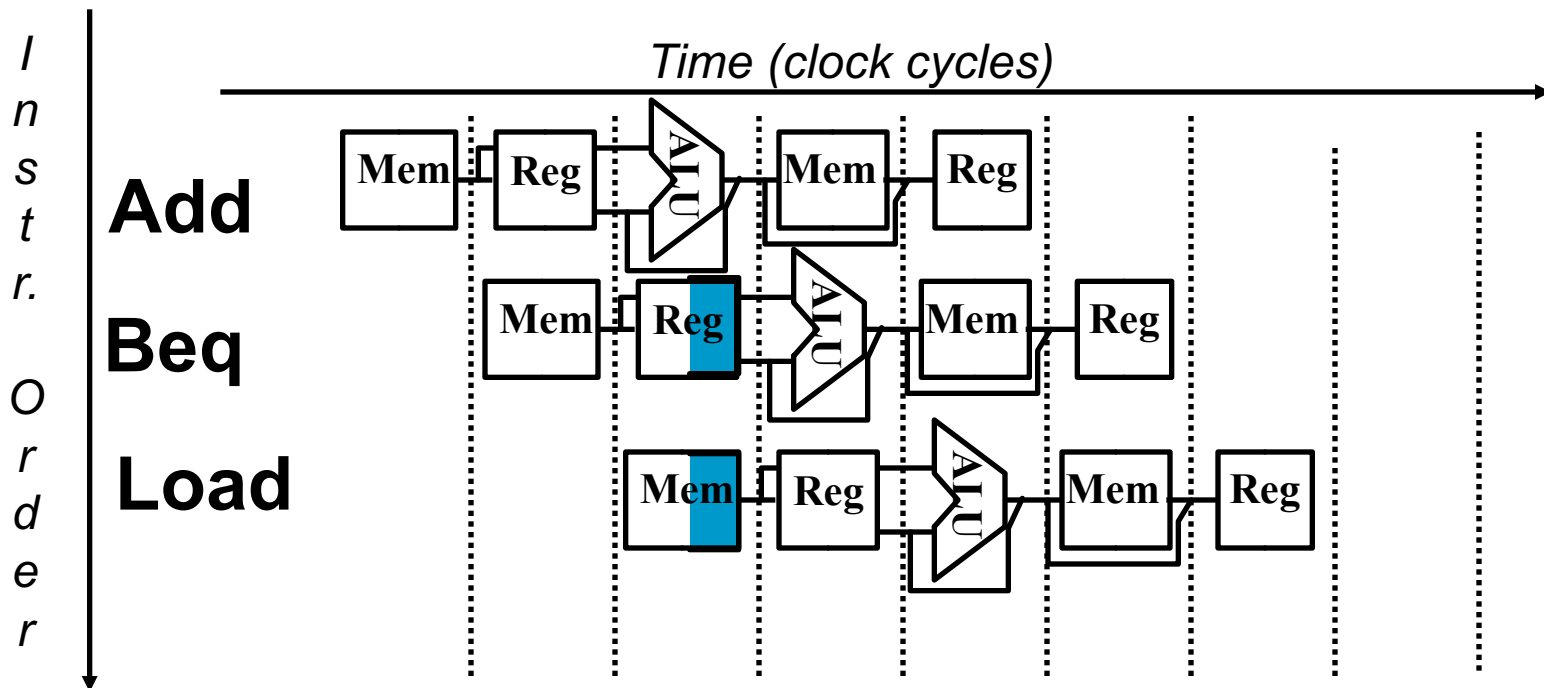
- Impact: 1 clock cycles per branch instruction if right, 2 if wrong (right 50% of time)
- More dynamic scheme: history of 1 branch (90%)

Predict branch not taken

```
Branch Inst (i)    IF   ID  EX   MEM  WB
Inst i+1                IF   ID    EX     MEM  WB
Inst i+2                     IF    ID     EX     MEM    WB
Inst i+3                          IF     ID     EX       MEM    WB
Inst i+4                                IF     ID      EX       MEM
```
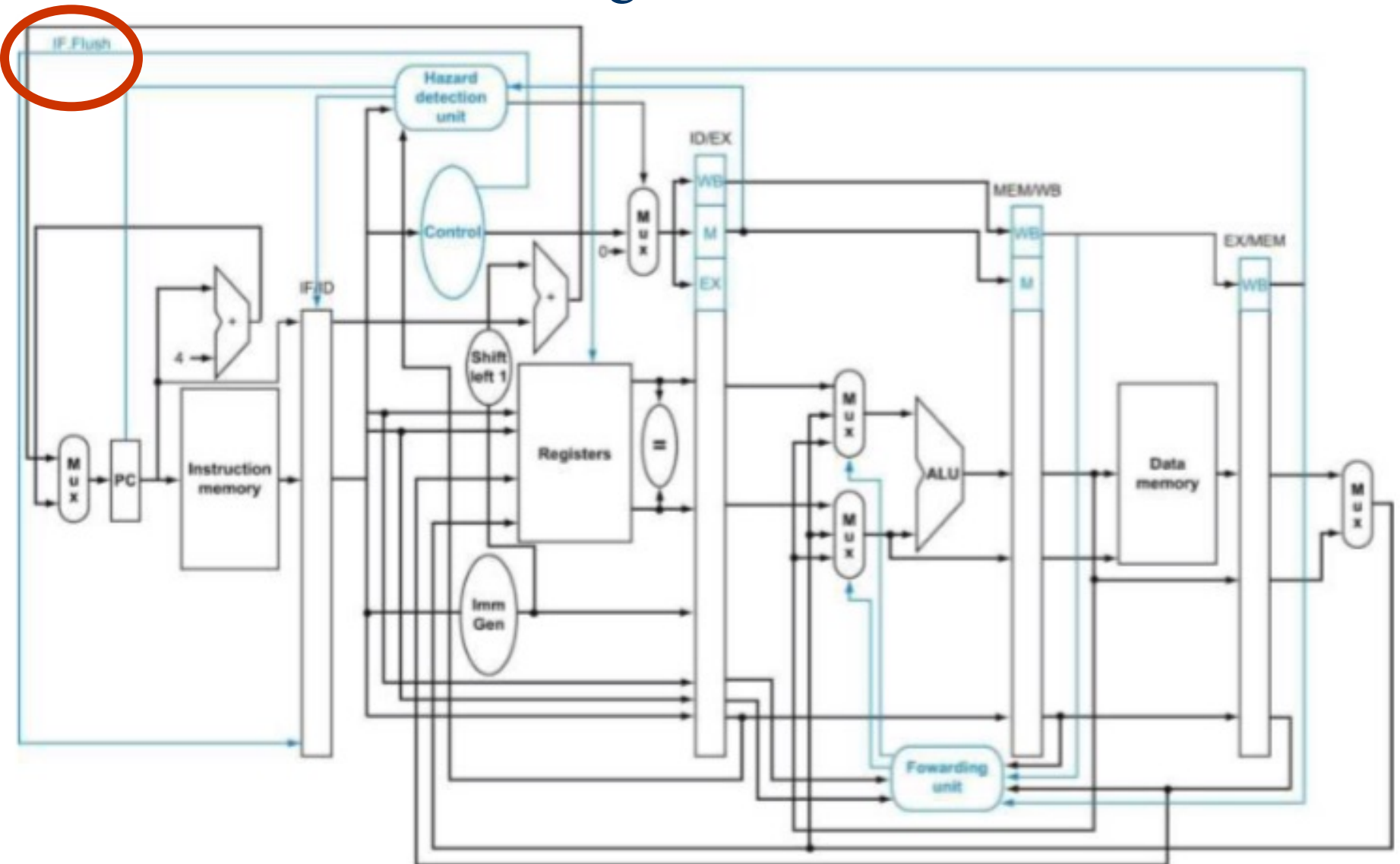
Correct Prediction : Zero Cycle Branch Penalty!

```
Branch Inst (i)    IF   ID  EX   MEM  WB
Inst i+1                IF   nop  nop    nop    nop
Branch target          IF    ID     EX     MEM   WB
```

Incorrect Prediction - waste one cycle
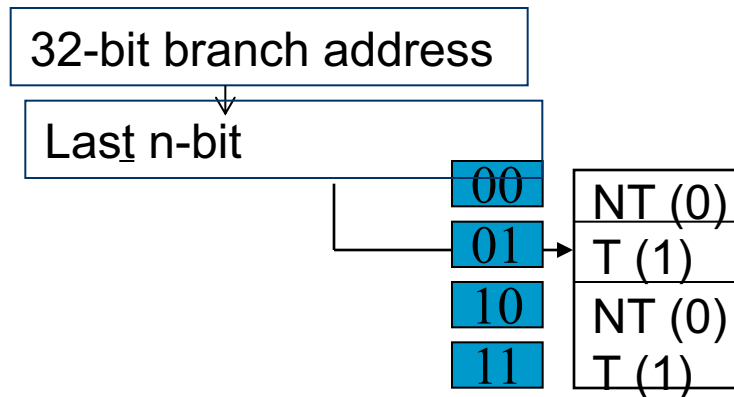**How to flush pipeline?**

Flushing Instructions

Zero the instruction field of the IF/ID pipeline register   40

Dynamic Branch Prediction

■ Branch History Table: Lower bits of PC address index table of 1-bit values
  – Says whether or not branch taken last time
  – No address check (saves HW, but may not be right branch)
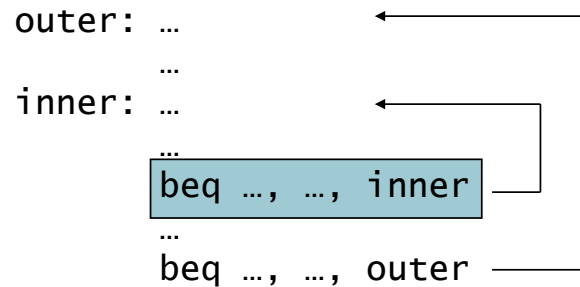
32-bit branch address

Last n-bit

00	NT (0)
01	T (1)
10	NT (0)
11	T (1)

branch history table = 2^n

Example: --00 T, --01 NT, --10 T, --11 T, --00 NT, --01 NT, --10 NT, --11 T,

Dynamic Branch Prediction

- Problem: in a loop, 1-bit BHT will cause
  2 mispredictions (avg is 9 iterations before exit):

```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

| loop	1	2	3	4	5	6	7	8	9	10
predict	NT									
real										

Dynamic Branch Prediction

- **Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):**
  - End of loop case, when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts *exit* instead of looping
  - Only 80% accuracy even if loop 90% of the time

```
outer: …
       …
inner: …
       …
beq …, …, inner
       …
beq …, …, outer
```
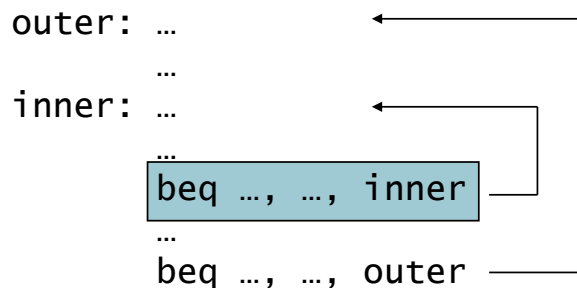
loop	1	2	3	4	5	6	7	8	9	10
predict	NT	T	T	T	T	T	T	T	T	T
real	T	T	T	T	T	T	T	T	T	NT

Dynamic Branch Prediction

- Solution: 2-bit scheme which changes prediction only if get misprediction *twice:*
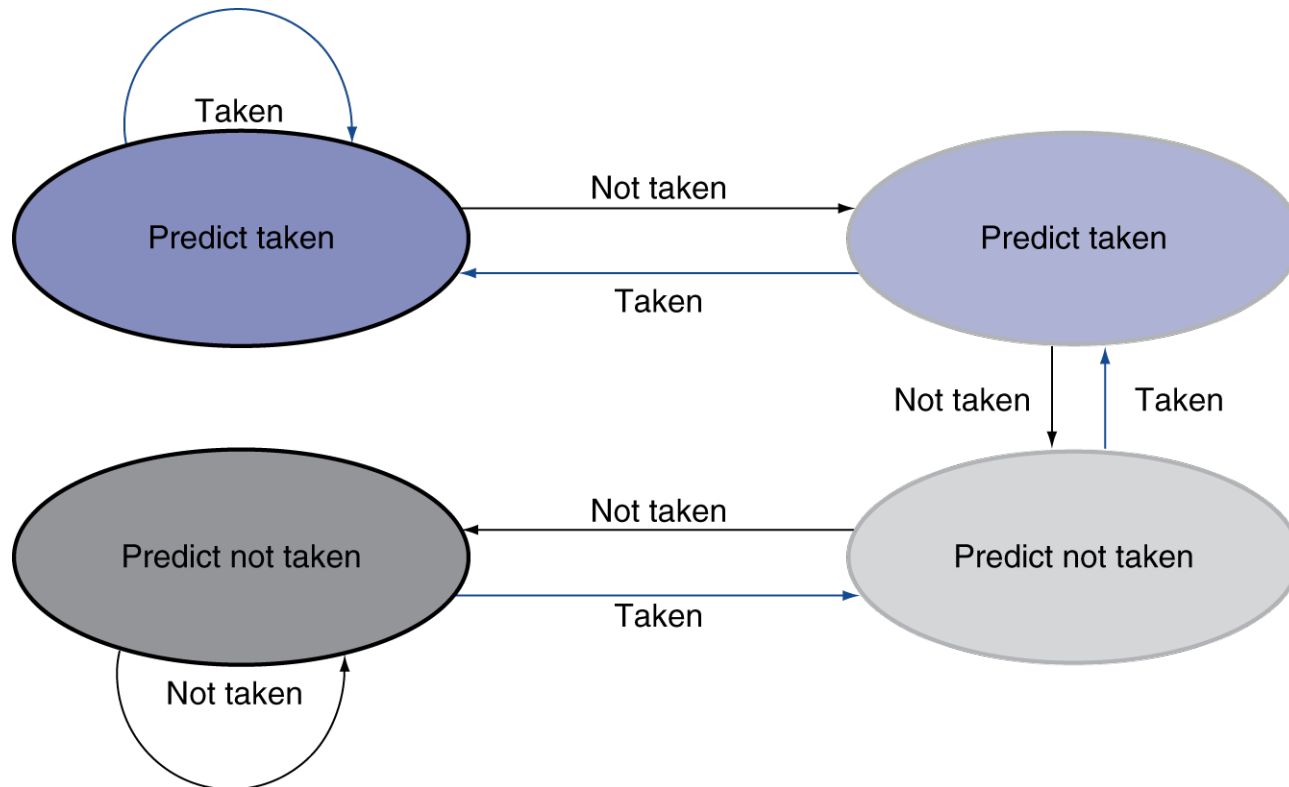
Exercise

- Branch outcome of a single branch
    - T T T N N N T T T

- How many instances of this branch instruction are mis-predicted with a 1-bit predictor?

- How many instances of this branch instruction are mis-predicted with a 2-bit predictor?

1-bit

branch	1	2	3	4	5	6	7	8	9
predict	NT								
real	T	T	T	NT	NT	NT	T	T	T

2-bit

branch	1	2	3	4	5	6	7	8	9
predict	NT								
real	T	T	T	NT	NT	NT	T	T	T

More on branch prediction

- **Problems with predicted taken?**
  - Need to calculate target address
  - Solution: branch target buffer
- **Correlating predictor**
  - A branch predictor that combines local behavior of a particular branch and global information about the behavior of some recent number of executed branches
- **Tournament branch predictor**
  - A branch predictor with multiple prediction for each branch and a selection mechanism that chooses which predictor to enable for a given branch

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, syscall, …
- Interrupt
  - From an external I/O controller
- Dealing with them without sacrificing performance is hard

Handling Exceptions

- Save PC of offending (or interrupted) instruction
    - In RISC-V: Supervisor Exception Program Counter (SEPC)

- Save indication of the problem
    - In RISC-V: Supervisor Exception Cause Register (SCAUSE)
    - 64 bits, but most bits unused
        - Exception code field: 2 for undefined opcode, 12 for hardware malfunction, …

- Jump to handler
    - Assume at 0000 0000 1C09 0000$_{hex}$

An Alternate Mechanism

- Vectored Interrupts
    - Handler address determined by the cause
- Exception vector address to be added to a vector table base register:
    - Undefined opcode $0000100\ 0000_{two}$
    - Hardware malfunction: $01\ 1000\ 0000_{two}$
    - …:   …
- Instructions either
    - Deal with the interrupt, or
   - Jump to real handler

Handler Actions

- Read cause, and transfer to relevant handler

- Determine action required

- If restartable

  - Take corrective action
  - use SEPC to return to program

- Otherwise

  - Terminate program
  - Report error using SEPC, SCAUSE, …

Exceptions in a Pipeline

- Another form of control hazard
- Consider malfunction on add in EX stage

  `add x1, x2, x1`
    - Prevent x1 from being clobbered
    - Complete previous instructions
    - Flush add and subsequent instructions
    - Set SEPC and SCAUSE register values
    - Transfer control to handler
- Similar to mispredicted branch
    - Use much of the same hardware

Pipeline with Exceptions
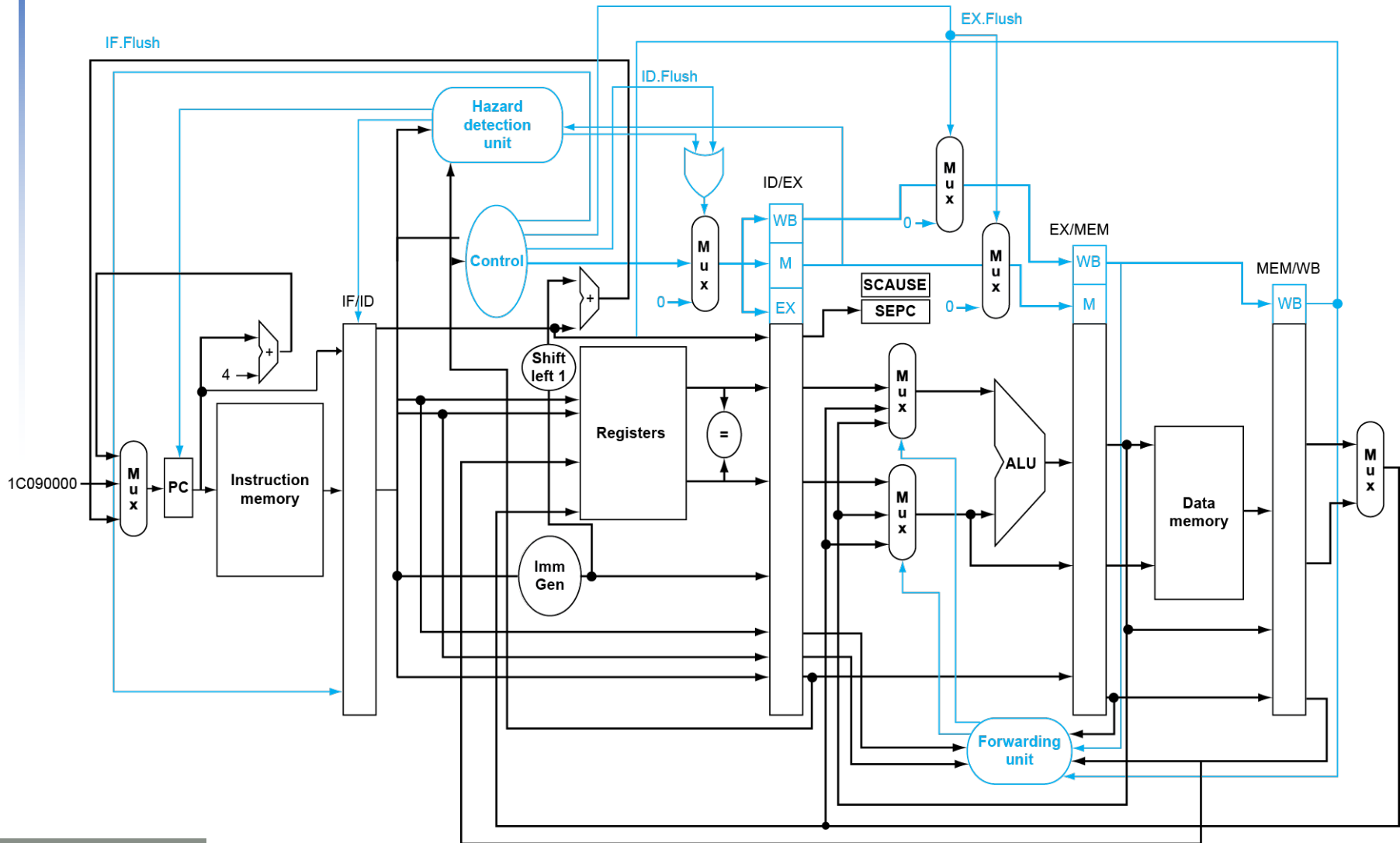
Exception Properties

- Restartable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch
- PC saved in SEPC register
  - Identifies causing instruction

Exception Example

- Exception on add in

```
40      sub   x11, x2, x4
44      and   x12, x2, x5
48      orr   x13, x2, x6
4c      add   x1,  x2, x1
50      sub   x15, x6, x7
54      ld    x16, 100(x7)
…
```
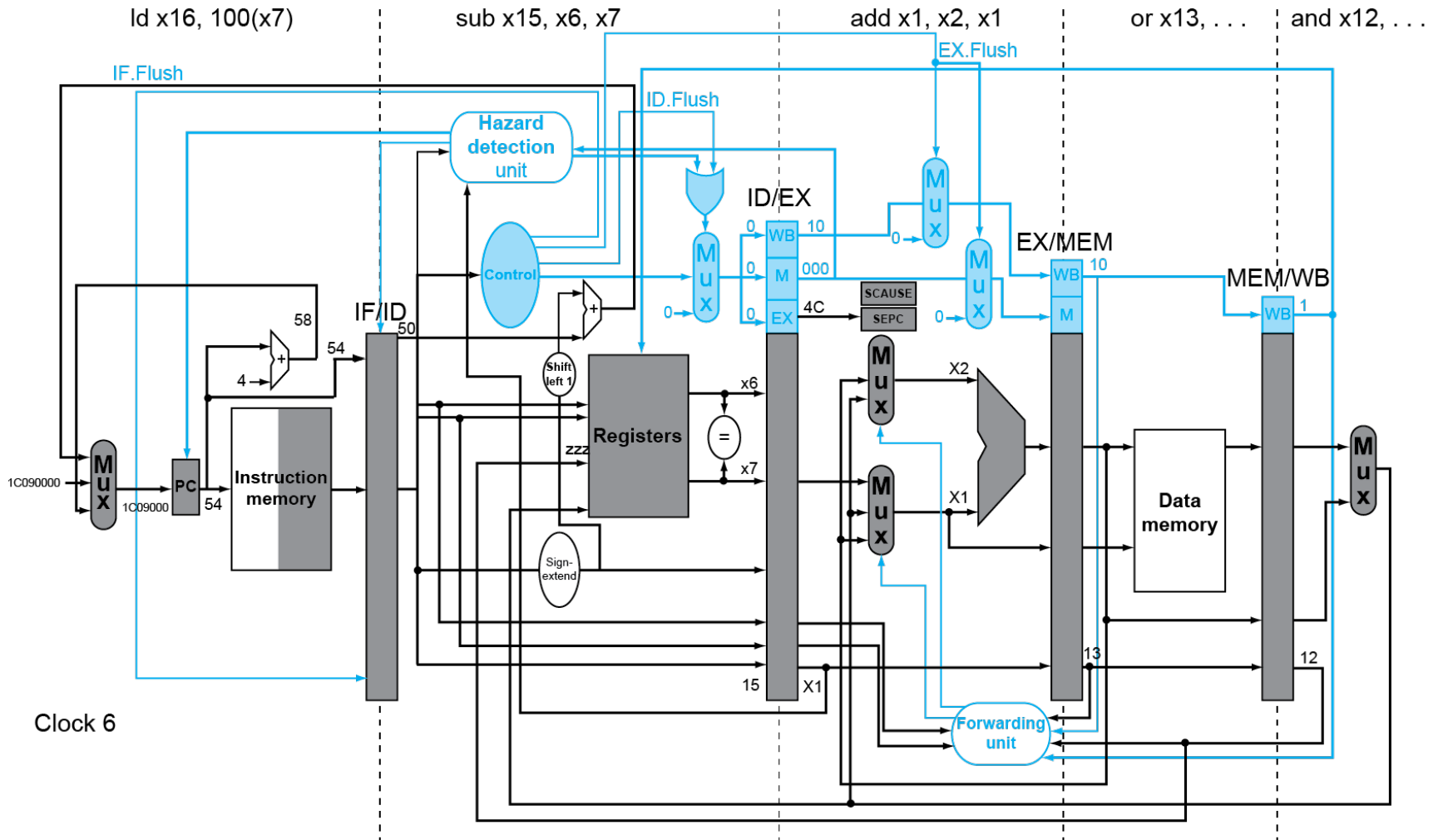
- Handler

```
1c090000    sd   x26, 1000(x10)
1c090004    sd   x27, 1008(x10)
…
```

Exception Example

Exception Example