RISC-V Pipelined Architecture II

Muhammad Tahir

Lecture 10-11

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Pipeline Limitations

- Major drawback of pipelining: Hazards
- Hazards are a consequence of instruction dependence

Pipeline Limitations

- Major drawback of pipelining: Hazards
- Hazards are a consequence of instruction dependence
- Broadly there are three types of hazards
  - Structural hazards
  - Data hazards
  - Control hazards

Structural Hazards

- A Structural Hazard arises when two instructions (in the pipeline) vie for the same (hardware) resource

Structural Hazards

- A Structural Hazard arises when two instructions (in the pipeline) vie for the same (hardware) resource
- Consider the pipelined execution of the following program (assuming single memory for instructions and data)

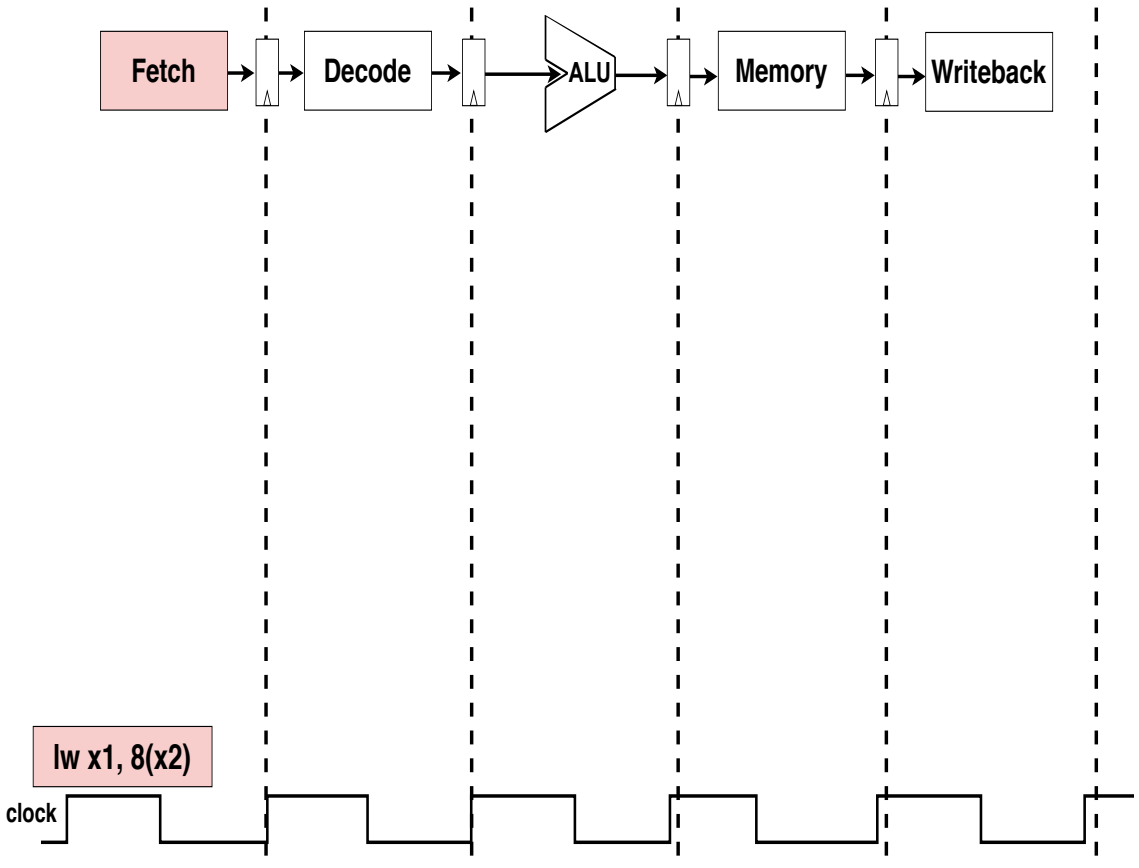
Structural Hazards

- A Structural Hazard arises when two instructions (in the pipeline) vie for the same (hardware) resource
- Consider the pipelined execution of the following program (assuming single memory for instructions and data)

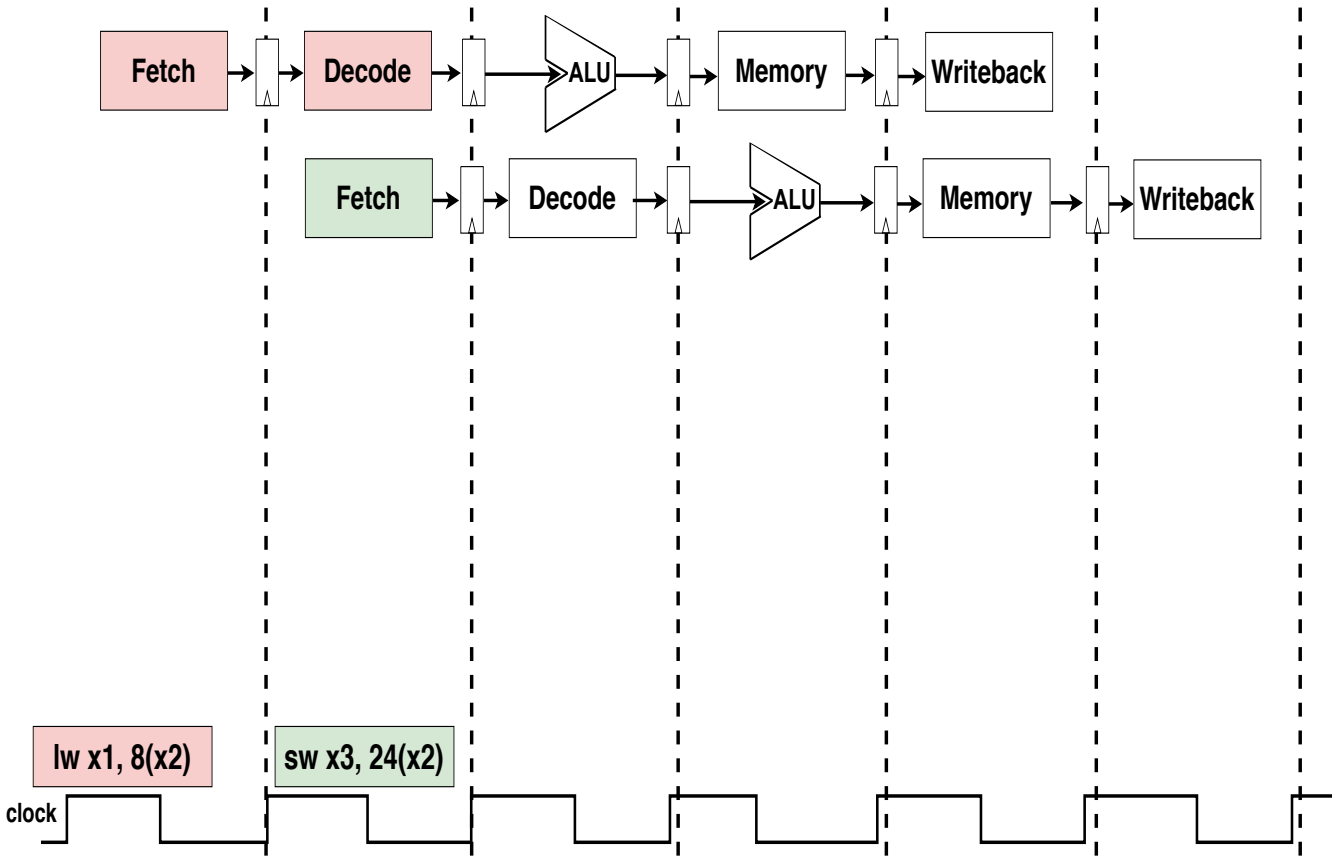
Example program.

```
lw    x1,  8(x2)
sw    x3,  24(x4)
or    x5,  x6,  x7
add   x8,  x6,  x7
lw    x1,  4(x2)
```

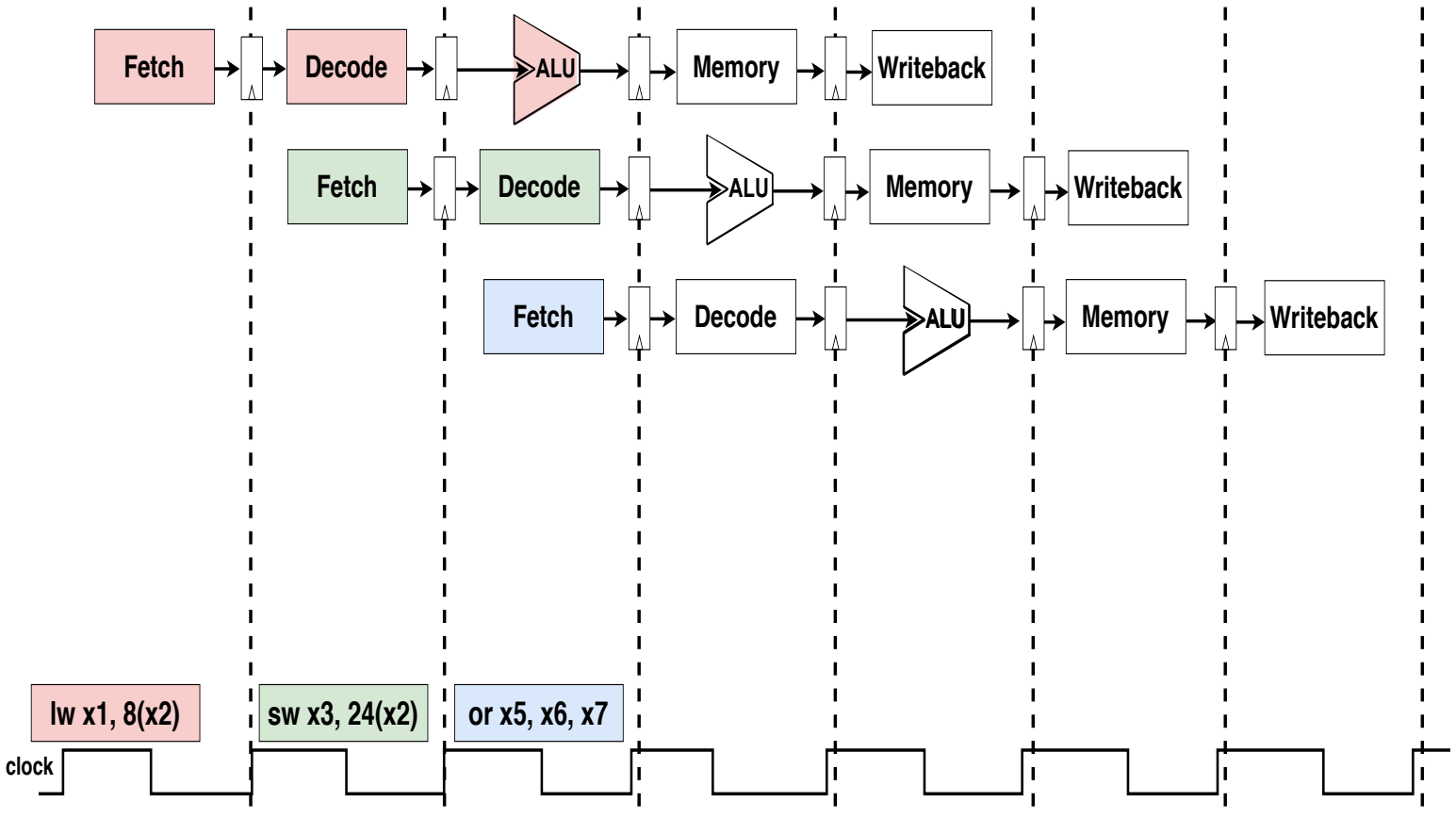
Instruction Execution



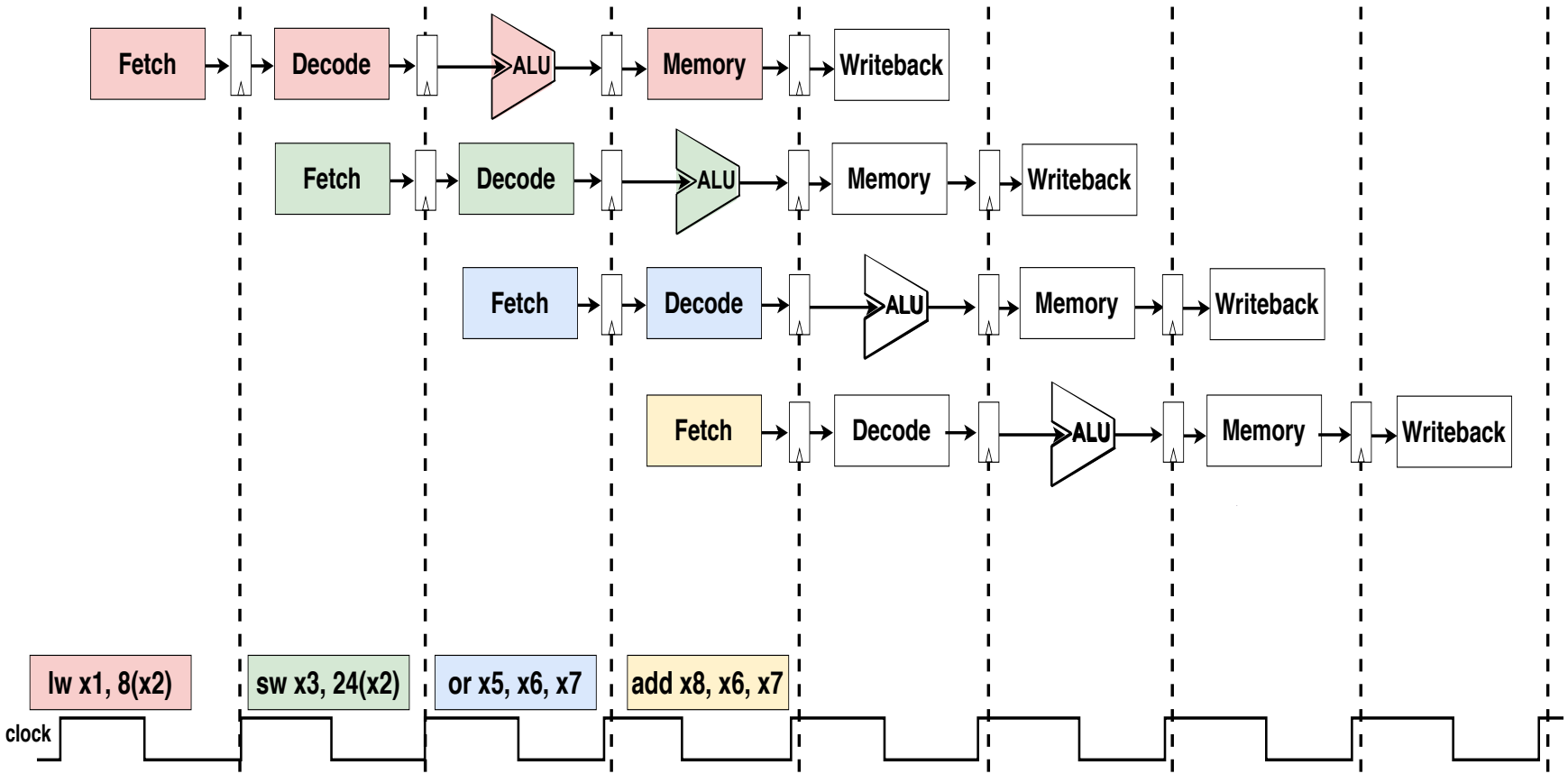
Instruction Execution: Cont'd



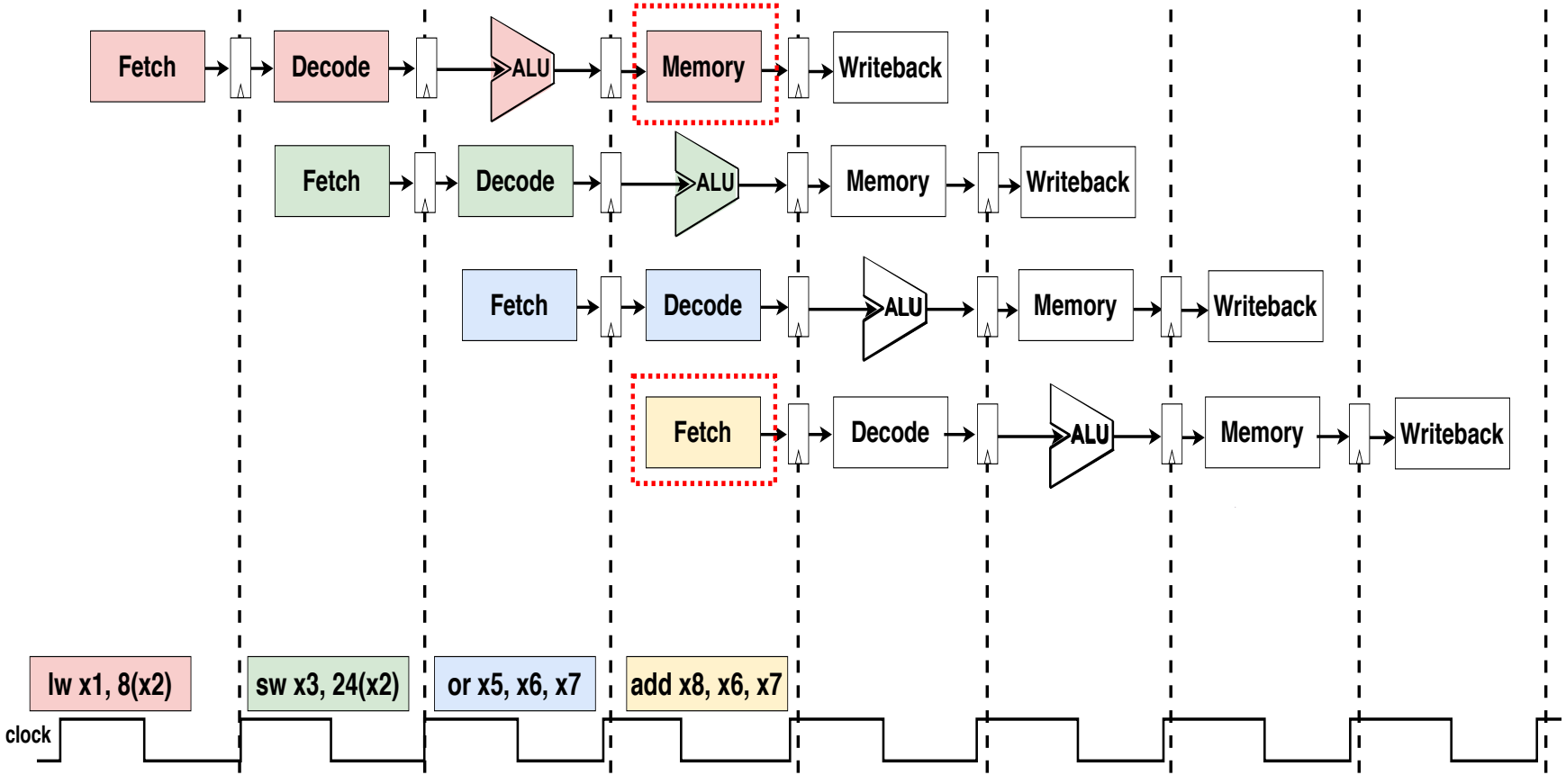
Instruction Execution: Cont'd



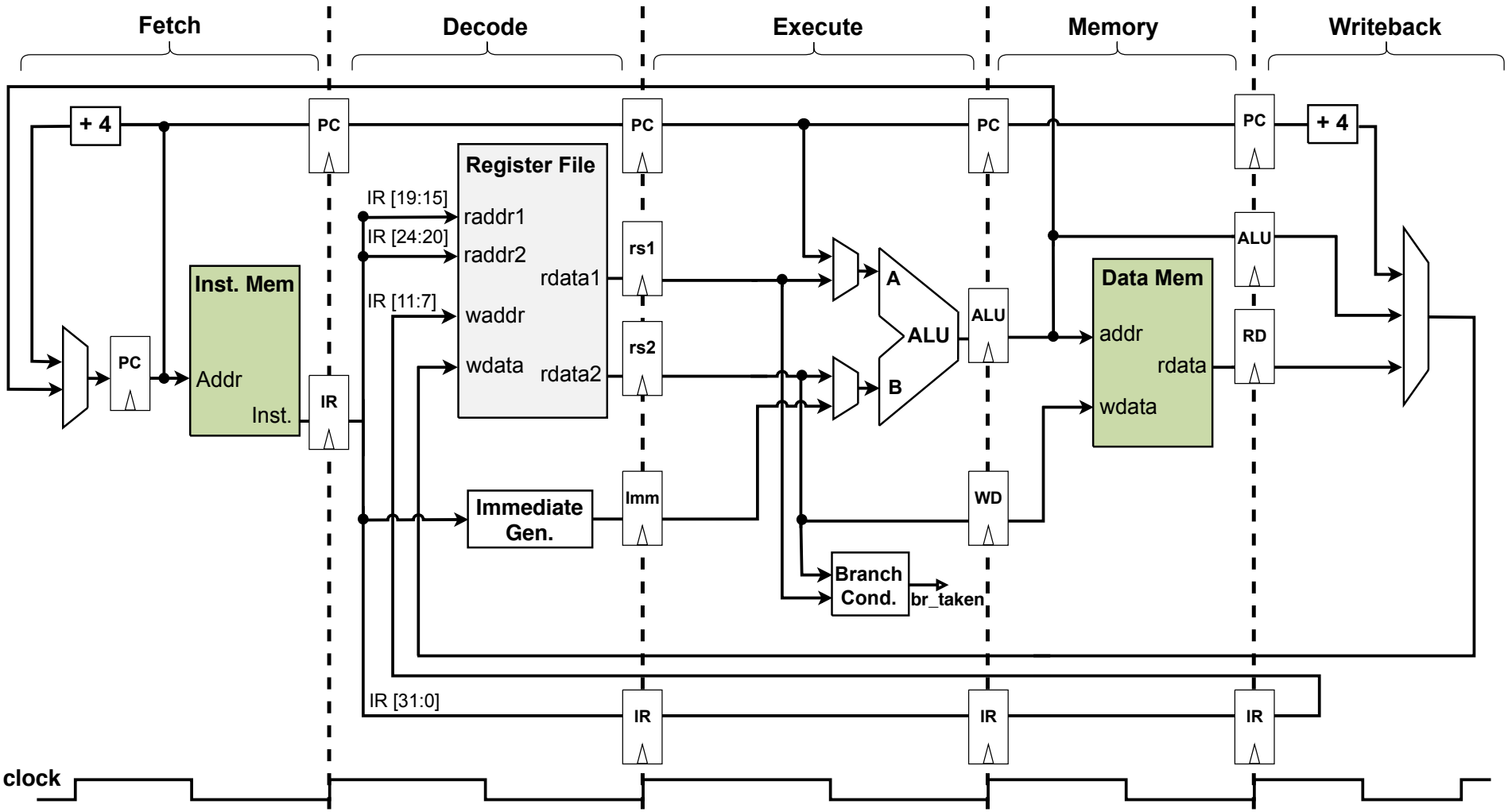
Instruction Execution: Cont'd



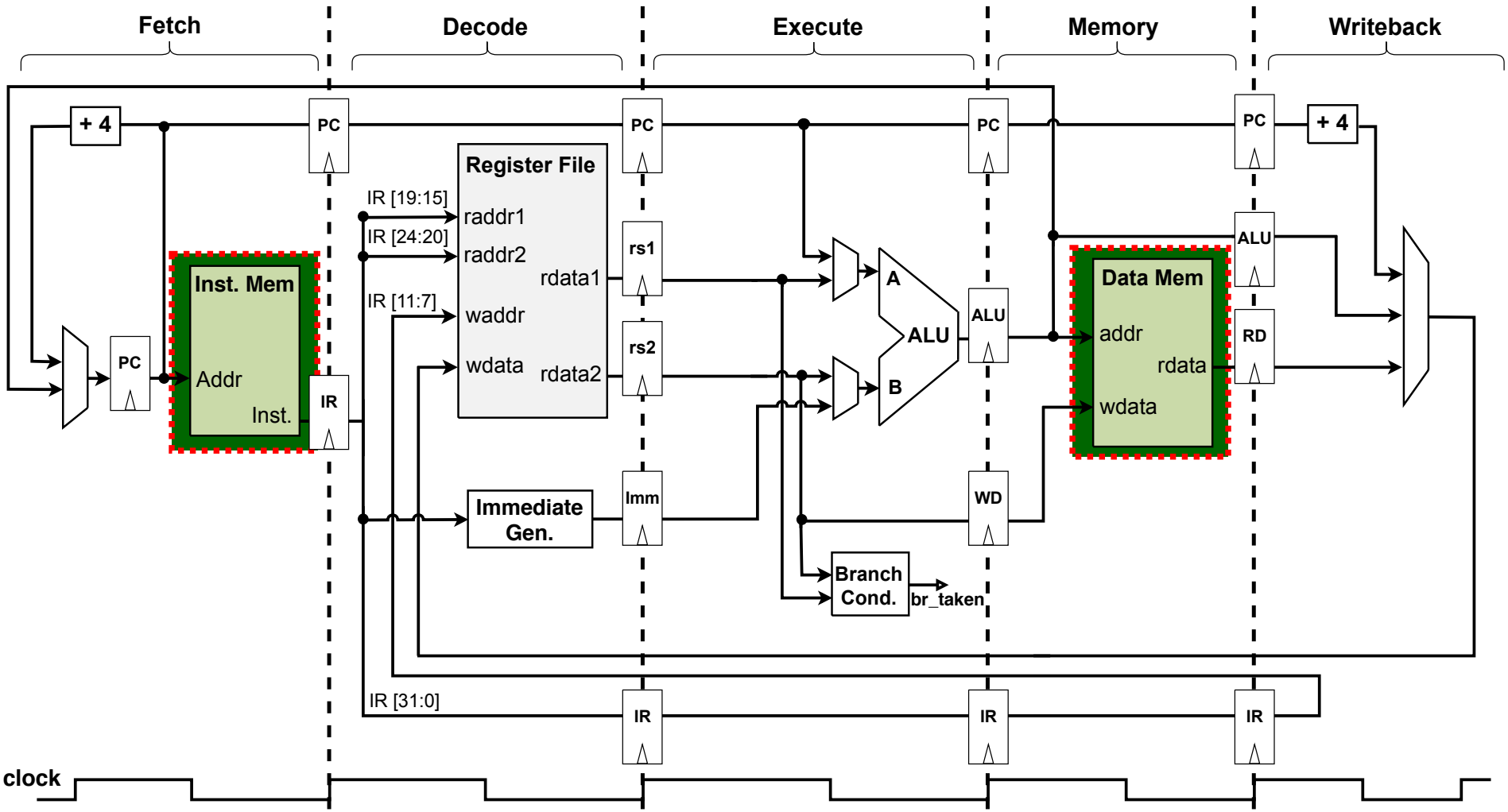
Instruction Execution: Cont'd



Structural Hazard in Our RV32I Design?



No Structural Hazard in Our RV32I Design



Data Hazards

- Data Hazard is a consequence of operand(s) dependence
- Applicable to those instructions which are simultaneously present in the pipeline
- Data hazards can be resolved by waiting

Data Hazards Cont'd

- Data hazards can be of three different types
  - Read After Write (RAW) hazard, *true data dependence*
    x3 ← x1 op x2
    x5 ← x3 op x4
  - Write After Read (WAR) hazard, anti-dependence (a type of name dependence)
    x1 ← x3 op x2
    x3 ← x4 op x5
  - Write After Write (WAW) hazard, output-dependence (a type of name dependence)
    x3 ← x1 op x2
    x3 ← x6 op x7

RAW Data Hazards

- True Data Dependence (RAW Hazard): An instruction j is said to be data dependent on instruction i if
  - Instruction j uses a result that is produced by instruction i, or
  - Instruction j is *data dependent* on instruction k, and instruction k is *data dependent* on instruction i

RAW Hazard

```
add   x3,  x2,  x1
addi  x4   x3,  5
```

RAW Data Hazards

- True Data Dependence (RAW Hazard): An instruction j is said to be data dependent on instruction i if
  - Instruction j uses a result that is produced by instruction i, or
  - Instruction j is *data dependent* on instruction k, and instruction k is *data dependent* on instruction i

RAW Hazard

```
add   x3,  x2,  x1
addi  x4   x3,  5
```

```
add   x3,  x2,  x1
or    x5,  x6,  x7
addi  x4   x3,  5
```

Resolving RAW Data Hazards

- Solution 1: A simple approach is to freeze the earlier pipeline stages while waiting for the result to be available (Interlock or Stall).
- Solution 2: Route data, as soon as it is available, to the earlier pipeline stage (Bypassing or Forwarding).

RAW Data Hazard Illustration

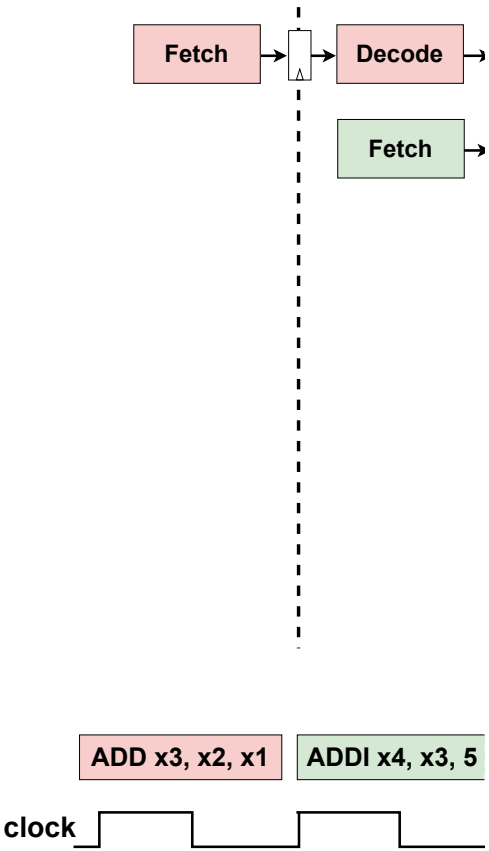
- Example program for data hazards

Example program.

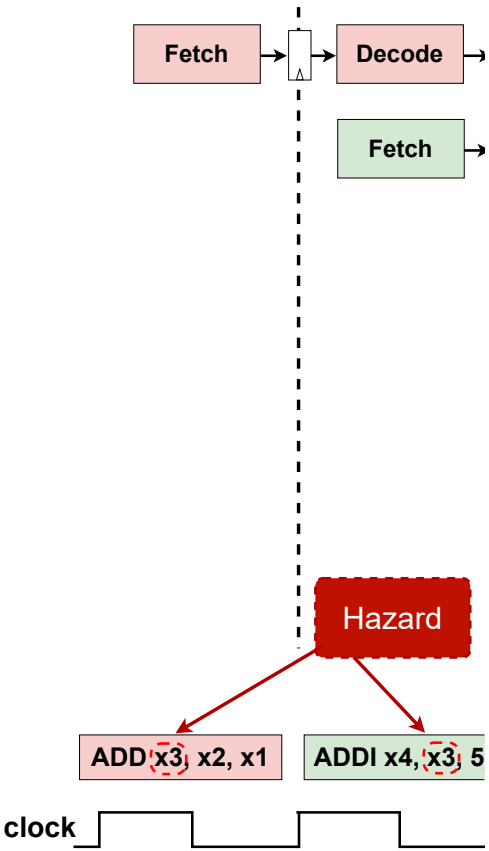
```
add   x3,    x2,    x1
addi  x4,    x3,    5
sub   x7,    x6,    x8
and   x9,    x6,    x8
lw    x10,   4(x12)
add   x11,   x8,    x2
```

Instructions Execution

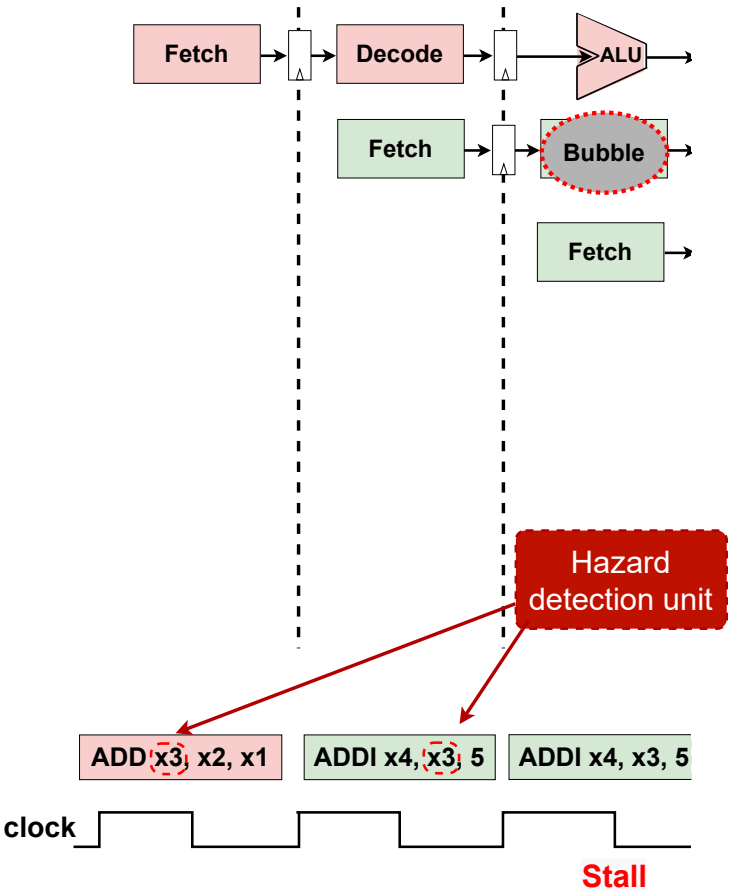
Instructions Execution Cont'd



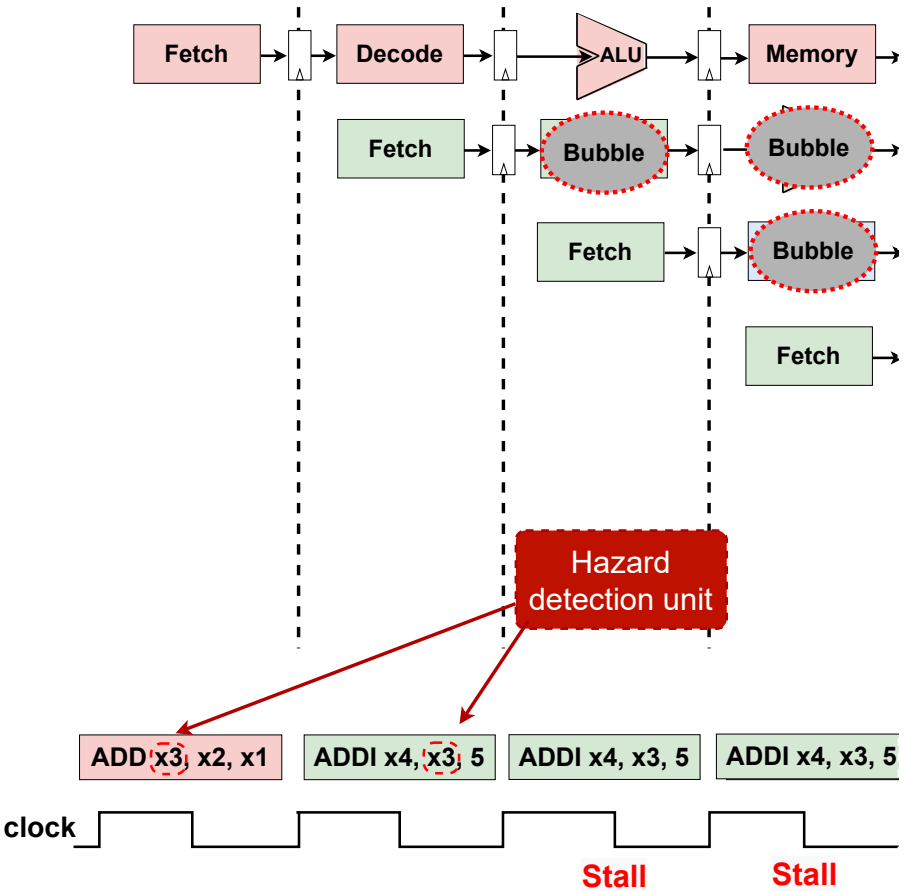
Instructions Execution Cont'd



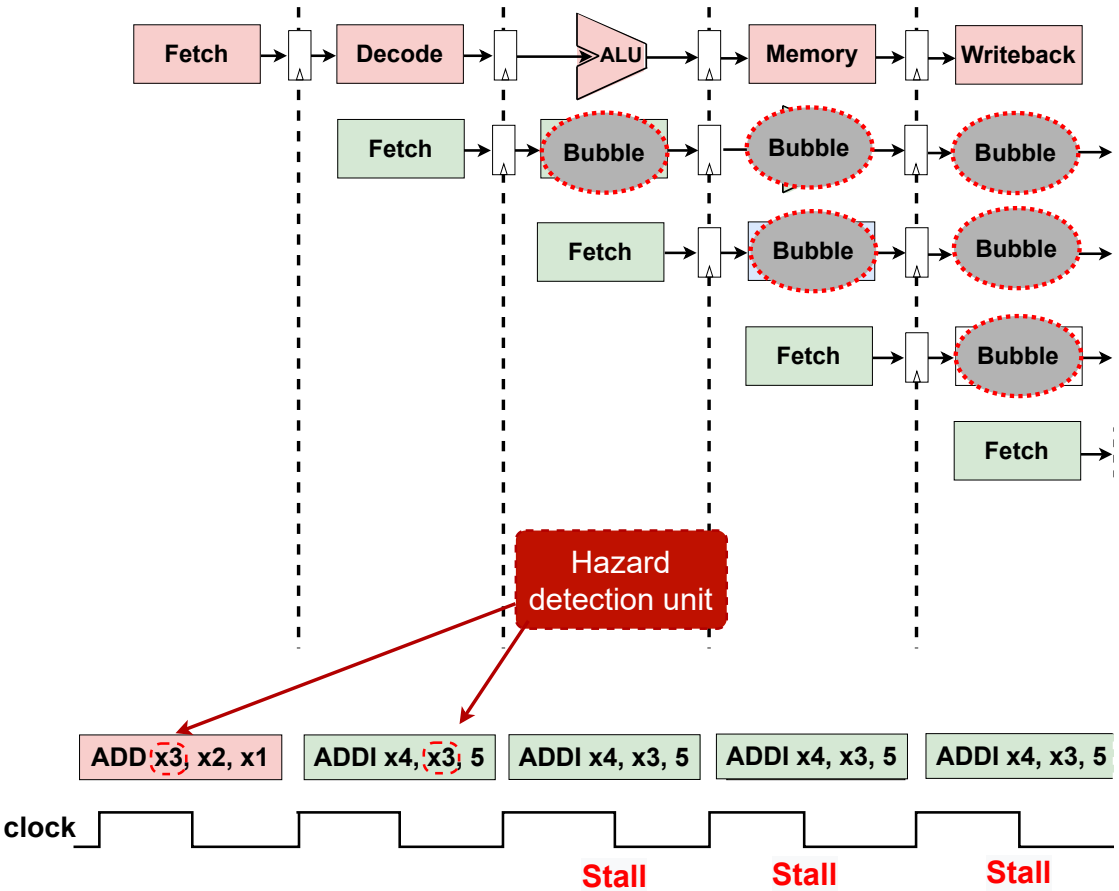
Instructions Execution Cont'd



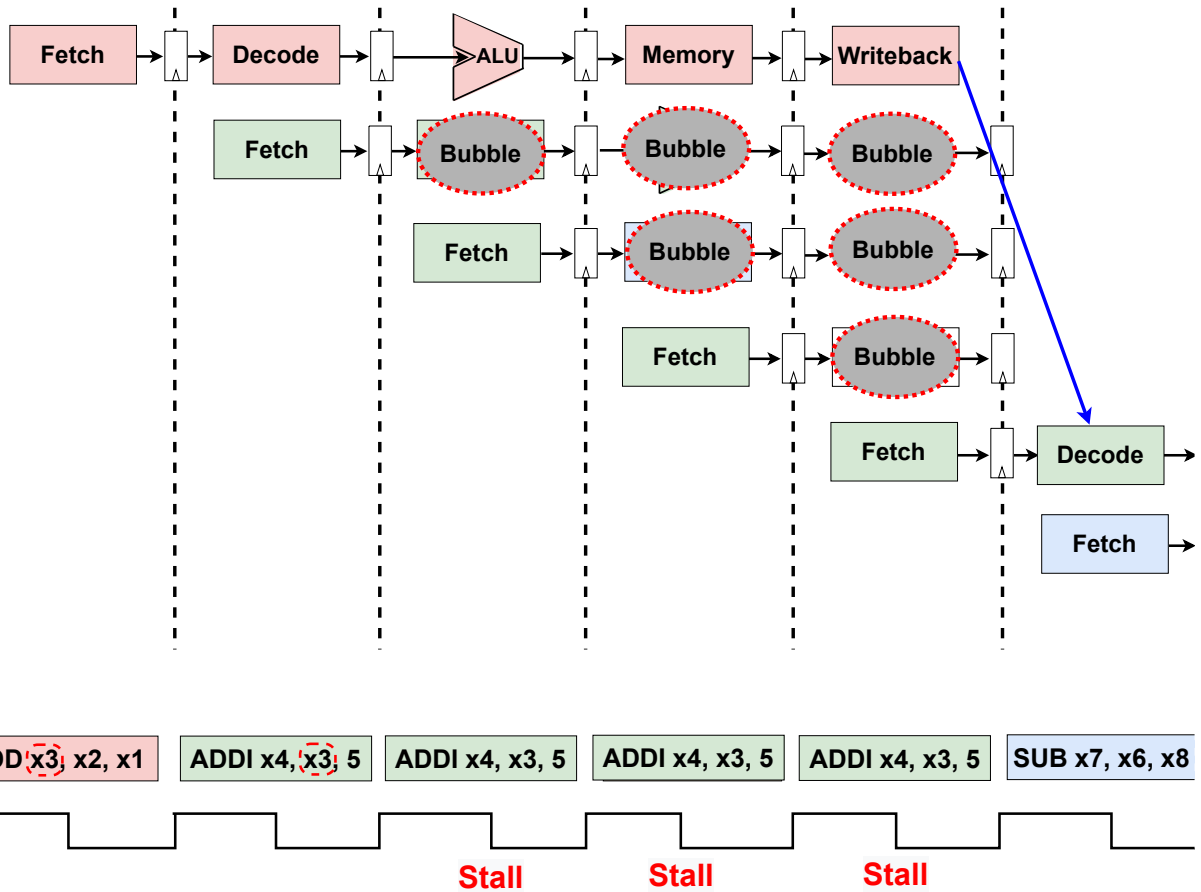
Instructions Execution: Pipeline Stalling



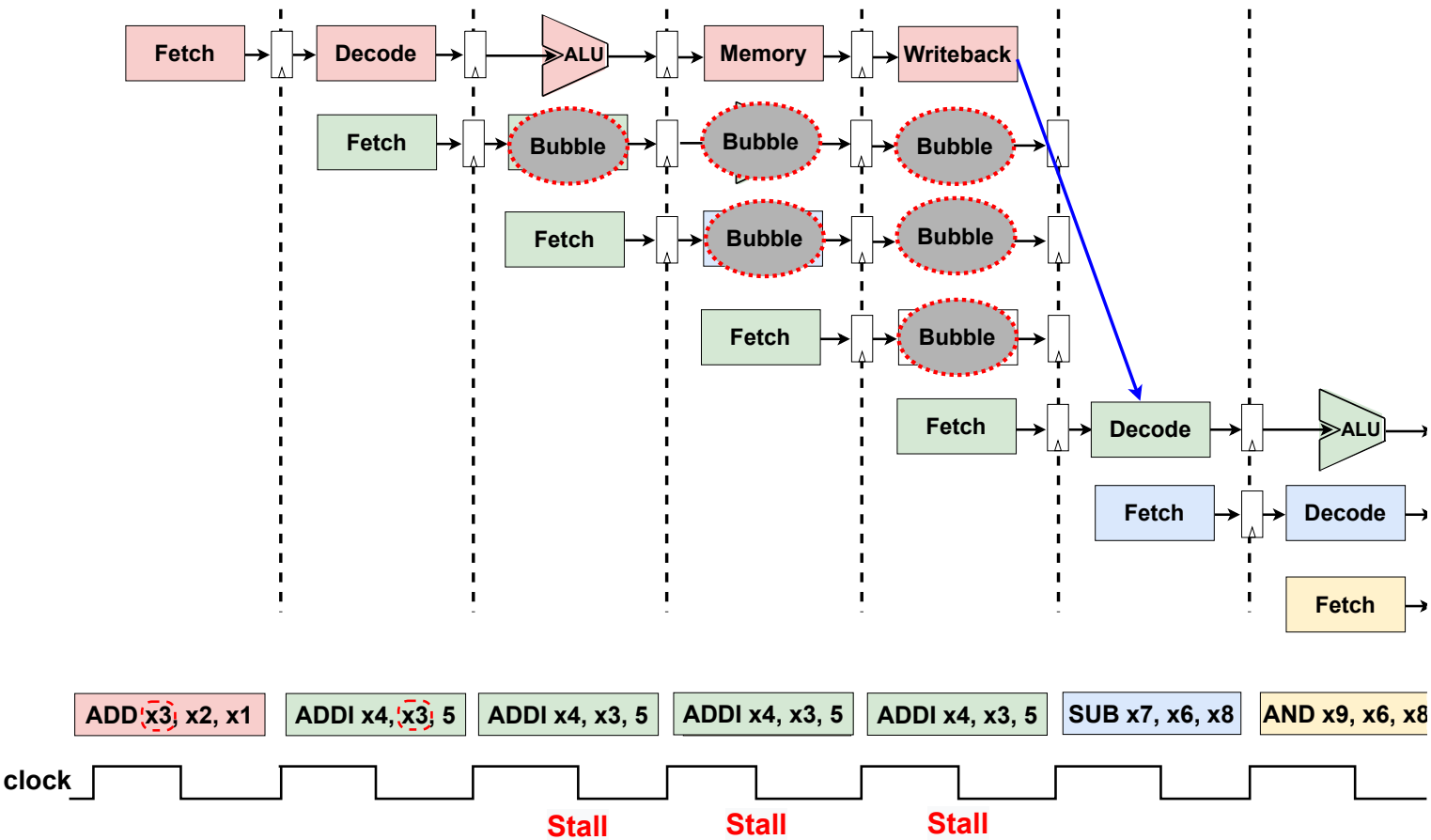
Instructions Execution: Pipeline Stalling



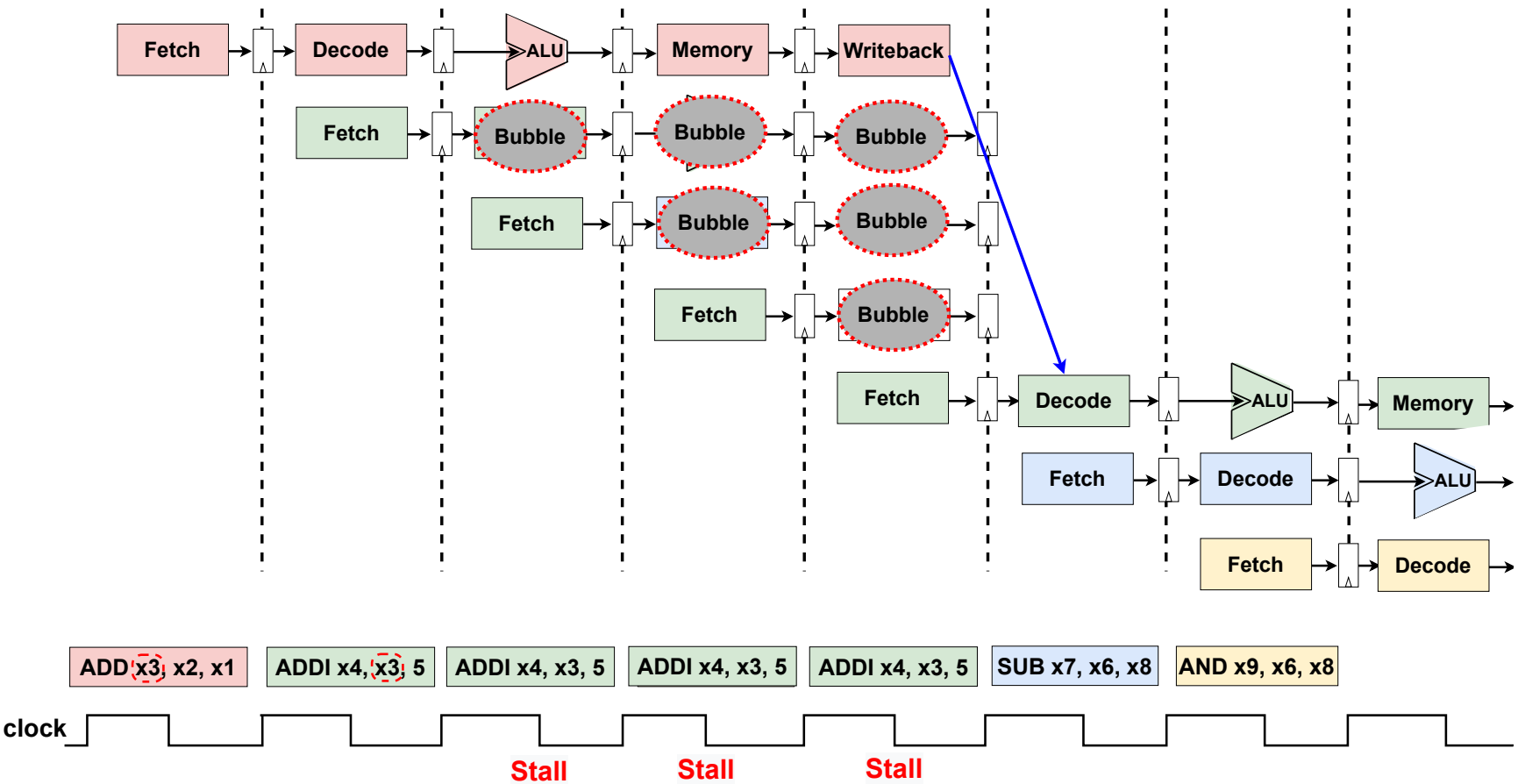
Instructions Execution: Pipeline Stalling



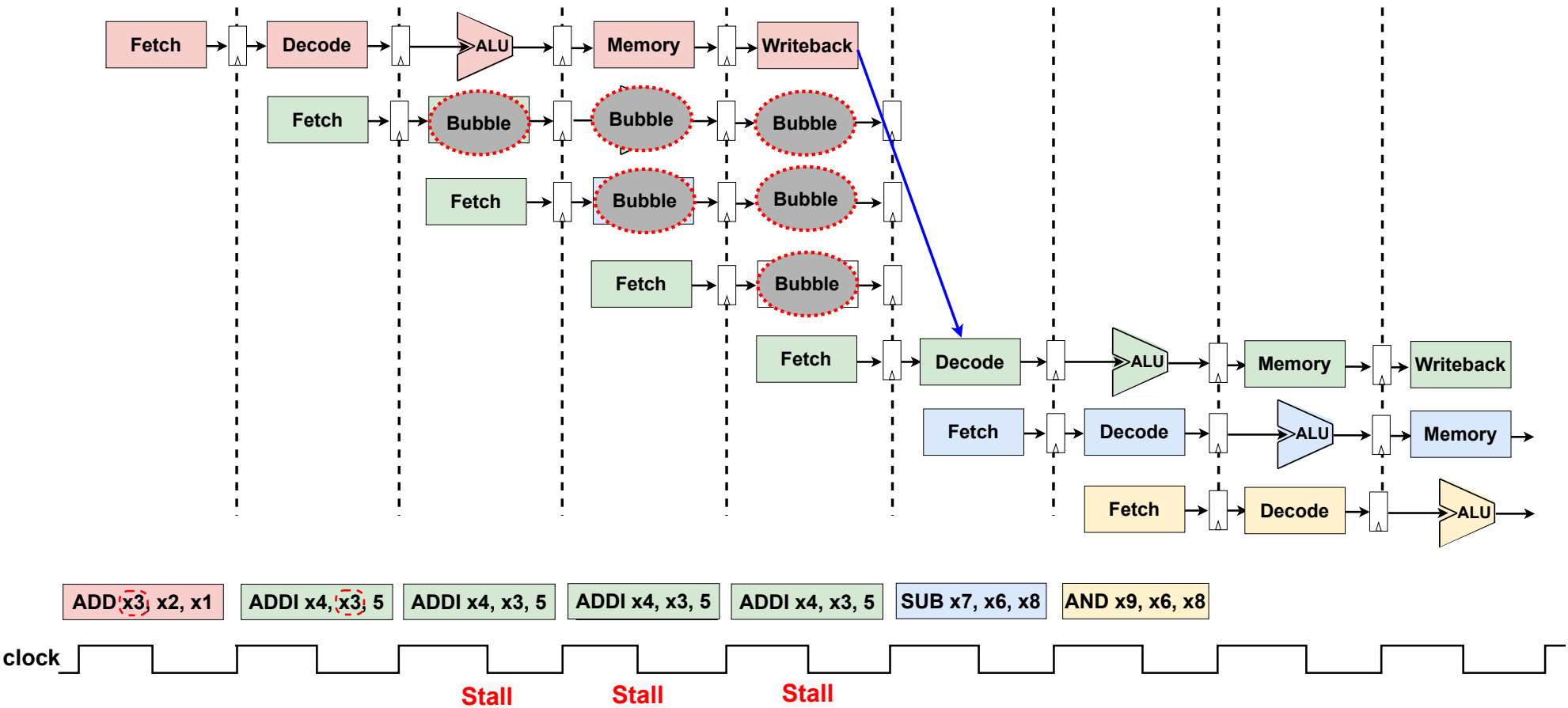
Instructions Execution: Pipeline Stalling



Instructions Execution: Pipeline Stalling



Instructions Execution: Pipeline Stalling



Instructions Execution: Pipeline Stalling

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
I_1	IF_1	ID_1	EX_1	MA_1	WB_1				
I_2		IF_2	ID_2	ID_2	ID_2	ID_2	EX_2	MA_2	WB_2
I_3			IF_3	IF_3	IF_3	IF_3	ID_3	EX_3	MA_3

Stalling Drawback

- Using Stalling to resolve RAW data hazard
  - The strategy of inserting NOP's is not a good idea
  - Introduces delays in executing a particular set of instructions

Stalling Drawback

- Using Stalling to resolve RAW data hazard
  - The strategy of inserting NOP's is not a good idea
  - Introduces delays in executing a particular set of instructions
- Pipelining was introduced to improve performance

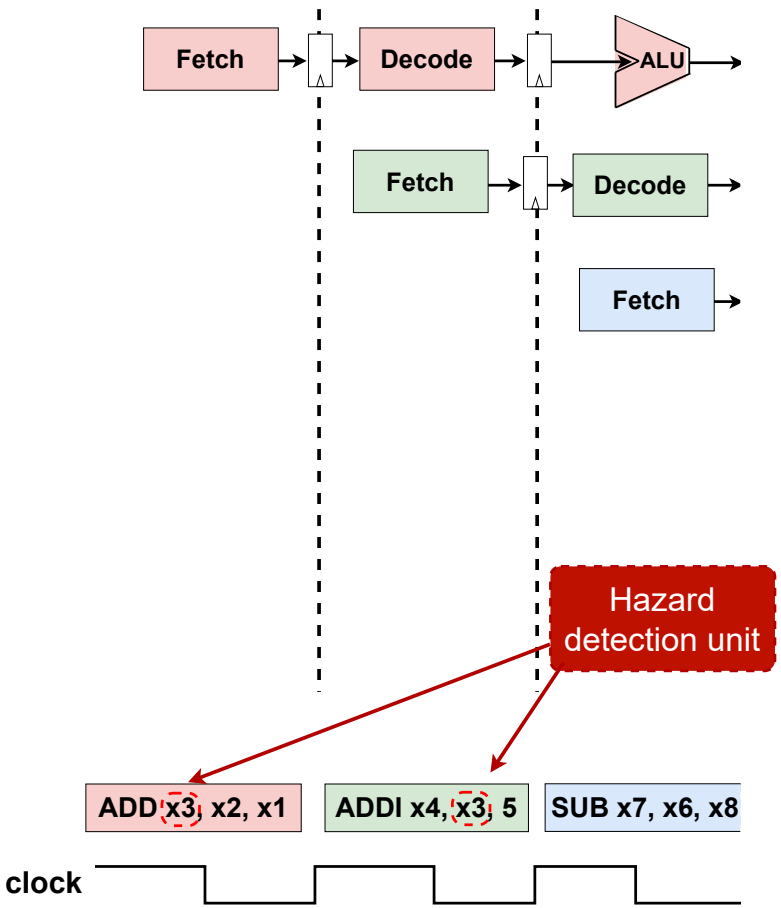
$$\frac{Time}{Program} = \frac{Instructions}{Program} x \frac{Cycles}{Instruction} x \frac{Time}{Cycle}$$

- But with stalling, CPI (cycles per instruction) will go up
- Need an alternative method to overcome stalling limitation

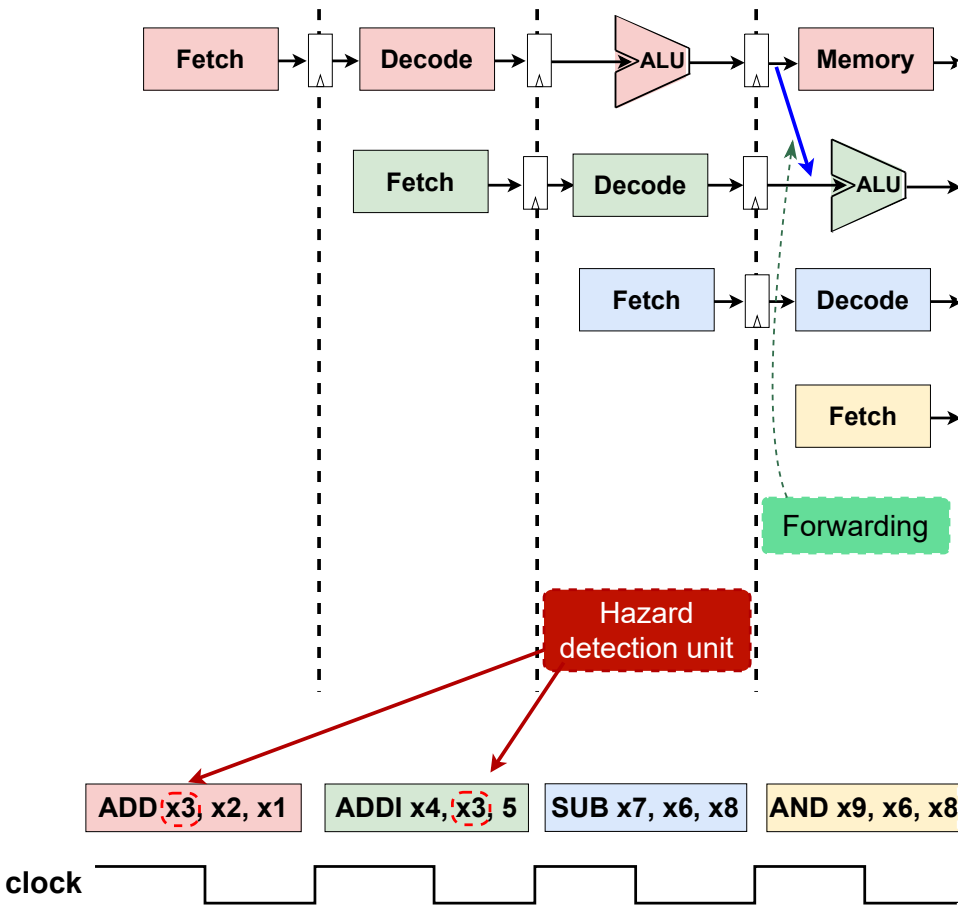
Resolving RAW Data Hazard: Forwarding

- `Forwarding` to resolve RAW data hazard
  - An alternative method with additional hardware for data forwarding
  - Route data to the input of pipeline execution stage (for subsequent instruction(s)) as soon as it becomes available
  - Requires additional signal path and multiplexer
  - Does not work for all scenarios

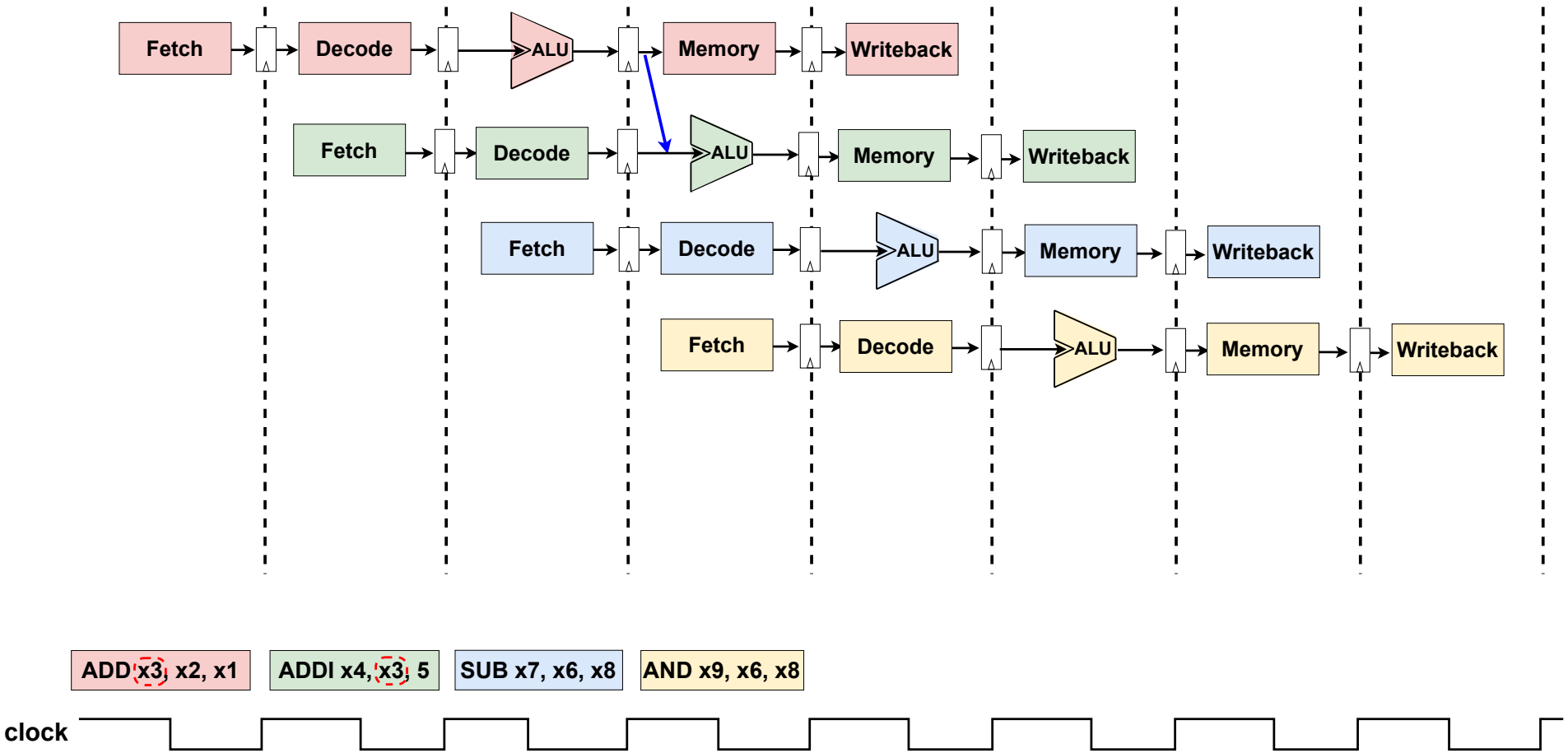
Resolving RAW Data Hazard: Forwarding



Resolving RAW Data Hazard: Forwarding



Resolving RAW Data Hazard: Forwarding



Load-Use RAW Data Hazard

- Forwarding performance for Load-Use hazard

Load-Use Hazard with Forwarding

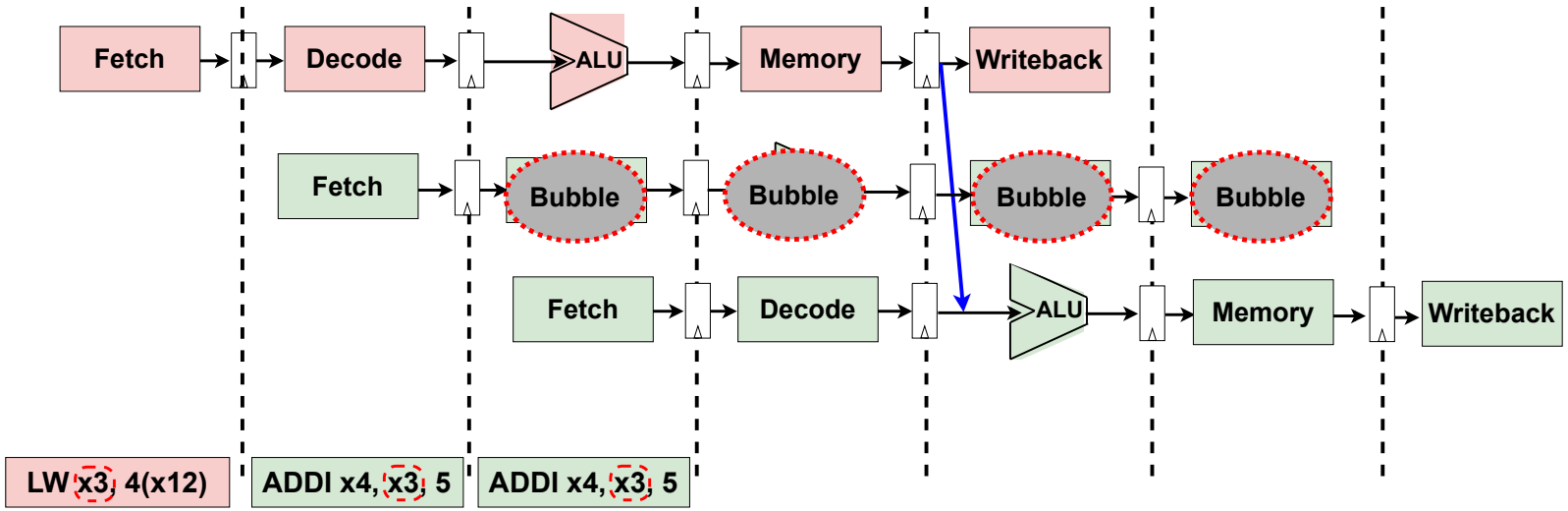
```
lw    x3, 4(x12)
addi  x4, x3, 5
sub   x7, x6, x8
```

Stall cycles = 1

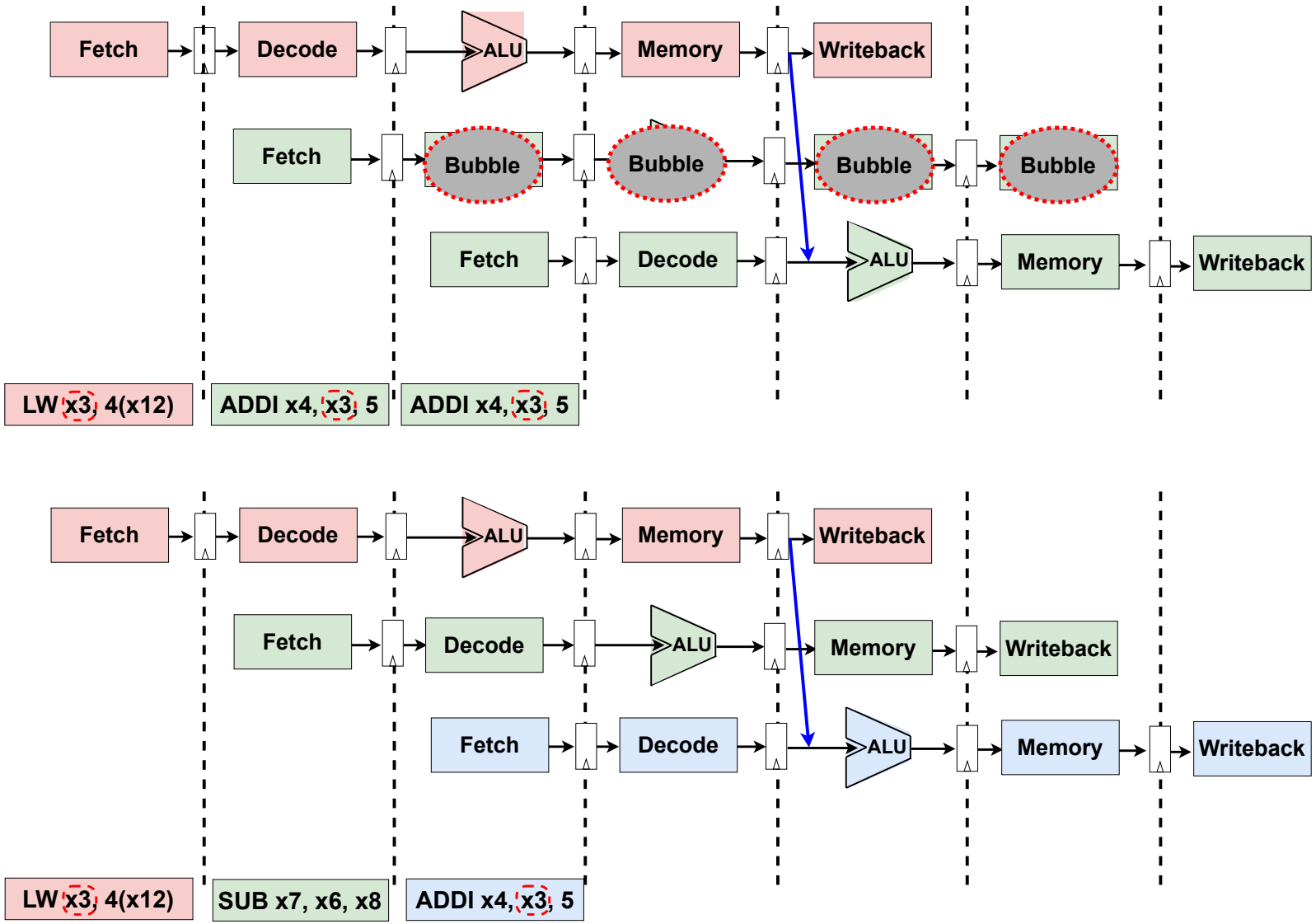
```
lw    x3, 4(x12)
sub   x7, x6, x8
addi  x4, x3, 5
```

Stall cycles = 0

Resolving Load-Use Hazard: Forwarding



Resolving Load-Use Hazard: Forwarding



RAW Data Hazard: Forwarding Performance

- Pipeline with single execution stage

RAW Hazards with Forwarding

```
add   x3,  x2,  x1
addi  x4   x3,  5
```

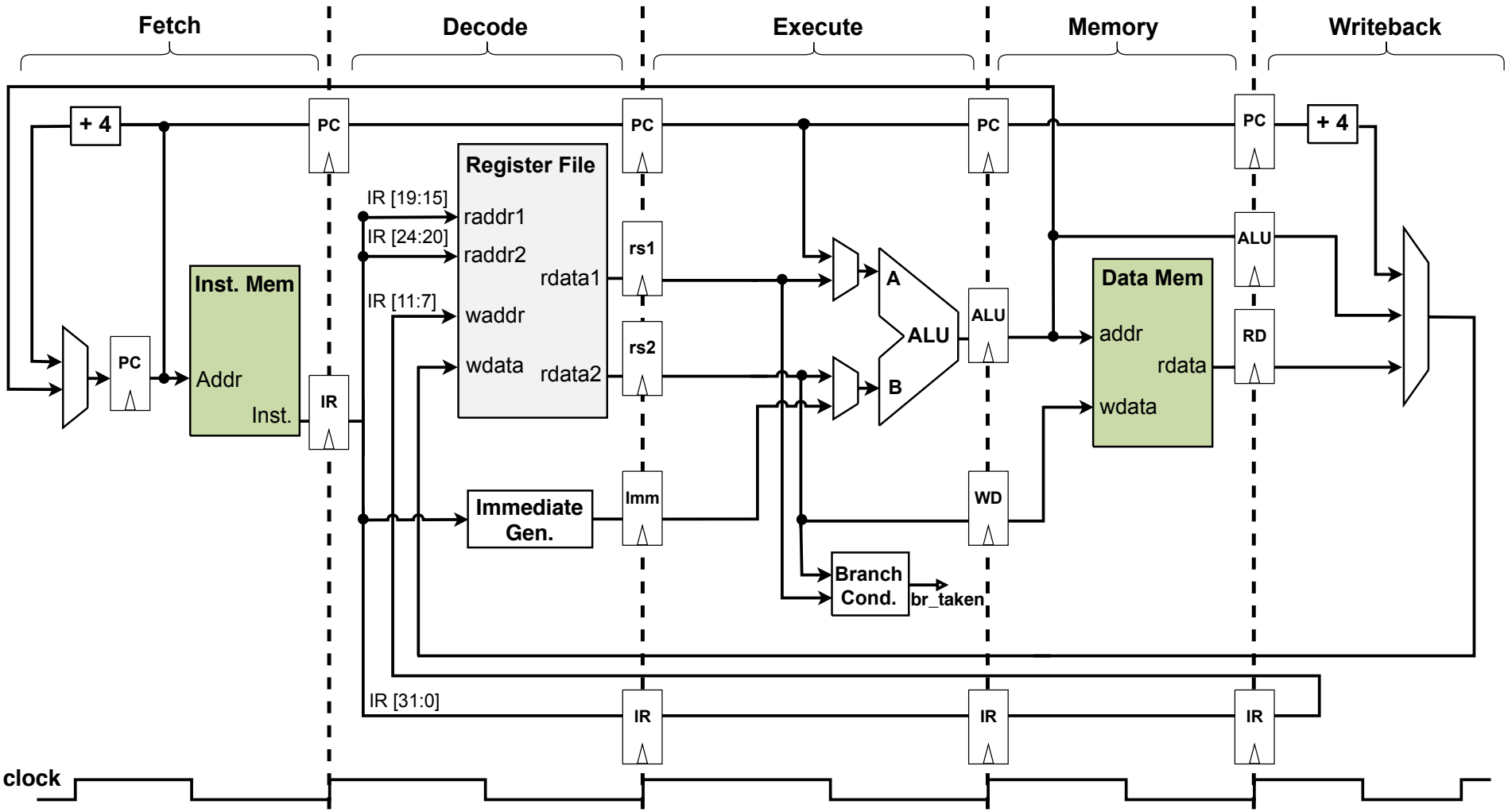
Stall cycles = 0

```
lw    x3,  4(x12)
addi  x4,  x3,      5
```

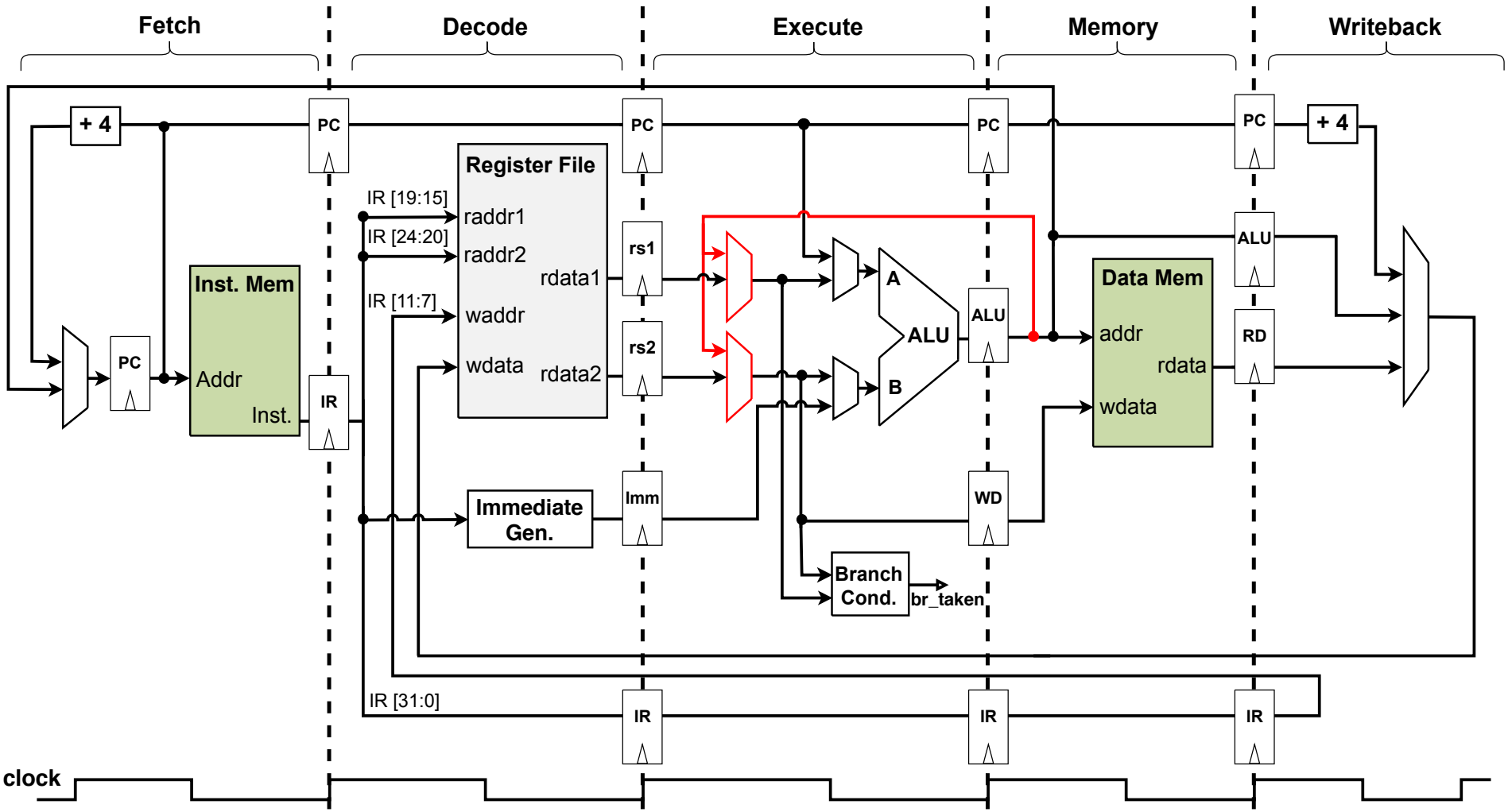
Stall cycles = 1

- Pipeline with n execution stages
  - No. of stall cycles $\geq n - 1$

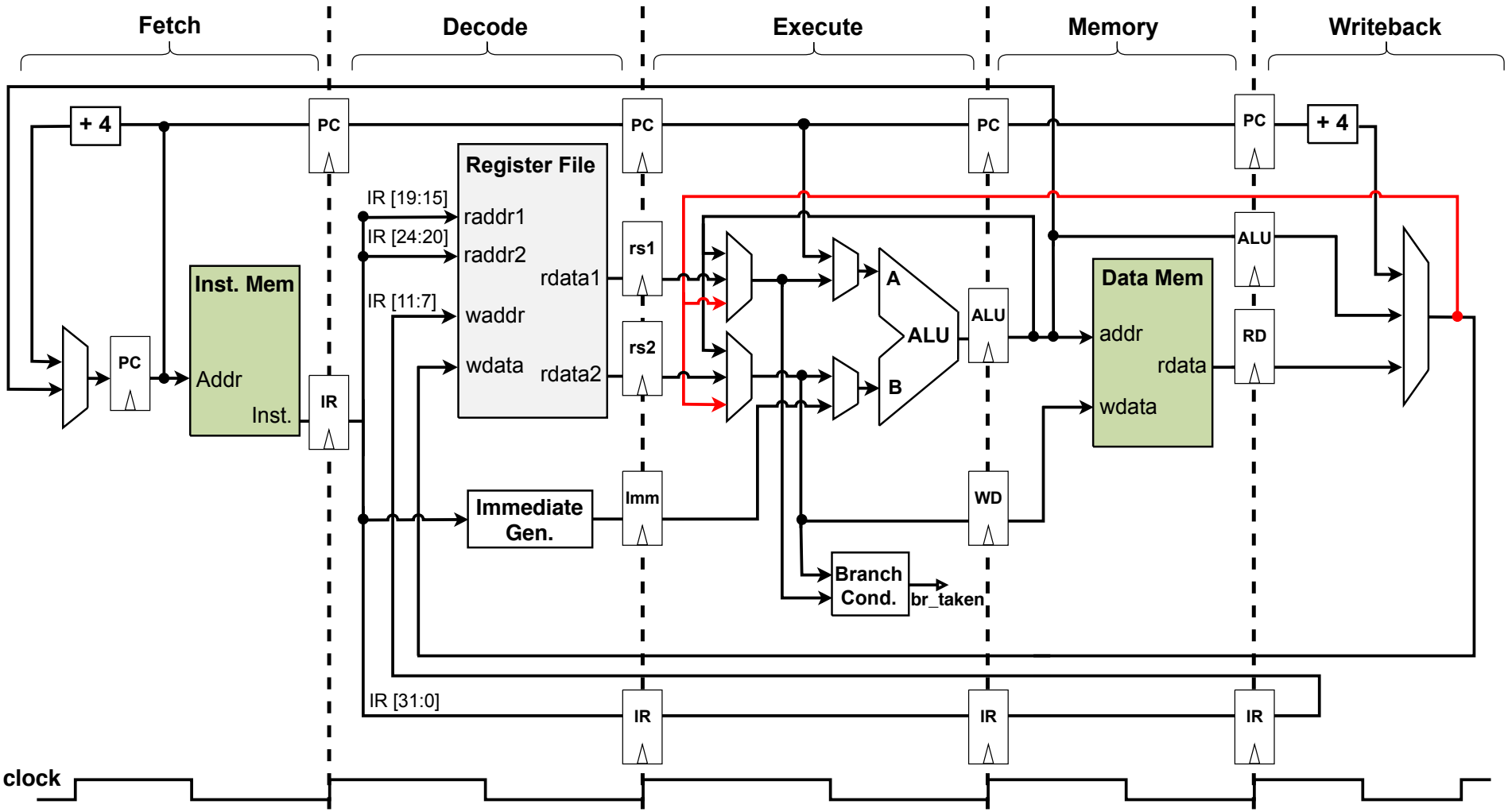
Datapath without Forwarding



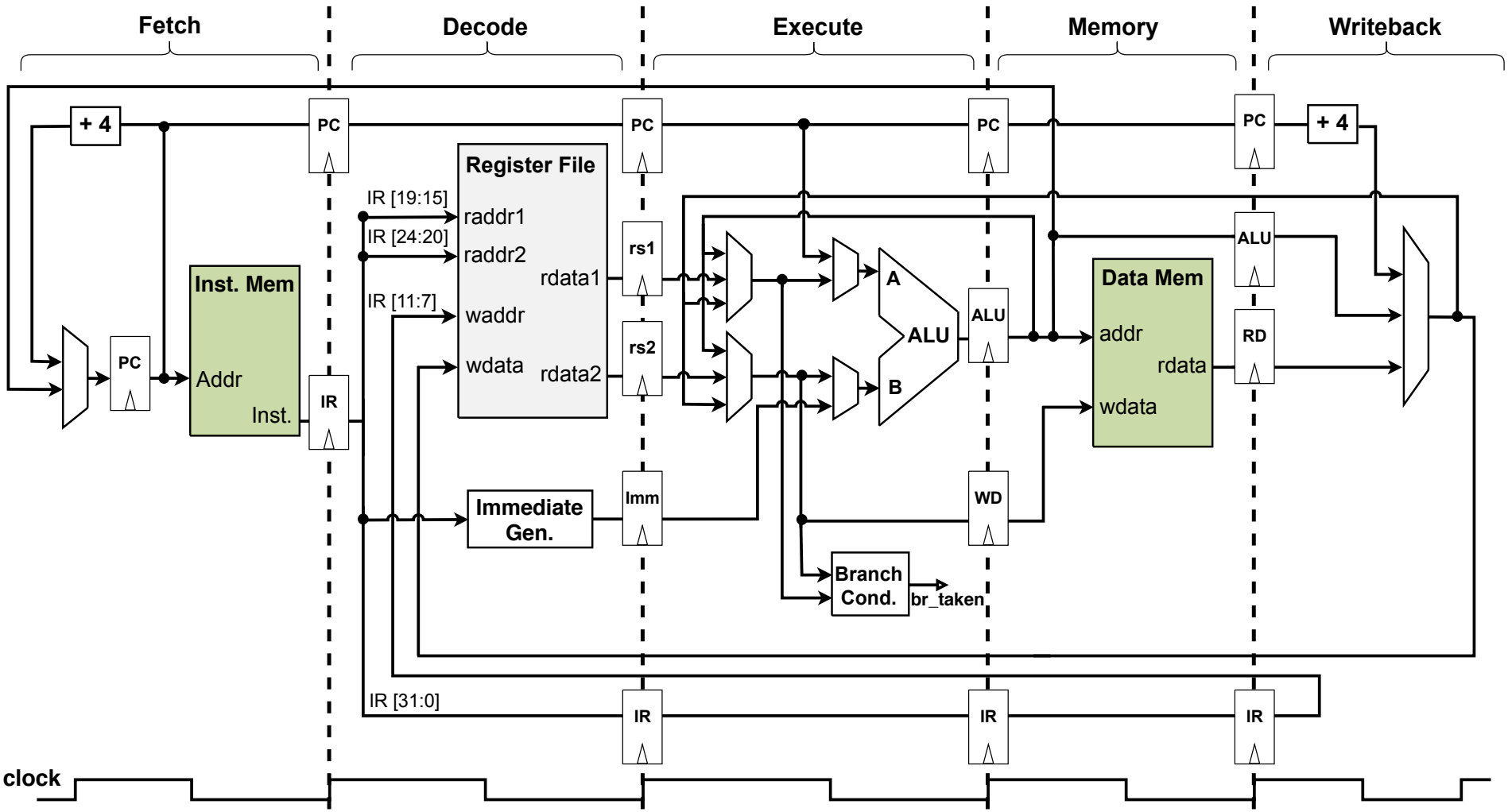
Datapath with Forwarding



Datapath with Forwarding Cont'd



Updated Datapath with Forwarding



Resolving RAW DATA Hazards: Hardware Solution

- `Interlocking` or `Stalling`: Freeze earlier pipeline stages till the data becomes available
- `Bypassing` or `Forwarding`: Route the data to the earlier pipeline stage(s) as soon as it has become available
- `Out of Order Execution`: Dynamic instruction scheduling to avoid data hazards (to be discussed later)

Resolving RAW DATA Hazards: Software Solution

Example program

```
x = a + b;
y = b + c;
```

Code Scheduling to resolve RAW Hazard

```
lw    x3,  0(x10)
lw    x4,  4(x10)
add   x5   x3, x4
sw    x5,  12(x10)
lw    x6,  8(x10)
add   x7   x3, x6
sw    x7,  16(x10)
```

Resolving RAW DATA Hazards: Software Solution

Example program

```
x = a + b;
y = b + c;
```

Code Scheduling to resolve RAW Hazard

```
lw   x3, 0(x10)
lw   x4, 4(x10)
add  x5  x3, x4
sw   x5, 12(x10)
lw   x6, 8(x10)
add  x7  x3, x6
sw   x7, 16(x10)
```

```
lw   x3, 0(x10)
lw   x4, 4(x10)
lw   x6, 8(x10)
add  x5  x3, x4
sw   x5, 12(x10)
add  x7  x3, x6
sw   x7, 16(x10)
```

RAW Hazard Detection Unit Cont'd

- Assume **Current** instruction is in the D/E pipeline register
- Then **Previous** instruction is in the E/M pipeline register and **Second-previous** instruction is in the M/W pipeline register

RAW Hazard Detection Unit Cont'd

- Assume **Current** instruction is in the D/E pipeline register
- Then **Previous** instruction is in the E/M pipeline register and **Second-previous** instruction is in the M/W pipeline register
- Detecting RAW hazards between **Current and Previous** instructions
  (E/M.IR.Rd == D/E.IR.Rs1) OR (E/M.IR.Rd == D/E.IR.Rs2)
- Detecting RAW hazards between **Current and Second-previous** instructions
  (M/W.IR.Rd == D/E.IR.Rs1) OR (M/W.IR.Rd == D/E.IR.Rs2)

Forwarding for RAW Hazard

- Forwarding for RAW hazards between **Current and Previous** instructions

  if ((E/M.CR.WB_REG == WB_ALU) && (E/M.IR.Rd $\neq$ 0) &&
  (E/M.IR.Rd == D/E.IR.Rs1))
  then Forward to A input of ALU

Pipeline Performance

- Let n instructions (excluding branch or jump) are executed using k stage pipeline

$$\text{clock cycles required} = k + (n - 1)$$

- CPI for pipelined implementation

$$\begin{aligned} \text{CPI}_{piplined} &= \frac{k + (n-1)}{n} \\ \text{CPI}_{piplined} &\approx \lim_{n \to \infty} \frac{k + (n-1)}{n} \\ &\approx 1. \end{aligned}$$

Pipeline Performance Cont'd

- Recall

$$\text{Execution Time} = N \times CPI \times \frac{time}{cycle}$$

- For datapath implementation having 5 phases (fetch, decode, execute, memory, write-back)

Implementation	No. of Inst.	CPI	$\frac{time}{cycle}$
Single Cycle	N	1	Large ($\approx 5x$)
Multicycle	N	> 1 (≈ 4)	Small ($\approx 1x$)
Pipelined	N	≈ 1	Small ($\approx 1x$)

Pipeline Performance Cont'd

- Speedup due to pipelining

$$\text{Speedup} = \frac{CPI_{unpipelined}}{CPI_{pipelined}} \times \frac{\text{Cycle time}_{unpipelined}}{\text{Cycle time}_{pipelined}}$$

$$\text{Speedup} \approx \frac{1}{1 + \text{stall cycles per instruction}} \times \text{Pipeline depth}$$

where

$$\text{Pipeline depth} \approx \frac{\text{Cycle time}_{unpipelined}}{\text{Cycle time}_{pipelined}}$$

Suggested Reading

- Read relevant sections of Chapter 4 of [Patterson and Hennessy, 2021].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2021).

Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.

Morgan Kaufmann.