

Data Hazards

– Timing error or race between dependent instructions

- **WAW Dependency/Hazard:**

- Between instructions that write to the same register (or memory location)

| | |
|-----|-------------|
| ADD | R1, R2, R3 |
| LD | R1, 100(R5) |

- **WAR Dependency/Hazard**

Between instruction that writes to a register and a subsequent instruction that reads the same register

| | |
|-----|-------------|
| ADD | R1, R2, R3 |
| LD | R2, 100(R5) |

- **RAW Dependency/Hazard**

Between instruction that reads a register and a subsequent instruction that writes the same register

| | |
|-----|-------------|
| ADD | R5, R2, R3 |
| LD | R2, 100(R5) |

RAW Hazards

- 5-stage pipeline has no WAR and WAW Hazards
- RAW Hazards
 - Simple solutions based on Stall
 - Software Solution
 - Hardware Solution
 - Reducing Performance Penalty
 - Software
 - Hardware

RAW Hazard

A : ADD **R1**, R2, R3

B : ADD R4, **R1**, R5

Hazard possible since register reads occur earlier stage than writes

- A writes R1 at cycle 5
- B may read R1 at cycle 3, 4 or 5
 - If B reads R1 at cycle 3 or 4: RAW Hazard
 - If B reads R1 at cycle 5 or later; No Hazard
- Example: A and B consecutive instructions

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----|----|-----------|-----|-----------|----|
| A | IF | ID | EX | MEM | WB | |
| B | | IF | ID | EX | MEM | WB |

Instruction B reads stale value in R1, before its update by A

Solutions for RAW Hazards

- **Correctness:**

- a) Introduce **stall cycles** (delays) to **avoid hazard**

- **Delay second instruction till write is complete**











- **Software**

- Insert NOPs into delay slots between the instructions
 - At least 2 independent instructions

- **Hardware**

- Hazard Detection Unit detects the hazard and stalls the pipeline

Compiler Inserted NOPs

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----|----|---|---|--|---|---|---|----|
| A | IF | ID | EX | MEM | WB | | | |
| NOP | |  |  |  |  |  | | |
| NOP | | |  |  |  |  |  | |
| B | | | | IF | ID | EX | MEM | WB |

Consecutive instructions A, B forced apart by 2 NOPs to avoid RAW hazard

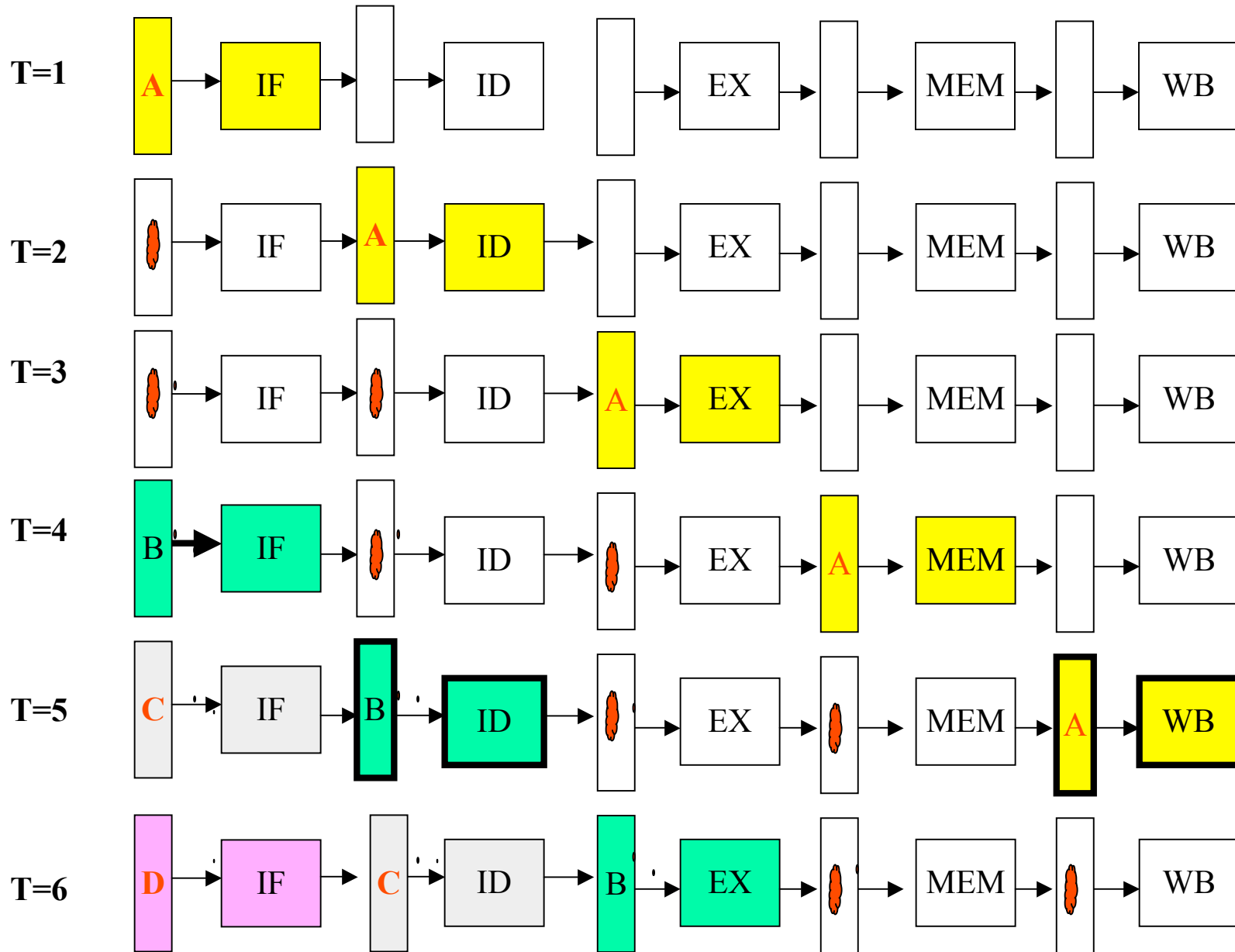
A : ADD R1, R2, R3

B : ADD R4, R1, R5

- NOPS add 2 cycles to the execution time
- Suppose 40% of ALU instructions are followed by a dependent ALU instruction with separation 1 or 2. 90% of instructions are ALU instructions

$$\text{Worst-case CPI} = 1.0 + 90\% \times 40\% \times 2 = 1.72$$

Compiler Inserted NOPS



Hardware-Controlled Pipeline Stall

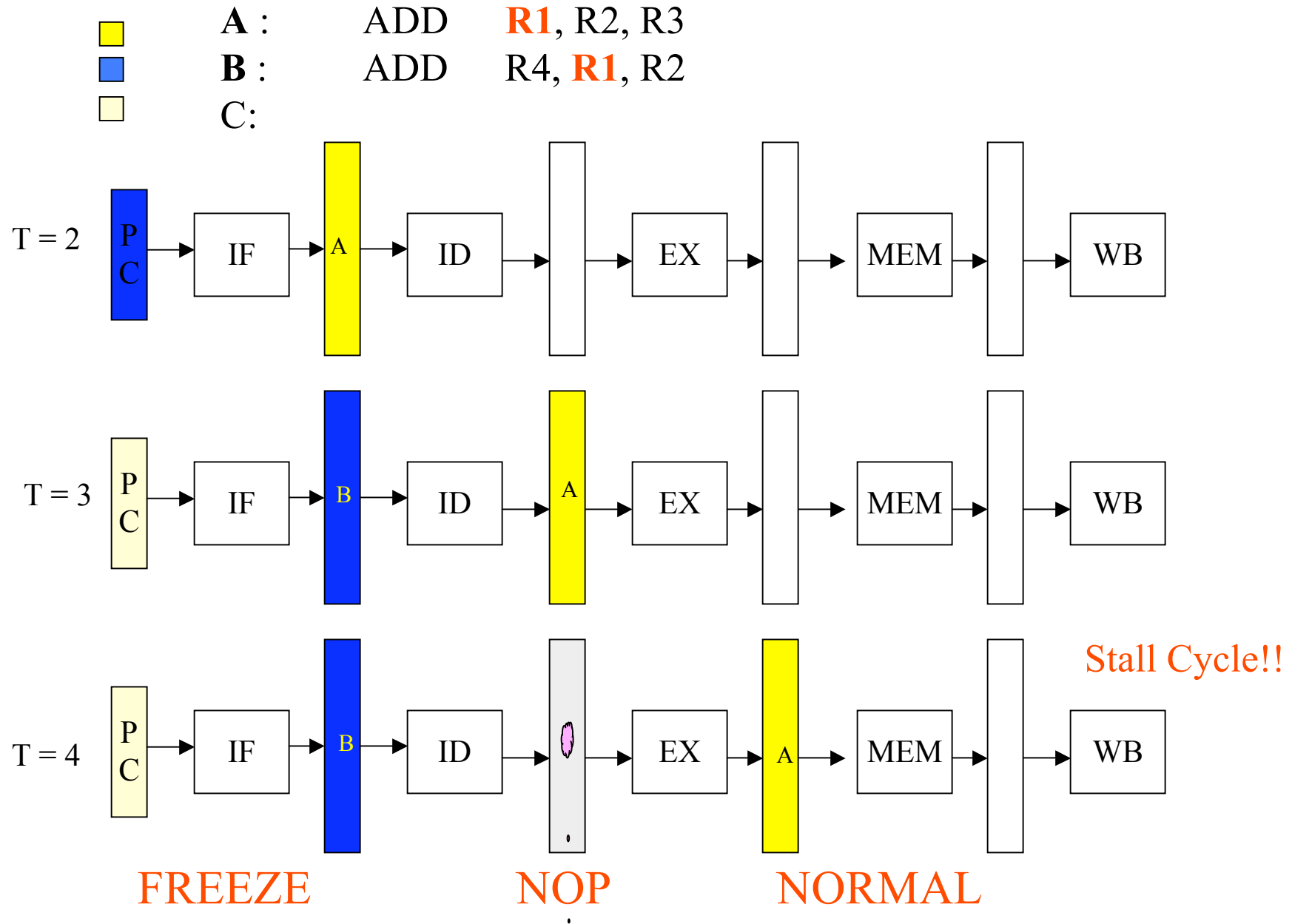
A : ADD **R1**, R2, R3

B : ADD R4, **R1**, R2

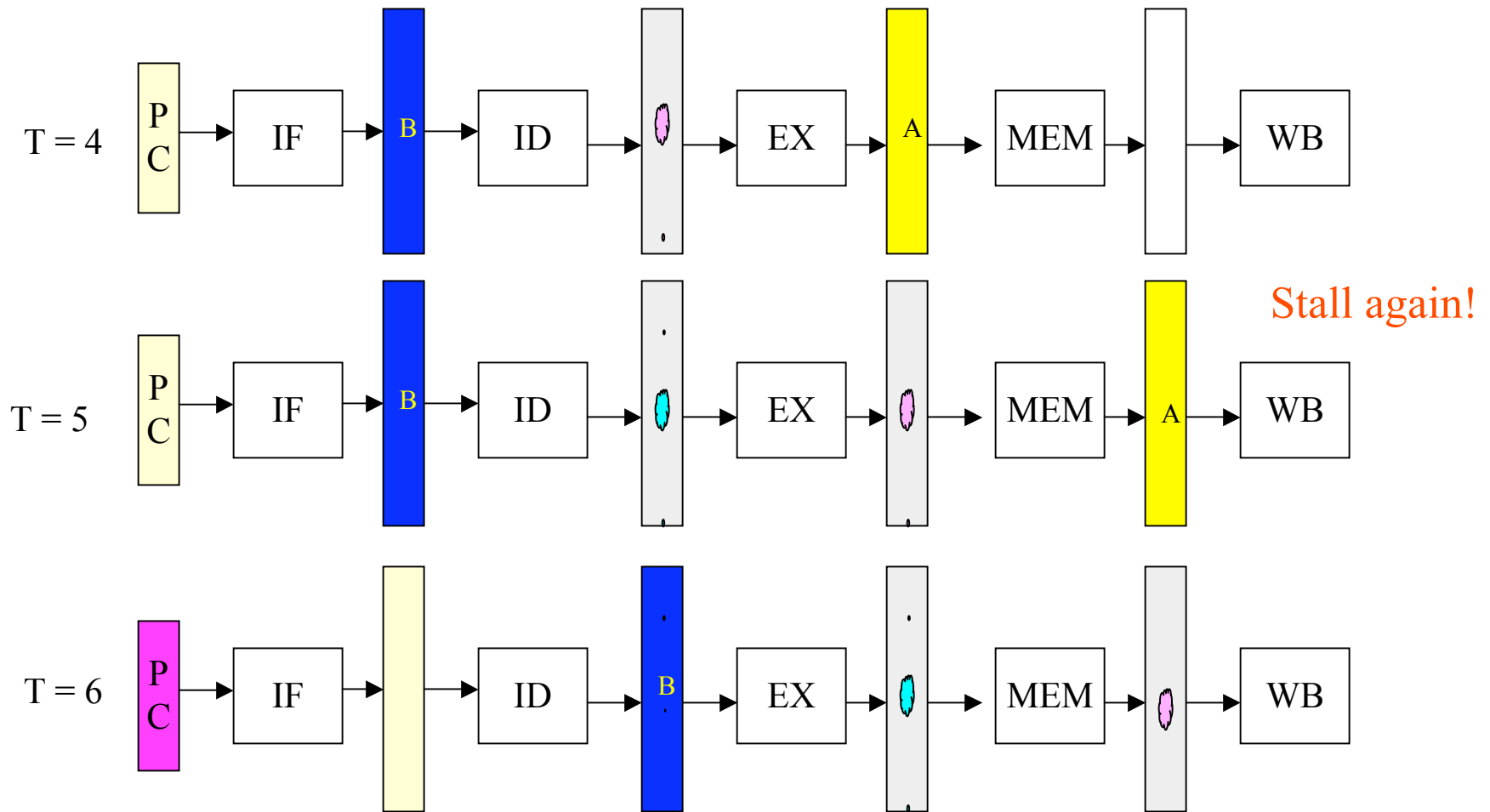
- Hazard Detection unit detects hazardous RAW dependency
- Stalls the pipeline till hazard is avoided
- Delays B by 2 cycles

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|----|----|-----------|-----------|-----------|----|-----|-----|-----|
| A | IF | ID | EX | MEM | WB | | | | |
| B | | IF | ID | ID | ID | EX | MEM | WB | |
| C | | | IF | IF | IF | ID | EX | MEM | WB |
| D | | | | | | IF | ID | EX | MEM |

Hardware Controlled Pipeline Stall

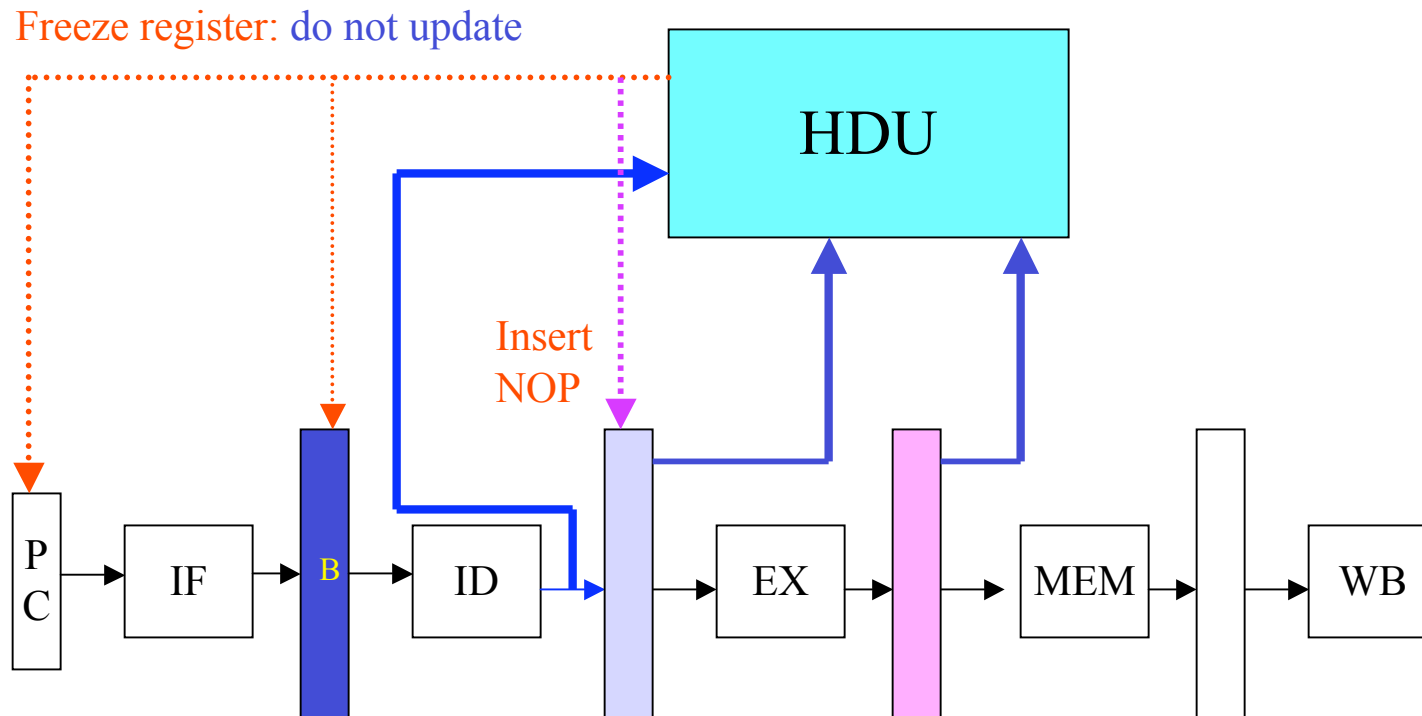


Hardware Controlled Pipeline Stall



- Instruction B held in IF/ID register until A reaches WB stage
- Internally generated NOPs propagated forward while B is stalled

Hazard Detection Unit



Stall Pipeline if instruction in **IF/ID register** reads register **W** and the instruction in either the **ID/EX register** or the **EX/MEM register** will write register **W**

W is in either the **rt** (RI) or **rd** (RR) field of the writing instruction and the rs or rt field of the reading instruction

Operation of Hazard Detection Unit

Compare Register numbers of the

READ REGISTER of instruction in **IF/ID Pipeline Register**

with the

WRITE REGISTER of the instruction in the **ID/EX Pipeline Register** **and**

WRITE REGISTER of the the instruction in the **EX/MEM Pipeline Register**

If any of the comparisons succeed: **Insert Stall Cycle**

FREEZE PC and **IF/ID Pipeline Register**

Insert NOP into **ID/EX Pipeline Register**

Write Register

Read Register

| | | |
|-----|----|--------|
| R-R | rd | rs, rt |
| R-I | rt | rs |
| LD | rt | rs |
| SD | -- | rs, rt |
| Bcc | -- | rs |
| Bcc | -- | rs, rt |

Solutions for RAW Hazards

- **Correctness:**
 - Introduce **stall cycles** (delays) to **avoid hazard** manifestation
 - compiler
 - hardware
- **Correctness + Performance:**
 - **Reduce** or **eliminate stall** cycles
 - program optimization
 - additional hardware
 - Combination
 - **Mask delay**
 - overlap stall cycles with other useful operations

Solutions for RAW Hazards

- Correctness + Performance

- a) Reduce or eliminate stall cycles

- Software

- Restructure code to fill delay slots with independent instructions
 - (Compiler Optimizations)

- Hardware

- Forwarding (Register bypass)
 - Provide alternate datapaths within the pipeline to communicate values
 - Instruction gets value directly from source instruction bypassing the register

- Combination

- Load Delay Slot
 - Mask delay

- b) Overlap stall cycles with other useful operations

Performance Issues

NOPS and stalls consume cycles and reduce throughput

- Software

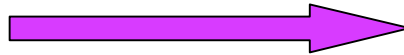
Reorganize assembly code

Move an independent instruction in the delay slot (where the NOP was inserted)

ADD R1, R2, R3
SUB R4, R1, R5
XOR R3, R2, R7
AND R8, R7, R7

Original Code

Compiler Optimization



ADD R1, R2, R3
XOR R3, R2, R7
AND R8, R7, R7
SUB R4, R1, R5

Optimized Code

- Hardware:

Forwarding and Bypass hardware

Forwarding

Example: Two R-R Type instructions with RAW dependencies

A : ADD **R1**, R2, R3

B : ADD R4, **R1**, R2

What is the effect of these two instructions?

- Value to be written into R1 by A **computed** in EX stage (**cycle 3**)
- Value **used** by B in EX stage (**cycle 4**)

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----|----|-----------|-----------|-----|----|
| A | IF | ID | EX | MEM | WB | |
| B | | IF | ID | EX | MEM | WB |

Forwarding

Example: Two R-R Type instructions with RAW dependencies

A : ADD **R1**, R2, R3

B : ADD R4, **R1**, R2

- Why wait till value written into register R1?
- Why use R1 to communicate the result of A to B ?
- **Directly forward result of A to B**
- Provide **alternate datapaths** from ALU output back to its input

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----|----|----|-----|-----|----|
| A | IF | ID | EX | MEM | WB | |
| B | | IF | ID | EX | MEM | WB |

Forwarding R-R Type Instructions

A : ADD **R1**, R2, R3

B : ADD R4, **R1**, R5

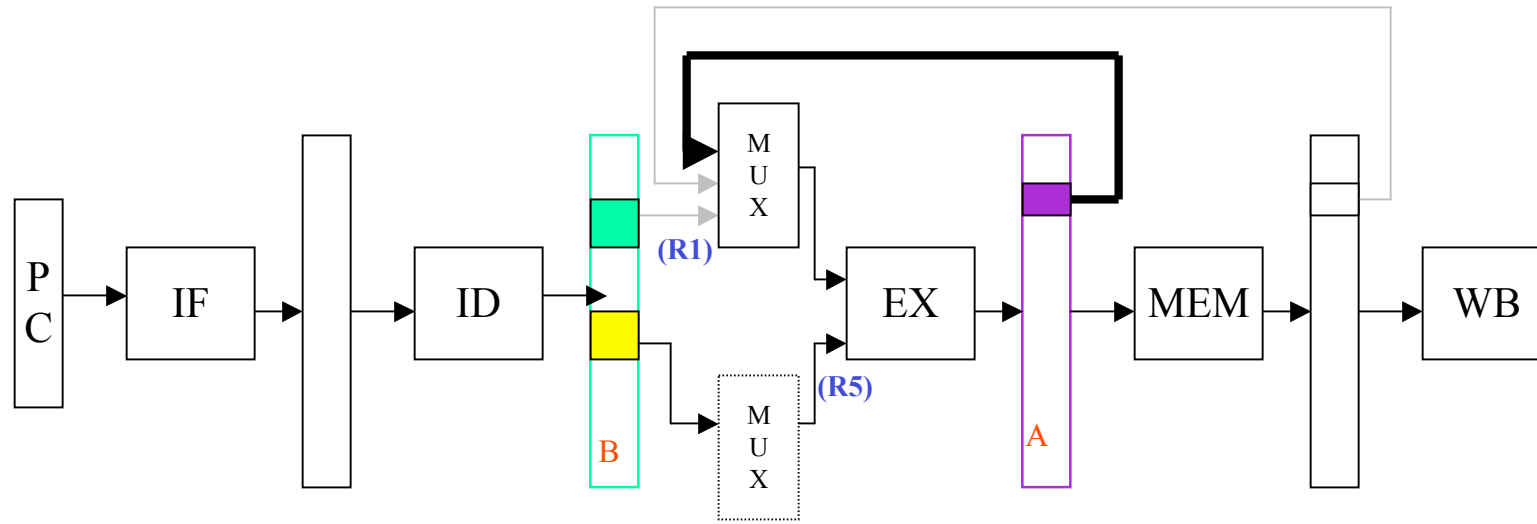
Result of **A** in EX/MEM register at end of cycle 3



Stale R1 value read by **B** in ID/EX register at end of cycle 3



B uses value forwarded from EX/MEM register in EX stage (cycle 4)



Forwarding


Example: Two R-R Type instructions with RAW dependencies

A : ADD **R1**, R2, R3

X: ADD R6, R7, R8

B : ADD R4, **R1**, R2

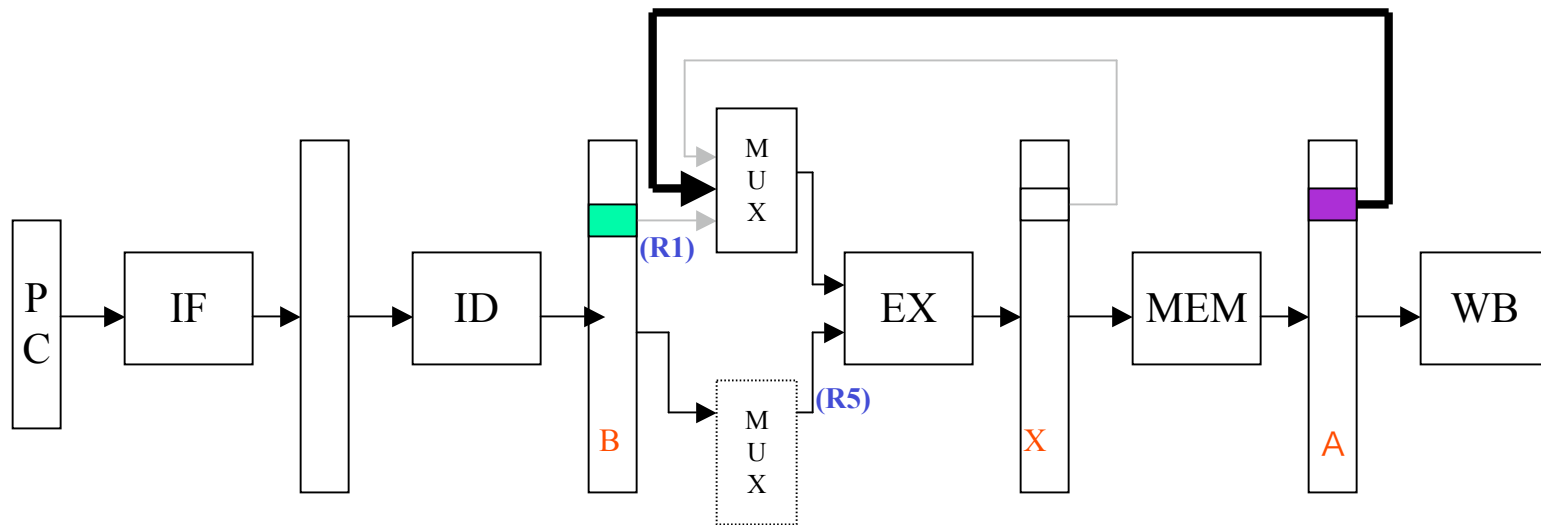
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|----|----|----|-----|-----|-----|----|
| A | IF | ID | EX | MEM | WB | | |
| X | | IF | ID | EX | MEM | WB | |
| B | | | IF | ID | EX | MEM | WB |



Forwarding R-R Type Instruction

A : ADD **R1**, R2, R3
X: ADD R6, R7, R8
B : ADD R4, **R1**, R5

- Result of A in MEM/WB register at end of cycle 4
 - Stale R1 value read by B in ID/EX register at end of cycle 4
- B uses value **forwarded** from MEM/WB register in EX stage (cycle 5)



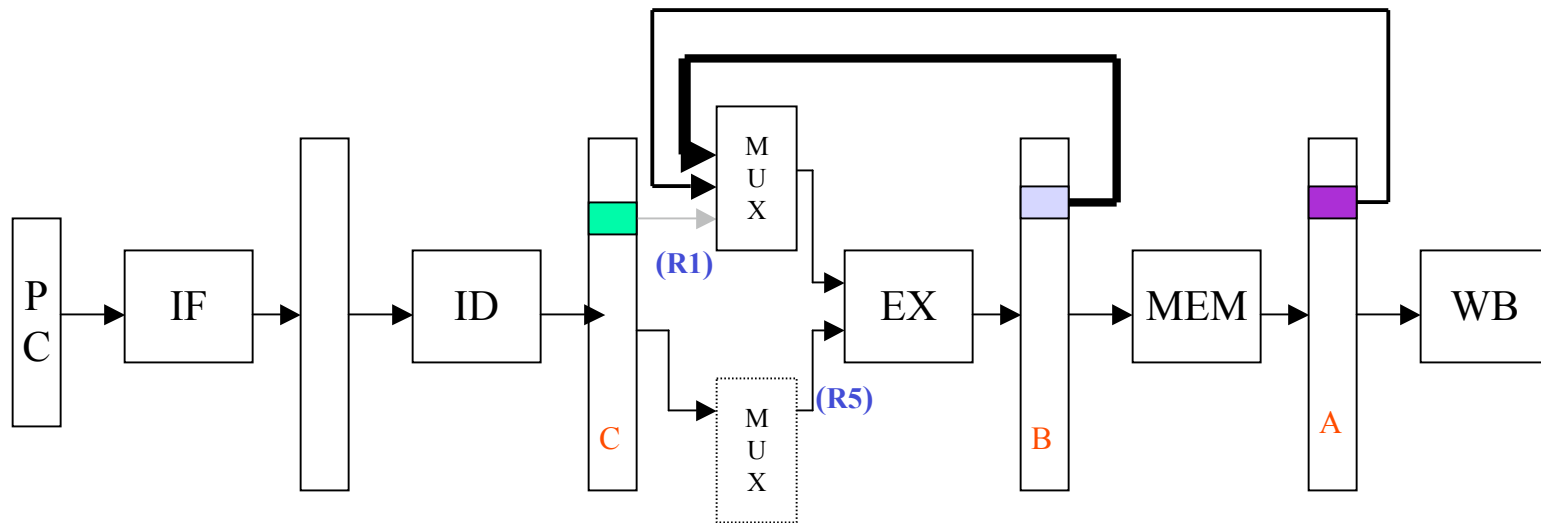
Forwarding R-R Type Instruction

A : ADD **R1**, R2, R3

B: ADD **R1**, R6, R7

C : ADD R4, **R1**, R5

- • Result of **A** in MEM/WB register at end of cycle 4
- • Result of **B** in EX/MEM register at end of cycle 4
- • **Stale** R1 value read by C in ID/EX register at end of cycle 4
- C uses value forwarded from EX/MEM register in EX stage (cycle 5)



Forwarding for Load Instructions

Example:

■ A: LD **R1**, 0(R2)

■ B: ADD R3, **R1**, R4

Forwarding is **insufficient** to resolve RAW data hazard

A obtains value from memory at end of cycle 4

B computes with R1 during cycle 4

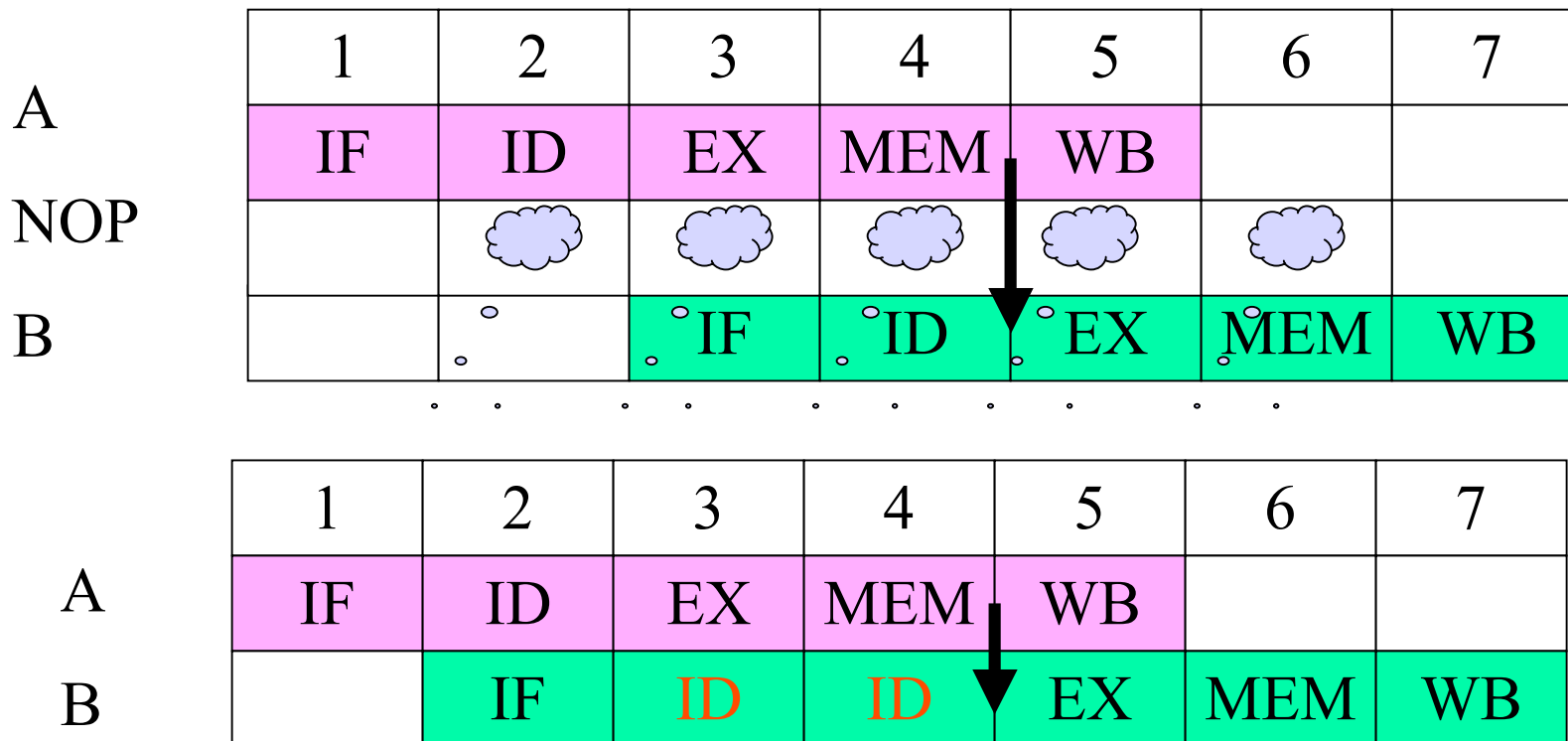
| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----|----|----|-----|-----|----|
| A | IF | ID | EX | MEM | WB | |
| B | | IF | ID | EX | MEM | WB |

Load Hazard

- Requires Delay even with Forwarding hardware
- In DLX the delay is done by software:
 - Instruction following LD executes in the Load Delay slot
 - Load Delay slot exposed to the programmer
 - Software must ensure that the instruction following a LD must not have RAW dependence with the LD
 - Explicitly insert NOP (or independent instruction) after load instruction

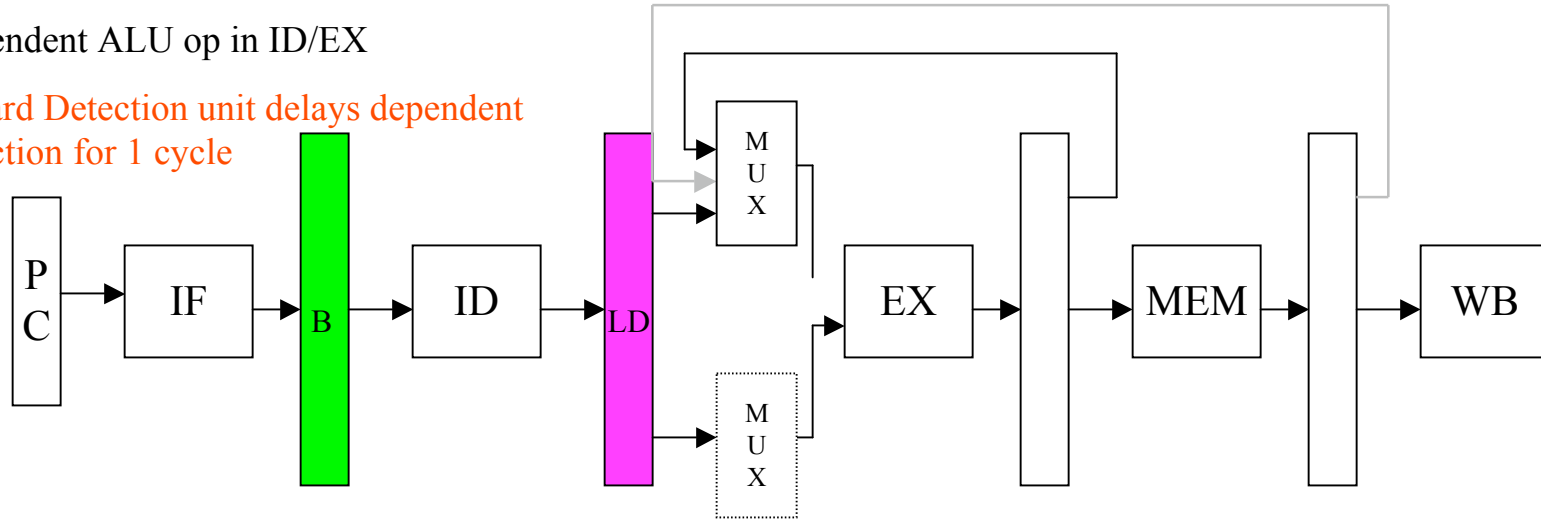
Load Hazard

1. Need to **delay B** for 1 cycle
 - (a) Software: Add **NOP** (or independent instruction) between **A** and **B** **OR**
 - (b) Hardware: **Stall B** for 1 cycle in **ID** stage using HDU
2. **Forward** data read from memory (in **MEM/WB register**) to **EX** stage



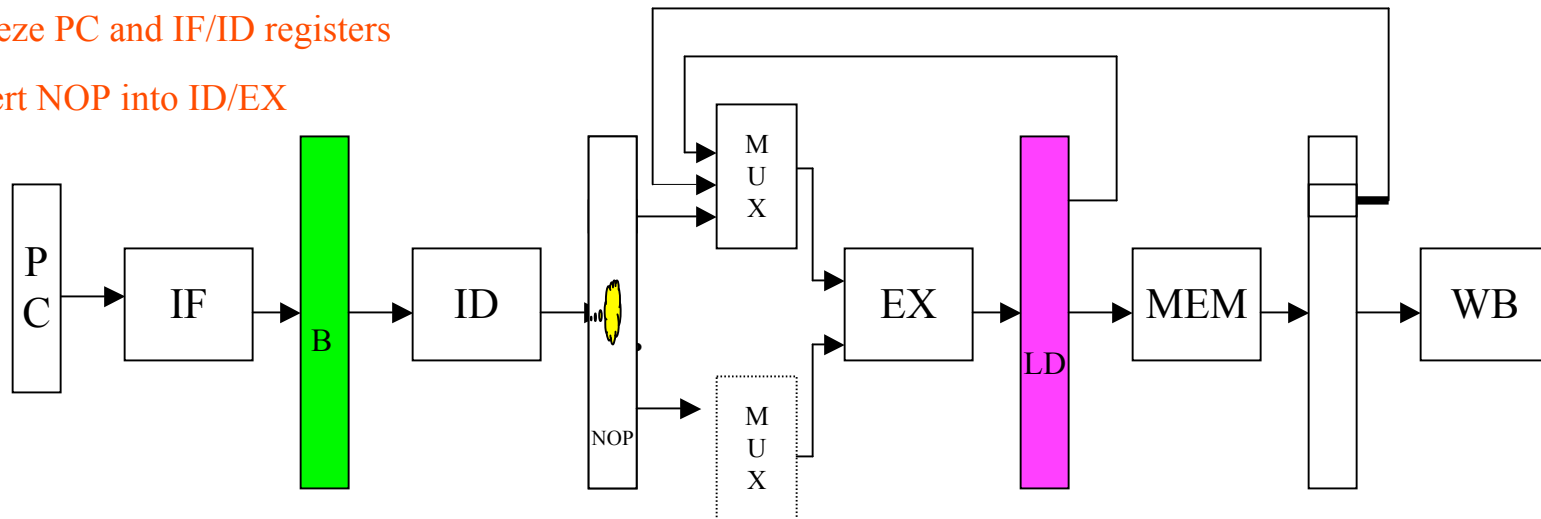
LD with Stall and Forwarding

- LD in ID/EX
- Dependent ALU op in ID/EX
- Hazard Detection unit delays dependent instruction for 1 cycle



Stall Step: Insert NOP

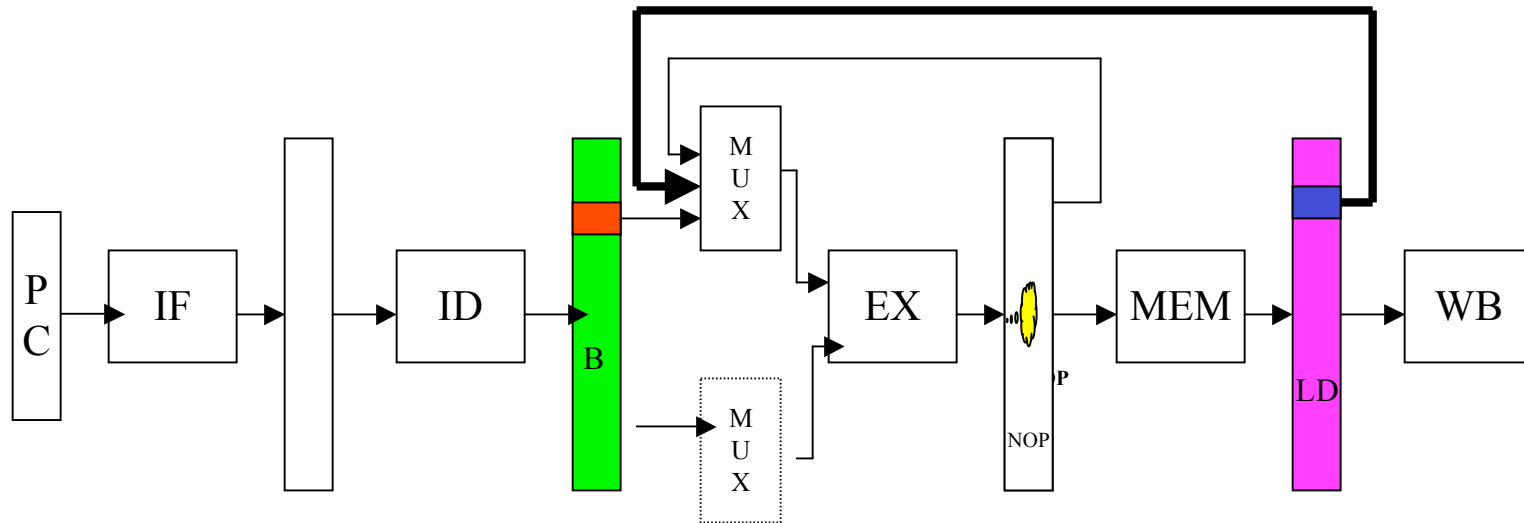
- Freeze PC and IF/ID registers
- Insert NOP into ID/EX



LD with Stalls and Forwarding

Forwarding Step: Forward data from MEM/WB to ALU Input

- LD in MEM/WB
- Dependent ALU instruction in ID/EX
- Forward output from MEM/WB to ALU input



What if dependent instruction was 2 cycles behind the LD?

LD and SD combinations?

Other instruction combinations?