# Pipelining Stall Due To Data Dependency

Dr. Suman Kumar Maji

Faculty of CSE Department
Indian Institute of Technology Patna

August 17, 2023

# Pipeline Stalls due to Data Dependencies

- Data dependencies occur when the execution of an instruction depends on the results of a preceding instruction.
- In a pipeline, data dependencies can lead to pipeline stalls or delays.
- One common type of data dependency is a **read-after-write** dependency, where a source operand of instruction $I_i$ depends on the results of executing a preceding instruction $I_j$, where $i > j$.
- In such scenario, $I_j$ must complete execution and write its result to the register before $I_i$ can proceed.
- This dependency introduces a pipeline stall, as $I_i$ has to wait for $I_j$ to complete before it can fetch its operands.

# Pipeline Execution Example

Example Instructions:

- ADD R1, R2, R3
- SL R3
- SUB R5, R6, R4

| Step | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | IF | ID | OF | OE | OS | | | | |
| 2 | | IF | ID | – | – | OF | OE | OS | |
| 3 | | | IF | – | – | ID | OF | OE | OS |

Table: 5 Phase Pipeline

# Example

consider the execution of the following sequence of instruction on a 5 phase pipeline. IF, ID, OF, OE, OS . Show the execution of this instruction pipeline.

- I1: Load -1, R1 R1 $\leftarrow$ (-1)
- I2: Load 5, R2 R2 $\leftarrow$ 5
- I3: Sub R2, 1, R2 R2 $\leftarrow$ R2 - 1
- I4: Add R1, R2, R3 R3 $\leftarrow$ R1 + R2
- I5: Add R4, R5, R6 R6 $\leftarrow$ R4 + R5
- I6: SL R3 R3 $\leftarrow$ SL(R3)
- I7: Add R6, R4, R7 R7 $\leftarrow$ R4 + R6

# Methods to Reduce Pipeline Stall due to Instruction Dependency

- **Re-ordering:** Sequence of instructions reordered while guaranteeing the final results.
- **Use of dedicated hardware:** Implement specialized hardware to handle and recognize the branch address without additional time.
- **Precomputing:** Compute and reorder branches in advance to reduce stalls.

# Methods to Reduce Pipeline Stall due to Instruction Dependency

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| IS | | | I1 | I4 | I2 | I3 | Ij | Ij+1 |
| IE | | I1 | I4 | I2 | I3 | Ij | Ij+1 | |
| IF | I1 | I4 | I2 | I3 | Ij | Ij+1 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Table: Table A

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| IS | | | I1 | I2 | I3 | I4 | Ij | Ij+1 |
| IE | | I1 | I2 | I3 | I4 | Ij | Ij+1 | |
| IF | I1 | I2 | I3 | I4 | Ij | Ij+1 | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Table: Table B

# Methods to Reduce Pipeline Stall due to Data Dependency

- Hardware operand forwarding: Result of 1 ALU operation made available to another ALU operation in the cycle that immediately follows.
  ADD R1,R2,R3 ; R3$\leftarrow$ R1 + R2
  SUB R3,1,R4; R4$\leftarrow$ R3 - 1

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| IS | | | | | I1 | – | I2 | | |
| IE | | | | I1 | – | I2 | | | |
| OF | | | I1 | – | I2 | | | | |
| ID | | I1 | I2 | | | | | | |
| IF | I1 | I2 | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Software Operand Forwarding

- A smart compiler performs data dependency analysis to mitigate pipeline stalls.
- It recognizes three forms of data dependencies:

## STORE-FETCH

- Before software operand forwarding:
    - Instruction 1: Store R2, (R3); M[R3] $\leftarrow$ R2
    - Instruction 2: Load (R3), R4; R4 $\leftarrow$ M[R3]
- After software operand forwarding:
    - Instruction 1: Store R2, R3; M[R3] $\leftarrow$ R2
    - Instruction 2: Move R2, R4; R4 $\leftarrow$ R2

# Software Operand Forwarding

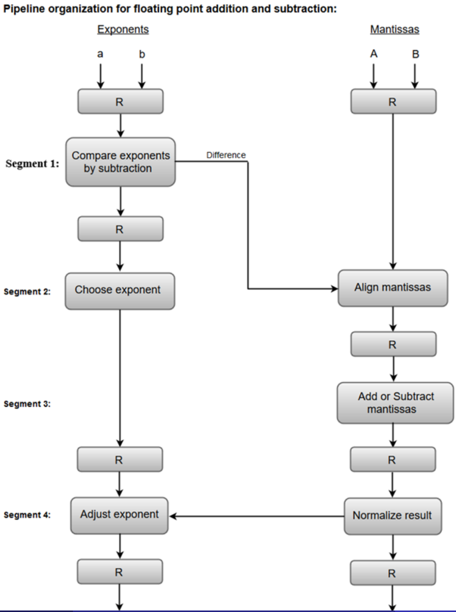
## FETCH-FETCH

- Before software operand forwarding:
  - Instruction 1: Load (R3), R2; R2 $\leftarrow$ M[R3]
  - Instruction 2: Load (R3), R4; R4 $\leftarrow$ M[R3]
- After software operand forwarding:
  - Instruction 1: Load (R3), R2; R2 $\leftarrow$ M[R3]
  - Instruction 2: Move R2, R4; R4 $\leftarrow$ R2

## STORE-STORE

- Before software operand forwarding:
  - Instruction 1: Store R2, (R3); M[R3] $\leftarrow$ R2
  - Instruction 2: Store R4, (R3); M[R3] $\leftarrow$ R4
- After software operand forwarding:
  - Instruction 1: Store R2, (R1); M[R1] $\leftarrow$ R2
  - Instruction 2: Store R4, (R3); M[R3] $\leftarrow$ R4

# Arithmetic Pipeline



# Floating Point Addition

- Inputs to the floating-point adder pipeline:
  - $X = A \times 2^a = 0.9504 \times 10^3$
  - $Y = B \times 2^b = 0.8200 \times 10^2$
- Segment 1: Compare exponents by subtraction
  - The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.
  - The difference of the exponents, i.e., $3 - 2 = 1$, determines how many times the mantissa associated with the smaller exponent must be shifted to the right.
- Segment 2: Align the mantissas
  - The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.
  - $X = 0.9504 \times 10^3$
  - $Y = 0.08200 \times 10^3$
- Segment 3: Add mantissas
  - The two mantissas are added in segment three.
  - $Z = X + Y = 1.0324 \times 10^3$
- Segment 4: Normalize the result
  - The result is written as: $Z = 0.1324 \times 10^4$