Pipelining Calculation

- Minimum time for a pipeline stage

\[ \tau = \max_i (T_i) + d = T_m + d \]

\( T_i \): the delay of combinational logic for stage \( i \).
\( T_m \): the maximum delay of combination logic for any stage.
\( d \): latch delay

- Speedup

\[ \text{Speedup} = \frac{\text{Time to execute N instructions on 1 stage}}{\text{Time to execute N instructions on K stages}} = \frac{N \tau_1}{[k + (N - 1)] \tau} \]

\( T_1 \): the delay of single-layer combination logic plus the latch delay

- Example: four-layer CPU, 5 ns for each step, 1 ns for the latch delay.
  - Time to execute one instruction on the 4-stage CPU
    - \( (5 + 1) \times 4 = 24 \) ns
  - Time to execute one instruction on the single-stage CPU
    - \( 5 \times 4 + 1 = 21 \) ns
  - Time to execute five instructions on the single-layer CPU
    - \( 5 \times 21 = 105 \) ns
  - Time to execute five instructions on the 4-layer CPU
    - \( (5 + 1) \times (4 + 5 - 1) = 48 \) ns
  - What is the speedup for 5 instructions using a 5-stage CPU
    - \( 105 / 48 = 2.2 \)

- Structural Hazard
  - A required resource is busy
  - Solutions:
    - More stages, more control hardware
    - Waiting
  - Data Hazards
• Need to wait for previous instruction to complete its data read/write

• Solutions:
  - Forwarding: retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible register or memory.
    
    \[
    \begin{align*}
    &\text{ADD} \ R1 \ R2 \ R0 \ # \ R1 + R2 \Rightarrow R0 \\
    &\text{SUB} \ R0 \ R3 \ R4 \ # \ R0 - R3 \Rightarrow R4 \\
    &\text{add} \ F \ D \ E \ W \\
    &\text{sub} \ F \ D \ E \ W
    \end{align*}
    \]

• Cannot prevent Load-use data hazard

  • Load-use data hazard: a specific form of data hazard in which the date being loaded by a load instruction has not yet become available when it is needed by another instruction
  - Waiting/pipeline stall

• Control Hazards

  • Fetching next instruction depends on branch outcome

  • Solutions:
    - Wait until branch outcome determined before fetching next instruction
    - Branch prediction