Pipelining – the Idea

- Similar to assembly line in a factory
- Divide instruction into smaller tasks
- Each task is performed on subset of resources
- Overlap the execution of multiple instructions by completing different tasks from different instructions in parallel
- Ideally this should lead to throughput of one instruction per clock cycle

Unpipelined Machine

- 2 instructions completed in 6 cycles

Pipelined Machine

- 2 instructions completed in 4 cycles

Pipelining Overview

- Each task is called pipe stage or pipe segment
- All stages must be able to proceed at the same time:
  - Stage duration is called the processor cycle
  - It is determined by the slowest stage
- Goal of pipelining is to increase throughput – number of instructions completed per clock cycle
- In ideally balanced pipeline with $n$ stages:
  $$ T_{\text{Instruction, unipipelined}} = \frac{n}{n} $$
  $$ \text{Speedup} = n $$

MIPS Unpipelined

- Fetch instruction
- PC=PC+4
- IF
- ID
- EX
- MEM
- WB
- Write data into register file either for LOAD or for ALU instruction
- If instruction is
  - Load: load data from memory
  - Store: store data from register to memory
- If instruction is
  - Memory reference: add base register and offset to form memory address
  - ALU: perform the operation, for immediate sign-extend the second operand
MIPS Pipelined

- Each task becomes a pipeline stage
- Start a new instruction in each clock cycle

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

- For this to work we have to make sure that pipeline stages use distinct resources

Separate instruction and data memory
- Memory must deliver 5 times the bandwidth
- Register file must support two reads and one write
  - We will perform them in half-cycles, first write and then read
  - We need adder to increment PC and to perform branch target calculation

Example
Consider the unpipelined processor and assume that it has a 1 ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20% and 40%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock. Ignoring any latency impact:
- how much speedup will we gain from a pipeline?
- how many stages has this pipeline?

Pipeline Hazards

- Conflicts that prevent the instruction A from executing during its designated clock cycle:
  1. Structural hazards – instruction A requires a resource occupied by some previous instruction
  2. Data hazards – a source operand of instruction A is the output of some previous instruction and is not ready
  3. Control hazards – pipelining branches and other instructions that change PC

Hazards may make it necessary to stall the pipeline:
- Instruction A and all subsequent instructions are delayed until the hazard resolves
- All previous instructions proceed with execution.
Performance Degradation Due to Hazards

\[
\text{Speedup} = \frac{T_{\text{per instruction unpipelined}}}{T_{\text{per instruction pipelined}}}
\]

\[
C\pi_{\text{unpipelined}} = \frac{C\pi_{\text{pipelined}}}{C\pi_{\text{unpipelined}}}
\]

\[
1 + \text{Stall}_n \cdot \text{Cycles per instruction}
\]

Structural Hazards

- Resource conflicts:
  - E.g. using same memory for data and instructions will create a hazard whenever we have STORE or LOAD
  - Instructions are stalled until resource becomes available
  - Conflicts can be avoided through resource duplication
  - If structural hazards are not that frequent it may be cheaper to allow them

Example

Suppose that data references constitute 40% of the mix, and that the ideal CPI of the pipelined processor, ignoring the structural hazard is 1. Assume that the processor with the structural hazard (data and instructions are stored in the same memory) has a clock rate that is 1.05 times higher than the clock rate of the processor without the hazard. Is the pipeline with or without structural hazard faster, and by how much?

Data Hazards

- Source operand of the current instruction is the output of a previous instruction and is not ready
  - DADD R1, R2, R3
  - DSUB R4, R1, R5
  - AND R6, R1, R7
  - OR R8, R1, R9
  - XOR R10, R1, R11

Data Hazards

- DADD R1, R2, R3
- DSUB R4, R1, R5
- AND R6, R1, R7
- OR R8, R1, R9
- XOR R10, R1, R11
Forwarding

DADD R1, R2, R3
DSUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11

Data Hazards

DADD R1, R2, R3
LD R4, 0(R1)
SD R4, 12(R1)

Forwarding

Forwarding can be generalized as passing the result from one functional unit to another unit that needs it

This is done through pipeline registers, not directly
Data Hazards Requiring Stalls

- Load followed by ALU instructions that use the result
  
  ```
  LD R1, 0(R2)
  DSUB R4, R1, R5
  AND R6, R1, R7
  OR R8, R1, R9
  ```

A piece of hardware called pipeline interlock is added to detect a hazard and stall the pipeline.