Pipelining

- RISC/MIPS64 five stage pipeline
- Basic pipeline performance
- Pipeline hazards
- Branch hazards
- More pipeline performance
- Challenges of pipelining

RISC Datapath

1. Instruction Fetch Cycle (IF)
2. Instruction decode/register fetch cycle (ID)
3. Execution/effective address cyclic (EX)
4. Memory Access (MEM)
5. Write-back (WB)
Avoiding contention for HW resources

- Separate data and instruction caches
- Pipeline registers
- Two stage register usage
  - Write first half of clock cycle
  - Read second half of clock cycle
Pipelining yields performance gains

- Pipelining increases instruction throughput, but does not reduce the execution time of any single instruction.
- Idealized maximum speedup = number of pipeline stages
- Idealized maximum speedup achieved when one instruction completes each clock cycle.

\[
\begin{align*}
\text{CPI( unpipelined)} &= P \\
\text{CPI( pipelined)} &= 1 \\
\text{Speedup} &= (IC \times P \times \text{clock}) / (IC \times 1 \times \text{clock}) \\
&= P
\end{align*}
\]
Limits on pipeline performance

- Imbalance among pipeline stages (limited by the slowest stage)
- Overhead due to pipeline register delays
- Overhead due to clock skew (max delay between when clock signal arrives at any two registers)
- Overhead to fill/drain the pipeline and
- **PIPELINE HAZARDS**

Pipeline Hazards

- **Structural hazards** - conflicts for HW resources
- **Data hazards** - conflicts for access to data
- **Control hazards** - uncertainty about the next instruction to enter the pipeline
Pipelining Performance w/ hazards

- Hazards are dealt with by **stalling** the pipeline. Stalls delay one or more stages for the instruction in the pipeline, causing the CPI to increase.

\[
\begin{align*}
\text{CPI(unpipelined)} &= P \\
\text{CPI(pipelined)} &= 1 + \text{stall cycles per instrct}. \\
\text{Speedup} &= \frac{P}{(1 + \text{stall cycles per instrct})}
\end{align*}
\]
Dealing with structural hazards

- Add more hardware
- Stall the pipeline

Data Hazards:
Incorrect access to data because pipelining changes the order of read/write accesses to data

- DADD R1, R2, R3
- DSUB R4, R1, R5
- AND R6, R1, R7
- OR R8, R1, R9
- XOR R10, R1, R11
Dealing with data hazards

- **Forwarding** - HW technique that gets needed data from the pipeline registers (rather than waiting for it to be WB in the regular registers) and moving it to where that data is needed in a timely manner.

- **Stalling** (with pipeline interlock)
Overcoming data hazard example #1 with forwarding
(Fig. A.7)

Data hazard example 2: forwarding doesn’t work (Fig. A.9)
Overcoming data hazard example #2 by stalling

<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>LD R1, 0(02)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB R4, R1, 05</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>AND R6, R1, 07</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>OR R8, R1, 09</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>LD R1, 0(02)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB R4, R1, 05</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>AND R6, R1, 07</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>OR R8, R1, 09</td>
<td></td>
<td></td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

Figure A.10: In the top half, we can see why a stall is needed: The MEM cycle of the load produces a value that is needed in the EX cycle of the SUB, which occurs at the same time. This problem is solved by inserting a stall, as shown in the bottom half.