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)
### Classic RISC Five Stage Pipeline (Fig. A.1)

<table>
<thead>
<tr>
<th>Instruction number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction /</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>Instruction / + 1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>Instruction / + 2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>Instruction / + 3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>Instruction / + 4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
</tbody>
</table>

*Figure A.1: Simple RISC pipeline. On each clock cycle, another instruction is fetched and begins its 5-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: IF = instruction fetch, ID = instruction decode, EX = execution, MEM = memory access, and WB = write back.*

### Classic RISC Five Stage Pipeline (Fig A.2)
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.
  \[ \text{CPI}(\text{unpipelined}) = P \]
  \[ \text{CPI}(\text{pipelined}) = 1 \]
  \[ \text{Speedup} = \frac{\text{IC} \times P \times \text{clock}}{\text{IC} \times 1 \times \text{clock}} = P \]

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.

\[
\begin{align*}
\text{CPI(unpipelined)} &= P \\
\text{CPI(pipelined)} &= 1 + \text{stall cycles per instruct.} \\
\text{Speedup} &= P / (1 + \text{stall cycles per instruct.})
\end{align*}
\]
Structural Hazards: insufficient HW resources

Dealing with structural hazards
- Add more hardware
- Stall the pipeline

Figure 4.5: A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load instruction effectively stalls an instruction-fetch cycle, causing the pipeline to stall—no instruction is issued on clock cycle 4 (which normally would initiate instruction i = 3). Because the instruction being stalled is stalled, all other instructions in the pipeline before the stalled instruction can proceed normally. The stall cycle will continue to pass through the pipeline, so that no instruction completes on clock cycle 3. Sometimes these pipeline diagrams are drawn with the stall occupying an entire horizontal row and instruction 3 being moved to the next row in either case, the effect is the same, since instruction i = 3 does not begin execution until cycle 5. We use the form above since it takes less space in the figure.
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)
Data hazard example #1 (Fig. A.6)

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