Extending simple pipeline to multiple pipes

- Single issue: in ID stage direct to one of several EX stages
- Common WB stage
- EX of various pipelines might take more than one cycle
- Latency of an EX unit = Number of cycles before its result can be forwarded = Number of stages – 1
- Not all EX need be pipelined
- IF EX is pipelined
  - A new instruction can be assigned to it every cycle (if no data dependency) or, maybe only after x cycles, with x depending on the function to be performed

Hazards in example multiple cycle pipeline

- **Structural:** Yes
  - Divide unit is not pipelined. In the example processor two Divides separated by less than 25 cycles will stall the pipe
  - Several writes might be “ready” at the same time and want to use WB stage at the same time (not possible if single write port)
- **RAW:** Yes
  - Essentially handled as in integer pipe (the dependent inst is stalled at the beginning of its EX stage) but with higher frequency of stalls. Also more forwarding needed.
- **WAW:** Yes (see later)
- **WAR** no since read is in the ID stage
- **Out of order completion:** Yes (see later)

Conflict in using the WB stage

- Several instructions might want to use the WB stage at the same time
  - E.g., A Mult issued at time t and an addd issued at time t + 3
- **Solution 1:** reserve the WB stage at ID stage (scheme already used in CRAY-1 built in 1976)
  - Keep track of WB stage usage in shift register
  - Reserve the right slot. If busy, stall for a cycle and repeat
  - Shift every clock cycle
- **Solution 2:** Stall before entering either Me or WB
  - Pre-easier detection than solution 1
  - Con: need to be able to trickle the stalls “backwards”.

Example on how to reserve the WB stage
(Solution 1 in previous slide)

<table>
<thead>
<tr>
<th>Time in ID stage</th>
<th>Operation</th>
<th>Shift register</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>multd</td>
<td>000 000 000</td>
</tr>
<tr>
<td>t + 4</td>
<td>int</td>
<td>001 000 000</td>
</tr>
<tr>
<td>t + 2</td>
<td>int</td>
<td>011 000 000</td>
</tr>
<tr>
<td>t + 3</td>
<td>adddl</td>
<td>110 00X 000</td>
</tr>
</tbody>
</table>

Note: multd and adddl want WB at time t + 9. adddl will be asked to stall one cycle
Instructions complete out of order (e.g., the two int terminate before the multd)

RAW: Example from the book

F4 < M
F0 < F4 * F6
F5 < F0 + F8
M < F2

In blue: data dependencies hazard
In red: structural hazard
In green: stall cycles
WAW Hazards

- Instruction $i$ writes f-p register $F_x$ at time $t$
- Instruction $i + k$ writes f-p register $F_x$ at time $t - m$
- But no instruction $i + 1, i + 2, i + k$ uses (reads) $F_x$ (otherwise there would be a stall)
- Only requirement is that $i + k$’s result be stored
  - Note: this situation should be rare (useless instruction $i$)
- Solutions:
  - Squash $i$: difficult to know where it is in the pipe
  - At ID stage check that result register is not a result register in all subsequent stages of other units. If it is, stall appropriate number of cycles.

Out-of-order completion

- Instruction $i$ finishes at time $t$
- Instruction $i + k$ finishes at time $t - m$
  - No hazard etc. (see previous example on integer completing before mult)
- What happens if instruction $i$ causes an exception at a time in $[t - m, t]$ and instruction $i + k$ writes in one of its own source operands (i.e., is not restartable)?

Exception handling

- Solutions (cf. book pp 194-196 for more details)
  - Do nothing (imprecise exceptions: bad with virtual memory)
  - Have a precise (by use of testing instructions) and an imprecise mode; effectively restricts concurrency of f-p operations
  - Buffer results in a “history file” (or a “future file”) until previous (in-order) instructions have completed; can be costly when there are large differences in latencies but a similar technique is used for OOO execution
  - Restrict concurrency of f-p operations and on an exception “simulate in software” the instructions in between the faulting and the finished one.
  - Flag early those operations that might result in an exception and stall accordingly