Lecture Topics

• Pipelining
  – Difficulties in implementing pipelines
    • Exceptions
    • ISA-related complications
    • Multi-Cycle operations

Reference:
• Appendix C: Sections C.4 and C.5
Handling Exceptions

- Must handle exceptions in instruction sequence order, not in the exception occurrence order (which is a pipelining artifact)
- Exception status vector carried along the pipeline with each instruction. Any exception caused by an instruction in a given stage cause an entry to be made in the status vector
- Control logic in each stage prevents data writes by the instruction if an exception has been posted to the exception status vector
- When instruction enters WB, the exception status vector is checked and the exceptions are handled in the instruction sequence order
Further Complications

- ISA can further complicate things ...
  - "commit": when an instruction is guaranteed to complete and writes can be permitted to occur or become permanent
  - MIPS instructions have single result (register or memory)
  - MIPS instructions commit on MEM/WB
- Some ISAs have instructions which change processor state during execution (e.g., IA-32 VAX)
  - Auto-increment addressing modes
  - String copy instruction
- Need the ability to "back out" state changes
- Need the ability to resume execution of instructions
  - Registers with intermediate results saved on exception
Further Complications

• “Odd” bits of state ...
  – Condition codes implicitly set by preceding instructions
  – May prevent effective scheduling of instruction in delay slot
  – When is the branch condition “fixed”?
    • Depends on which instructions are capable of setting condition codes
    • Delay branch condition evaluation until all prior instructions have had a chance to set condition codes
    • Treat condition codes as operands in RAW hazard detection
Further Complications

- Multi-cycle operations
  - VAX instructions can require widely different number of cycles to complete
  - VAX instructions can perform none to hundreds of memory references
  - A solution: IA-32 from 1995
    - “Microcoded” implementation: pipeline the microcode
Incorporating Floating Point Operations
Multicycle Operations

• Impractical to require all FP operations to complete in 1 cycle
  • Would slow down the clock speed
• Longer FP operations (e.g., FP multiply and divide) can take multiple cycles (EX stage repeated as many times as needed)
• Latency vs. Initiation Interval
  • **Latency**: Number of intervening cycles between an instruction that produces a result and an instruction that uses the result
  • **Initiation interval**: Number of cycles that must elapse between issuing two operations of a given type

<table>
<thead>
<tr>
<th>Functional unit</th>
<th>Latency</th>
<th>Initiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer ALU</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Data memory (integer and FP loads)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FP add</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FP multiply (also integer multiply)</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>FP divide (also integer divide)</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>
For multi-cycle operations, we add additional “EX” stages separated by pipeline registers
MIPS Pipelines with Multicycle Operations
Issues with Multicycle Operations

• Divide unit not fully pipelined => structural hazards can occur

• Instructions have varying execution times => number of register writes required in a cycle can be > 1

• Instructions no longer reach WB stage in order => Write-after-write (WAW) hazards are possible

• Instructions can complete in a different order than they were issued => can cause problems with exceptions

• Longer latency of operations => RAW hazards become more frequent
### Multi-cycle Operation: RAW Hazards

**Longer pipeline increases the stall penalty (both stall frequency and number of stall cycles)**

<table>
<thead>
<tr>
<th>Instruction</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>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F4,0(R2)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL.D F0,F4,F6</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td>stall</td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
<td>M6</td>
<td>M7</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F2,F0,F8</td>
<td>IF</td>
<td></td>
<td>stall</td>
<td>ID</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F2,0(R2)</td>
<td>IF</td>
<td></td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>ID</td>
<td></td>
<td>EX</td>
<td></td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>MEM</td>
</tr>
</tbody>
</table>
Multi-cycle Operation: Structural Hazards

3 instructions in WB in the same cycle
Solutions for Structural hazards

• **Solution # 1:** Detect structural hazards in the ID stage
  - Track the use of write port in the ID stage and stall the instruction for enough number of cycles to avoid structural hazard
  - Tracking can be done with a shift register
    - The register indicates when already-issued instructions will write to the register file
    - During ID, instruction examines bit corresponding to the time when its write would occur
      - If zero, set to 1 and proceed with execution
      - If one, stall one cycle (repeat)
      - Shift register left at each clock cycle

• **Advantage:** All hazard detection and stalling occurs in ID stage
Solutions for Structural hazards

- **Solution # 2**: Detect the hazard at beginning of stage with hazard
  - Stall a conflicting instruction when it tries to enter the MEM or WB stage
  - A simple heuristic is to prioritize the instruction with the longest latency

- **Advantage**: Does not require us to detect the conflict until the entrance of MEM (or WB) stage, where it is easy to see

- **Disadvantages**:
  - Complicates pipeline control (stalls can arise at multiple places)
  - Stalls may need to be propagated backward