Lecture Topics

• Pipelining
  – Structural Hazards
  – Pipeline Performance
    • Effects of Stalls and Penalties
  – Superscalar Processors
  – CISC Processors

• Reference:
  – Chapter 6: Sections 6.7, 6.8, 6.9 and 6.10 (Pages 209 – 214, 218 - 220)
Structural Hazards

• A **structural hazard** occurs when there are insufficient hardware resources to permit all actions to proceed correctly.

• In a pipelined processor, if two (or more) instructions require the use of a given resource in the same clock cycle, one instruction must be **stalled** to allow the other instruction to use the resource.

• **For example:** Consider a processor that has a single cache that supports only one access per cycle.
  - An instruction fetch operation may need to be stalled to allow a load instruction in its memory stage to access the cache.

• Structural hazards are prevented by providing additional hardware.
  - In the above example, using separate caches for instructions and data allows the Fetch and Memory stages to proceed simultaneously without stalling.
Pipeline Performance
Performance Evaluation

• Recall that the execution time (T) of a program is given by:
  \[ T = \frac{N \times CPI}{R} \]

• Instruction throughput (P) is the rate of instruction execution:
  \[ P = \frac{N}{T} = \frac{R}{CPI} \]

• For a non-pipelined processor with \( n \) stages, each instruction is completed in \( n \) cycles (assuming single-cycle memory access) and instructions are executed one after the other. Therefore throughput of a non-pipelined processor (\( P_{np} \)) is given by:
  \[ P_{np} = \frac{R}{n} \]
Performance Evaluation (cont.)

• For a pipelined processor, throughput depends on how frequently instructions can complete and come out of the pipeline.

• In the ideal case of no stalls, one instruction is completed every cycle. Therefore the ideal throughput with pipelining \( (P_p) \) is given by:

\[
P_p = \frac{R}{1} = R
\]

• Clock rate \( R \) is determined by the delay of the longest pipeline stage.
  • For example, if the longest pipeline stage requires a delay of 2 ns, then \( R = 1/2 \text{ ns} = 500 \text{ MHz} \) & \( P_p = 500 \text{ MIPS} \) (million instructions per second).
Performance Evaluation (cont.)

• An $n$-stage pipeline has the potential to increase throughput by $n$ times

• Two questions regarding pipeline performance:
  • How much of the potential increase in instruction throughput can actually be realized in the presence of stalls?
  • What is a good value for $n$ (number of pipeline stages)?
Impact of Pipeline Stalls on Throughput

- In the presence of stalls, there are cycles in which no instruction is completed in a pipelined processor.
- The impact of pipeline stalls on throughput depends on:
  - How frequently do stalls occur?
  - How much is the penalty for each stall in terms of number of cycles?
- To calculate the impact of stalls on performance, we need to first calculate the number of stall cycles per instruction (\(\delta\)).
- \(\delta\) is the product of stall frequency and stall penalty.
  - For example if 40% of the instructions stall for 1 cycle each, \(\delta = 0.4\).
- All the different sources of stalls are added to compute total \(\delta\).
- Throughput in the presence of stalls is given by:
  \[ P_p = \frac{R}{1 + \delta} \]
Impact of Data Hazards

- Recall that with operand forwarding, data hazards have no penalties except a 1-cycle penalty when a Load instruction is immediately followed by a dependent instruction.

- **Example:** Assume that load instructions constitute 25% of dynamic instruction count and assume that 40% of Load instructions are followed by a dependent instruction. What is the reduction in throughput compared to ideal case?

- **Solution:**
  \[
  \delta = \text{Stall frequency} \times \text{stall penalty} = (0.25 \times 0.4) \times 1 = 0.1
  \]
  \[
  P_p = \frac{R}{1 + \delta} = \frac{R}{1 + 0.1} = 0.91R
  \]
  Throughput is reduced by 9%
Impact of Branch Penalties

• Recall that branch prediction incurs a penalty whenever the branch predictor mispredicts the outcome of a branch instruction.

• Branch penalty depends on when the actual branch decision and target address are computed:
  – If the branch computation is done in Decode stage, penalty is one cycle.

• **Example:** Assume that branches constitute 20% of dynamic instruction count and the average branch prediction accuracy is 90%. What is the impact of branch misprediction on throughput?

• **Solution:**
  
  \[ \delta = \text{Stall frequency} \times \text{stall penalty} = (0.2 \times (1-0.9)) \times 1 = 0.02 \]

  \[ P_p = \frac{R}{1+\delta} = \frac{R}{1 + 0.02} = 0.98R \]

  Throughput is reduced by 2%.
Impact of Cache Misses

• Cache misses cause pipeline stalls in two scenarios:
  – Cache miss during instruction fetch operation for any instruction
  – Cache miss during data read/write operation for a Load/Store instruction
• These two components of pipeline stall are additive

• Example: Assume that 5% of all instruction fetch operations incur a cache miss, 30% of all instructions executed are Load/Store, and 10% of data accesses incur a cache miss. Assume that the penalty to access the main memory for a cache miss is 10 cycles? What is the impact of cache misses on throughput?
• Solution:
\[ \delta = \text{Stall frequency} \times \text{stall penalty} = (0.05 + (0.3 \times 0.1)) \times 10 = 0.8 \]
\[ P_p = R / (1 + \delta) = R / (1 + 0.8) = 0.56R \]
Throughput is reduced by 44%
Number of Pipeline Stages

• Since an $n$-stage pipeline has the potential to increase the throughput by $n$ times, how about we use a 10,000-stage pipeline?
• As the number of stages increase, the number of pipeline stalls also increase because:
  – More instructions being overlapped => more potential data dependencies => higher stall probability
  – Branch penalty may be larger than one cycle, if a longer pipeline moves the branch decision to a later stage
• There is also an inherent limit to pipelining, because it is impractical to place pipeline registers at arbitrarily fine granularities
• Some recent processor implementations have used ~ 20 pipeline stages, enabling clock rates of several GHz
• However further increases are unlikely, due to power/area constraints and limited performance benefits
Superscalar Processors
Overview

- The max. throughput of a pipelined processor is 1 instr. per cycle
- If we equip the processor with multiple execution units, each capable of executing an independent instruction, several instructions start execution in the same clock cycle – *multiple-issue*
- Such processors are capable of achieving a throughput of more than one instruction per cycle – *superscalar processors*
  - Most modern high performance processors are superscalar processors
- To enable *multiple-issue* execution, a pipelined processor needs modification to all the pipeline stages
  - Multiple instruction fetched per cycle and placed in an instruction queue
  - Multiple instruction decoded in each cycle
  - More register ports to read register operands for multiple instrs. in parallel
Multiple-Issue Execution

• To enable *multiple-issue* execution, a pipelined processor needs modification to all the pipeline stages
  
  – **Fetch stage**: Multiple instruction fetched from the instruction cache per cycle and placed in an instruction queue
  
  – **Decode stage**: Multiple instruction decoded in each cycle and sent to the appropriate execution units. More register read ports are needed to enable source register reads for multiple instructions
  
  – **Compute Stage**: Multiple execution units (e.g., separate units for arithmetic instructions and load/store instructions)
  
  – **Writeback Stage**: Multiple instructions write their results to register file in parallel => need more register write ports
### Example of Superscalar Execution

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add R2, R3, #100</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Load R5, 16(R6)</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subtract R7, R8, R9</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Store R10, 24(R11)</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Throughput = 2 instructions per clock cycle
Stalls in Superscalar Processors

• Even though superscalar processors use multiple execution units, pipeline hazards can still hurt their throughput.

• Examples:
  • **Structural Hazard:** The execution units may not necessarily match the available instructions. For example, two arithmetic instruction may be ready for issue, but there is only arithmetic unit and one load/store unit
  • **Data Hazards:** The instructions that are ready for issue may have a data dependency. The second instruction needs to stall, even if there is an idle execution unit (operand forwarding does not solve this problem)
  • **Instruction Hazards:** All the instructions fetched and executed after a mispredicted branch need to be discarded and their results annulled
Example of Structural Hazard

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add R2, R3, #100</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subtract R7, R8, R9</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Load R5, 16(R6)</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Store R10, 24(R11)</strong></td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stall due to unavailability of arithmetic unit
CISC Processors
Pipelining in CISC Processors

- CISC-style instructions complicate pipelining because of their irregular encoding formats complex behaviors

- Some of the common reasons why CISC pipelining is complex:
  - Instructions have variable sizes => complicates fetching and decoding
  - Instructions have multiple memory operands => some instructions require more time while others require less time in Memory stage; complicates the pipeline stalling logic
  - Instructions use complex addressing modes which sometimes cause both the source and destination registers to change values => hard to track instruction dependences and complicates operand forwarding

Complex pipelining in CISC processors was one of the key reasons for developing the RISC approach
Pipelining in Intel Processors

- Intel processors achieve high performance with superscalar execution and deep pipelines
  - Intel Core-i7 uses issue width of 4 instructions and a 14-stage pipeline

- Even though Intel processors use CISC instruction sets, the execution hardware is designed like a RISC processor to reduce hardware complexity
  - CISC-style instructions are dynamically converted into simpler RISC-style micro-operations (*micro-ops*)
  - Micro-ops are issued to the execution units to complete the tasks specified by the original CISC instructions
  - This approach preserves (backward) code compatibility while enabling the advantages of simpler RISC-style pipelining