**Branch Prediction**

COMP375 Computer Architecture and Organization

---

**Hazards**

- A hazard is a situation that reduces the processors ability to pipeline instructions.
- **Resource** – When different instructions want to use the same CPU resource.
- **Data** – When the data used in an instruction is modified by the previous instruction.
- **Control** – When a jump is taken or anything changes the sequential flow.

---

**Control Hazards**

- A jump or function call changes the sequential execution of instructions.
- The pipelined instruction fetch stage continually fetches sequential instructions.
- When a jump occurs, the previously fetched instructions should not be executed.
- Instructions in the pipe may have to be discarded before the write back stage.
Unconditional Jump
• The CPU can detect that an instruction is an unconditional jump during the instruction decode state.
• The next instruction that has already been fetched must be discarded.

Conditional Jump
• The processor cannot determine if the jump will be taken until the execution stage.
• Three instructions in the pipe must be discarded.

Queues
• The problem of operand and instruction fetch conflicts can be reduced by pre-fetching several instructions.
• The queue will have to be flushed if there is a branch instruction.

Delayed Branch
• When a jump or function call occurs, the processing that has been done on the following instructions is wasted.
• To avoid wasting some of the processing, some machines always execute the instruction immediately after a jump.
• The compiler must rearrange the instructions so a useful instruction occurs after the jump. A NOOP can be inserted if no useful instruction is available.
Delayed Branch Example

\[
\begin{align*}
X &= 7; \\
\text{if } (Y == 0) & \ Z = 5; \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Original</th>
<th>Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load R1, 7</td>
<td>Load R1, 7</td>
</tr>
<tr>
<td>Store R1, X</td>
<td>Load R2, Y</td>
</tr>
<tr>
<td>Load R2, Y</td>
<td>BZ R2,there</td>
</tr>
<tr>
<td>BZ R2,there</td>
<td>Store R1, X</td>
</tr>
<tr>
<td>Load R3, 5</td>
<td>Load R3, 5</td>
</tr>
<tr>
<td>Store R3, Z</td>
<td>Store R3, Z</td>
</tr>
</tbody>
</table>

Branch Prediction

- The processor tries to guess if the jump will be taken.
- Correct prediction allows the processor to start fetching the proper instructions immediately.
- This is particularly useful when the processor has already executed these instructions and has previously fetched the branch location.

Types of Branching

<table>
<thead>
<tr>
<th>Type of Instruction</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconditional jump</td>
<td>28%</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>42%</td>
</tr>
<tr>
<td>Loop</td>
<td>10%</td>
</tr>
<tr>
<td>Function calls &amp; returns</td>
<td>20%</td>
</tr>
</tbody>
</table>

Static Branch Prediction

<table>
<thead>
<tr>
<th>Type of Instruction</th>
<th>frequency</th>
<th>Branch Taken?</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconditional jump</td>
<td>28%</td>
<td>Yes</td>
<td>28%</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>42%</td>
<td>No</td>
<td>25%</td>
</tr>
<tr>
<td>Loop</td>
<td>10%</td>
<td>Yes</td>
<td>9%</td>
</tr>
<tr>
<td>Function calls &amp; returns</td>
<td>20%</td>
<td>Yes</td>
<td>20%</td>
</tr>
</tbody>
</table>

Overall prediction accuracy of 82%
Dynamic Branch Prediction

- The prediction of a conditional branch is based on the previous execution of this instruction.
- The processor keeps a table of previous execution results to predict if the branch will be taken.

Execution Trace Cache

- It caches decoded instructions (microcode steps), thus removing the latency associated with the instruction fetch and decode.
- Instructions are stored in the order of likely execution. The target of a branch is kept immediately after the branch.

Instruction Set Considerations

- Architectures with many registers make it easier to avoid data hazards.
- The use of condition codes set by many instructions make it difficult to reorder instructions.
- Complex addressing modes can make the operand fetch stage take much longer.