Pipelining Wrapup

- All modern processors all use pipelining to improve performance. Additional performance gains are achieved through:
  - Superscalar execution (multiple functional units)
  - Superpipelining (deeper pipelines)
  - Dynamic scheduling (finding the “best” sequence of instructions to execute)
  - Branch prediction: Deep pipelines incur larger branch penalties
  - Speculative execution (execute instructions beyond branches)

Organization

- Modern processors look more like:

  - Instruction queue: holds a pile of to-be-worked on instructions; these may come from different paths of a branch...
  - Instruction scheduler: attempts to find the best set of instructions to send to the functional units. It may also “rename” registers
  - Functional units: integer ALU, floating point ALU, branch computation, load/store unit
  - Branch predictor: maintains a table of past branches and history
  - Instructions need to wait until all of their operands are available. When they are, they are “executed”
  - Results are passed to the commit unit.
  - The commit unit is responsible for committing results in the right order (or maybe not at all, if a branch was or was not taken!)

What do the units do?

- Instruction queue: holds a pile of to-be-worked on instructions; these may come from different paths of a branch...
- Instruction scheduler: attempts to find the best set of instructions to send to the functional units. It may also “rename” registers
- Functional units: integer ALU, floating point ALU, branch computation, load/store unit
- Branch predictor: maintains a table of past branches and history
- Instructions need to wait until all of their operands are available. When they are, they are “executed”
- Results are passed to the commit unit.
- The commit unit is responsible for committing results in the right order (or maybe not at all, if a branch was or was not taken!)

Dealing with branches.

- Below we’ll assume:
  - \[ B \] = % of instructions that are branches
  - \[ D \] = number of cycles of branch “delay/penalty”
- Always stalling:
  \[ CPI = (1-B) + (B \times (1+D)) \]
- Delayed branches (C is the % of the delay slots the compiler can fill):
  \[ CPI = (1-B) + (B \times (1+(1-C)*D)) \]
- Predict not taken (T is the % of branches taken)
  \[ CPI = (1-B) + ((1-T) \times B) + (T \times B \times (1+D)) \]
- Predict taken (T is the % of branches taken)
  \[ CPI = (1-B) + (T \times B) + ((1-T) \times B \times (1+D)) \]
- Branch prediction (P is the % of the time our prediction is wrong):
  \[ CPI = (1-B) + ((1-P) \times B) + (P \times B \times (1+D)) \]
Comparing the schemes

- The below table assumes:
  - The program is 15% branches, all other instructions are CPI = 1
  - Penalty (D) is 2 or 4 cycles
  - The compiler can fill 60% of the 2 delay slots, 30% of 4
  - 70% of branches are taken
  - Branch prediction is 90% accurate

<table>
<thead>
<tr>
<th>Scheme</th>
<th>CPI (when D=2)</th>
<th>CPI (when D=4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always stall</td>
<td>1.3</td>
<td>1.6</td>
</tr>
<tr>
<td>Delayed branches</td>
<td>1.12</td>
<td>1.42</td>
</tr>
<tr>
<td>Predict not taken</td>
<td>1.21</td>
<td>1.42</td>
</tr>
<tr>
<td>Predict taken</td>
<td>1.09</td>
<td>1.18</td>
</tr>
<tr>
<td>Full prediction</td>
<td>1.03</td>
<td>1.06</td>
</tr>
</tbody>
</table>

Branch Prediction

- Keep a table mapping branches to history (did we take the branch the last time?)
- Suppose we have a 256 entry table.
- If we find a branch at address N, we locate its entry like this:
  \[ \text{index} = N \% 256 \]
  (or easier: \( \text{index} = N \& 0x000000FF \))
- The simplest predictor just keeps one bit of state: taken/not-taken
- Most predictors use a two bit scheme:

The AMD Athlon

- 35+ million transistors; clock speeds in excess of 1 GHz
- The “front end” of the processor translates the incoming CISC instructions (up to 3 x86 instructions) into RISC86/MacroOp instructions
- The RISC86 instructions are passed to the instruction control unit, which can manage up to 72 instructions (= a maximum of 36 x86 instructions) at a time. The instructions are scheduled here, and the unit may issue up to 9 instructions per cycle.
- There are 9 pipelines: 3 integer, 3 address calculation, 3 floating-point/MMX instructions. The integer pipelines have 10 stages.
- Finally, the instruction control unit handles committing completed instructions (up to 9 per cycle)
- Branch prediction: 2048 entry history table & branch target table