Lecture 21: BPred, OOO, Memory Hierarchy

• Today’s topics:
  - Branch Predictors
  - Out-of-order execution
  - Cache intro
Pipeline with Branch Predictor

IF (br) → PC → Branch Predictor

Reg Read Compare Br-target
Bimodal Predictor

14 bits

Branch PC

Table of 16K entries of 2-bit saturating counters
2-Bit Prediction

• For each branch, maintain a 2-bit saturating counter:
  if the branch is taken: counter = min(3,counter+1)
  if the branch is not taken: counter = max(0,counter-1)
  ... sound familiar?

• If (counter >= 2), predict taken, else predict not taken

• The counter attempts to capture the common case for each branch

Indexing functions
Multiple branch predictors
History, trade-offs
Multicycle Instructions

- Multiple parallel pipelines – each pipeline can have a different number of stages

- Instructions can now complete out of order – must make sure that writes to a register happen in the correct order
An Out-of-Order Processor Implementation

- **Branch prediction and instr fetch**
  - R1 ← R1+R2
  - R2 ← R1+R3
  - BEQZ R2
  - R3 ← R1+R2
  - R1 ← R3+R2

- **Instr Fetch Queue**

- **Decode & Rename**

- **Reorder Buffer (ROB)**
  - Instr 1
  - Instr 2
  - Instr 3
  - Instr 4
  - Instr 5
  - Instr 6
  - T1
  - T2
  - T3
  - T4
  - T5
  - T6

- **Register File R1-R32**

- **Issue Queue (IQ)**
  - T1 ← R1+R2
  - T2 ← T1+R3
  - BEQZ T2
  - T4 ← T1+T2
  - T5 ← T4+T2

Results written to ROB and tags broadcast to IQ
## Example Code

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Completion times with in-order</th>
<th>Completion times with ooo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>ADD R4, R1, R2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>LW R5, 8(R4)</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>ADD R7, R6, R5</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>ADD R8, R7, R5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>LW R9, 16(R4)</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>ADD R10, R6, R9</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>ADD R11, R10, R9</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>
Cache Hierarchies

• Data and instructions are stored on DRAM chips – DRAM is a technology that has high bit density, but relatively poor latency – an access to data in memory can take as many as 300 cycles today!

• Hence, some data is stored on the processor in a structure called the cache – caches employ SRAM technology, which is faster, but has lower bit density

• Internet browsers also cache web pages – same concept
Memory Hierarchy

- As you go further, capacity and latency increase

**Registers**
- 1KB
- 1 cycle

**L1 data or instruction Cache**
- 32KB
- 2 cycles

**L2 cache**
- 2MB
- 15 cycles

**Memory**
- 16GB
- 300 cycles

**Disk**
- 1 TB
- 10M cycles
Locality

• Why do caches work?
  ▪ Temporal locality: if you used some data recently, you will likely use it again
  ▪ Spatial locality: if you used some data recently, you will likely access its neighbors

• No hierarchy: average access time for data = 300 cycles

• 32KB 1-cycle L1 cache that has a hit rate of 95%:
  average access time = 0.95 x 1 + 0.05 x (301) = 16 cycles