Lecture 19: Branches, OOO

• Today’s topics:
  - Instruction scheduling
  - Branch prediction
  - Out-of-order execution
Control Hazards

• Simple techniques to handle control hazard stalls:
  ➢ for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  ➢ assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  ➢ fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
  ➢ make a smarter guess and fetch instructions from the expected target
Branch Delay Slots

a. From before:

\[
\text{add $s1, s2, s3} \\
\text{if $s2 = 0 \text{ then}} \\
\text{Delay slot}
\]

Becomes:

\[
\text{if $s2 = 0 \text{ then}} \\
\text{add $s1, s2, s3}
\]

b. From target:

\[
\text{sub $t4, t5, t6} \\
\text{...} \\
\text{add $s1, s2, s3} \\
\text{if $s1 = 0 \text{ then}} \\
\text{Delay slot}
\]

Becomes:

\[
\text{add $s1, s2, s3} \\
\text{if $s1 = 0 \text{ then}} \\
\text{sub $t4, t5, t6}
\]
Pipeline without Branch Predictor

IF (br)

PC

Reg Read
Compare Br-target

PC + 4
Pipeline with Branch Predictor

PC → IF (br) → Reg Read Compare Br-target → Branch Predictor → IF (br)
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
Slowdowns from Stalls

• Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) → speedup = increase in clock speed = num pipeline stages

• With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

• Total cycles = number of instructions + stall cycles
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

Instr Fetch Queue

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2

Decode & Rename

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6

T1
T2
T3
T4
T5
T6

Reorder Buffer (ROB)

Register File
R1-R32

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6

T1 ← R1+R2
T2 ← T1+R3
BEQZ T2
T4 ← T1+T2
T5 ← T4+T2

Issue Queue (IQ)

ALU
ALU
ALU

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>In-Order Time</th>
<th>OOO Time</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>
Title

• Bullet