Lecture 20: Branches, OOO

• Today’s topics:
  - Branch prediction
  - Out-of-order execution
  - (Also see class notes on pipelining, hazards, etc.)
Pipelining Example (Recap)

• Unpipelined design: the entire circuit takes 10ns to finish
  Cycle time = 10ns; Clock speed = 1/10ns = 100 MHz
  CPI = 1 (assuming no stalls)
  Throughput in instructions per second =
  \#cycles in a second x instructions-per-cycle =
  100 M \times 1 = 100 M \text{ instrs per second} = 0.1 \text{ BIPS (billion instrs per sec)}

• 5-stage pipeline: under ideal conditions, each stage takes 2ns
  Cycle time = 2ns; Clock speed = 1/2ns = 500 MHz (5x higher)
  CPI = 1 (continuing to assume no stalls)
  Throughput = \# cycles in a second x instrs-per-cycle
  = 500 M \times 1 = 500 \text{ MIPS} = 0.5 \text{ BIPS}
  Under ideal conditions, a 5-stage pipeline gives a 5x speedup.
Control Hazards

• Simple techniques to handle control hazard stalls:
  ➢ for every branch, introduce a stall cycle (note: every 6\textsuperscript{th} 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

```
add $s1, $s2, $s3
if $s2 = 0 then
    Delay slot
```

Becomes

```
if $s2 = 0 then
    add $s1, $s2, $s3
```

b. From target

```
sub $t4, $t5, $t6
... 
add $s1, $s2, $s3
if $s1 = 0 then
    Delay slot
```

Becomes

```
add $s1, $s2, $s3
if $s1 = 0 then
    sub $t4, $t5, $t6
```
Pipeline without Branch Predictor

IF (br)

PC + 4

PC

Reg Read Compare Br-target
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
Slowdowns from Stalls

• Perfect pipelining with no hazards $\rightarrow$ an instruction completes every cycle (total cycles $\sim$ num instructions) $\rightarrow$ 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**

- **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

- **ALU ALU ALU**

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

<table>
<thead>
<tr>
<th>Completion times</th>
<th>with in-order</th>
<th>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>