Branch Prediction schemes

- Two implementations of branches in the 5-stage pipelined datapath:

1. Branch target and decision resolved in the EX stage:

<table>
<thead>
<tr>
<th></th>
<th>Right prediction 😊</th>
<th>Wrong prediction 😞</th>
</tr>
</thead>
<tbody>
<tr>
<td>No prediction</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Predict not-taken</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Predict taken</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

2. Branch target and decision resolved in the ID stage

<table>
<thead>
<tr>
<th></th>
<th>Right prediction 😊</th>
<th>Wrong prediction 😞</th>
</tr>
</thead>
<tbody>
<tr>
<td>No prediction</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Predict not-taken</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Predict taken</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Predict “taken” only makes sense if target resolved before decision
Original pipelined datapath: everything in EX

Instruction memory

- Read Instruction address [31-0]
- Instruction memory

Control

ID/EX

- Read register 1
- Read data 1
- Read register 2
- Read data 2
- Write register
- Write data

Registers

- Instr [15 - 0]
- Sign extend
- Instr [15 - 11]
- Instr [20 - 16]

RegWrite

Shift left 2

ALU

- ALUOp
- Zero
- ALUSrc

RegDst

MemWrite

MemRead

Data memory

- Address
- Write data
- Read data

MemToReg

EX/MEM

- MemWrite
- MemRead

EX

MEM/WB

- WB
- M
- EX

PCSsrc

Add

PC

IF/ID

- Add
- 4

- 1
- 0

P C
The branch target resolution (i.e., “PC + 4 + 4×offset” computation) is done in the ID stage, and the branch is resolved in the EX stage.

<table>
<thead>
<tr>
<th></th>
<th>Right prediction 😊</th>
<th>Wrong prediction 😞</th>
</tr>
</thead>
<tbody>
<tr>
<td>No prediction</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Predict not-taken</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Predict taken</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Claim”: Predict not-taken is the best strategy, since its best-case behavior is the best, and its worst-case behavior is no worse than others.

No! It depends on whether branches are actually taken or not.

A common heuristic: Forward-branches predicted not taken, backward branches predicted taken (good for loops).
Aside: Improvement when target in ID, decision in EX

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next</td>
<td></td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Either next+4 or target+4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

flush one of these
Advanced Branch Prediction

- Most modern processors maintain run-time statistics on branch behavior to try and predict the branch outcome.

- A common strategy: 2-bit counter

- More strategies:
  - Remember \( n \)-bit history, and for each history record what happened
  - Use “neural predictors” to avoid exponential storage (but slow!)
  - Instead of “local” history, record “global” history to detect branch-dependencies
Branch Prediction: Algorithmic Issues

- A great deal of research has gone into sorting algorithms
  - Software libraries implement highly optimized sorting routines

- Comparison-based algorithms often have the instruction sequence:
  
  ```
  compare a[i], a[j]
  branch based on comparison
  ```

- Theoretical lower bound: $\Omega(n \log n)$ comparisons needed to sort $n$ objects
  - Any algorithm that makes $dn \log n$ comparisons makes $\Omega(n \log_d n)$ branch mispredictions

- Cryptographic issues (side-channel attack into RSA):
  
  ```
  encrypt(M, e) = (M^e) \mod N
  ```

- The bit-pattern of $e$ dictates how long encryption takes
  - Snooper can analyze encryption times for chosen $M$ values
  - Snooper can indirectly analyze prediction history table