CIS 429/529

Branch Prediction

Prof. Michel A. Kinsky
Phases of Instruction Execution

- **Fetch**: Instruction bits retrieved from cache.

- **Decode**: Instructions placed in appropriate issue (aka “dispatch”) stage buffer.

- **Execute**: Instructions and operands sent to execution units. When execution completes, all results and exception flags are available.

- **Commit**: Instruction irrevocably updates architectural state (aka “graduation” or “completion”).
Control Flow Penalty

- Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution!
- How much work is lost if pipeline doesn’t follow correct instruction flow?

![Diagram showing the pipeline stages: PC, Fetch, Decode, Execute, Commit, Arch. State, Branch executed, Next fetch started]
**Average Branches Distance**

- **Average Run-Length between Branches**

  - Average dynamic instruction mix from SPEC92:

    |                | SPECInt92 | SPECFp92 |
    |----------------|-----------|----------|
    | ALU            | 39 %      | 13 %     |
    | FPU Add        |           | 20 %     |
    | FPU Mult       |           | 13 %     |
    | load           | 26 %      | 23 %     |
    | store          | 9 %       | 9 %      |
    | branch         | 16 %      | 8 %      |
    | other          | 10 %      | 12 %     |

SPECInt92: compress, eqntott, espresso, gcc, li
SPECFp92: doduc, ear, hydro2d, mdijdp2, su2cor
MIPS Branches and Jumps

- Each instruction fetch depends on one or two pieces of information from the preceding instruction:
  1. Is the preceding instruction a taken branch?
  2. If so, what is the target address?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Taken known?</th>
<th>Target known?</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>After Inst. Decode</td>
<td>After Inst. Decode</td>
</tr>
<tr>
<td>JR</td>
<td>After Inst. Decode</td>
<td>After Reg Fetch</td>
</tr>
<tr>
<td>BEQZ/BNEZ</td>
<td>After Reg Fetch</td>
<td>After Inst. Decode</td>
</tr>
</tbody>
</table>
Branch Penalties in Modern Pipelines

- UltraSPARC-III instruction fetch pipeline stages (in-order issue, 4-way superscalar, 750MHz, 2000)

- PC Generation/Mux
- Instruction Fetch Stage 1
- Instruction Fetch Stage 2
- Branch Address Calc/Begin Decode
- Complete Decode
- Steer Instructions to Functional units
- Register File Read
- Integer Execute
- Remainder of execute pipeline (+ another 6 stages)

Branch Target Address Known

Branch Direction & Jump Register Target Known
Reducing Control Flow Penalty

• Software solutions
  ‣ Eliminate branches - loop unrolling increases the run length
  ‣ Reduce resolution time - instruction scheduling compute the branch condition as early as possible (of limited value)

• Hardware solutions
  ‣ Find something else to do - delay slots replaces pipeline bubbles with useful work (requires software cooperation)
  ‣ Speculate - branch prediction speculative execution of instructions beyond the branch
Branch Prediction

• Motivation:
  ‣ Branch penalties limit performance of deeply pipelined processors
  ‣ Modern branch predictors have high accuracy (>95%) and can reduce branch penalties significantly
Branch Prediction

• Required hardware support:
  ‣ Prediction structures: branch history tables, branch target buffers, etc.
  ‣ Mispredict recovery mechanisms:
    • Keep result computation separate from commit
    • Kill instructions following branch in pipeline
    • Restore state to state following branch
Static Branch Prediction

- Overall probability a branch is taken is ~60-70% but:

  ![Diagram showing branch prediction]

  - **backward** 90%
  - **forward** 50%

- ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110
  - bne0 (preferred taken)  beq0 (not taken)

- ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64
  - Typically reported as ~80% accurate
Dynamic Prediction

Prediction as a feedback control process

Operations
• Predict
• Update
Predictor Primitive

- Indexed table holding values
- Operations
  - Predict
  - Update
- Algebraic notation
  - Prediction = P[Width, Depth] (Index; Update)
Dynamic Branch Prediction

• Learning based on past behavior

• Temporal correlation
  ‣ The way a branch resolves may be a good predictor of the way it will resolve at the next execution

• Spatial correlation
  ‣ Several branches may resolve in a highly correlated manner (a preferred path of execution)
One-bit Predictor

- $A_{21064}(PC; T) = P[1, 2K](PC; T)$
- What happens on loop branches?
  - At best, mispredicts twice for every use of loop.
Branch Prediction Bits

- Assume 2 BP bits per instruction
- Use saturating counter

<table>
<thead>
<tr>
<th>On taken</th>
<th>On taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Two-bit Predictor

- Counter\([W,D](I; T) = P[W, D]\)
  - \((I; \text{if } T \text{ then } P+1 \text{ else } P-1)\)
- \(\text{A21164}(PC; T) = \text{MSB}(\text{Counter}[2, 2K](PC; T))\)
Branch History Table

4K-entry BHT, 2 bits/entry, ~80-90% correct predictions
Exploiting Spatial Correlation

- If first condition false, second condition also false
- History register, H, records the direction of the last N branches executed by the processor

```plaintext
if (x[i] < 7) then
  y += 1;
if (x[i] < 5) then
  c -= 4;
```
History Register

- History(PC, T) = P(PC; P || T)
GHist(;T) = Counter(History(0, T); T)
Ind-Ghist(PC;T) = Counter(PC || Hist(GHist(;T);T))

- Can we take advantage of a pattern at a particular PC?