Branch Prediction
Branch Penalty

Example: Comparing perfect branch prediction to 90%, 95%, 99% prediction accuracy, and to no branch prediction

- Processor has a 20-stage 6-wide pipeline, incorrectly predicted branch leads to pipeline flush
- Program can have an average of 4 instructions retire per cycle, has 100,000 conditional branches out of 1 million instructions
- Perfect BP: IPC = 1,000,000/250,000 = 4.00
- 90% BP accuracy: 1/10 branches incorrectly predicted
  - IPC = 1,000,000/(250,000 + 0.1x100,000x20) = 2.22 (80% slower)
- 95% BP accuracy: 1/20 branches incorrectly predicted
  - IPC = 1,000,000/(250,000 + 0.05x100,000x20) = 2.85 (40% slower)
- 99% BP accuracy: 1/100 branches incorrectly predicted
  - IPC = 1,000,000/(250,000 + 0.01x100,000x20) = 3.70 (8% slower)
- No BP: Fetch stalled until branch is resolved (5 pipeline stages)
  - IPC = 1,000,000/(250,000 + 100,000x5) = 1.33 (300% slower – 3x)
Reducing Branch Costs with Dynamic Hardware Prediction

Branch prediction basics:

- We need to predict conditional branch outcome to select the address for next instruction fetch
  - PC + 4
  - Or branch *target* address
- Also we need to quickly determine the branch *target* address
  - Direct branches
  - Register indirect branches
  - Returns
Predicting Conditional Branch Outcomes

- Simplest dynamic branch prediction scheme uses a *branch-prediction buffer* or *branch history table*
  - Small memory indexed by the lower portion of the branch address
  - Stores previous branch outcomes to predict next outcome
  - Memory is not *tagged*
  - Prediction may have been put in the entry by a different branch (Aliasing)
Predicting Conditional Branch Outcomes

- 1-bit prediction buffer stores the last executed branch outcome, and uses it to predict the next outcome
  - If bit = 1, branch is predicted taken
  - If bit = 0, branch is predicted not-taken

- A simple 1-bit scheme may not perform well
  - Example: Below is a series of branch outcomes and corresponding predictions:
    
    outcomes: 1111011110111101
    predictions: 111101111011110
    mispredictions: 111101111011110
Predicting Conditional Branch Outcomes

- 2-bit saturating counter often used
  - Branch taken $\Rightarrow$ increment state
    - Max state “11” stays at “11” when incremented
  - Branch not-taken $\Rightarrow$ decrement state
    - Min state “00” stays at “00” when decremented
  - “11” and “10” are predict taken states
  - “00” and “01” are predict not-taken states
2-bit Saturating Counter State Machine

Predict taken “11”

Predict not taken “00”

Predict taken “10”

Predict not taken “01”

T

T

NT

NT
Assuming initial state to be “11”, i.e., 3, branch outcomes and corresponding predictions now look as follows:

<table>
<thead>
<tr>
<th>outcomes</th>
<th>1111011110111101</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>3333233332333232</td>
</tr>
<tr>
<td>predictions</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>mismatches</td>
<td>1111111111111111</td>
</tr>
</tbody>
</table>
Correlating Branch Predictors

- 2-bit prediction schemes use the recent behavior of a single branch to predict the future behavior of that branch.
- Behavior of longer sequence of branch execution history often provides more accurate prediction outcome.
- Behavior of other branches rather than just the branch we are trying to predict is sometimes important:
  - Because outcomes of different branches often correlate.
  - Global branch history.
- For some branches, prior history execution of the branch is important:
  - Because of loops.
  - Local branch history.
Correlating Branch Predictors: Code Example

if (aa == 2)
    aa = 0;
if (bb == 2)
    bb = 0;
if (aa != bb) {
    if (aa == 2)
        BNEZ R3,L1
    DADD R1,R0,R0
    L1:  DSUBUI R3,R2,#2
        BNEZ R3,L2
        DADD R2,R0,R0
    L2:  DSUBU R3,R1,R2
         BEQZ R3,L3
Correlating Branch Predictor with 2-bit Global History Register

Branch address

2-bit per-branch predictors

XX

XX prediction

2-bit global branch history

© 2003 Elsevier Science (USA). All rights reserved.
Two-Level Adaptive Branch Prediction

- Two main structures
  - Branch History Register (BHR) or Branch History Table (BHT)
  - Pattern History Table (PHT)
  - Basic structure of the branch predictor: Yeh&Patt Figure 1
  - Updating predictions using automatons: Yeh&Patt Figure 2

- Three different flavors (Yeh&Patt Figure 3)
  - Global History Register and Global Pattern History Table (GAg)
  - Per-address Branch History Table and Global Pattern History Table (PAg)
  - Per-address Branch History Table and Per-address Pattern History Tables (PAp)
Two-Level Adaptive Branch Prediction: Discussion

- **Cost-effectiveness of three flavors**
  - GAg has too much branch interference, needs long history
  - PAp needs lots of space for Per-address PHT
  - PAg is the most cost-effective

- **Context switch**
  - GAg almost unaffected
  - PAg, PAp degraded
  - Pros and cons for saving branch history on a context switch?
Other Branch Prediction Strategies

- McFarling’s Paper:
  - Bimodal Predictor: Figure 1
  - PAg and GAg: Figure 4 and 6
  - Global Predictor with Index Selection: Figure 8
  - Global History with Index Sharing (GShare): Figure 10

- Using perceptrons instead of 2-bit saturating counters
  - Jimenez&Lin’s paper (skim, not in exam)
  - Provides higher prediction accuracy
Adaptively Combining Branch Predictors

- Some branches are predicted more accurately with *global* predictors
- Other branches are predicted better with *local* predictors
- It is possible to combine both types of predictors, and dynamically select the right predict for the right branch
- The selector is yet another predictor with 2-bit state machine per entry
Branch Target Buffer (BTB)

- A cache that stores branch targets
- Accessed by the address of the instruction currently fetched
- Allows branch target to be read in the IF stage
  - When a branch is predicted taken, the fetch of the instruction at the branch target address can proceed immediately in the next cycle
  - Stall cycles that would have been needed to wait for the decoding of the branch and the computation of the target are saved
Branch Target Buffer

- PC of instruction to fetch
- Look up
- Predicted PC
- Number of entries in branch-target buffer

Yes: then instruction is branch and predicted PC should be used as the next PC

No: instruction is not predicted to be branch; proceed normally

Branch predicted taken or not taken

© 2003 Elsevier Science (USA). All rights reserved.
Predicting Return Address Using Return Address Stack (RAS)

- Indirect branches have multiple potential targets, since address comes from a register, which can have many possible values
- Branch target buffers could be used for indirect branch target prediction
  - However, many mispredictions can happen because the BTB can store only one target per branch
- Most indirect branches come from return instructions
Return Address Stack

- A small address buffer organized as a stack
- When a Call is encountered, the Return address (which is Call address + 4) is pushed onto the RAS
- When a Return instruction is encountered, the address from the top of the RAS is popped and used as the target
Reading Assignment

Thursday
- Simplescalar technical report (Read)
- Tutorial (Skim)
- No reviews due

Tuesday
- Daniel Jimenez and Calvin Lin, “Dynamic Branch Prediction with Perceptrons,” HPCA 2001 (Skim)