Branch Prediction

PSU ECE587

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 is a branch-prediction buffer or branch history table.

- It is a 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.

```
P C
1K entries prediction buffer
```

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 has a performance shortcoming:
- A series of branch outcomes and corresponding predictions:
  - outcomes: 1111011110111101
  - predictions: 1111011110111101
  - mispredictions: 1111011110111110
Predicting conditional branch outcomes

- 2-bit saturating counter often used
  - Branch taken ==> increment state
    - Max state “11” stays at “11” when incremented
  - Branch not-taken ==> 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
Predicting conditional branch outcomes

- Assuming initial state to be “11”, 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>33332333233332</td>
</tr>
<tr>
<td>predictions</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>mispredictions</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 other branches rather than just the branch we are trying to predict is sometimes important.
  - Outcomes of different branches often correlate.
Correlating branch predictors: code example

if (aa == 2)              DSUBUI   R3,R1,#2
    aa = 0;
if (bb == 2) L1:          BNEZ      R3,L1
    bb = 0;
if (aa != bb) {           DADD      R1,R0,R0
    L2: DSUBUI   R3,R2,#2
        BNEZ      R3,L2
        DADD      R2,R0,R0
        DSUBU     R3,R1,R2
        BEQZ      R3,L3

Correlating branch predictor with 2-bit global history register
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.

State machine transitions for a selector
Branch target buffer (BTB)

- It is a cache that stores branch targets
- It is 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
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.
Paper Reading assignment

- Trace Cache
- Implementation of Precise Faults