Announcements

- Lab2 and prelim grades

- Back to the regular office hours
Overview

- Dynamic scheduling handles various types of data hazards
  - BUT, how about control hazard?

- Today – dealing with control hazards through prediction
  - Branch prediction
    - Temporal correlation
    - Spatial correlation
  - Branch target buffer (BTB)

- Reading
  - H&P Chapter 2.3
  - Scott McFarling, “Combining Branch Predictors”, WRL Technical Note TN-36

Run-Length Between Branches

Average dynamic instruction mix from SPEC92:

<table>
<thead>
<tr>
<th></th>
<th>SPECint92</th>
<th>SPECfp92</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>39 %</td>
<td>13 %</td>
</tr>
<tr>
<td>FPU Add</td>
<td></td>
<td>20 %</td>
</tr>
<tr>
<td>FPU Mult</td>
<td></td>
<td>13 %</td>
</tr>
<tr>
<td>load</td>
<td>26 %</td>
<td>23 %</td>
</tr>
<tr>
<td>store</td>
<td>9 %</td>
<td>9 %</td>
</tr>
<tr>
<td>branch</td>
<td>16 %</td>
<td>8 %</td>
</tr>
<tr>
<td>other</td>
<td>10 %</td>
<td>12 %</td>
</tr>
</tbody>
</table>

SPECint92:         compress, eqntott, espresso, gcc, li
SPECfp92:          doduc, ear, hydro2d, mlidjp2, su2cor

What is the average run length between branches?

Based on percentage: \( \frac{100}{16}, \frac{100}{12}, \frac{100}{8} \)
MIPS Branches and Jumps

Each instruction fetch depends on one or two pieces of information from the preceding instruction:

1) Is it a jump or branch?
   Is it a taken branch?

2) Target address (if taken branch or jump)

Branch Penalties in Modern Pipelines

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

- A: PC Generation/Mux
- P: Instruction Fetch Stage 1
- F: Instruction Fetch Stage 2
- B: Branch Address Calc/Begin Decode
- I: Complete Decode
- J: Steer Instructions to Functional units
- R: Register File Read
- E: Integer Execute

... Remainder of execute pipeline (+ another 6 stages)
Branch Prediction

- Motivation: branch penalties limit performance of deeply pipelined processors
  
  - it gets worse with issue width and pipeline depth
  
  - # of simultaneous instructions

- Required hardware support:
  1) Prediction
     a) is it a branch?
     b) outcome?
     c) if taken, target?
  2) Recovery
     fix pipeline

Static Branch Predication

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

- 90% chance to take it
- 50% chance to take it

Methods:
1) SW ⇒ ISA hint
   - bne/p
     preferred taken
   - bne/n
     preferred not taken

2) HW determine based on forward/backwards
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*)

Branch History Table (BHT)

- Small memory to store the history of each branch
  - Sometimes called Pattern History Table (PHT)
    - Use LSB’s of branch PC as index
    - No tag = always true
    - Doesn’t affect correctness, only performance
    - Rather use the memory to store history

- Simplest: remember last outcome – one-bit BHT
  - Targets highly biased branches
  - Ex) loop branch taken 9/10 times; what is the misprediction rate?

<table>
<thead>
<tr>
<th>Branch Instance</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Outcome</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
2-bit BHT

- Solution: two-bit prediction *(saturating counter, generalize to n-bits)*

<table>
<thead>
<tr>
<th>On taken</th>
<th>1 1</th>
<th>Strongly taken</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 0</td>
<td>Weakly taken</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>Weakly not taken</td>
</tr>
<tr>
<td></td>
<td>0 0</td>
<td>Strongly not taken</td>
</tr>
</tbody>
</table>

• (note: textbook & HW3 follows different encoding)

<table>
<thead>
<tr>
<th>Prediction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outcome</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

**Branch Instance**

- From state 11, need 2 NT's to switch prediction

---

2-bit BHT Implementation

**Fetch PC**

- **isBranch?**
- **opcode**
- **offset**
- Target

**BHT**

- 2 entries
- 2 bits/entry
Accuracy of 2-bit BHT

- 4K vs. infinite # entries for SPEC89 (average)

![Graph showing the accuracy of 2-bit BHT with a trend line indicating 93.5% accuracy.]

From McFarling, 1993

Local History Predictor

```cpp
for (i = 0; i < N; c++) {
  for (j = 0; j < 3; j++) {
    ...
  }
}
```

- Exploit the history of a particular branch
  - The loop: 2 taken branches + 1 not taken branch
  - Use this to figure out branch behavior

- Select a prediction based on a branch’s recent history
  - Local history table remembers the history of a particular branch
  - Use 2-bit BHT for a prediction
Local Predictor Example

Local History Table

Branch Instance

<table>
<thead>
<tr>
<th>Prediction</th>
<th>NT</th>
<th>NT</th>
<th>NT</th>
<th>NT</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>NT</th>
<th>NT</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outcome</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
<td>NT</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Local Predictor Accuracy

From McFarling, 1993
Exploiting Spatial Correlation
Yeh and Patt, 1992

- Look at behavior of recently executed branches
  - Ex) If first condition false, second condition also false

- Select a prediction based on the outcome of past branches (global history)

Global-Select Branch Predictor

Pentium Pro uses the result from the last two branches to select one of the four sets of BHT bits (~95% correct)
(m,n) Branch Predictors

- Generalized global-select BPs
  - m-bit GHR
  - n-bit BHTs

- m = 0?
  no global history, just 1 BHT

- How many bits are required for (2,2) BP?

\[
(m, n) \text{ BP} \? \quad 2^m \times \left( \frac{\text{# entries}}{\text{BHTs}} \right) \times n \text{ bits} \]

Global Predictor Example

B1: if \((x[i] < 7)\) then \(y += 1;\)
B2: if \((x[i] < 5)\) then \(c = 4;\)

<table>
<thead>
<tr>
<th>Prediction</th>
<th>NT</th>
<th>NT</th>
<th>NT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outcome</td>
<td>T</td>
<td>T</td>
<td>NT</td>
</tr>
<tr>
<td></td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td></td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td></td>
<td>NT</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Global Predictor Variants

- Global

  ![Global Predictor Diagram](image)

- Global-Share

  ![Global-Share Predictor Diagram](image)

Tournament Predictors

- Multilevel predictors – predicting predictors!
  - which predictor works best for this particular branch?
- Useful to pick between local, global
- All predictors cast vote
  - only predicted predictor's used
  - all predictions compared against branch outcome, then
    - each branch predictor updates its content
    - predictor history table updated
- Ex.: two-bit saturating counter, select between two predictors
  - 0X use pred. #1, 1X use pred. #2
Tournament Predictor: Alpha 21264

Combined Predictor Performance

- doduc
- egrottt
- espess
- tpppp
- gcc
- il
- mat300
- nasa7
- spice
- tomcatv
- average

<table>
<thead>
<tr>
<th>Conditioned Branch Prediction Accuracy (%)</th>
<th>bimodal</th>
<th>gshare</th>
<th>bimodal/gshare</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Limitations of BHTs

Only predicts branch direction. Therefore, cannot redirect fetch stream until after branch target is determined.

**UltraSPARC-III fetch pipeline**

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

Branch Target Buffer (BTB)

BP bits are stored with the predicted target address.

**IF stage:** If (BP=taken) then nPC=target else nPC=PC+4

**Later:** check prediction, if wrong then kill the instruction and update BTB & BPb else update BPb
Address Collisions

Assume a 64-entry BTB

What will be fetched after the instruction at 0x220?
- BTB prediction = 0x200
- Correct target = 0x214, PC + 4

BTB is only for Control Instructions

BTB contains useful information for branch and jump instructions only

How to achieve this effect without decoding the instruction?

=> store tag!
Branch Target Buffer (BTB)

- Keep both the branch PC and target PC in the BTB
- PC+4 is fetched if match fails
- Only taken branches and jumps held in BTB
- Next PC determined before branch fetched and decoded

Consulting BTB Before Decoding

- For PC = 0x210
  - Tag mismatch → PC = PC + 4
  - = 0x214

- Add ...
- Jump 0x200
- 0x110
Combining BTB and BHT

- BTB entries are considerably more expensive than BHT, but can redirect fetches at earlier stage in pipeline and can accelerate indirect branches (JR)
- BHT can hold many more entries and is more accurate

BHT in later pipeline stage corrects when BTB misses a predicted taken branch

BTB/BHT only updated after branch resolves in E stage

Uses of Jump Register (JR)

- Switch statements (jump to address of matching case)
  
  \[ \text{BTB works well if you jump to same case} \]

- Dynamic function call (jump to run-time function address)
  
  \[ \text{BTB works well if jump to same function} \]

- Subroutine returns (jump to return address)
  
  \[ \text{BTB works well if you return to the same place} \]

How well does BTB work for each of these cases?
Return Address Stack

Small structure to accelerate JR for subroutine returns, typically much more accurate than BTBs.

```plaintext
fa() { fb(); }
fb() { fc(); }
fc() { fd(); }
```