Branch Prediction

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen
Updated by Mikko Lipasti
Lecture Overview

• Program control flow
  – Implicit sequential control flow
  – Disruptions of sequential control flow
• Branch Prediction
  – Branch instruction processing
  – Branch instruction speculation
• Key historical studies on branch prediction
  – UCB Study [Lee and Smith, 1984]
  – IBM Study [Nair, 1992]
• Branch prediction implementation (PPC 604)
  – BTAC and BHT design
  – Fetch Address Generation
Program Control Flow

• Implicit Sequential Control Flow
  – Static Program Representation
    • Control Flow Graph (CFG)
    • Nodes = basic blocks
    • Edges = Control flow transfers
  – Physical Program Layout
    • Mapping of CFG to linear program memory
    • Implied sequential control flow
  – Dynamic Program Execution
    • Traversal of the CFG nodes and edges (e.g. loops)
    • Traversal dictated by branch conditions
  – Dynamic Control Flow
    • Deviates from sequential control flow
    • Disrupts sequential fetching
    • Can stall IF stage and reduce I-fetch bandwidth
Program Control Flow

- Dynamic traversal of static CFG
- Mapping CFG to linear memory
Disruption of Sequential Control Flow

- Fetch
  - Instruction/Decode Buffer
- Decode
- Dispatch Buffer
- Dispatch
- Reservation Stations
- Execute
- Branch
- Reorder/Completion Buffer
- Finish
- Complete
- Store Buffer
- Retire
Branch Prediction

• Target address generation → **Target Speculation**
  – Access register:
    • PC, General purpose register, Link register
  – Perform calculation:
    • +/- offset, autoincrement, autodecrement

• **Condition resolution** → **Condition speculation**
  – Access register:
    • Condition code register, General purpose register
  – Perform calculation:
    • Comparison of data register(s)
Condition Resolution

Fetch
Decode Buffer
Decode
Dispatch Buffer
Dispatch
Store Buffer
Complete
Retire
Stations
Issue
Execute
Finish
Completion Buffer
Branch
CC reg.
GP reg.
value comp.
Branch Instruction Speculation

Branch to I-cache
PC(seq.) = FA (fetch address)

Branch Predictor (using a BTB)

Prediction
Spec. target
Spec. cond.

BTB update (target addr. and history)

FA-mux

Fetch
PC(seq.)

Decode Buffer

Decode

Dispatch Buffer

Dispatch

Issue

Reservation Stations

Branch

Execute

Finish

Completion Buffer
Branch/Jump Target Prediction

<table>
<thead>
<tr>
<th>Branch inst. address</th>
<th>Information for predict.</th>
<th>Branch target address (most recent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0348</td>
<td>0101 (NTNT)</td>
<td>0x0612</td>
</tr>
</tbody>
</table>

- **Branch Target Buffer**: small cache in fetch stage
  - Previously executed branches, address, taken history, target(s)
- Fetch stage compares current FA against BTB
  - If match, use prediction
  - If predict taken, use BTB target
- When branch executes, BTB is updated
- Optimization:
  - Size of BTB: increases hit rate
  - Prediction algorithm: increase accuracy of prediction
Branch Prediction: Condition Speculation

1. Biased Not Taken
   - Hardware prediction
   - Does not affect ISA
   - Not effective for loops

2. Software Prediction
   - Extra bit in each branch instruction
     • Set to 0 for not taken
     • Set to 1 for taken
   - Bit set by compiler or user; can use profiling
   - Static prediction, same behavior every time

3. Prediction based on branch offset
   - Positive offset: predict not taken
   - Negative offset: predict taken

4. Prediction based on dynamic history
UCB Study [Lee and Smith, 1984]

- **Benchmarks used**
  - 26 programs (IBM 370, DEC PDP-11, CDC 6400)
  - 6 workloads (4 IBM, 1 DEC, 1 CDC)
  - Used trace-driven simulation

- **Branch types**
  - Unconditional: always taken or always not taken
  - Subroutine call: always taken
  - Loop control: usually taken
  - Decision: either way, if-then-else
  - Computed goto: always taken, with changing target
  - Supervisor call: always taken
  - Execute: always taken (IBM 370)

<table>
<thead>
<tr>
<th></th>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.640</td>
<td>0.657</td>
<td>0.704</td>
<td>0.540</td>
<td>0.738</td>
<td>0.778</td>
<td>0.676</td>
</tr>
<tr>
<td>NT</td>
<td>0.360</td>
<td>0.343</td>
<td>0.296</td>
<td>0.460</td>
<td>0.262</td>
<td>0.222</td>
<td>0.324</td>
</tr>
</tbody>
</table>

IBM1: compiler
IBM2: cobol (business app)
IBM3: scientific
IBM4: supervisor (OS)
## Branch Prediction Function

- Prediction function $F(X_1, X_2, \ldots)$
  - $X_1$ – opcode type
  - $X_2$ – history

- Prediction effectiveness based on opcode only, or history

<table>
<thead>
<tr>
<th></th>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode only</td>
<td>66</td>
<td>69</td>
<td>71</td>
<td>55</td>
<td>80</td>
<td>78</td>
</tr>
<tr>
<td>History 0</td>
<td>64</td>
<td>64</td>
<td>70</td>
<td>54</td>
<td>74</td>
<td>78</td>
</tr>
<tr>
<td>History 1</td>
<td>92</td>
<td>95</td>
<td>87</td>
<td>80</td>
<td>97</td>
<td>82</td>
</tr>
<tr>
<td>History 2</td>
<td>93</td>
<td>97</td>
<td>91</td>
<td>83</td>
<td>98</td>
<td>91</td>
</tr>
<tr>
<td>History 3</td>
<td>94</td>
<td>97</td>
<td>91</td>
<td>84</td>
<td>98</td>
<td>94</td>
</tr>
<tr>
<td>History 4</td>
<td>95</td>
<td>97</td>
<td>92</td>
<td>84</td>
<td>98</td>
<td>95</td>
</tr>
<tr>
<td>History 5</td>
<td>95</td>
<td>97</td>
<td>92</td>
<td>84</td>
<td>98</td>
<td>96</td>
</tr>
</tbody>
</table>
Example Prediction Algorithm

- Hardware table remembers last 2 branch outcomes
  - History of past several branches encoded by FSM
  - Current state used to generate prediction

- Results:

<table>
<thead>
<tr>
<th>Workload</th>
<th>IBM1</th>
<th>IBM2</th>
<th>IBM3</th>
<th>IBM4</th>
<th>DEC</th>
<th>CDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>93</td>
<td>97</td>
<td>91</td>
<td>83</td>
<td>98</td>
<td>91</td>
</tr>
</tbody>
</table>
• Combining prediction accuracy with BTB hit rate (86.5% for 128 sets of 4 entries each), branch prediction can provide the net prediction accuracy of approximately 80%. This implies a 5-20% performance enhancement.
IBM Study [Nair, 1992]

• Branch processing on the IBM RS/6000
  – Separate branch functional unit
  – Five different branch types
    • b: unconditional branch
    • bl: branch and link (subroutine calls)
    • bc: conditional branch
    • bcr: conditional branch using link register (returns)
    • bcc: conditional branch using count register
  – Overlap of branch instructions with other instructions
    • Zero cycle branches
  – Two causes for branch stalls
    • Unresolved conditions
    • Branches downstream too close to unresolved branches
## Branch Instruction Distribution

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>% of each branch type</th>
<th>% bc with penalty cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>bl</td>
</tr>
<tr>
<td>spice2g6</td>
<td>7.86</td>
<td>0.30</td>
</tr>
<tr>
<td>doduc</td>
<td>1.00</td>
<td>0.94</td>
</tr>
<tr>
<td>matrix300</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>tomcatv</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>gcc</td>
<td>2.30</td>
<td>1.32</td>
</tr>
<tr>
<td>espresso</td>
<td>3.61</td>
<td>0.58</td>
</tr>
<tr>
<td>li</td>
<td>2.41</td>
<td>1.92</td>
</tr>
<tr>
<td>eqntottt</td>
<td>0.91</td>
<td>0.47</td>
</tr>
</tbody>
</table>
Exhaustive Search for Optimal 2-bit Predictor

- There are $2^{20}$ possible state machines of 2-bit predictors.
- Some machines are uninteresting, pruning them out reduces the number of state machines to 5248.
- For each benchmark, determine prediction accuracy for all the predictor state machines.
- Find optimal 2-bit predictor for each application.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>spice2g6</td>
<td>97.2</td>
</tr>
<tr>
<td>doduc</td>
<td>94.3</td>
</tr>
<tr>
<td>gcc</td>
<td>89.1</td>
</tr>
<tr>
<td>espresso</td>
<td>89.1</td>
</tr>
<tr>
<td>li</td>
<td>87.1</td>
</tr>
<tr>
<td>eqntott</td>
<td>87.9</td>
</tr>
</tbody>
</table>
Number of History Bits Needed

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>3 bit</th>
<th>2 bit</th>
<th>1 bit</th>
<th>0 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>spice2g6</td>
<td>97.0 (0.009)</td>
<td>97.0 (0.009)</td>
<td>96.2 (0.013)</td>
<td>76.6 (0.031)</td>
</tr>
<tr>
<td>doduc</td>
<td>94.2 (0.003)</td>
<td>94.3 (0.003)</td>
<td>90.2 (0.004)</td>
<td>69.2 (0.022)</td>
</tr>
<tr>
<td>gcc</td>
<td>89.7 (0.025)</td>
<td>89.1 (0.026)</td>
<td>86.0 (0.033)</td>
<td>50.0 (0.128)</td>
</tr>
<tr>
<td>espresso</td>
<td>89.5 (0.045)</td>
<td>89.1 (0.047)</td>
<td>87.2 (0.054)</td>
<td>58.5 (0.176)</td>
</tr>
<tr>
<td>li</td>
<td>88.3 (0.042)</td>
<td>86.8 (0.048)</td>
<td>82.5 (0.063)</td>
<td>62.4 (0.142)</td>
</tr>
<tr>
<td>eqntott</td>
<td>89.3 (0.028)</td>
<td>87.2 (0.033)</td>
<td>82.9 (0.046)</td>
<td>78.4 (0.049)</td>
</tr>
</tbody>
</table>

- **Branch history table size**: Direct-mapped array of $2^k$ entries
- Some programs, like gcc, have over 7000 conditional branches
- In collisions, multiple branches share the same predictor
  - Constructive and destructive interference
  - Destructive interference
- Marginal gains beyond 1K entries (for these programs)
Branch Prediction Implementation (PPC 604)

- **Prediction**
- **Fetch**
- **Decode**
- **Dispatch**
- **Issue**
- **Execute**
- **Finish**

- **Branch Predictor**
- **Update**
- **Branch Predictor**
- **Fetch**
- **Decode**
- **Dispatch**
- **Reservation Stations**
- **Completion Buffer**
- **BRN**
- **SFX**
- **SFX**
- **CFX**
- **FPU**
- **LS**
BTAC and BHT Design (PPC 604)

**BTAC**:
- 64 entries
- fully associative
- hit => predict taken

**BHT**:
- 512 entries
- direct mapped
- 2-bit saturating counter
  history based prediction
- overrides BTAC prediction