Correlating (Global) Branch Predictors

- Assign two prediction bits, one if the previous branch was not taken, the other if it was taken.

  b1: if \(d=0\)  
  d=1;  
  b2: if \(d=1\)

  If b1 is taken, b2 is taken

  b1:  
  BNEZ R1, L1 
  DADDUI R1, R0, #1 
  L1:  
  DSUBUI R3, R1, #1 
  b2:  
  BNEZ R3, L2 
  L2:  
  …

  If b1 is not taken, b2 is not taken

One bit indicating what to do if one previous branch was taken

One bit indicating what to do if one previous branch was not taken

Correlating Branch Predictors

- Assign two prediction bits, one if the previous branch was not taken, the other if it was taken.

  b1: BNEZ R1, L1 
  DADDUI R1, R0, #1 
  L1: DSUBUI R3, R1, #1 
  b2: BNEZ R3, L2 
  L2: …

  This is (1,1) predictor \(\Rightarrow\) it uses outcome of 1 previous branch to do prediction with 1-bit predictor

<table>
<thead>
<tr>
<th>R1=?</th>
<th>b1 prediction</th>
<th>b1 action</th>
<th>New b1 prediction</th>
<th>b2 prediction</th>
<th>b2 action</th>
<th>New b2 prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT/NT</td>
<td>T m</td>
<td>T/NT</td>
<td>NT/NT</td>
<td>T m</td>
<td>NT/T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>T/NT</td>
<td>NT/T</td>
<td>NT</td>
<td>NT/T</td>
</tr>
<tr>
<td>2</td>
<td>T/NT</td>
<td>T</td>
<td>T/NT</td>
<td>NT/T</td>
<td>T</td>
<td>NT/T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>T/NT</td>
<td>NT/T</td>
<td>NT</td>
<td>NT/T</td>
</tr>
</tbody>
</table>

Correlating Branch Predictors (m,n)

- Observe behavior of \(m\) previous branches, use \(n\)-bit predictor

  One bit indicating what to do if one previous branch was not taken

  One bit indicating what to do if one previous branch was taken

  \((1,1)\)

  One bit indicating what to do if \(m\) previous branches were not taken

  One bit indicating what to do if \(m\) previous branches were taken

  \((m,1)\)

  \(n\) bits indicating what to do if \(m\) previous branches were not taken

  \(n\) bits indicating what to do if \(m\) previous branches were taken

  \((m,n)\)

Correlating Branch Predictors (m,n)

- How many bits do we need for \((m,n)\) predictor?

  \(2^m\) combinations, \(n\)-bits each, suppose we use last \(t\) bits of branch target to select prediction

  \(2^m \times n \times 2^t\)

Tournament Predictors

- Combine one global and one local predictor with a selector

  First selector was right

  Second selector was wrong

  Use predictor 1

  Use predictor 2

  1/0 0/1 0/1

  1/0 0/1 1/0

  0/0, 1/1

  0/0, 1/1
High-performance Instruction Delivery

- Issue multiple instructions per clock cycle
- It is not enough to predict branches correctly, we also must resolve branch target quickly (end of IF)
- Use branch target buffer, cache branch target address for every branch
- For MIPS pipeline we will resolve branch during ID stage
  - Branch target buffer along with accurate prediction saves 1 cycle as everything is done during IF stage

Branch Target Buffer

- In IF, look up instruction address in BTB
- If we find the address (exact match), it is a branch
- It is predicted taken (otherwise it wouldn’t be there)
  - Use cached address in BTB to fetch next instruction

Example

Determine total branch penalty for a BTB assuming the following penalty cycles

<table>
<thead>
<tr>
<th>Instruction in buffer</th>
<th>Prediction</th>
<th>Outcome</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>taken</td>
<td>taken</td>
<td>0</td>
</tr>
<tr>
<td>yes</td>
<td>taken</td>
<td>not taken</td>
<td>2</td>
</tr>
<tr>
<td>no</td>
<td>taken</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>not taken</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Assume 60% of branches are taken, prediction accuracy is 90% and hit rate in BTB is 90%

Branch Target Buffer

- Sometimes BTB stores a few instructions from the target, instead of target address
  - This allows for 0-cycle unconditional branches

Return Address Predictors

- Procedure may be called from several places, return addresses will differ
- Use a stack to store return address as procedure is called, pushing address on call, popping on return

Multiple Issue

- We will issue several instructions per cycle
  - Superscalar processors issue varying number of MIPS-like instructions per cycle
    - Statically scheduled (in-order execution)
    - Dynamically scheduled (out-of-order execution)
  - Very Long Instruction Word (VLIW) processors have long instruction words that contain fixed number of MIPS-like instructions
    - One large instruction or several instructions with explicitly indicated parallelism
    - Statically scheduled (in-order execution)
    - They issue one such word per cycle
**Statically Scheduled Superscalar Processor**

- Issue instructions in issue packets
  - From 0 to 8 instructions per issue packet
- All hazards are checked in hardware when instructions are issued (*dynamic issue capability*)
  - Among issuing instructions in the issue packet
  - Between issuing instructions in the issue packet and the ones still in execution
  - If one instruction from the issue packet cannot be issued due to hazard, only preceding instructions are issued

**Statically Scheduled Superscalar Processor**

- Hazard checks take a long time so issue stage is split into two stages and pipelined
  - Hazards among instructions in the current issue packet are checked in the first issue stage
  - Hazards between current issue packet and others in the pipeline are checked in the second issue stage

**Statically Scheduled Superscalar MIPS**

- Assume 2 instructions issue per cycle
  - One is load, store, branch or integer ALU
  - Other is FP operation
  - This combination reduces risk of hazards between these two instructions
- Fetch two instructions, check for hazards and issue them
  - If one instruction is load, store or move than we could have data hazard and maybe structural hazard for FP register file ports

**Statically Scheduled Superscalar MIPS**

1          2           3         4         5         6          7          8          9

<table>
<thead>
<tr>
<th>Int instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>Int instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>EX</td>
<td>MEM</td>
</tr>
</tbody>
</table>

**Statically Scheduled Superscalar MIPS**

- Problems might arise:
  - We will need additional hardware in the pipeline
  - Maintaining precise exceptions is hard because instructions may complete out of order
  - Hazard penalties are longer

**Dynamically Scheduled Superscalar MIPS**

- Extend Tomasulo’s algorithm to support issue of 2 instructions per cycle
- We must issue instructions to reservation stations in order
- Issue stage can either be
  - Pipelined – issue one instruction in half cycle, another one in another half
  - Extended – add more hardware and issue instructions simultaneously
**Dynamically Scheduled Superscalar MIPS**

Loop:  
\[ \text{L.D F0,0(R1)} \]
\[ \text{ADD D F4, F0, F2} \]
\[ \text{S.D F4, 0(R1)} \]
\[ \text{DADDIU R1, R1, #-8} \]
\[ \text{BNE R1, R2, LOOP} \]

- Any two instruction can be issued (not only integer + FP)
- One integer unit used both for ALU operations and effective address calculation
- Integer ALU takes 1 cycle, load 2, FP add 3
- Pipelined FP units, 2 CDB, perfect branch prediction
- One cycle is needed for issue and one for write results (this stage adds one cycle delay)
- Show when each instruction issues, begins execution and writes to CDB for the first 3 iterations of the loop
- Show resource usage for integer unit, FP unit, data cache and CDB
- Assume instructions following branch cannot proceed with

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Memory</th>
<th>Write CDB</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L.D F0, 0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>wait for L.D</td>
</tr>
<tr>
<td>1</td>
<td>ADD D F4, F0, F2</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>1</td>
<td>S.D F4, 0(R1)</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU R1, R1, #-8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>wait for next.</td>
</tr>
<tr>
<td>1</td>
<td>BNE R1, R2, Loop</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>1</td>
<td>L.D F0, 0(R1)</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>2</td>
<td>ADD D F4, F0, F2</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>wait for L.D</td>
</tr>
<tr>
<td>2</td>
<td>S.D F4, 0(R1)</td>
<td>5</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU R1, R1, #-8</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>15</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>3</td>
<td>ADD D F4, F0, F2</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>15</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>3</td>
<td>S.D F4, 0(R1)</td>
<td>8</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>wait for ALU</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU R1, R1, #-8</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>wait for DAMO</td>
</tr>
<tr>
<td>3</td>
<td>BNE R1, R2, Loop</td>
<td>9</td>
<td>16</td>
<td>17</td>
<td>15</td>
<td>wait for DAMO</td>
</tr>
</tbody>
</table>

\[ \text{CPI=16/15=1.07} \]

---

**Dynamically Scheduled Superscalar MIPS**

Assume different units for effective address and int ALU

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Memory</th>
<th>Write CDB</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L.D F0, 0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>wait for L.D</td>
</tr>
<tr>
<td>1</td>
<td>ADD D F4, F0, F2</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>1</td>
<td>S.D F4, 0(R1)</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU R1, R1, #-8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>wait for next.</td>
</tr>
<tr>
<td>1</td>
<td>BNE R1, R2, Loop</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>1</td>
<td>L.D F0, 0(R1)</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>2</td>
<td>ADD D F4, F0, F2</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>wait for L.D</td>
</tr>
<tr>
<td>2</td>
<td>S.D F4, 0(R1)</td>
<td>5</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU R1, R1, #-8</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>15</td>
<td>wait for BNE</td>
</tr>
<tr>
<td>3</td>
<td>ADD D F4, F0, F2</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>15</td>
<td>wait for ADD.D</td>
</tr>
<tr>
<td>3</td>
<td>S.D F4, 0(R1)</td>
<td>8</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>wait for ALU</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU R1, R1, #-8</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>wait for DAMO</td>
</tr>
<tr>
<td>3</td>
<td>BNE R1, R2, Loop</td>
<td>9</td>
<td>16</td>
<td>17</td>
<td>15</td>
<td>wait for DAMO</td>
</tr>
</tbody>
</table>

\[ \text{CPI=11/15=0.73} \]