Branch Prediction Key Points

- The better we predict, the behinder we get.
- 2-bit predictors capture tendencies well.
- Correlating predictors improve accuracy, particularly when combined with 2-bit predictors.
- Accurate branch prediction does no good if we don’t know there was a branch to predict.
- BTB identifies branches in (or before) IF stage.
- BTB combined with branch prediction table identifies branches to predict, and predicts them well.

Now what?

- CPI = 1.0 + BSPI + FPSPI + LdSPI

Multiple Instruction Issue

- Superscalar
  - variable number of instructions issued each cycle
  - parallelism detected in hardware
- Very Long Instruction Words (VLIW)
  - fixed number of instructions issued each cycle
  - parallelism scheduled by the compiler
  - IA-64 (Merced, Itanium)
Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar DLX: 2 instructions, 1 FP & 1 anything else
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues

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

VLIW Processors

- Very Long Instruction Word
- N-wide VLIW issues packets of N instructions simultaneously. Compiler guarantees independence of those N instructions.

Loop Unrolling in VLIW

- Unrolled 7 times to avoid delays
- 7 results in 9 clocks, or 1.3 clocks per iteration
- Need more registers in VLIW
Limits to Multi-Issue Machines

- 1 branch in 5 instructions => how to keep a 5-way VLIW busy?
- Latencies of units => many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent operations to keep machines busy
- Instruction mix may not match hardware mix
  - Need duplicate FUs, increased flexibility
- Increase ports to Register File (VLIW example needs 6 read and 3 write for Int. Reg. & 6 read and 4 write for FP reg)
- Increase ports to memory
- Decoding/rename SS and impact on clock rate

Superscalar vs. VLIW

- Superscalar Positives
- VLIW Positives

VLIW Software Techniques

- VLIW processors only work in the presence of abundant software ILP (and reduced number of branches).
- The following techniques were originally motivated by VLIW processors, although they have since been applied to superscalar processors.
  - Software pipelining
  - Trace scheduling
  - Predicated execution

Compiler support for ILP: Software Pipelining

- Observation: if iterations from loops are independent, then can get ILP by taking instructions from different iterations
- Software pipelining: reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop
SW Pipelining Example

Unrolled 3 times
1. LD F0,0(R1)  
2. ADDD F4,F0,F2  
3. SD 0(R1),F4  
4. LD F6,-8(R1)  
5. ADDD F8,F6,F2  
6. SD -8(R1),F8  
7. LD F10,-16(R1)  
8. ADDD F12,F10,F2  
9. SD -16(R1),F12  
10. SUBI R1,R1,#24  
11. BNEZ R1,LOOP

Software Pipelined
1. LD F0,0(R1)  
2. ADDD F4,F0,F2  
3. SD 0(R1),F4  
4. ADDD F8,F6,F2  
5. SD -8(R1),F8  
6. SUBI R1,R1,#8  
7. BNEZ R1,LOOP

Compiler Support for ILP: Trace Scheduling

Trace Scheduling
- Parallelism across IF branches vs. LOOP branches
- Two steps:
  - Trace Selection
    - Find likely sequence of basic blocks (trace) of (statically predicted) long sequence of straight-line code
  - Trace Compaction
    - Squeeze trace into few VLIW instructions
    - Need bookkeeping code in case prediction is wrong

HW support for More SW ILP
- Avoid branch prediction by turning branches into conditionally executed instructions: (aka predicated instructions)
  - if (x) then A = B op C else NOP
    - If false, then neither store result or cause exception
    - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr, IA64 can predicate any instruction (even have multiple predicates)
Predicated Execution

- Drawbacks to conditional instructions
  - Still takes a clock & alu even if “annulled”
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline
  - Requires more operands! Typically only available as conditional move.
  - Complicates evaluation of dependences/renaming.
- Advantages
  - Eliminate prediction, misprediction
  - Longer basic blocks, ...
  - Creates long basic blocks by finding long paths in the code

HW support for More (HW) ILP

- Speculation: allow an instruction to issue that is dependent on branch predicted to be taken without any consequences (including exceptions) if branch is not actually taken (“HW undo”)
- Often combined with dynamic scheduling
- Tomasulo: separate speculative bypassing of results from real bypassing of results
  - When instruction no longer speculative, write results (instruction commit)
  - Execute out-of-order but commit in order

Hardware Speculative Execution

- Need HW buffer for results of uncommitted instructions: reorder buffer
  - Reorder buffer can be operand source
  - Once operand commits, result is found in register
  - 3 fields: instr. type, destination, value
  - Use reorder buffer number instead of reservation station
  - Instructions commit in order
  - As a result, easy to undo speculated instructions on mispredicted branches or on exceptions

Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination. Operands may be read from register file or reorder buffer.
2. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute
3. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. Commit—update register with reorder result
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.
Tomasulo – cycle 0

Loop:
- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- SUBI ...
- BNEZ ..., Loop

Instruction Queue

FP adders

FP mult's

Dean Tullsen

Tomasulo – cycle 1

Loop:
- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- SUBI ...
- BNEZ ...

Instruction Queue

FP adders

FP mult's

Dean Tullsen

Tomasulo – cycle 2

Loop:
- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- SUBI ...
- BNEZ ...

Instruction Queue

FP adders

FP mult's

Dean Tullsen

Tomasulo – cycle 3

Loop:
- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- SUBI ...
- BNEZ ...

Instruction Queue

FP adders

FP mult's

Dean Tullsen
Speculative Execution

- The re-order buffer and in-order commit allow us to flush the speculative instructions from the machine when a misprediction is discovered.
- ROB is another possible source of operands
- ROB can provide precise exceptions in an out-of-order machine
- ROB allows us to ignore exceptions on speculative code.

- Compiler speculation vs. hardware speculation?

PowerPC 620

- speculative, out-of-order, superscalar processor
- fetches 4, issues 4, completes 4, 6 independent functional units
- Reservation stations, reorder buffer, register renaming in the register file

PowerPC 620

- Based on all kinds of ideal assumptions. Further limited by:
  - realistic branch prediction
  - finite renaming registers
  - imperfect alias analysis for memory operations
PowerPC 620 Pipeline Structure

- Fetch
- Decode
- Issue
- Execute
- Commit

PowerPC 620 Performance

- Parallelism is absolutely critical to modern computer system performance, but at a very fine level.
- Mechanisms that create, or expose parallelism: loop unrolling, software pipelining, code motion
- Mechanisms that allow the machine to exploit ILP: pipelining, superscalar, dynamic scheduling, speculative execution

FIGURE 4.52 The average number of instructions that the fetch unit can provide to the issue unit varies between 3.2 and 4, with an average of 3.4 for the integer benchmarks and 3.8 for the FP benchmarks.

FIGURE 4.57 The breakdown of the ideal IPC of 4.0 into its components.