Outline

• History of Itanium
• Itanium Architecture Features
  – Registers
  – Instructions
  – Speculation
  – Predication and Hints
• Itanium 2 vs. Itanium
  – Execution Units
  – Cache system
  – Pipeline

History

• 1994: Intel and HP began working on Merced
• 2001: released Itanium processor at 733, 800MHz
• 2002: released Itanium 2 processor at 900MHz and 1GHz, codenamed McKinley
• 2003:
  – Madison was introduced with three version
  – Hondo was announced as the HP mx2 dual-processor module
  – Deerfield was released as the first low voltage Itanium processor
• 2004:
  – released first processor in Madison 9M series
  – Fanwood core debuted
• Upcoming: Montecito

Application Register State

• 128 General Registers
  – GR0 = 0 (read-only)
  – GR0-GR31 are static, GR32-GR127 are stacked
• 128 Floating-point Registers
  – FR0 = 0.0 (read-only), FR1 = 1.0 (read-only)
  – FR0-FR21 are static, FR32-FR127 are rotating
• 64 Predicate Registers
  – PR0 = 1 (read-only)
  – PR0-PR15 are static, PR16-PR63 are rotating
• 8 Branch Registers
  – Holds target address for indirect branches
• 128 Application Registers
Application Register State (cont)

- Current Frame Marker
- Instruction Pointer
- User Mask

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfh</td>
<td>mfl</td>
<td>ac</td>
<td>up</td>
<td>be</td>
<td>rv</td>
</tr>
</tbody>
</table>

543210

rrb.pr
rrb.fr
rrb.gr
sol
sor

rv: Reserved
be: IA-64 big-endian memory access enable
up: User performance monitor enable
ac: Alignment check for data memory references
mfl: Lower floating-point registers written
mfh: Upper floating-point registers written

Register Stack

Register Stack (cont)

- Register stack frame can be resized using alloc instruction
- Register Stack Engine (RSE)
  - Automatically save and restore register stack without explicit software intervention
  - Use spare memory bandwidth in the background
  - Spill and fill may cause RSE traffic
  - RSE traffic degrades performance

Register Rotation

- Register renaming mechanism that enables the concurrent execution of multiple iterations of a loop
- Rotating register
  - PR32-PR127, FR32-FR127, programmable sized GR starting from GR32
  - Size of rotating area in GR file determined by alloc instruction (size either be 0 or 8x)
  - Rotate toward larger register number
  - Renaming register number = rotate register number + value of rrb

Register Rotation (cont)

Instruction Types

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Description</th>
<th>Execution Unit Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Integer ALU</td>
<td>I-unit or M-unit</td>
</tr>
<tr>
<td>I</td>
<td>Non-ALU loops</td>
<td>I-unit</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>M-unit</td>
</tr>
<tr>
<td>F</td>
<td>Floating-point</td>
<td>F-unit</td>
</tr>
<tr>
<td>B</td>
<td>Branch</td>
<td>B-unit</td>
</tr>
<tr>
<td>L+X</td>
<td>Extended</td>
<td>L-unit/B-unit</td>
</tr>
</tbody>
</table>

Example:
A: add, or
I: mov, shif
B: br, brp
M: load L<K: brl
Bundle and Template

- Three instructions are grouped together into a 128-bit container called bundle

<table>
<thead>
<tr>
<th>Instruction slot 0</th>
<th>Instruction slot 1</th>
<th>Instruction slot 2</th>
<th>Template</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Template
  - 12 basic types: MII, MI\|I, MLX, MMI, MMF, MFI, MMF, MIB, MBB, BBB, MMB, MFB
  - Mapping of instruction to execution unit
  - Stop

Control Speculation

- Overlap long load latency
- NaT/NaTVal support

```plaintext
if (a > b) {
  ld.s target, addr
  ...
  br next
}
```

```plaintext
chk.s target, recovery
next:
    ...
```

Id.s – control speculative load
chk.s – check instruction

Data Speculation

- Advanced load (ld.a, ldf.a and ldfp.a)
  - Compute ALAT tag
  - If ALAT entry exists, remove it
  - Allocate a new ALAT entry
  - Load the value into target register
- Advanced load check
  - Looks for a matching ALAT entry, if found, falls through to next instruction
  - Otherwise, branches to recovery
- Two kinds of compiler-generated recovery
  - Check load instructions: ld.c, ldf.c or ldfp.c
  - Advanced load check: chk.a

Predication

- Conditional execution of an instruction base on a qualifying predicate
  - Convert branch conditions to predicate registers
  - Convert control dependences to data dependences
- Most instructions can be predicated

Branch Hints

- Mechanism to decrease the branch misprediction rate
- Do not affect the functional behavior of the program and may be ignored by the processor
Execution Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>New</th>
<th>Legacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Load Port</td>
<td>2</td>
<td>1 cycle (L1)</td>
</tr>
<tr>
<td>Memory Store Port</td>
<td>2</td>
<td>NA</td>
</tr>
<tr>
<td>ALU's (integer)</td>
<td>6</td>
<td>1 cycle</td>
</tr>
<tr>
<td>Integer Units</td>
<td>2</td>
<td>1 cycle</td>
</tr>
<tr>
<td>Integer Shift</td>
<td>1</td>
<td>1 cycle</td>
</tr>
<tr>
<td>Multiplication ALUs</td>
<td>6</td>
<td>2 cycles</td>
</tr>
<tr>
<td>Parallel Multiply Units</td>
<td>2</td>
<td>2 cycles</td>
</tr>
<tr>
<td>Parallel Shift-Add Units</td>
<td>2</td>
<td>2 cycles</td>
</tr>
<tr>
<td>FP FMAC (multiply-accumulate)</td>
<td>2</td>
<td>4 cycles</td>
</tr>
<tr>
<td>FP FMAC (compare, merge, etc)</td>
<td>2</td>
<td>4 cycles</td>
</tr>
<tr>
<td>Branch Unit</td>
<td>3</td>
<td>3-&gt;4 cycles</td>
</tr>
</tbody>
</table>

Issue Combinations for 2 Bundles

Cache System Distinction

- **Cache Latency**
  - L1, Ld: 2 cycles => 1 cycle
  - L2 (I, FP): 6, 9 cycles => 5, 6 cycles
  - L3 (I, FP): 21, 24 cycles => 12, 13 cycles
- **Virtual address and physical address**
  - 50-bit => 64-bit for virtual address
  - 44-bit => 50-bit for physical address
- **Cache line size**
  - Doubled for every level of cache
- **Page size**
  - Up to 4GB, used to be up to 256MB
- **Cache line transfer bandwidth**
  - Doubled

Pipeline

- In-order pipeline
- Pipeline deduction leads to 4.6% performance improvement

- IPG = Instruction pointer, FET = Fetch, ROT = Rotate, EXP = Expand
- REN = Rename, WLD = Word-line decode, REG = Register read
- EXE = Execute, DET = Exception detect, WRB = Write-back