Multiple Issue
Dynamic Speculation

Getting CPI < 1:
Issuing Multiple Instructions/Cycle

- Vector Processing: Explicit coding of independent loops as operations on large vectors of numbers
  - Multimedia instructions being added to many processors

- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, Pentium III/4

- (Very) Long Instruction Words (V)LIW:
  fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates (TBD)
  - Intel Architecture-64 (IA-64) 64-bit address
    » Renamed: “Explicitly Parallel Instruction Computer (EPIC)”
  - Will discuss in 2 lectures

- Anticipated success of multiple instructions lead to Instructions Per Clock_cycle (IPC) vs. CPI
Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar MIPS: 2 instructions, 1 FP & 1 anything
  - Fetch 64-bits/clock cycle: Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

<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>
<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>

- 1 cycle load delay expands to 3 instructions in SS
  - Instruction in right half can’t use it, nor instructions in next slot

Multiple Issue Issues

- **issue packet**: group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard or a data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction does not issue
  - 0 to N instruction issues per clock cycle, for N-issue

- Performing issue checks in 1 cycle could limit clock cycle time: \( O(n^2-n) \) comparisons
  - \( \Rightarrow \) issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue, 2nd stage examines hazards among selected instructions and those already been issued
  - \( \Rightarrow \) higher branch penalties \( \Rightarrow \) prediction accuracy important
Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations AND No hazards

- If more instructions issue at same time, greater difficulty of decode and issue:
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue; (N-issue ~O(N^2-N) comparisons)
  - Register file: need 2x reads and 1x writes/cycle
  - Rename logic: must be able to rename same register multiple times in one cycle! For instance, consider 4-way issue:

    ```
    add r1, r2, r3    add p11, p4, p7
    sub r4, r1, r2    sub p22, p11, p4
    lw r1, 4(r4)      lw p23, 4(p22)
    add r5, r1, r2    add p12, p23, p4
    ```

    Imagine doing this transformation in a single cycle!

  - Result buses: Need to complete multiple instructions/cycle
    » So, need multiple buses with associated matching logic at every reservation station.
    » Or, need multiple forwarding paths

Dynamic Scheduling in Superscalar
The easy way

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point

- Issue 2X Clock Rate, so that issue remains in order

- Only loads/stores might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW
Register renaming, virtual registers versus Reorder Buffers

• Alternative to Reorder Buffer is a larger virtual set of registers and register renaming

• Virtual registers hold both architecturally visible registers + temporary values
  - replace functions of reorder buffer and reservation station

• Renaming process maps names of architectural registers to registers in virtual register set
  - Changing subset of virtual registers contains architecturally visible registers

• Simplifies instruction commit: mark register as no longer speculative, free register with old value

• Adds 40-80 extra registers: Alpha, Pentium, ...
  - Size limits no. instructions in execution (used until commit)

How much to speculate?

• Speculation Pro: uncover events that would otherwise stall the pipeline (cache misses)

• Speculation Con: speculate costly if exceptional event occurs when speculation was incorrect

• Typical solution: speculation allows only low-cost exceptional events (1st-level cache miss)

• When expensive exceptional event occurs, (2nd-level cache miss or TLB miss) processor waits until the instruction causing event is no longer speculative before handling the event

• Assuming single branch per cycle: future may speculate across multiple branches!
Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola AltaVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

Initial HW Model here; MIPS compilers.
Assumptions for ideal/perfect machine to start:

1. Register renaming - infinite virtual registers => all register WAW & WAR hazards are avoided
2. Branch prediction - perfect; no mispredictions
3. Jump prediction - all jumps perfectly predicted
   2 & 3 => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address alias analysis - addresses are known & a store can be moved before a load provided addresses not equal

Also:
- unlimited number of instructions issued/clock cycle;
- perfect caches;
- 1 cycle latency for all instructions (FP *, /);
Upper Limit to ILP: Ideal Machine
(Figure 3.34, page 294)

- Integer: 18 - 60
- FP: 75 - 150

Instruction issues per cycle

<table>
<thead>
<tr>
<th>Program</th>
<th>IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>54.8</td>
</tr>
<tr>
<td>espresso</td>
<td>62.6</td>
</tr>
<tr>
<td>li</td>
<td>17.9</td>
</tr>
<tr>
<td>fppp</td>
<td>75.2</td>
</tr>
<tr>
<td>doducd</td>
<td>118.7</td>
</tr>
<tr>
<td>tomcatv</td>
<td>150.1</td>
</tr>
</tbody>
</table>

More Realistic HW: Branch Impact
(Figure 3.38, Page 300)

Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle
- Integer: 6 - 12
- FP: 15 - 45

Instruction issues per cycle

<table>
<thead>
<tr>
<th>Program</th>
<th>IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>50</td>
</tr>
<tr>
<td>espresso</td>
<td>63</td>
</tr>
<tr>
<td>li</td>
<td>16</td>
</tr>
<tr>
<td>fppp</td>
<td>81</td>
</tr>
<tr>
<td>doducd</td>
<td>58</td>
</tr>
<tr>
<td>tomcatv</td>
<td>80</td>
</tr>
</tbody>
</table>

- Perfect
- Tournament
- BHT (512)
- Profile
- No prediction

Page <#>
More Realistic HW: Renaming Register Impact

Figure 3.41, Page 304

FP: 11 - 45

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction

Integer: 5 - 15

More Realistic HW: Memory Address Alias Impact

Figure 3.43, Page 306

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers

Integer: 4 - 9

FP: 4 - 45 (Fortran, no heap)

IPC
How to Exceed ILP Limits of this study?

- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
- Unnecessary dependences (compiler not unrolling loops so iteration variable dependence)
- Overcoming the data flow limit: value prediction, predicting values and speculating on prediction
  - Address value prediction and speculation predicts addresses and speculates by reordering loads and stores; could provide better aliasing analysis, only need predict if addresses =
### Workstation Microprocessors 3/2001

<table>
<thead>
<tr>
<th>Processor</th>
<th>Alpha 21264B</th>
<th>AMD Athlon</th>
<th>HP PA-8700</th>
<th>IBM PowerPC 740</th>
<th>Intel Pentium III</th>
<th>Intel Pentium 4</th>
<th>MIPS R12000</th>
<th>Sun Ultra II</th>
<th>UU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Rate</td>
<td>660MHz</td>
<td>660MHz</td>
<td>660MHz</td>
<td>533MHz</td>
<td>1.066GHz</td>
<td>1.333GHz</td>
<td>1.666GHz</td>
<td>1.800MHz</td>
<td>1.20GHz</td>
</tr>
<tr>
<td>Caches</td>
<td>64K/64K</td>
<td>512K/1M</td>
<td>16K/256K</td>
<td>16K/16K</td>
<td>128K/256K</td>
<td>32K/32K</td>
<td>16K/16K</td>
<td>180MHz</td>
<td></td>
</tr>
<tr>
<td>Issue Rate</td>
<td>4 issue</td>
<td>3 x 060 instr</td>
<td>4 issue</td>
<td>4 issue</td>
<td>8 x 060</td>
<td>4 x 060</td>
<td>4 x 060</td>
<td>4 issue</td>
<td>4 issue</td>
</tr>
<tr>
<td>Pipeline Stages</td>
<td>7/9 stages</td>
<td>4/5 stages</td>
<td>7/9 stages</td>
<td>7/9 stages</td>
<td>12/14 stages</td>
<td>12/14 stages</td>
<td>12/14 stages</td>
<td>12/14 stages</td>
<td>12/14 stages</td>
</tr>
<tr>
<td>Out of Order</td>
<td>80 metal</td>
<td>72ROPs</td>
<td>66 metal</td>
<td>40 ROps</td>
<td>128 ROps</td>
<td>88 metal</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Rename regs</td>
<td>46/41</td>
<td>36/36</td>
<td>56 total</td>
<td>60 total</td>
<td>120 total</td>
<td>56 total</td>
<td>32/32</td>
<td>32/32</td>
<td>32/32</td>
</tr>
<tr>
<td>BHT Entries</td>
<td>4K x 9 bit</td>
<td>4K x 9 bit</td>
<td>7K x 7 bit</td>
<td>7K x 7 bit</td>
<td>12K x 12 bit</td>
<td>16K x 12 bit</td>
<td>4K x 9 bit</td>
<td>512 x 9 bit</td>
<td>10K</td>
</tr>
<tr>
<td>Memory UW</td>
<td>2.66GB/s</td>
<td>1.96GB/s</td>
<td>1.54GB/s</td>
<td>1.66GB/s</td>
<td>1.50GB/s</td>
<td>1.20GB/s</td>
<td>1GB/s</td>
<td>1GB/s</td>
<td>1GB/s</td>
</tr>
<tr>
<td>IC Process</td>
<td>0.18µm</td>
<td>0.18µm</td>
<td>0.25µm</td>
<td>0.18µm</td>
<td>0.18µm</td>
<td>0.18µm</td>
<td>0.18µm</td>
<td>0.18µm</td>
<td>0.18µm</td>
</tr>
<tr>
<td>Die Size</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
<td>151mm²</td>
</tr>
<tr>
<td>Transition</td>
<td>80.5 million</td>
<td>70 million</td>
<td>100 million</td>
<td>120 million</td>
<td>120 million</td>
<td>120 million</td>
<td>120 million</td>
<td>120 million</td>
<td>120 million</td>
</tr>
<tr>
<td>Est. mfg. cost</td>
<td>$100</td>
<td>$200</td>
<td>$500</td>
<td>$1000</td>
<td>$1000</td>
<td>$1000</td>
<td>$1000</td>
<td>$1000</td>
<td>$1000</td>
</tr>
<tr>
<td>Power (Max)</td>
<td>75W</td>
<td>80W</td>
<td>100W</td>
<td>120W</td>
<td>200W</td>
<td>200W</td>
<td>200W</td>
<td>200W</td>
<td>200W</td>
</tr>
<tr>
<td>Availability</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
</tr>
</tbody>
</table>

- **Max issue**: 4 instructions (many CPUs)
- **Max rename registers**: 128 (Pentium 4)
- **Max BHT**: 4K x 9 (Alpha 21264B), 16Kx2 (Ultra III)
- **Max Window Size (OOO)**: 126 instructions (Pent. 4)
- **Max Pipeline**: 22/24 stages (Pentium 4)

---

**SPEC 2000 Performance 3/2001**

<table>
<thead>
<tr>
<th>Processor</th>
<th>Alpha 21264B</th>
<th>AMD Athlon</th>
<th>HP PA-8700</th>
<th>IBM PowerPC 740</th>
<th>Intel Pentium III</th>
<th>Intel Pentium 4</th>
<th>MIPS R12000</th>
<th>Sun Ultra II</th>
<th>Sun Ultra III</th>
<th>Blade 10G</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-Ratio</td>
<td>3/2</td>
<td>2/1</td>
<td>2/1</td>
<td>2/1</td>
<td>3/2</td>
<td>3/2</td>
<td>3/2</td>
<td>3/2</td>
<td>3/2</td>
<td>3/2</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>8KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
<td>32KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>64KB</td>
<td>128KB</td>
<td>256KB</td>
<td>512KB</td>
<td>1MB</td>
<td>2MB</td>
<td>4MB</td>
<td>8MB</td>
<td>16MB</td>
<td>32MB</td>
</tr>
<tr>
<td>FPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MMX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SSE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SSE2</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IA64</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**SPEC base2000**: 3.104, 3.04, 3.00, 3.00, 3.00, 3.00, 3.00, 3.00, 3.00, 3.00
If time permits: "A Language for Describing Predictors and its Application to Automatic Synthesis", by Emer and Gloy

- What was dynamic branch mechanisms they looked at?
- How did they explore space?
- Did they improve upon current practice?
- How was did they choose between options?

Conclusion

- 1985-2000: 1000X performance
  - Moore’s Law transistors/chip => Moore’s Law for Performance/MPU
- Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore’s Law to get 1.55X/year
  - Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, ...
- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?
  - Otherwise drop to old rate of 1.3X per year?
  - Less than 1.3X because of processor-memory performance gap?
- Impact on you: if you care about performance, better think about explicitly parallel algorithms vs. rely on ILP?