Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Two variations
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun SuperSparc, DEC Alpha, HP 7100
- Very Long Instruction Words (VLIW): fixed number of instructions (16) scheduled by the compiler
  - IA64 (EPIC)
Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar Ex: 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
  - More ports for FP registers to do FP load & FP op in a pair

Type | Pipe | Stages
--- | --- | ---
FP instruction | IF | ID | EX | MEM | WB
FP instruction | IF | ID | EX | MEM | WB
Int. instruction | IF | ID | EX | MEM | WB
FP instruction | IF | ID | EX | MEM | WB
FP instruction | IF | ID | EX | MEM | WB
Int. instruction | IF | ID | EX | MEM | WB
FP

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

- Dependencies stop instruction issue
- Code compiler for scalar version will run poorly on SS
  - May want code to vary depending on how superscalar
- Simple approach: separate Tomasulo Control for separate reservation stations for Integer FU/Reg and for FP FU/Reg
- Result: more complicated Tomasulo scheme
• While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - 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
• VLIW: tradeoff instruction space for simple decoding
  - The long instruction word has room for many operations
  - By definition, all the operations the compiler puts in the long instruction word can execute in parallel
  - E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch
    • 16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide
  - Need compiling technique that schedules across several branches
Limits to Multi-Issue Machines

- Inherent limitations of ILP
  - 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

- Difficulties in building HW
  - Duplicate FUs to get parallel execution
  - 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 SS and impact on clock rate, pipeline depth
Limits to Multi-Issue Machines

- Limitations specific to either SS or VLIW implementation
  - Decode issue in SS
  - VLIW code size: unroll loops + wasted fields in VLIW
  - VLIW lock step => 1 hazard & all instructions stall
  - VLIW & binary compatibility is practical weakness
Review: Summary

- **Branch Prediction**
  - Branch History Table: 2 bits for loop accuracy
  - Correlation: Recently executed branches correlated with next branch
  - Branch Target Buffer: include branch address & prediction

- **SuperScalar and VLIW**
  - CPI < 1
  - Dynamic issue vs. Static issue
  - More instructions issue at same time, larger the penalty of hazards
HW support for More 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 try to combine 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
Four Steps of Speculative Tomasulo Algorithm

1. **Issue**—get instruction from FP Op Queue
   If reservation station or reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination.

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.
Limits to ILP

- Conflicting studies of amount of parallelism available in late 1980s and early 1990s. Different assumptions about:
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
Limits to ILP

Initial HW Model here; MIPS compilers

1. Register renaming—infinite virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction—perfect; no mispredictions
3. Jump prediction—all jumps perfectly predicted => 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

1 cycle latency for all instructions
Upper Limit to ILP

Programs

- gcc: 54.8
- espresso: 62.6
- li: 17.9
- fppp: 75.2
- doudcd: 118.7
- tomcatv: 150.1
## Combos

<table>
<thead>
<tr>
<th>Common name</th>
<th>Issue structure</th>
<th>Hazard detection</th>
<th>Scheduling</th>
<th>Distinguishing characteristic</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar (static)</td>
<td>dynamic</td>
<td>hardware</td>
<td>static</td>
<td>in-order execution</td>
<td>Sun UltraSPARC II/III</td>
</tr>
<tr>
<td>Superscalar (dynamic)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic</td>
<td>some out-of-order execution</td>
<td>IBM Power2</td>
</tr>
<tr>
<td>Superscalar (speculative)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic with speculation</td>
<td>out-of-order execution with speculation</td>
<td>Pentium III/4, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64III</td>
</tr>
<tr>
<td>VLIW/LIW</td>
<td>static</td>
<td>software</td>
<td>static</td>
<td>no hazards between issue packets</td>
<td>Trimedia, i860</td>
</tr>
<tr>
<td>EPIC</td>
<td>mostly static</td>
<td>mostly software</td>
<td>mostly static</td>
<td>explicit dependences marked by compiler</td>
<td>Itanium</td>
</tr>
</tbody>
</table>
# Water Cooler Science: Intel Pentium III Processor Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Speeds</td>
<td>1.40GHz, 1GHz, 933MHz, 866MHz, 850MHz, 800MHz, 750MHz, 733MHz, 700MHz, 667MHz and 650MHz</td>
</tr>
<tr>
<td>Features</td>
<td>P6 Dynamic Execution micro-architecture</td>
</tr>
<tr>
<td></td>
<td>Internet Streaming SIMD Extensions</td>
</tr>
<tr>
<td></td>
<td>Non-Blocking Level 1 Cache</td>
</tr>
<tr>
<td></td>
<td>256 KB Level 2 Advanced Transfer Cache</td>
</tr>
<tr>
<td></td>
<td>Intel MMX media enhancement technology</td>
</tr>
<tr>
<td>Cache</td>
<td>Level 1: 32K (16K for infrastructure and 16K for data)</td>
</tr>
<tr>
<td></td>
<td>Level 2: 512KB unified, non-blocking, or an integrated 256KB Advanced Transfer Cache</td>
</tr>
<tr>
<td>RAM</td>
<td>SDRAM and Rambus* technology RDRAM</td>
</tr>
<tr>
<td>System Bus</td>
<td>100 MHz</td>
</tr>
<tr>
<td>Frequency</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>
Pentium vs. Pentium Pro System Architectures

APIC = Advanced Programmable Interrupt Controller
Glue-less Multiprocessors
## Pentium Pro, Pentium II, Pentium III

<table>
<thead>
<tr>
<th>Processor</th>
<th>First ship date</th>
<th>Clock rate range</th>
<th>L1 cache</th>
<th>L2 cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium Pro</td>
<td>1995</td>
<td>100–200 MHz</td>
<td>8 KB instr. + 8 KB data</td>
<td>256 KB–1024 KB</td>
</tr>
<tr>
<td>Pentium II</td>
<td>1998</td>
<td>233–450 MHz</td>
<td>16 KB instr. + 16 KB data</td>
<td>256 KB–512 KB</td>
</tr>
<tr>
<td>Pentium II Xeon</td>
<td>1999</td>
<td>400–450 MHz</td>
<td>16 KB instr. + 16 KB data</td>
<td>512 KB–2 MB</td>
</tr>
<tr>
<td>Celeron</td>
<td>1999</td>
<td>500–900 MHz</td>
<td>16 KB instr. + 16 KB data</td>
<td>128 KB</td>
</tr>
<tr>
<td>Pentium III</td>
<td>1999</td>
<td>450–1100 MHz</td>
<td>16 KB instr. + 16 KB data</td>
<td>256 KB–512 KB</td>
</tr>
<tr>
<td>Pentium III Xeon</td>
<td>2000</td>
<td>700–900 MHz</td>
<td>16 KB instr. + 16 KB data</td>
<td>1 MB–2 MB</td>
</tr>
</tbody>
</table>
P6 details

- Dynamic scheduling of micro-operations for execution
- IA-32 instructions translated into micro-operations executed by pipeline
  - Up to 3 IA-32 instructions fetched, decoded, translated per cycle (ideal CPI = .33)
  - Complex IA-32 instructions (> 4 micro-ops) uses microcode
  - 6 slots for micro-instructions, 4 for first instr, 1 each for other two
- Micro-operations execution
  - Out-of-order speculative pipeline
  - Register renaming (OOO) and re-order buffering (speculation)
  - Up to 3 instructions per cycle can be renamed and dispatched to RS
  - Up to 3 instructions per cycle can be committed
P6 Processor Pipeline Overview

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Pipeline stages</th>
<th>Repeat rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer ALU</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Integer load</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Integer multiply</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FP add</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FP multiply</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>FP divide (64-bit)</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>
P6 Processor Pipeline Overview

Fetch/Decode/Dispatch
8-stages for in-order instr fetch/decode/dispatch
Next instruction selected during fetch using 512-entry, 2-level branch pred
Decode and issue stages include register renaming
40 virtual regs, 20 RS, 40-entry ROB

Out-of-order execution
3 stages, 5 separate functional units (int, fp, branch, addr, mem access)
Some instructions fully pipelined, others not

Instruction Commit
3 stages
P6 Processing Unit Interface with Memory

- **Fetch/Decode Unit**
  - In-order
  - Gets instructions from icache
  - Decodes insts into uops
  - Includes speculative prefetch

- **Dispatch/Execute Unit**
  - Out-of-order
  - Accesses decoded instrs from pool (ROB)
  - Dynamic/speculative scheduling of uops

- **Retire Unit**
  - In-order
  - Commits temp ROB results to arch state

- **Bus Interface Unit**
  - Partially in-order
  - Connects 3 other units to system bus / L2
Fetch/Decode/Rename Unit Details

- Icache is local instr cache
- Next_IP provides cache index
  - From branch target buffer
  - Trap/interrupt status
  - Branch misprediction result
- Icache fetchs 16 aligned bytes to decoder
- Three parallel decoders translate IA-32
  - Triadic uops (3 operands)
  - Mostly 1:1 conversion, some 1:4, some complex
  - uops queued and sent to RAT (renaming)
  - uops placed in pool (ROB)
Decode/Execute Details (includes renaming in RS)

- Selects uops from pool (ROB) dep on status
  - Status = operands avail
  - Dispatch = exec unit ready and status ready
- Results later forwarded to pool
- 5 ports to ROB
  - Max rate of 5 uops per cycle (one to each unit)
  - Dynamical scheduling key to ILP
  - FIFO scheduling among ready instructions
- Branch instr predicted by branch target buffer
  - At EX, branch pred tested
  - JEU restarts pipeline on mispred

Figure 2: Pentium II Dispatch/Execute units
The Retire Unit

- Selects uops from pool (ROB) dep on status
  - uops that have executed
  - Remove and commit in program order
- Must work under
  - Interrupts
  - Traps
  - Faults
  - Breakpoints
  - Mispredictions
- Update pool (ROB) and reg file
- Up to 3 uops per cycle