**Summary of discussions**

- ILP processors
  - VLIW/EPIC, Superscalar
- Superscalar has hardware logic for extracting parallelism
  - Solutions for stalls etc. must be provided in hardware
- Stalls play an even greater role in ILP processors
- Software solutions, such as code scheduling through code movement, can lead to improved execution times
  - More sophisticated techniques needed
  - Can we provide some HW support to help the compiler – leads to EPIC/VLIW

**Multiple Issue ILP Processors**

- In statically scheduled superscalar instructions issue in order, and all pipeline hazards checked at issue time
  - Inst causing hazard will force subsequent inst to be stalled
- In statically scheduled VLIW, compiler generates multiple issue packets of instructions
- During instruction fetch, pipeline receives number of inst from IF stage – issue packet
  - Examine each inst in packet; if no hazard then issue else wait
  - Issue unit examines all inst in packet
  - Complexity implies further splitting of issue stage

**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/IV
- **(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)"
- 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>
</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; \((N\text{-issue} \sim O(N^2-N)\) 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:
    - \(\text{add } r_1, r_2, r_3\) => \(\text{add } p11, p4, p7\)
    - \(\text{sub } r_4, r_1, r_2\) => \(\text{sub } p22, p11, p4\)
    - \(\text{lw } r_1, 4(r_4)\) => \(\text{lw } p23, 4(p22)\)
    - \(\text{add } r_5, r_1, r_2\) => \(\text{add } p12, p6, p4\)
- \(\Rightarrow\) higher branch penalties \(\Rightarrow\) prediction accuracy important
  - Performance and Cost

Summary of Course

- **Performance and Cost**
  - **Amdahl’s Law**:
    \[ \text{Speedup}_{\text{average}} = \frac{1}{\frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} \left(1 - \text{Fraction}_{\text{enhanced}}\right) + \text{Fraction}_{\text{enhanced}} \} \]
  - **CPI Law**:
    \[ \text{CPU time} = \text{Seconds} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}} \]
  - **Designing to Last through Trends**
    - Capacity
      - Logic: 2x in 3 years
      - DRAM: 4x in 4 years
      - Disk: 4x in 3 years
      - Processor: 2x every 1.5 years
    - Speed
      - 2x in 3 years
      - 2x in 10 years
      - 2x in 5 years
### Performance and Cost

- **Thumb rules for performance:**
  - Common case fast
  - Locality
  - Parallelism
  - Critical path

- **Cost vs. Price**
  - Can PC industry support engineering/research investment?

For better or worse, benchmarks shape a field

**Interested in learning more on performance?**

CS 238 "Computer Systems Performance"
- Don't ask me when 😊

### Goodbye to Performance and Cost

- Will sustain 2X every 1.5 years?
  - Can integrated circuits improve below 1.8 micron in speed as well as capacity?

- 5-6 yrs to PhD =>
  - 16X CPU speed, 10XDRAM Capacity, 25X Disk capacity?
  (20 GHz CPU, 10GB DRAM, 2TB disk?)

### Processor Architecture

- **ISA**
  - What ISA looks like to pipeline?
    - Cray: load/store machine; registers; simple instr. format
  - RISC: Making an ISA that supports pipelined execution
  - 80x86: importance of being their first
  - VLIW/EPIC: compiler controls Instruction Level Parallelism (ILP)
  - Interested in learning more on compilers and ISA?
    CS 246 "Compiler Optimization"

### Superscalar Execution (Chapter 3)

- Dynamic scheduling with Tomasulo
  - Why does it work, and why is it the de-facto method today?
- Multiple Instruction issue
  - Challenges?
- Did ILP limits really restrict practical machines to 4-issue, 4-commit?
- Did we ever really get CPI below 1.0?
- Branch prediction: How accurate did it become?
  - For real programs, how much better than 2 bit table?
Software Scheduling

- Instruction Level Parallelism (ILP) found either by compiler or hardware.
- Loop level parallelism is easiest to see
  - SW dependencies/compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine => Memory disambiguation
  - Very sophisticated transformations available
- Trace Scheduling to Parallelize If statements
- Superscalar and VLIW: CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - More instructions issue at same time => larger hazard penalty
  - Limitation is often number of instructions that you can successfully fetch and decode per cycle

EPIC/VLIW

- What did IA-64/EPIC do well besides floating point programs?
  - Was the only difference the 64-bit address v. 32-bit address?
  - What happened to the AMD 64-bit address 80x86 proposal?
- What happened on EPIC code size vs. x86?
- Did anybody propose anything at ISA to help with software quality? availability? Security?

Hardware versus Software Speculation Mechanisms

- To speculate extensively, must be able to disambiguate memory references
  - Much easier in HW than in SW for code with pointers
- HW-based speculation works better when control flow is unpredictable, and when HW-based branch prediction is superior to SW-based branch prediction done at compile time
  - Mispredictions mean wasted speculation
- HW-based speculation maintains precise exception model even for speculated instructions
- HW-based speculation does not require compensation or bookkeeping code

Hardware versus Software Speculation Mechanisms cont’d

- Compiler-based approaches may benefit from the ability to see further in the code sequence, resulting in better code scheduling
- HW-based speculation with dynamic scheduling does not require different code sequences to achieve good performance for different implementations of an architecture
  - may be the most important in the long run?
IA-64 EPIC vs. Classic VLIW

- **Similarities:**
  - Compiler generated wide instructions
  - Static detection of dependencies
  - ILP encoded in the binary (a group)
  - Large number of architected registers

- **Differences:**
  - Instructions in a bundle can have dependencies
  - Hardware interlock between dependent instructions
  - Accommodates varying number of functional units and latencies
  - Allows dynamic scheduling and functional unit binding

  - Code compatibility across generations
  - But software won’t run at top speed until it is recompiled so “shrink-wrap binary” might need to include multiple builds

EPIC and Compiler Optimization

- EPIC requires dependency free “scheduled code”
- Burden of extracting parallelism falls on compiler
- Success of EPIC architectures depends on efficiency of Compilers!!
- We provide overview of Compiler Optimization techniques (as they apply to EPIC/ILP)

Introduction to Compiler Optimization

Hardware-Software Interface

- Machine
  - Available resources statically fixed
  - Designed to support wide variety of programs
  - Interested in running many programs fast
- Program
  - Required resources dynamically varying
  - Designed to run well on a variety of machines
  - Interested in having itself run fast

![Equation](image)

Performance reflects how well the machine resources match the program requirements
Compiler Tasks

- Code Translation
  - Source language → target language
    - FORTRAN → C
    - C → MIPS, PowerPC or Alpha machine code
    - MIPS binary → Alpha binary

- Code Optimization
  - Code runs faster
  - Match dynamic code behavior to static machine structure

Compiler Structure

Midterm Exam

- When: Tuesday, October 28th, 6:15pm
  - 2 hours
- What?
  - Chapters 1,2,3, Appendix AB,G...(all material upto/including EPIC/VLIW), Homeworks 1,2,3.
- Looks like?
  - 3 parts: (1) multiple choice, (2) short answers (2 sentence), and (3) detailed/analytical questions (like homeworks?)
- If you miss exam then makeup IF you have valid excuse
  - Sickness, your company sent you out on travel, others?
- NOT valid excuses:
  - You need to get your car serviced at the same time as exam
  - You have a party to attend the previous night