Announcements

Homework #1 due today
Homework #2 & Reader #2 coming on Wednesday
Project descriptions coming soon
Compiler 101

Compiler goals:
- all correct programs execute correctly
- most compiled programs execute fast
- compile fast
- provide support for debugging

Multiple phases to manage complexity
- Lexical analysis
- Parsing => Intermediate representation
- Optimization
  - Procedure inlining
  - Loop optimizations
  - Common sub-expression elimination
  - Jump optimization
  - Constant propagation
  - Register allocation
  - Strength reduction
  - Pipeline scheduling
  - Parallelism extraction
- Code generation
William Wulf’s Paper

What compiler writers want?
- regularity
- orthogonality
- composability

Compilers perform a giant case analysis
- too many choices make it hard

Orthogonal instruction sets
- operation, addressing mode, data type

One solution or all possible solutions
- 2 branch conditions, eq and lt
- or all six conditions, eq, ne, lt, gt, le, ge
- not 3 or 4

Primitives not solutions
“...by giving too much semantic content to the instruction, the machine designer made it possible to use the instruction only in limited contexts. In many cases the complex instructions are synthesized from more primitive operations, which if the compiler had access to, could be recomposed to more closely model the feature actually needed.”
Bhandarkar’s and Clark’s Paper

VAX 8700 vs MIPS M/R2000
Combines 3 features:
- Architecture
- Implementation
- Compilers and OS

Argues that
- implementation effects are second order
- compilers are similar
- RISCs are better than CISCs: fair comparison?

RISC factor: \( \frac{\text{CPI}_{\text{VAX}} \times \text{Instr}_{\text{VAX}}}{\text{CPI}_{\text{MIPS}} \times \text{Instr}_{\text{MIPS}}} \)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instruction ratio</th>
<th>CPI_{MIPS}</th>
<th>CPI_{VAX}</th>
<th>CPI ratio</th>
<th>RISC factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li</td>
<td>1.6</td>
<td>1.1</td>
<td>6.5</td>
<td>6.0</td>
<td>3.7</td>
</tr>
<tr>
<td>Eqntott</td>
<td>1.1</td>
<td>1.3</td>
<td>4.4</td>
<td>3.5</td>
<td>3.3</td>
</tr>
<tr>
<td>Fpppp</td>
<td>2.9</td>
<td>1.5</td>
<td>15.2</td>
<td>10.5</td>
<td>2.7</td>
</tr>
<tr>
<td>Tomcatv</td>
<td>2.9</td>
<td>2.1</td>
<td>17.5</td>
<td>8.2</td>
<td>2.9</td>
</tr>
</tbody>
</table>
Bhandarkar’s and Clark’s Paper

RISC vs. CISC

Compensating factors
- increase VAX CPI but decrease VAX instruction count
- increase MIPS instruction count
- e.g. 1: loads/stores vs. operand specifiers

Factors favoring VAX
- big immediate values
- not-taken branches incur no delay

Factors favoring MIPS
- operands specifier decoding
- number of registers
- separate floating-point unit
- simple branches/jumps (lower latency)
- no complex instructions
- instruction scheduling
- translation buffer
- branch displacement size
Smith’s and Weiss’s Paper

Two RISCs: Alpha and PowerPC

“At this point many RISC purists will undoubtedly claim that this is not a RISC design.....this is a second generation RISC, representing a reasonable melding of RISC and CISC concepts likely to be the direction for many future RISC designs”

P. Hester, RS/6000 Hardware Background & Philosophies

“We re-applied the principles of RISC to processor design to get maximum clock speed”

R. Sites, RISC enters new generation. An insider’s look at the development of DEC’s Alpha 1

Smith’s and Weiss’s Paper

Recall the Iron Law

Time = Instr count x CPI x clock speed

Alpha optimizes the last two (emphasis on the 3rd)

PowerPC optimizes the first two
Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>PowerPC</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Arch</td>
<td>Ld/St</td>
<td>Ld/St</td>
</tr>
<tr>
<td>Instr Length</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>Byte/Halfword/ ld/st</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Condition Codes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Conditional Moves</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Integer Registers</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Integer Register Size</td>
<td>32/64 bits</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

Features

- Instruction formats similar
  - PowerPC has slightly longer immediate field
  - PowerPC provides 3 source FP regs
  - Alpha has longer branch/jump displacements

- Loads and stores
  - Alpha has only register+displacement
  - PowerPC has register+register
  - PowerPC has update addressing
**ISA Comparisons**

**Floating point**
- PowerPC supports A*B+C and A*B-C instructions

**Data Alignment**
- PowerPC allows any alignment
- Alpha faster if aligned
- Alpha uses multiple instruction sequence if unaligned

**Byte operations**
- PowerPC has byte ld/st
- Alpha has only ld/st 32 & 64 bits
- More instructions (insert/extract) for other lengths

**Branches**
- PowerPC uses explicit condition codes and CTR
- Alpha uses general registers