Evolution of ISAs

Characterizing ISAs

• Number of operands per instruction:
  • how many operands are specified per instruction
  • is the number fixed/variable

• Number of addresses per instruction: how many operands may be memory addresses

• Regularity of format:
  • Variable/fixed length instructions
  • Few or many formats

• Number of addressing modes
  • Registers: special/general purpose, are they implied in instructions?

Tour of the Addressing Modes

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Register</td>
<td>$6</td>
<td>Register($6)</td>
</tr>
<tr>
<td>Register deferred</td>
<td>[6]</td>
<td>Memory([Reg[6]])</td>
</tr>
<tr>
<td>Base/Displacement</td>
<td>-[6]</td>
<td>Memory([Reg[6]] + 100)</td>
</tr>
<tr>
<td>PC-relative</td>
<td>-[6PC]</td>
<td>PC + 100</td>
</tr>
<tr>
<td>Deferred</td>
<td>[6][100]</td>
<td>Memory([Reg[6]] + 100)</td>
</tr>
</tbody>
</table>

Accumulator Machines

• Early machines and many microcontrollers use an implied register called an accumulator
  • Operands per instruction: at most 1
  • Addresses per instruction: at most 1
  • Formats: variable length, few formats for ease of programming
  • Addr Modes: few (typically immediate and PC relative)
  • Registers: one, implied

• Encode A = B + C
  load addressB
  add addressC
  store addressA

Stack Machines

• Machines where data is on an implied stack
  • Operands per instruction: at most 1
  • Addresses per instruction: at most 1
  • Formats: variable length, few formats for ease of programming
  • Addr Modes: few (typically immediate and PC relative)
  • Registers: none (but there are often hidden registers for performance)

• Encode A = B + C
  push addressB
  push addressC
  add
  pop addressA

CISC Machines

• Intel x86, Motorola 68x0 are examples
  • They are register-memory architectures (some operands may be memory addresses)
    • Operands per instruction: variable, up to 2
    • Addresses per instruction: 1
    • Formats: variable length (x86 is between 1 and 17 bytes), many formats
    • Addr Modes: x86 has at least 7, 68k has more
    • Registers: usually some special purpose and some general

• Encode A = B + C
  load r1, address
  add r1, address
  store r1, address
**True CISC**

- The VAX was/is the ultimate CISC machine
- Operands per instruction: variable, up to 3
- Addresses per instruction: variable, up to 3
- Formats: variable length (1 to 54 bytes!), many formats
- Addr Modes: more than 10
- Registers: 16 general purpose
- Encoding A=B+C is easy: ADD addressA, addressB, addressC
- VAX included loop instructions, as well as call & return
- VAX was an orthogonal instruction set -- very complicated implementation

**RISCs**

- Typically load-store architectures
- Operands per instruction: 3
- Addresses per instruction: 0
- Formats: few formats, fixed length
- Addr Modes: few (usually 3 or 4)
- Registers: many general purpose
- Encoding A=B+C.

**Summary**

<table>
<thead>
<tr>
<th></th>
<th>Accumulator</th>
<th>Stack</th>
<th>CISC</th>
<th>RISC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implementation</strong></td>
<td>easy</td>
<td>easy</td>
<td>hard</td>
<td>easy</td>
</tr>
<tr>
<td><strong>Instruction density</strong></td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td><strong>Assembly coding</strong></td>
<td>medium</td>
<td>medium</td>
<td>easy</td>
<td>excessive</td>
</tr>
<tr>
<td><strong>Compilation</strong></td>
<td>medium</td>
<td>easy</td>
<td>easy</td>
<td>hard</td>
</tr>
<tr>
<td><strong>Memory overhead</strong></td>
<td>high</td>
<td>high</td>
<td>highest!?</td>
<td>lowest</td>
</tr>
<tr>
<td><strong>Instruction count</strong></td>
<td>medium</td>
<td>medium</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td><strong>CPI</strong></td>
<td>medium</td>
<td>medium</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td><strong>Cycle time</strong></td>
<td>medium</td>
<td>medium</td>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>

**Trends**

- The 1960s: expensive memory, poor compilers, poor implementation technologies:
  - Goals: simple compilers/assemblers, simple implementation, good density
  - Results: simple ISAs, regular formats, compact encoding
- The 1970s: better implementation technologies, poor compilers, expensive (but fast) memory, high software costs
  - Goals: simple compilers, high code density, easy assembly coding
  - Results: powerful ISAs, irregular formats, complicated implementation
- The 1980s: improved implementation, compilers, cheap (slow) memory
  - Goals: high performance by pipelining, simple implementations
  - Results: simple ISAs, regular formats, lots of registers