Instruction Set Architecture

Virendra Singh
Associate Professor

Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay

http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

Computer Organization & Architecture

Lecture 5 (22 March 2013)
What Are the Components of an ISA?

- Sometimes known as *The Programmer’s Model* of the machine
- Storage cells
  - General and special purpose registers in the CPU
  - Many general purpose cells of same size in memory
  - Storage associated with I/O devices
- The machine instruction set
  - The instruction set is the entire repertoire of machine operations
  - Makes use of storage cells, formats, and results of the fetch/execute cycle
  - i.e., register transfers
What Are the Components of an ISA?

• The instruction format
  - Size and meaning of fields within the instruction

• The nature of the fetch-execute cycle
  - Things that are done before the operation code is known
Instruction

- **C Statement**
  
  \[ f = (g+h) - (i+j) \]

- **Assembly instructions**
  
  - `add t0, g, h`
  - `add t1, l, j`
  - `sub f, t0, t1`

- **Opcode/mnemonic, operand, source/destination**
Why not Bigger Instructions?

• Why not “f = (g+h) – (i+j)” as one instruction?
• Church’s thesis: A very primitive computer can compute anything that a fancy computer can compute – you need only logical functions, read and write to memory, and data dependent decisions
• Therefore, ISA selection is for practical reasons
  – Performance and cost not computability
• Regularity tends to improve both
  – E.g, H/W to handle arbitrary number of operands is complex and slow, and UNNECESSARY
What Must an Instruction Specify?(I)

• Which operation to perform  \textbf{add} r0, r1, r3
  – Ans: Op code: add, load, branch, etc.

• Where to find the operands: add r0, \textcolor{red}{r1}, \textcolor{red}{r3}
  – In CPU registers, memory cells, I/O locations, or part of instruction

• Place to store result  \textbf{add} r0, r1, r3
  – Again CPU register or memory cell
What Must an Instruction Specify?(II)

• Location of next instruction

  add r0, r1, r3
  br endloop

  – Almost always memory cell pointed to by program counter—PC

• Sometimes there is no operand, or no result, or no next instruction. Can you think of examples?
Instructions Can Be Divided into 3 Classes (I)

• Data movement instructions
  – Move data from a memory location or register to another memory location or register without changing its form
  – Load—source is memory and destination is register
  – Store—source is register and destination is memory

• Arithmetic and logic (ALU) instructions
  – Change the form of one or more operands to produce a result stored in another location
  – Add, Sub, Shift, etc.

• Branch instructions (control flow instructions)
  – Alter the normal flow of control from executing the next instruction in sequence
  – Br Loc, Brz Loc2,—unconditional or conditional branches
Use of Computers

- Desktop
  - Performance of program with Floating point, integer data type
  - Little regard to program size
- Servers
  - Data bases, file servers. Etc.
  - Integer and character strings
- Embedded Systems
  - Values cost, energy, code size
ISA Classification

• Type of internal storage in a processor is the most basic differentiator
• Stack Architecture
• Accumulator Architecture
• General Purpose Register Architecture
## ISA Classification

<table>
<thead>
<tr>
<th># Memory Address</th>
<th>Max. no. of operands allowed</th>
<th>Type of architecture</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Load-Store</td>
<td>Alpha, ARM, MIPS, PowerPC</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Reg-Mem</td>
<td>IBM360, Intel x86, 68000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Mem-Mem</td>
<td>VAX</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Mem-Mem</td>
<td>VAX</td>
</tr>
</tbody>
</table>
## ISA Classification

<table>
<thead>
<tr>
<th>Type</th>
<th>Adv</th>
<th>Disadv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg-Reg</td>
<td>Simple, fixed length encoding, simple code generation, all instr. Take same no. of cycles</td>
<td>Higher instruction count, lower instruction density</td>
</tr>
<tr>
<td>Reg-Mem</td>
<td>Data can be accessed without separate load instruction first, instruction format tend to be easy to encode and yield good density</td>
<td>Encoding register no and memory address in each instruction may restrict the no. of registers.</td>
</tr>
<tr>
<td>Mem-Mem</td>
<td>Most compact, doesn’t waste registers for temporaries</td>
<td>Large variation in instruction size, large variation in in amount of work (NOT USED TODAY)</td>
</tr>
</tbody>
</table>
Thank You