Computer Organization
CS 231-01

Instruction Set Architecture Part 2

Dr. William H. Robinson
December 1, 2004

http://eecs.vanderbilt.edu/courses/cs231/

Topics

“Now throw the switch and let us begin the battle for the planet.”

– The Brain
(Pinky and the Brain)

• Instruction Set Architecture design
• Administrative stuff
  – Programming assignment #3
  – TA evaluation

From Last Time...

• Described purpose of ISA
  – The common language between hardware and software
• Introduced specifications for ISA
• Operations supported by ISA
• Location of operands/results
  – Registers
  – Addressing modes

Instruction Set Architecture Level

• Interface between software and hardware
• Both compilers and hardware must understand ISA
  – Compilers translate high-level language into object code
  – Hardware must directly execute or interpret ISA
ISA: What Must be Specified?

- Instruction format or encoding
  - how is it decoded?
- Location of operands and result
  - where other than memory?
  - how many explicit operands?
  - how are memory operands located?
  - which operands can or cannot be in memory?
- Data type and size
- Operations
  - what are supported
- Successor instruction
  - jumps, conditions, branches
  - fetch-decode-execute is implicit!

Typical Operations (little change since 1960)

<table>
<thead>
<tr>
<th>Data Movement</th>
<th>Load (from memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Store (to memory)</td>
</tr>
<tr>
<td></td>
<td>memory-to-memory move</td>
</tr>
<tr>
<td></td>
<td>register-to-register move</td>
</tr>
<tr>
<td></td>
<td>input (from I/O device)</td>
</tr>
<tr>
<td></td>
<td>output (to I/O device)</td>
</tr>
<tr>
<td></td>
<td>push, pop (to/from stack)</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>integer (binary + decimal) or FP</td>
</tr>
<tr>
<td></td>
<td>Add, Subtract, Multiply, Divide</td>
</tr>
<tr>
<td>Shift</td>
<td>shift left/right, rotate left/right</td>
</tr>
<tr>
<td>Logical</td>
<td>not, and, or, set, clear</td>
</tr>
<tr>
<td>Control (Jump/Branch)</td>
<td>unconditional, conditional</td>
</tr>
<tr>
<td>Subroutine Linkage</td>
<td>call, return</td>
</tr>
<tr>
<td>Interrupt</td>
<td>trap, return</td>
</tr>
<tr>
<td>Synchronization</td>
<td>test &amp; set (atomic r-m-w)</td>
</tr>
<tr>
<td>String</td>
<td>search, translate</td>
</tr>
<tr>
<td>Graphics (MMX)</td>
<td>parallel subword ops (4 16-bit add)</td>
</tr>
</tbody>
</table>

Operation Summary

Support these simple instructions, since they will dominate the number of instructions executed:

load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch, jump, call, return;

Instruction Types (from Tanenbaum)

- Data movement
- Dyadic (two operands)
- Monadic (one operand)
- Comparisons and conditional branches
- Procedure calls
- Loop control
- Input/output
Specifying Operand Locations

- **Zero operands**
  - Stack architecture

- **One operand (accumulator)**
  - Accumulator = Accumulator + MemoryRef

- **Two operands**
  - Register2 = Register2 + Source1

- **Three operands**
  - Destination = Source1 + Source2

### Basic ISA Classes

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Add R4,R3</td>
<td>R4 ← R4+R3</td>
</tr>
<tr>
<td>Immediate</td>
<td>Add R4,#3</td>
<td>R4 ← R4+3</td>
</tr>
<tr>
<td>Displacement</td>
<td>Add R4,100(R1)</td>
<td>R4 ← R4+Mem[100+R1]</td>
</tr>
<tr>
<td>Register indirect</td>
<td>Add R4,(R1)</td>
<td>R4 ← R4+Mem[R1]</td>
</tr>
<tr>
<td>Base-Indexed</td>
<td>Add R3,(R1+R2)</td>
<td>R3 ← R3+Mem[R1+R2]</td>
</tr>
<tr>
<td>Direct or absolute</td>
<td>Add R1,(1001)</td>
<td>R1 ← R1+Mem[1001]</td>
</tr>
<tr>
<td>Memory indirect</td>
<td>Add R1,@(R3)</td>
<td>R1 ← R1+Mem[Mem[R3]]</td>
</tr>
<tr>
<td>Post-increment</td>
<td>Add R1,(R2)+</td>
<td>R1 ← R1+Mem[R2]; R2 ← R2+d</td>
</tr>
<tr>
<td>Pre-decrement</td>
<td>Add R1,–(R2)</td>
<td>R2 ← R2–d; R1 ← R1+Mem[R2]</td>
</tr>
<tr>
<td>Scaled</td>
<td>Add R1,100(R2)[R3]</td>
<td>R1 ← R1+Mem[100+R2+R3]*d</td>
</tr>
</tbody>
</table>

Adapted from John Kubiatowicz’s CS 152 lecture notes. Copyright © 2003 UCB.
Addressing Modes

• Based-indexed
  – Adds two registers plus an optional offset

• Direct or absolute
  – Give full memory address of operand

• Stack
  – Memory referenced with a Last-In First-Out (LIFO) queue

Generic Examples: Instruction Format Widths

- Variable:
  ![Variable Format](image1)

- Fixed:
  ![Fixed Format](image2)

- Hybrid:
  ![Hybrid Format](image3)

UltraSPARC II

- Format 1a: DEST OFCODE SRC1 0 FP.OP SRC2
- Format 1b: DEST OFCODE SRC1 1 IMMEDIATE CONSTANT
- Format 2: DEST OP IMMEDIATE CONSTANT SETHI
- Format 3: A|COND OP PC-RELATIVE DISPLACEMENT BRANCH
- Format 4: PC-RELATIVE DISPLACEMENT CALL

Figure 5-14. The original SPARC instruction formats.

Java Virtual Machine (JVM)

- Format 1: OPCODE
- Format 2: OPCLASS BYTE BYTE = index, constant or type
- Format 3: OPCLASS SHORT SHORT = index, constant or offset
- Format 4: OPCLASS INDEX CONST
- Format 5: OPCLASS INDEX DIMENSIONS
- Format 6: OPCLASS INDEX PARAMETERS 0
- Format 7: OPCLASS INDEX CONST
- Format 8: OPCLASS SUB-BRANCH OFFSET
- Format 9: OPCLASS VARIABLE LENGTH

Figure 5-15. The JVM instruction formats.
Instruction Formats

• If code size is most important, use variable length instructions
• If performance is most important, use fixed length instructions
• Recent embedded machines (ARM, MIPS) added optional mode to execute subset of 16-bit wide instructions (Thumb, MIPS16); per procedure decide performance or density
• Some architectures actually exploring on-the-fly decompression for more density.

Data Types

• Some data types may not be supported by H/W
  – Would have to rely on S/W
• Two categories of H/W supported data types
  – Numeric
  – Non-numeric

Numeric Data Types

• Integers
  – Signed and unsigned
  – Lengths of 8, 16, 32, and 64 bits
• Floating point
  – Follow IEEE Standard 754
  – Lengths of 32, 64, and 128 bits
• Decimal
  – Binary coded decimal for COBOL
  – Pack 2 decimal values per byte

Non-Numeric Data Types

• Characters
  – ASCII
  – UNICODE
• Boolean
  – Zero means FALSE
  – Everything else means TRUE
• Pointers
  – Machine address
Data Types

Bit: 0, 1

Bit String: sequence of bits of a particular length
- 4 bits is a nibble
- 8 bits is a byte
- 16 bits is a half-word
- 32 bits is a word
- 64 bits is a double-word

Character:
- ASCII: 7 bit code
- UNICODE: 16 bit code

Decimal:
- digits 0-9 encoded as 0000b thru 1001b
- two decimal digits packed per 8 bit byte

Integers:
- 2’s Complement

Floating Point:
- Single Precision
- Double Precision
- Extended Precision
- M x R
- exponent
- mantissa
- Where is decimal pt?
- How many +/- #’s?
- How are +/- exponents represented?

Overflow Using Two’s Complement

- When the leftmost bits of the operands are the same, the leftmost bit of the answer must also be the same, otherwise overflow has occurred.

OR...looked at another way...

- The carry-in bit to the leftmost bit must be the same as the carry-out bit from the leftmost bit.

IEEE Floating Point Standard 754

- Provided designers with a correct model
- Allowed FP data to be exchanged among different computer systems
- Defines three formats
  - Single precision (32 bits)
  - Double precision (64 bits)
  - Extended precision (80 bits)
    - Only occurs within FP units

IEEE Numerical Types

- Underflow handled gracefully
  - Use denormalized numbers instead of jumping to zero
- Overflows becomes infinity
- Includes Not a Number (NaN)
Operand Size Usage (32-bit Word)

- Support for these data sizes and types:
  - 8-bit, 16-bit, 32-bit integers and
  - 32-bit and 64-bit IEEE 754 floating point numbers

Summary

- ISA is the common language between the compiler and the hardware

- ISA must support/define
  - Instruction format or encoding
  - Location of operands and result
  - Data type and size
  - Operations
  - Successor instruction