ISA is the HW/SW interface

ISA choice determines:

• program size (& memory size)
• complexity of hardware (CPI and f)
• execution time for different applications and domains
• power consumption
• die area (cost)
Stored program concept (von Neumann model)

- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs
Steps in execution of a program

- What is instruction format / size?
- How is it decoded?
- Where are the operands located? What are their sizes?
- What are supported operations?
- How to determine the successor instruction?

1. Fetch instruction @ PC
2. Decode instruction
3. Fetch Operands
4. Execute instruction
5. Store result
6. Update PC
Example of an instruction

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

add $t0, $s1, $s2

**assembly language**

<table>
<thead>
<tr>
<th>special</th>
<th>$s1</th>
<th>$s2</th>
<th>$t0</th>
<th>0</th>
<th>add</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>18</td>
<td>8</td>
<td>0</td>
<td>32</td>
</tr>
</tbody>
</table>

| 000000 | 10001 | 10010 | 01000 | 00000 | 100000 |

0000001000110010010000000001000000₂ = 02324020₁₆

**machine language**
ISA design choices

- Number, size (fixed/variable) and format of instructions
- Operations supported (arithmetic, logical, string, floating point, jump, etc)
- Operands supported (bytes, words, signed, unsigned, floating, etc)
- Operand storage (accumulator, stack, registers, memory)
- Addressing modes
## Typical operations

**Data Movement**
- Load (from memory)
- Store (to memory)
- memory-to-memory move
- register-to-register move
- input (from I/O device)
- output (to I/O device)
- push, pop (to/from stack)

**Arithmetic**
- integer (binary + decimal) or FP
- Add, Subtract, Multiply, Divide

**Shift**
- shift left/right, rotate left/right

**Logical**
- not, and, or, set, clear

**Control (Jump/Branch)**
- unconditional, conditional

**Subroutine Linkage**
- call, return

**Interrupt**
- trap, return

**Synchronization**
- test & set (atomic r-m-w)

**String**
- search, translate
x86 ISA

- Backward compatibility ⇒ instruction set doesn’t change
  - But they do accumulate more instructions

<table>
<thead>
<tr>
<th>Rank</th>
<th>Instruction</th>
<th>Integer</th>
<th>Average</th>
<th>Percent</th>
<th>Total executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>load</td>
<td></td>
<td></td>
<td>22%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>conditional branch</td>
<td></td>
<td></td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>compare</td>
<td></td>
<td></td>
<td>16%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>store</td>
<td></td>
<td></td>
<td>12%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td></td>
<td></td>
<td>8%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>and</td>
<td></td>
<td></td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sub</td>
<td></td>
<td></td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>move register-register</td>
<td></td>
<td></td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>call</td>
<td></td>
<td></td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>return</td>
<td></td>
<td></td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
<td>96%</td>
<td></td>
</tr>
</tbody>
</table>

- Simple instructions dominate instruction frequency
Classification of ISAs

[Figure from D. Brooks -- Harvard]

These ISAs give different characteristics in terms of size of programs, number of instructions and CPI.
Examples of ISA

Instruction sequence for $C = A + B$ for the four ISAs

<table>
<thead>
<tr>
<th>Stack</th>
<th>Accumulator</th>
<th>Register (register-memory)</th>
<th>Register (load-store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push A</td>
<td>Load A</td>
<td>Load R1, A</td>
<td>Load R1, A</td>
</tr>
<tr>
<td>Push B</td>
<td>Add B</td>
<td>Add R1, B</td>
<td>Load R2, B</td>
</tr>
<tr>
<td>Add</td>
<td>Store C</td>
<td>Store C, R1</td>
<td>Add R3, R1, R2</td>
</tr>
<tr>
<td>Pop C</td>
<td></td>
<td>Store C, R1</td>
<td>Store C, R3</td>
</tr>
</tbody>
</table>

Some architectures (e.g. x86) support hybrid ISAs for different classes of instructions and/or for backward compatibility.
What makes a good ISA?

- Efficiency of hardware implementation
- Convenience of programming / compiling
- Matches target applications (or generality)
- Compatibility and portability

Four design principles for ISA

1. Simplicity favors regularity
2. Smaller is faster
3. Make the common case fast
4. Good design demands good compromises

*ISA design is an art!*
Popular ISAs

- x86 from Intel (laptops, servers)
- ARM (mobile devices)
- MIPS (embedded devices)
- Power and PowerPC from IBM (servers, old Macs)
- and many others still spoken and dead ISAs