Appendix A: Instruction Set Principles
Chapter Overview

- Classification
- Memory Addressing
- Operand Types
- Basic Operations
- Encoding Instruction Sets
- The Role of Compilers
- Putting it all together: The RISC-V Architecture
Classification

- General-purpose register (GPR) architectures
  - Register-memory architectures
  - Load-store architectures
- Stack architectures
- Memory-memory architectures
- Accumulator architecture
Memory Addressing

- Little Endian (word address = rightmost byte address)
- Big Endian (word address = leftmost byte address)
- The significance of alignment
The book begins to really open up and clearly show examples of quantitative measures to support design decisions. Examine the graphs of quantitative measures and corresponding prose closely. Establish this habit for the remainder of your studies in this textbook.
Frequency of Addressing Modes

- **Memory indirect**
  - TeX: 1%
  - spice: 6%
  - gcc: 1%
- **Scaled**
  - TeX: 0%
  - spice: 16%
  - gcc: 6%
- **Register indirect**
  - TeX: 3%
  - spice: 24%
  - gcc: 11%
- **Immediate**
  - TeX: 3%
  - spice: 43%
  - gcc: 39%
- **Displacement**
  - TeX: 3%
  - spice: 55%
  - gcc: 40%

Frequency of the addressing mode: 0% 10% 20% 30% 40% 50% 60%
Distance of Operand Displacements

![Graph showing distance of operand displacements. The graph plots percentage of displacement against number of bits of displacement. There are two lines: Integer average and Floating-point average. The graph shows a decreasing trend for integer average as the number of bits increases, while the floating-point average has a more fluctuating pattern with peaks at certain bit numbers.](image-url)
Frequency of Immediates

- Loads: 23% (Floating-point average), 22% (Integer average)
- ALU operations: 25% (Floating-point average), 19% (Integer average)
- All instructions: 21% (Floating-point average), 16% (Integer average)
Size of Immediates

- Percentage of immediates
  - Floating-point average
  - Integer average

Number of bits needed for immediate

- 0% to 45%
- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Note: The graph shows the distribution of immediates in terms of percentage and the number of bits needed for both floating-point and integer averages.
<table>
<thead>
<tr>
<th>Rank</th>
<th>80x86 instr</th>
<th>Frequency (% total executed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>load</td>
<td>22%</td>
</tr>
<tr>
<td>2</td>
<td>conditional branch</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>compare</td>
<td>16%</td>
</tr>
<tr>
<td>4</td>
<td>store</td>
<td>12%</td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td>8%</td>
</tr>
<tr>
<td>6</td>
<td>and</td>
<td>6%</td>
</tr>
<tr>
<td>7</td>
<td>sub</td>
<td>5%</td>
</tr>
<tr>
<td>8</td>
<td>move (reg-to-reg)</td>
<td>4%</td>
</tr>
<tr>
<td>9</td>
<td>call</td>
<td>1%</td>
</tr>
<tr>
<td>10</td>
<td>return</td>
<td>1%</td>
</tr>
</tbody>
</table>
Control Flow Instructions

- Call/return: 8% (Floating-point average), 19% (Integer average)
- Jump: 10% (Floating-point average), 6% (Integer average)
- Conditional branch: 82% (Floating-point average), 75% (Integer average)

Frequency of branch instructions:
- 0% to 25%
- 25% to 50%
- 50% to 75%
- 75% to 100%
Bits of Branch Displacement

- Integer average
- Floating-point average

Percentage of distance vs. Bits of branch displacement graph.
Relational Tests in Conditional Branches

- **Greater than or equal**: 16% + 11% = 27%
- **Less than or equal**: 33% + 34% = 67%
- **Not equal**: 5% + 2% = 7%

**Frequency of comparison types in branches**

- **Floating-point average**
  - Greater than or equal: 0%
  - Less than or equal: 44%
  - Not equal: 5%
- **Integer average**
  - Greater than or equal: 11%
  - Less than or equal: 35%
  - Not equal: 2%
- PC relative addressing
- Indirect jumps
- Caller saving/Callee saving
- Instruction set encoding: 16-bit instrs vs compression
  - impact on caches
The Role of Compilers

- Optimization (multi-pass [phases])
- Effective optimization has dramatic impact: 25%-90%
- Register allocation
- How to help the compiler writer:
  - Provide regularity
  - Provide primitives not solutions
  - Simplify trade-offs among alternatives
  - Provide abilities to bind quantities known at compile time
## Impact of Compiler Optimization

<table>
<thead>
<tr>
<th>Program, compiler optimization level</th>
<th>Branches/calls</th>
<th>Floating-point ALU ops</th>
<th>Loads-stores</th>
<th>Integer ALU ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcf, level 0</td>
<td>100%</td>
<td>0%</td>
<td>20%</td>
<td>40%</td>
</tr>
<tr>
<td>mcf, level 1</td>
<td>76%</td>
<td>76%</td>
<td>84%</td>
<td>100%</td>
</tr>
<tr>
<td>mcf, level 2</td>
<td>84%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>mcf, level 3</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>lucas, level 0</td>
<td>100%</td>
<td>11%</td>
<td>21%</td>
<td>100%</td>
</tr>
<tr>
<td>lucas, level 1</td>
<td>76%</td>
<td>12%</td>
<td>21%</td>
<td>100%</td>
</tr>
<tr>
<td>lucas, level 2</td>
<td>76%</td>
<td>12%</td>
<td>21%</td>
<td>100%</td>
</tr>
<tr>
<td>lucas, level 3</td>
<td>100%</td>
<td>11%</td>
<td>21%</td>
<td>100%</td>
</tr>
</tbody>
</table>
The RISC-V instruction set is used throughout the textbook in examples and to discuss program structures that can impact design decisions. You will not be examined on your knowledge of the RISC-V instruction set, but familiarity with it will assist you in later chapters of this textbook.