Assignment 2
Solution Set
CS152 Computer Architecture and Engineering

3.14 Instruction Reads = 1/instruction
Data Reads = \( \frac{2}{3} + \frac{1}{3} \times \frac{3}{3} \) /instruction
Data Reads = \( \frac{1}{3} + \frac{3}{3} \) /instruction

Total Memory Accesses = InstructionReads + DataReads + DataWrites = 1.33/instruction
Total Instruction Memory Accesses = InstructionReads = 1/instruction
Total Data Memory Accesses = DataReads + DataW rites = .33/instruction
Total Read Memory Accesses = DataReads + DataW rites = 1.22/instruction
Total Write Memory Accesses = DataW rites = .11/instruction

a. \( \frac{33}{133} = .24 \) or 24% of accesses for data
b. \( \frac{122}{133} = .91 \) or 91% of accesses are reads

3.19 Initially all data in memory. At the end all results in memory.

- **Accumulator:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instr bytes</th>
<th>Data Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. load addressB</td>
<td>1+2=3</td>
<td>4</td>
</tr>
<tr>
<td>2. add addressC</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3. store addressA</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4. add addressC</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5. store addressB</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6. load addressA</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7. sub addressB</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8. store addressD</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>24</strong></td>
<td><strong>32</strong></td>
</tr>
</tbody>
</table>

Each instruction is 3 bytes long (1 byte for opcode, 2 for address), so 24 bytes of code fetched.
Each instruction loads or stores 4 bytes of data, so 32 bytes of data transferred.

- **Memory-memory:**
Each instruction is 7 bytes long (1 opcode, 3 address), so 21 bytes of code fetched. Each instruction loads 8 bytes and stores 4, so 36 bytes of data transferred.

### Stack:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instr bytes</th>
<th>Data Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>push addressB</td>
<td>1+2=3</td>
<td>4</td>
</tr>
<tr>
<td>push addressC</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>pop addressA</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>push addressA</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>push addressC</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>pop addressB</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>push addressA</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>push addressB</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>pop addressA D</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>30</td>
<td>36</td>
</tr>
</tbody>
</table>

Push and pop instructions are 3 bytes long, add and sub just 1 byte. So, 30 bytes of code fetched. Each push and pop transfers 4 bytes of data, so 36 bytes of data transferred.

### Load-store:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instr bytes</th>
<th>Data Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>load $r1, addressB</td>
<td>1+.5=2=4</td>
<td>4</td>
</tr>
<tr>
<td>load $r2, addressC</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>add $r3, $r2, $r1</td>
<td>1+.5+.5+.5=3</td>
<td>0</td>
</tr>
<tr>
<td>add $r2, $r3, $r2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>sub $r4, $r3, $r1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>store $r1, addressB</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>store $r3, addressA</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>store $r4, addressD</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>29</td>
<td>20</td>
</tr>
</tbody>
</table>

Each load/store instruction is 4 bytes long, each add/sub is 3 bytes long. So, 29 bytes of code fetched. Each load or store transfers 4 bytes of data, so 20 bytes of data transferred.

The memory-memory architecture is the most efficient in terms of code size. The load-store architecture is the most efficient in terms of total bytes fetched. Memory-memory has very powerful instructions but has to fetch all operands from memory. Load-store takes advantage of registers in order to reuse operands fetched as long as they are not overwritten.

A.3 These registers are considered by the OS as unused by user programs so it never saves and restores them. If using any OS and there is a chance that the user program will get interrupted