Instruction Set Architecture

Notation:

\[ a \leftarrow b: \text{the value of } b \text{ is placed in } a. \]

\[(a): \text{the (memory) location whose address is contained in } a.\]

ra, etc. denote registers, PC is the programme counter.

The numeric value X is denoted by \#X.

Here are some typical instructions in assembly language (using three addresses).

\[
\begin{align*}
\text{LD} & \text{ ra, (rb) } \quad \text{ra} \leftarrow (\text{rb}) \\
\text{ST} & \text{ (ra), rb } \quad (\text{ra}) \leftarrow \text{rb} \\
\text{MOV} & \text{ ra, rb } \quad \text{ra} \leftarrow \text{rb} \\
\text{ADD} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} + \text{rc} \\
\text{SUB} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} - \text{rc} \\
\text{MUL} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} \times \text{rc} \\
\text{DIV} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} / \text{rc} \\
\text{AND} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} \& \text{rc} \\
\text{OR} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} | \text{rc} \\
\text{NOT} & \text{ ra, rb } \quad \text{ra} \leftarrow \neg \text{rb} \\
\text{ASH} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} \text{ (arithmetically) shifted by } \text{rc} \text{ positions} \\
\text{LSH} & \text{ ra, rb, rc } \quad \text{ra} \leftarrow \text{rb} \text{ (logically) shifted by } \text{rc} \text{ positions} \\
\text{BR} & \text{ ra } \quad \text{PC} \leftarrow \text{ra} \\
\text{BEQ} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is equal to } \text{rc} \\
\text{BNE} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is not equal to } \text{rc} \\
\text{BLT} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is less than } \text{rc} \\
\text{BGT} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is greater than } \text{rc} \\
\text{BLE} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is less than or equal to } \text{rc} \\
\text{BGE} & \text{ ra, rb, rc } \quad \text{PC} \leftarrow \text{ra if } \text{rb is greater than or equal to } \text{rc} \\
\end{align*}
\]

Most instructions have a floating-point version when special floating-point registers are used. Some instructions have a version where a specific value is the operand (immediate addressing), e.g., \text{MOV ra, } \#X \text{ means } \text{ra} \leftarrow \#X \text{ and } \text{BR } \#I1 \text{ means to jump to instruction } I1 \text{ (i.e., load it into PC).}

Some architectures use two addresses, where the first argument serves both as the target and one of the sources, e.g., \text{ADD ra, rb} \text{ means } \text{ra} \leftarrow \text{ra} + \text{rb}.

We will use a five-stage pipeline: IF (instruction fetch), ID (instruction decode), RR (register read), EX (execute instruction), WB (write back result into register). We will assume that for data-movement instructions the data transfer happens in the execute stage.
Example

Write an assembly program that computes $5 + (3 \times 7) - 8$ and show the delay slots when executed on a five-stage pipeline. Try to minimize the number of registers and the number of delay slots.

Here is the code:

I1  MOV r1, #3
I2  MOV r2, #7
I3  MUL r1, r1, r2
I4  MOV r2, #5
I5  ADD r1, r1, r2
I6  MOV r2, #8
I7  SUB r1, r1, r2

and here is the diagram showing delay slots:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I4</td>
<td>I5</td>
<td>I6</td>
<td>I7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I5</td>
<td>I6</td>
<td>I7</td>
<td>I7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I5</td>
<td>I6</td>
<td>I7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I6</td>
<td>I6</td>
<td>I7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td>I1</td>
<td>I2</td>
<td>I3</td>
<td>I4</td>
<td>I5</td>
<td>I6</td>
<td>I6</td>
<td>I7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

Write an assembly program that computes \(((10 \times 8) + 4) - 7)^2\) and show the delay slots when executed on a five-stage pipeline. Try to minimize the number of registers and/or the number of delay slots.

Comprehensive Exercise

Consider the computation 
\((10 \times 8) + (4 - 7)\).

1. Write a program in assembly language that performs the above computation. Try to use a minimal number of registers (assume that initially all registers contain 0).

2. Draw a diagram showing the execution of your program in a five-stage pipeline.

3. Identify the dependencies in your code.

4. Remove as many dependencies as possible from your code (by using register renaming) and reorder the code to minimize the number of delay slots when executed on a five-stage pipeline.