1. List all the dependences in the following code fragment. Indicate whether the true dependences are loop-carried or not. Show why the loop is not parallel.

For (i=2;i<100;i++) {
    a[i] = b[i] + a[i]; /* S1 */
    c[i-1] = a[i] + d[i]; /* S2 */
    a[i-1] = 2 * b[i]; /* S3 */
    b[i+1] = 2 * b[i]; /* S4 */
}

2. Here is an unusual loop. First, list the dependences and then rewrite the loop so that it is parallel.

For (i=1;i<100;i++) {
    a[i] = b[i] + c[i]; /* S1 */
    b[i] = a[i] + d[i]; /* S2 */
    a[i+1] = a[i] + e[i]; /* S3 */
}

3. Assume the pipeline latencies from below and a one-cycle delayed branch. Unroll the following loop a sufficient number of times to schedule it without any delays. Show the schedule after eliminating any redundant overhead instructions. The loop is a dot product (assuming $f2$ is initially 0) and contains a recurrence. Despite the fact that the loop is not parallel, it can be scheduled with no delays.

Loop: ld $f0, 0($s1)
      ld $f4, 0($s2)
      multd $f0, $f0, $f4
      addd $f2, $f0, $f2
      subi $s1, $s1, 8
      subi $s2, $s2, 8
      bneqz $s1, Loop

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>Another FP ALU op</td>
<td>3</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>Store Double</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
</tbody>
</table>