Problem 1: The program below executes on the DLX implementation shown below. The implementation uses forwarding (bypassing) to avoid some data hazards and stalls to avoid others. All forwarding paths are shown. (If a needed forwarding path is not there, sorry, you’ll have to stall.) A value can be read from the register file in the same cycle it is written. The destination field in the \texttt{beqz} is zero. Instructions are nullled (squashed) in this problem by replacing with \texttt{slt r0,r0,r0}. All instructions stall in the ID stage.

Initially, $r1=0x1000$, $r2=0x2000$, $r3=0x3000$

\begin{verbatim}
MEM[0x1000] = 0xa0, MEM[0x1001] = 0xa1, MEM[0x1002] = 0xa2, etc.
\end{verbatim}

\begin{verbatim}
sub r0, r0, r0
sub r0, r0, r0
sub r0, r0, r0
sub r0, r0, r0
\end{verbatim}

\begin{verbatim}
START: ! START = 0x50
addi r1, r1, #8
lh r2, 2(r1)
sw 4(r1), r2
bneq r2, START (taken)
sub r2, r3, r1
sub r0, r0, r0
sub r0, r0, r0
\end{verbatim}

The table below shows the contents of pipeline registers and changes to architecturally visible registers $r1$-$r31$ over time. Cycle zero is the time that \texttt{addi} is in instruction fetch. The first two columns are completed; fill in the rest of the table. Use a “?” for the value of the “immediate field” of a type R instruction and for the output of the memory when no memory read is performed. Show pipeline register values even if they’re not used. Assume that the ALU performs the branch target computation even though it was already computed in ID. The row labeled “Reg. Chng.” shows a new register value that is available at the beginning of the cycle. If $r0$ is written leave the entry blank.

\textit{Hint: See Spring 1999 HW 3 and Fall 1999 HW 2 for similar problems.}

Completed table appears below. Numbers in table are in hexadecimal.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</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>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>50</td>
<td>54</td>
<td>58</td>
<td>5c</td>
<td>5c</td>
<td>60</td>
<td>50</td>
<td>54</td>
<td>58</td>
<td>5c</td>
<td></td>
</tr>
<tr>
<td>IF/ID.IR</td>
<td>sub</td>
<td>addi</td>
<td>lh</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>slt</td>
<td>bneq</td>
<td>sub</td>
<td>addi</td>
<td>lh</td>
</tr>
<tr>
<td>Reg. Chng.</td>
<td>r0=0</td>
<td>r0=0</td>
<td>r0=0</td>
<td>r0=0</td>
<td>r1=1008</td>
<td>r2=1008</td>
<td>r0=0</td>
<td>r0=0</td>
<td>r0=0</td>
<td>r0=0</td>
<td></td>
</tr>
<tr>
<td>ID/EX.IR</td>
<td>sub</td>
<td>sub</td>
<td>addi</td>
<td>lh</td>
<td>sit</td>
<td>sit</td>
<td>sit</td>
<td>sw</td>
<td>bneq</td>
<td>sub</td>
<td>addi</td>
</tr>
<tr>
<td>ID/EX.A</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1008</td>
<td>1008</td>
<td>1008</td>
<td>1008</td>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>ID/EX.B</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
<td>1008</td>
<td>1008</td>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>ID/EX.IMM</td>
<td>?</td>
<td>?</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>-4</td>
<td>?</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>EX/MEM.IR</td>
<td>sub</td>
<td>sub</td>
<td>sub</td>
<td>addi</td>
<td>lh</td>
<td>sit</td>
<td>sit</td>
<td>sit</td>
<td>sw</td>
<td>bneq</td>
<td>sub</td>
</tr>
<tr>
<td>EX/MEM.ALU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1008</td>
<td>100a</td>
<td>1</td>
<td>1</td>
<td>100c</td>
<td>14</td>
<td>1ff</td>
<td>1ff</td>
</tr>
<tr>
<td>EX/MEM.B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
<td>1008</td>
<td>1008</td>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>MEM/WB.IR</td>
<td>sub</td>
<td>sub</td>
<td>sub</td>
<td>sub</td>
<td>addi</td>
<td>lh</td>
<td>sit</td>
<td>sit</td>
<td>sit</td>
<td>sw</td>
<td>bneq</td>
</tr>
<tr>
<td>MEM/WB.ALU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1008</td>
<td>100a</td>
<td>1</td>
<td>1</td>
<td>100c</td>
<td>14</td>
<td>1ff</td>
</tr>
</tbody>
</table>
LOOP:
!
addi r1, r1, #8
lh r2, 2(r1)
sw 4(r1), r2
bneq r2, LOOP
sub r2, r3, r1
}

Problem 2: The execution of the code in the problem above should suffer a stall (not including the branch delay). Add bypass path(s) to the diagram below needed to avoid the stall(s). Add only the bypass paths needed to avoid the stalls encountered in the problem above, and no others. (The diagram below is the same as the one in the first problem.)

START:
addi r1, r1, #8
lh r2, 2(r1)
sw 4(r1), r2
bneq r2, START
sub r2, r3, r1

Add a bypass path from the output of the WB-stage multiplexor to a new multiplexor feeding the MEM-stage memory data-in port. (The other input to the new multiplexor is from EX/MEM.B.)