ECE 232: Solutions to Test 2

(1)

\[
\begin{align*}
D_0 &= (Q1 \cdot In) + (Q3 \cdot In) = (Q1 + Q3) \cdot In \\
D_1 &= Q0 \cdot \overline{In} \\
D_2 &= (Q0 \cdot In) + ((Q1 + Q2) \cdot \overline{In}) \\
D_3 &= (Q2 \cdot In) + (Q3 \cdot \overline{In})
\end{align*}
\]

(2a)

| Cycles | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| I0     | IF | ID | EX | MEM| WB |     |    |    |    |    |    |    |    |    |    |
| I1     | IF | ID | EX | MEM| WB |     |    |    |    |    |    |    |    |    |    |
| I2     | IF | ID | ID | ID | EX | MEM| WB |    |    |    |    |    |    |    |    |

(2b)

| Cycles | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| I0     | IF | ID | EX | MEM| WB |     |    |    |    |    |    |    |    |    |    |
| I1     | IF | ID | EX | MEM| WB |     |    |    |    |    |    |    |    |    |    |
| I2     | IF | ID | ID | ID | EX | MEM| WB |    |    |    |    |    |    |    |    |

(3) Start by writing out the progress of each of these instructions down the pipeline. There is no dependence between instructions and therefore no stalling.

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In cycle 4, add is in the WB stage, sub in the MEM stage, or in the EX stage, and lw in the ID stage.

IF/ID.PC has the PC value as brought in by lw: from Figure 6.22, this is the output of the adder that produces PC+4, where PC is the address of the lw instruction. Since add has address 256, lw has address 268. Hence, IF/ID.PC has value 268 + 4 = 272.
For EX/MEM.PCAdder, we need to go back to Figure 6.22. The value produced by PCAdder is the sign-extended and shifted-left value of the immediate operand. The relevant instruction is the one which just moved into the MEM stage, namely sub.

We will need the LSB 16 bits of the sub instruction, since that is the part which gets sign-extended. Looking at the instruction format, we can see that the LSB 16 bits consist of the rd, shamt and funct fields of sub. The value of rd is \(12_{\text{ten}} = 01100_{\text{bin}}\). The value of shamt is 0000 and the value of funct can be read off the instruction table: it is 34_{ten} = 100010_{bin}. The LSB 16 bits are therefore 011000000001000100. Sign-extending them involves replicating the MSB of this 16-bit string 16 times, yielding 00000000000000001100000001000100. Shifting left by 2 gives us 00000000000000000000001000010000. This is added to PC+4: the PC of the sub instruction was 260_{ten}, so the PC+4 value is 264_{sub} = 00000000000000000000000100001000_{bin}

The value in EX/MEM.PCAdder is therefore

\[
\begin{align*}
00000000000000001 & 1000000001000100 \\
+ & 00000000000000000100001000 \\
= & 0000000000000000000000011000010000.
\end{align*}
\]

(4a) Nothing needs to be added to the datapath. The adder needs to have the sign-extended LSB 16 bits of the instruction sent to it, and all the links required for that already exist.

(4b) Upon decoding an addi instruction, the machine transits from state 1 of Figure 5.38 to a new state. In this new state (call it state 10), we set ALUSrcA=1, ALUSrcB=10, ALUop=10. Then, go to state 11, where we set RegDst=0, RegWrite=1, MemToReg=0. From state 11 go back to state 0.

(5)

```
Lg:  addi $sp, $sp, -12  # make room on the stack.
    sw  $s1, 8($sp)  # save $s1
    sw  $ra, 4($sp)  # save $ra on the stack.
    sw  $a0, 0($sp)  # save $a0 on the stack.
    sli  $t0, $a0, 1  # set $t0=1 if n<1
    beq  $t0, $zero, L1  # go to L1 if $t0=0 (i.e., if n >= 1)
    addi  $v0, $zero, 1  # prepare to return 1 (for the n<1 case)
    j  L2  # jump to the restoration (cleanup) code
L1:  add  $s1, $a0, $zero  # save n in $s1
    addi  $a0, $a0, -2  # prepare to call g() with argument n-2
    jal  Lg  # call g(n-2)
    add  $v0, $s1, $v0  # place n+g(n-2) in $v0.
L2:  lw  $a0, 0($sp)  # restore n into $a0
    lw  $ra, 4($sp)  # restore $ra register
    lw  $s1, 8($sp)  # restore $s1
    addi  $sp, $sp, 12  # restore stack pointer register
    jr  $ra  # return to calling program
```