This is a solution to Lab 6 which investigated Tomasulo’s algorithm and the use of a scoreboard in dynamically scheduling instructions in a loop. In this lab we are assuming that:

These is one functional unit for all integer operations (add, subtract, multiply, and divide)
There are four separate functional units for floating point add, subtract, multiply, and divide.
Each functional unit has three reservation stations available.
Loads and stores can be done in order by monitoring effective address calculation.
Branch prediction is 100% accurate.

Assume that $t1$ points to the 5-vector $V$ and that $t2$ points to the 5-vector $W$ and that lambda is in $f0$. We are going to calculate labmda * V + W and put the result in $w$. The loop in MIPS assembly is:

```mips
  addi $t3, $zero, 5       # init loop counter to 5

loop:
  1.d $f2, 0($t1)          # load vector V[i]
  mul.d $f4, $f2, $f0      # lambda * V[i]
  1.d $f6, 0($t2)          # load vector W[i]
  add.d $f8, $f6, $f4      # lambda * V[i] + W[i]
  s.d $f8, 0($t2)          # update W[i] = lambda * V[i] + W[i]
  addui $t1, $t1, 8        # update pointer $t1 -> V
  addui $t2, $t2, 8        # update pointer $t2 -> W
  addi $t3, $t3, -1        # decrement loop counter
  bne $t3, $zero, loop
```

Assignment:
Analyze the performance of the above loop with regard to Tomasulo’s algorithm by making a table and running the loop manually for 5 iterations. Assume that it takes one clock cycle for each of the actions: fetch instruction (IF), enter decode queue (DQ), issue instruction, if possible, to reservation station (RS), and write register result (WR). I am assuming that a completed result can be handed off at the end of execution (EX) to a dependent instruction which will start executing at the beginning of the next cycle and I am assuming no local bus contention for all transfers. Assume that all data is in the L1 data cache and that this can be read/written (LD and ST) in one clock cycle. Assume that integer addition and subtraction can be done in one clock cycle (1 EX), that floating point addition takes 4 clock cycles (4 EX), and that floating point multiplication takes 7 clock cycles (7 EX). What could you do in order to speed up this loop? Options might be: additional functional units (integer or floating point), more reservation stations per functional unit, etc. Does it matter in this example whether or not the floating point multiply unit is pipelined?
Solution:

First note that instructions issue to a reservation station as long as they are not writing one of the registers currently being written (we will eventually fix this restriction). Also note that l.d and s.d usually have an EX cycle to compute the address, BUT, since all the offsets below are 0 we can do the LD or ST without needing an extra cycle for address calculation. Look first at the loop itself and note that there is a block of 9 identical clock cycles which will be repeated 5 times. Registers being written and available are noted in parentheses:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Clock Cycles</th>
<th>Registers Written at DQ-&gt;RS Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>l.d $f2, 0($t1)</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>mul.d $f4,$f2,$f0</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>l.d $f6, 0($t2)</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>add.d $f8,$f6,$f4</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>s.d $f8, 0($t2)</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>addi $t1,$t1, 8</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>addi $t2,$t2, 8</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
<tr>
<td>bne $t3,$zero,loop</td>
<td>01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09</td>
<td>-</td>
</tr>
</tbody>
</table>

... (the pattern repeats) ...

Looking at the loop itself and there is a block of 9 identical clock cycles which will be repeated 5 times. Therefore, the total number of clock cycles will be: $4 + 9 \times 5 + 3 = 52$ clock cycles. Assuming that the number of clock cycles per functional unit remains the same there is nothing that can be done to speed up the loop because we are already fetching one instruction per clock cycle. Pipelining the floating point multiply unit is irrelevant since there is never more than one instruction at a time in it.