PROBLEM 1.  [15 points] In this exercise, we examine how pipelining affects the clock cycle time of the processor. Assume that individual stages of the data-path have the following latencies:

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>200ps</td>
<td>120ps</td>
<td>150ps</td>
<td>170ps</td>
<td>100ps</td>
</tr>
</tbody>
</table>

1.1 What is the clock cycle time in a pipelined and non-pipelined processor?
1.2 If you could split one of the pipeline stages into two equal halves, which one would you choose? What is the new cycle time?

PROBLEM 2.  [60 points] Consider the following C code:

```c
for (i = 0; i < n; ++i)  
a[i] = a[i] + b[i];
```

where `a` and `b` are 32-bit integer arrays of size `n`.

The program has been compiled in MIPS assembly code assuming that registers $t6$ and $t7$ have been initialized with values 0 and 4 N respectively. The symbols `VECTA` and `VECTB` are 16-bit constant. The processor clock frequency is 1 GHz.

```
FOR  
beq $t6, $t7, END  # if ($t6 == $t7) goto END  
lw $t2,VECTA($t6)  # $t2 <- VECTA [t6];  
lw $t3,VECTB($t6)  # $t3 <- VECTB [t6];  
add $t2,$t2,$t3  # $t2 = $t2 + $t3;  
sw $t2,VECTA($t6)  # VECTA[t6] <- $t2;  
addi $t6,$t6,4  # $t6 = $t6 + 4;  
j FOR  # goto FOR;
END:
```

Assume that the above MIPS instructions will be executed on a 5-stage pipelined processor (as defined in class).

2.1 Let us consider a single iteration of the loop executed by a 5-stage pipelined MIPS processor without any optimization in the pipeline. Note that, on Slide 16 of Lecture 6.2, although we use a line of bubbles, some optimizations are used; for instance, there is a forwarding line from the output of $D$ to
the input of ALU, which avoids two additional lines bubbles. Here, such forwarding line is not assumed.

Draw the pipeline execution diagram without unrolling and compute the average CPI (clock cycle per instruction) of the loop. Identify the RAW (Read After Write) data hazards by marking in RED and control hazards in BLUE. Identify the number of stalls to be inserted before each instruction (or between the stage IF and ID of each instruction) necessary to solve the hazards. You can assume that there are no structural hazards. (Hence, separate instruction and data memories ($I$ and $D$) are available. Moreover, you can assume that it is possible to read and write at the same address in the same clock cycle, in the Register File.) Therefore, the hazards to be considered here are the two types of data hazards studied in class as well as control hazards.

2.2 From now on and for the subsequent questions, assume the following optimization in the pipeline:

- structural hazards are avoided in the manner described in Question 2.1,
- forwarding computation of PC and TARGET ADDRESS for branch & jump instructions anticipated in the ID stage.

Identify the RAW (Read After Write) data hazards and the control hazards in the pipeline scheme. Identify the number of stalls to be inserted before each instruction (or between the stage IF and ID of each instruction) necessary to solve the hazards. Identify in the hazard type and the forwarding path used.

2.3 Apply loop unrolling (as well as instruction re-ordering, if you like) on the above MIPS code for two iterations. Write the corresponding MIPS instruction code.

2.4 Draw the pipeline execution diagram of your MIPS instructions (one iteration of the new loop would be enough) and compute the average CPI of the loop.

PROBLEM 3. [25 points]

A 4-processor shared-memory multiprocessor configuration implements write-back cache using the MESI (Modified, Exclusive, Shared, Invalid) algorithm for cache coherency. Assume that location 0x0010 is not in any cache at the start of the following sequence.

Consider the following read/write operations:

(a) Processor 0 reads from location 0x0010
(b) Processor 0 writes to location 0x0010
(c) Processor 2 reads from location 0x0010
(d) Processor 3 writes to location 0x0010
(e) Processor 2 writes to location 0x0010
(f) Processor 1 reads from location 0x0010
(g) Processor 3 writes to location 0x0010
(h) Processor 0 reads from location 0x0010
Show the state (M, E, S or I) for the cache line containing location 0x0010 in each processor cache after each operation. Also note any transfers to/from memory if any occurs.

Solutions to operations (a) and (b) are given in the following table. Complete the table for operations (c) - (h).

<table>
<thead>
<tr>
<th>State</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Memory transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a) P0 read miss</td>
<td>E</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>P0 reads a cache line from memory</td>
</tr>
<tr>
<td>(b) P0 write hit</td>
<td>M</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>