Exercise 1.  [10 points] The following statements are either [T]rue or [F]alse. Indicate T in [ ] if the statement is true, otherwise F in [ ].

1.1 [F] Pipelining helps latency of a single stage and throughput of entire workload.
1.2 [T] Time to “fill” pipeline and time to “drain” it reduce speedup.
1.3 [F] Pipelining reduces CPI (cycles per instruction), rather than the clock period.
1.4 [T] Multiple-cycle CPU clocking is limited by the slowest stage of an instruction.
1.5 [T] The potential speedup that a pipelined processor can achieve is the number of stages.
1.6 [F] In a pipelined processor, we can NOT solve all types of hazards by stalling.
1.7 [F] In the best case, a 2-issue pipelined processor can obtain IPC (instructions per cycle) of 1.
1.8 [T] In multicore processors, a cache coherence problem likely occurs when the cores share a common physical address space.
1.9 [F] In multicores, false sharing in separated cache does not increase cache miss rates.
1.10 [F] To exploit TLP (thread-level parallelism), multicore processors consider the dependencies among multiple instructions so as to issue them to independent threads.
Exercise 2. [20 points] Consider the following code sequence in MIPS to execute on a pipelined processor.

```
sub $s2, $s1, $s3  # $s2 = $s1 - $s3
and $t2, $s2, $t5  # $t2 = $s2 && $t5
or $t3, $t6, $s2   # $t3 = $t6 || $s2
lw $t5, 100($s2)  # $t5 = Mem[$s2+100]
sw $t3, 100($s2)  # Mem[$s2+100] = $t3
```

Assume that instruction re-ordering is not used.

2.1 Indicate those registers that have the RAW (read after write) dependence.

$s2 [2]
$t3 [2]

2.2 Indicate three potential hazards and their types (structural, data or control).

(1) $s2 from and and or depending on sub [1] has data hazard [1].

(2) $s2 from lw depending on sub [1] has structural hazard [1].

(3) $t3 from sw depending on or [1] has data hazard [1].
2.3 In the case that you are only allowed to stall the pipelining, how many extra clock cycles you have to pay?

3 cycles [2], including 2 cycles for (1) and 1 cycles for (3)

2.4 Provide your hazard resolution methods for each hazard occurred so as to avoid to stall the pipelining. If you use forwarding, indicate which type of forwarding (ALU-ALU, MEM-MEM, ALU-MEM) is.

For (1), ALU-ALU forwarding. [2]
For (2), register read and write can happen in the same clock cycle. [2]
For (3), ALU-MEM forwarding. [2]
2.5 What is the CPI if the pipeline doesn’t stall for those MIPS instructions?

It takes 9 clock cycles for 5 instructions. So, CPI = 9/5 = 1.8. [2]

Exercise 3. [10 points] Consider the sequence of MIPS instructions as below.

```
lp: addi $s1, $s1, -8 # decrement pointer
    lw $t0, 0($s1) # $t0 = array element
    lw $t1, 4($s1) # $t1 = array element
    addu $t0, $t0, $s2 # add scalar in $s2
    addu $t1, $t1, $s2 # add scalar in $s2
    sw $t0, 0($s1) # store result
    sw $t1, 4($s1) # store result
    bne $s1, $0, lp # branch if $s1 != 0
```

Propose a schedule of this sequence on a two-issue pipelined processor with the following constraints:

- the first issue deals only with ALU or branch instructions,
- the second issue deals only with data transfer (loads and stores),
- we wish to avoid pipeline stalls.

Assume that forwarding is used and there is no need for instruction re-ordering.

<table>
<thead>
<tr>
<th></th>
<th>ALU or branch</th>
<th>Data transfer</th>
<th>Clock Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp:</td>
<td>addi</td>
<td>lw</td>
<td>1</td>
</tr>
</tbody>
</table>
Exercise 4. [10 points] Processors P1 and P2 start with empty cache and have the read/write operations on words \(x[1]\) and \(x[2]\) as in Table 1. Assume words \(x[1]\) and \(x[2]\) will be fetched to the same cache line of the cache of P1 and P2, respectively. For each operation, indicate whether it is a cold miss (CM), false miss (FM), true miss (TM) or hit (H).

<table>
<thead>
<tr>
<th>Time</th>
<th>P1</th>
<th>P2</th>
<th>CM, FM, TM or H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write (x[1])</td>
<td></td>
<td>CM [2]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read (x[2])</td>
<td>CM [2]</td>
</tr>
<tr>
<td>3</td>
<td>Write (x[1])</td>
<td></td>
<td>H [2]</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Write (x[2])</td>
<td>FM [2]</td>
</tr>
<tr>
<td>5</td>
<td>Read (x[2])</td>
<td></td>
<td>TM [2]</td>
</tr>
</tbody>
</table>

Table 1: The read/write operations on words \(x[1]\) and \(x[2]\).