Question #1 (Multiprocessors) (40 points)

1.1. (5 points) Compare write-invalidate and write-update protocols for cache coherence (discuss advantages and disadvantages).

1.2. (5 points) What are four states of the MESI protocol? Give a short explanation of what each of these states means.
1.3. (5 points) What is false sharing? How we can cope with it?

1.4. (5 points) Why is Test-and-Test&Set generally a better lock implementation than Test&Set? Under what conditions would Test&Set be a better lock than Test-and-Test&Set?
1.5. (10 points)
Consider a three-processor (P1, P2, P3) bus-based shared memory multiprocessor protocol with write-back direct-mapped first level cache memories and MESI snooping cache coherence protocol. Show the state of a cache line in each processor’s cache and main memory after each of the following actions. Assume that memory blocks A and B map to the same cache line. Initially, A = 5, B = 7.

Actions:
1. P1: Write A, 4
2. P3: Write B, 8
3. P2: Read A
4. P3: Read A
5. P3: Write A, 3
6. P2: Read A
7. P2: Read B

Note: BRd – Bus Read, BWr – Bus Write, BRdX – Bus Read Exclusive, BInv – Bus Invalidate.
For P1, P2, P3 use A:S notation to indicate that the cache has block A in Shared state.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>-</td>
<td>A:5 B:7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.6. (10 points)
Consider a three-processor (P1, P2, P3) shared memory multiprocessor with distributed main memory, which uses the directory-based MSI write invalidate protocol with write-back direct-mapped first level cache memories. Show the state of a cache line in each processor’s cache and directories after each of the following actions. Assume that memory blocks A, and B map to the same cache line. Processor P1 is home for block A (initial value is 5), and processor P2 is home for block B (initial value is 7).

Actions:
(1) P1: Write A, 4
(2) P3: Write B, 8
(3) P2: Read A
(4) P3: Read A
(5) P3: Write A, 3
(6) P2: Read A
(7) P2: Read B

Legend: St (State), Bl (Block)= { A | B}, V(Value), IN (Interconnection Network), Act (Action = Message Type), Src (Source), Dst (Destination), DirP1 (Directory of processor P1), ProcList (List of Sharers). Each action on the IN should take one row. States in the directory: U (uncached), S (shared), E (exclusive). Cache States: M (Modified), S (Shared), I (Invalid).

Interconnection Network Messages:
- RM, Read Miss (Local Cache -> Home Directory)
- WM, Write Miss (Local Cache -> Home Directory)
- I, Invalidate (Home Directory -> Remote Caches)
- F, Fetch (Home Directory -> Remote Caches)
- FI, FetchInvalidate (Home Directory -> Remote Caches)
- DVR, DataValueReply (Home Directory -> Local Cache)
- DWB, DataWriteBack (Remote Cache -> Home Directory)
<table>
<thead>
<tr>
<th>Step</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>IN</th>
<th>DirP1</th>
<th>DirP2</th>
<th>DirP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>A:U:5</td>
<td>000</td>
<td>B:U:7</td>
</tr>
</tbody>
</table>

(1)
Question #2 Vector Processing

Show how a vectorizing compiler would translate the following loop into vector operations. For simplicity, assume that LEN is equal to the length of the system’s vector registers. Use Vn to indicate vector registers, and Rn to indicate scalar registers.

```
for (i = 0; i < LEN; i++) {
    a[i] = b[i] * c[i];
    d[i] = a[i] + 7
}
```

2.1. (10 points) Illustrate execution assuming one memory access pipe in VMIPS vector processor. What is execution time?

2.2. (10 points) Illustrate execution assuming 3 memory access pipes (2 for load and 1 for store) in VMIPS processor.
**Question #3 (Vector Processing) (20 points)**

Consider the following DAXPY-like loop where k is a parameter to the procedure containing the loop:

```fortran
  do 10 i=1, 128
      do 10 j=1, 128
          Y(k,j) = a*X(i,j) + Y(k,j)
          Z(j) = X(i,j)*Z(j)
      10 continue
```

3.1 (10 points) Analyze the original FORTAN code. Write an optimal vector code sequence for VMIPS processor. Identify convoys. VMIPS register banks have 64 registers. Use chaining assuming: 1 Load pipe, 1 Store pipe, 2 multiplier pipes, and 2 adder pipes.

3.2 (10 points) Estimate performance of your code from part (a). Startup penalties are the following: 6cc for vector add, 7 for vector multiply, 12 for vector load, 12 for vector store. Assume that the overhead for handling an outer loop is 15 clock cycles.
Question #4: (ILP and its Software Exploitation) (20 points)
Consider the following code.

Loop:    LD.D F0, 0(R1) ; load X(i)
         MUL.D F0, F0, F2 ; a*X(i)
         LD.D F4, 0(R2) ; load Y(i)
         ADD.D F0, F0, F4 ; a*X(i) + Y(i)
         S.D 0(R2), F0  ; store Y(i)
         DSUBUI R1, R1, #8 ; decrement X index
         DSUBUI R2, R2, #8 ; decrement Y index
         BNE R1 Loop ; loop if not done

Assume an extended MIPS pipeline with latencies given in the table below and a 1-cycle delayed branch.

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP Mul</td>
<td>FP ALU op / Store</td>
<td>4</td>
</tr>
<tr>
<td>FP Add</td>
<td>FP ALU op / Store</td>
<td>3</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
</tbody>
</table>

Assume a dual-issue processor as in Figure 4.2 (one integer and one FP instructions can be issued in one clock cycle). Unroll the loop (as many times as necessary) and schedule the code in order to remove all stalls. Show the schedule. What is the execution time per element? How many instruction issue slots are unused?