1. The four generic actions that hardware performs in response to an interrupt are: (1.5 pts. each)

1) save PC and PSR
2) restrict or disable further interrupts
3) set execution mode to kernel
4) load new PC from IVT

2. What is the concept of a privileged instruction? Also, identify one such instruction as an example. (3 pts.)

A privileged instruction is an instruction that is only valid in kernel mode, e.g., set the CPU timer.

Process/Thread. Circle only one of P or T. (1 pt. each)

3. P / T Has an associated SP (stack pointer).
4. P / T Has an associated PC (program counter).
5. P / T Is assigned the memory map (e.g., page table).
6. P / T Is the unit of resource allocation (e.g., open file).
7. P / T Is the unit of execution that is individually scheduled to run on a processor.

T, T, P, P, T

True/False. Circle only one of T or F. (1 pt. each)

8. T / F A reentrant program has no store instructions.
9. T / F A supervisor call instruction acts just like a subroutine call instruction.
10. T / F A store instruction should be invalid in user mode to prevent memory protection errors.
11. T / F When a user attempts to execute an instruction that is invalid in user mode, the CPU should interrupt.
12. T / F Users should not be allowed to write into the execution mode bit(s) in the processor status register.
13. T / F Remote Procedure Call wraps normal function call syntax around underlying message passing.
14. T / F Rendezvous is only possible when the message send uses asymmetric naming.
15. T / F Rendezvous cannot be used with mailboxes (i.e., indirect naming) since mailboxes always have buffers.
16. T / F An output mailbox has many senders and one receiver.
17. T / F Threads can execute semaphore signals but not semaphore waits.
18. T / F Threads can perform message sends as well as message receives.
19. T / F Interrupt service routines can execute semaphore signals but not semaphore waits.
20. T / F Interrupt service routines can perform message sends as well as message receives.
21. T / F Load Linked and Store Conditional can be used to implement a spin lock.
22. T / F Load Linked and Store Conditional can be used to implement optimistic concurrency control.
23. T / F Test and Set combines read, modify, and write actions into a single atomic instruction.

<answers>

11-13, 18-19, 21-23 are true; the rest are false

24-27. Peterson's algorithm / Test&Set / Compare&Swap / Load linked & store conditional / Semaphore.
Circle one or more of P, T, C, L, or S, as applies. (5 pts. each)
24. P / T / C / L / S Appropriate use is in OS only with interrupts disabled.
25. P / T / C / L / S Blocks user threads by causing thread state change (i.e., no busy wait at user level).
26. P / T / C / L / S Uses an address-watch register that snoops bus on shared-memory multiprocessor.
27. P / T / C / L / S Machine instruction in which read-modify-write actions are performed as atomic unit.

<answers>

24 - T only; 25 - S only; 26 - L only; 27 - T and C

28-31. Mutual exclusion / Restricted sharing / Notification. Circle one or more of M, R, or N, as applies. (3 pts. each)
28. M / R / N Requires a general semaphore.
29. M / R / N Semaphore that is used must be initialized to 0.
30. M / R / N Semaphore that is used must be initialized to 1.
31. M / R / N Semaphore wait operation placed in one thread and signal operation placed in a separate thread.

<answers>

28 - R only; 29 - N only; 30 - M only; 31 - N only
32. Here is the code for an approach to software synchronization using a turn variable. (6 pts.)

```c
int turn = 1; // initially
while(true) {
    T1S1 while( turn != 1 );
    T1S2 CS1
    T1S3 turn = 2;
    T1S4 // remainder
}
while(true) {
    T2S1 while( turn != 2 );
    T2S2 CS2
    T2S3 turn = 1;
    T2S4 // remainder
}
```

Show a scenario in which this approach fails one of the three rules for mutual exclusion. Give the exact sequence using the T<i>S<j> statement ids so that it is clear the order in which the statements are executed. Explain what type of failure occurs.

<answer>

There is a failure of robustness (“A thread outside its critical section and entry and exit sections cannot block another thread from entering its critical section.”) since one thread can fail or stop and then never participate in taking turns again. The remaining thread can visits its critical section but is then blocked from doing so again.

There are several possible scenarios in which this failure will occur. Here is one:

- turn = 1
- T1S1 - drops through while
- T1S2 - in CS1
- T1S3 - changes turn
- T1S4 - fails/halts in remainder section
- T2S1 - drops through while
- T2S2 - in CS2
- T2S3 - changes turn
- T2S4 - in remainder, loops to top
- T2S1 - stuck in infinite loop

33. What was the major problem we identified and discussed if users were allowed to implement their own test-and-set spin locks? (4 pts.)

<answer>

priority-based deadlock – a low-priority thread acquires a lock, then a high-priority thread awakens and tries to obtain the lock; the higher-priority thread is stuck in the spin lock, and the low-priority thread is unable to complete its critical section and release the lock
34. What is the name of the instruction that has the following definition? (2 pts.)

```java
boolean INSTRUCTION( &a, &b, c ){ // indivisible operation on shared item ‘b’
    if( a == b ){ b = c; return true; } // ‘a’ and ‘b’ are reference parameters
    else { a = b; return false; }
}
```

<answer>

compare and swap

35. Use the instruction from question 34 to implement an optimistic concurrency increment to a shared, global value of ‘X’. That is, X=X+1. Your code can be C-like pseudo code for one of the threads. The other threads will be assumed to have the same type of code. (6 pts.)

<answer>

```c
local = X;
repeat
    new = local + 1;
    rc = compare_and_swap( local, X, new );
until( rc == true );
```

36. Fill in the blanks for the semaphore signal logic. (6 pts.)

```c
if (empty(___________________________)) {
    S.value = S.value + 1;
} else{
    select thread on _____________________ and move to ______________________;
}
```

<answers>

S.blocked_list, S.blocked_list, ready_list
37. Use message passing to implement a solution to the mutual exclusion problem. Give a pre-protocol that must be executed prior to entering the critical section and a post-protocol that must be executed when leaving the critical section. Be sure to show how the mailbox is initialized in terms of how many messages (0, 1, more?) and the contents of any initial message(s). Your code can be pseudo code for one of the threads. The other threads will be assumed to have the same type of code. (7 pts.)

```c
mailbox_object MB; // MB is global name; initialize the contents to: ________________________________
...
void send( msg_object & msg, mailbox_object & to_mbox ); // prototype
void rcv( msg_object & msg, mailbox_object & from_mbox ); // prototype
...
// thread code
msg_object message; // local variable

// pre-protocol – fill-in

Crit_Sect_function( ... ); // critical section

//post-protocol – fill-in

<answer>

mailbox_object MB; // MB is global name; initialize the contents to: one NULL message
...
void send( msg_object & msg, mailbox_object & to_mbox ); // prototype
void rcv( msg_object & msg, mailbox_object & from_mbox ); // prototype
...
// thread code
msg_object message; // local variable

rcv( message, MB ); // pre-protocol – fill-in

Crit_Sect_function( ... ); // critical section

send( message, MB ); //post-protocol – fill-in
```
38. Use semaphore wait and signal to implement a solution to the mutual exclusion problem. Give a pre-protocol that must be executed prior to entering the critical section and a post-protocol that must be executed when leaving the critical section. Be sure to show the value to which the semaphore is initialized. Your code can be pseudo code for one of the threads. The other threads will be assumed to have the same type of code. (7 pts.)

```
semaphore S(______); // global semaphore; initialized to value in parentheses
int X = 0; // global variable
...
// thread code

// pre-protocol – fill-in

X = X + 1;
// critical section

//post-protocol – fill-in
```

<answer>

```
semaphore S(1); // global semaphore; initialized to value in parentheses
int X = 0; // global variable
...
// thread code

wait(S);
// pre-protocol – fill-in

X = X + 1;
// critical section

signal(S);
//post-protocol – fill-in
```

Extra credit. High in the Andes Mountains, there are two circular railway lines. One line is in Peru, the other in Bolivia. They share a common section of track where the lines cross a mountain pass that lies on the international border.

Unfortunately, the Peruvian and Bolivian trains occasionally collide when simultaneously entering the common section of track (the mountain pass). The trouble is, alas, that the drivers of the two trains are both blind and deaf, so they can neither see nor hear each other.

The two drivers agreed on the following method of preventing collisions. They set up a large bowl at the entrance to the pass. Before entering the pass, a driver must stop his train, walk over to the bowl, and reach into it to see if it contains a rock. If the bowl is empty, the driver finds a rock and drops it in the bowl, indicating that his train is entering the pass;
once his train has cleared the pass, he must walk back to the bowl and remove his rock, indicating that the pass is no longer being used. Finally, he walks back to the train and continues down the line.

![Diagram of the pass and train tracks]

If a driver arriving at the pass finds a rock in the bowl, he leaves the rock there; he repeatedly takes a siesta and rechecks the bowl until he finds it empty. Then he drops a rock in the bowl and drives his train into the pass.

Unfortunately, one day the two trains crashed. Explain why. (6 pts.)

<suggested answer>

This is an example of the test and set problem when the test is separated from the set.

Assume both drivers arrive about the same time. The first walks to the bowl and finds it empty. While he searches for a rock, the other arrives and also finds the bowl empty. While the second one searches for a rock, the first drops his rock. The second then drops his rock; and, since he is blind, he does not see that the bowl already has a rock. (He is relying on the check he just did a moment before.) Both drivers head to their trains and drive into the pass.

A possible scenario with a crash looks like this:

<table>
<thead>
<tr>
<th>Peruvian driver</th>
<th>bowl = empty</th>
<th>Bolivian driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrives and find bowl empty</td>
<td>empty</td>
<td>arrives and finds bowl empty</td>
</tr>
<tr>
<td>searches for rock</td>
<td>empty</td>
<td>searches for rock</td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td>one rock</td>
</tr>
<tr>
<td>drops rock in bowl</td>
<td>one rock</td>
<td>two rocks</td>
</tr>
<tr>
<td>starts train and drives into pass</td>
<td>two rocks</td>
<td>drops rock in bowl</td>
</tr>
<tr>
<td></td>
<td>two rocks</td>
<td>starts trains and drives in to pass</td>
</tr>
</tbody>
</table>

Both trains are in the pass at the same time.