1. Consider two functions, P and Q, and the main program creates two threads to execute P and Q as shown in the following code. Note that each statement is labeled and the label is just for convenience.

   function P();
   { a = b + 1; print(a); } 

   function Q();
   { b = a + 3; print (b); } 

   main()
   { a = 5; b = 1;
     create a thread to execute P();
     create a thread to execute Q(); } 

List all possible printouts after the execution of the program. You should consider potential process switches at the instruction level.

2. Consider the following program. Find a counter example (execution sequence) that demonstrates that this software solution is incorrect in terms of mutual exclusion violation.

   var blocked: array[0..1] of boolean;
   var turn: 0..1; // could be 0 or 1

   function p (id: integer)
   { repeat
     blocked[id] := true;
     while (turn != id) do
     { while blocked[1–id] do; // do nothing
       turn := id;
     }
     < critical section >
     blocked[id] := false;
     < remainder code for p>
     until false;
   }

   main ()
   { blocked[0] := false; blocked[1] := false;
     create thread to execute P(0);
     create thread to execute P(1);
   }

3. Consider a sequential program as follows. Assume that we try to execute all statements concurrently to achieve maximal parallelism, but we want to obtain the same output as the sequential execution below. Use (a) locks and (b) semaphores to control the synchronization so that the program can be executed in maximal concurrency while still get the same outcome as the sequential execution.

   x = y+z;
   w = y*z;
   u = w–3;
   v = x+t;
   k = x+w;
4. In the Jurassic Park Amusement center, tourists ride on a tour bus to visit the park. Each bus takes two visitors from the visitor center, drives around the park as long as the passengers wish, and return to the visitor center to drop off the passengers. There are N tour buses on service. If the N buses are all out, then a passenger who wants a ride must wait. If a bus is ready to load but there are no waiting visitors, then the bus waits. A bus does not leave until it loads two passengers. A bus does not load passengers if there is another bus that is only partly filled.

Consider the problem of synchronizing the bus processes and the visitor processes using semaphores. The code for visitor process follows. Three semaphores cust, bus, and busReady are used. Write the code for the bus process that performs the proper operations on the three semaphores. Use extra semaphores as needed. In the bus process, you can use the same pseudo-statements “visitor get in bus”, etc. as in the visitor process. Do not forget to properly initialize the semaphores (including the three given in the code and any new ones you may introduce).

\[
\begin{align*}
\text{var} & \quad \text{cust: semaphore (:= ?);} \\
& \quad \text{bus: semaphore (:= ?);} \\
& \quad \text{busReady: semaphore (:= ?);} \\
\end{align*}
\]

\[
\begin{align*}
\text{procedure visitor;} \\
& \quad \text{begin} \\
& \quad \quad \text{signal (cust);} \\
& \quad \quad \text{wait (bus);} \\
& \quad \quad \text{visitor get in bus;} \\
& \quad \quad \text{wait (busReady);} \\
& \quad \quad \text{drive around the park;} \\
& \quad \quad \text{visitor get off bus;} \\
& \quad \text{end;}
\end{align*}
\]

5. Implement a monitor solution for the bakery problem. Assume that you have N salesmen and customers arrive at arbitrary times. The code for customers and salesmen external to the monitor is given as follows.

\[
\begin{align*}
\text{customer} & \quad \{ \text{get service ();} \\
& \quad \quad \text{receive the service} \\
& \quad \quad \text{release service ();} \\
& \quad \}\n\end{align*}
\]

\[
\begin{align*}
\text{salesman} & \quad \{ \text{prepare service();} \\
& \quad \quad \text{provide the service} \\
& \quad \quad \text{complete service();} \\
& \quad \}\n\end{align*}
\]

6. When we implement the code for realizing the monitor, there is one important issue. Monitor can only allow one active thread in it. When signaling a condition variable, we need to decide whether to let the signaler or the signalee to continue. We have discussed how to implement the monitor with the choice of letting the signaler continue. Now, discuss the high-level implementation idea if we choose to let the signalee continue.