COMP 525
Assignment 1
Due Date: 24th January 2017

Prakash Panangaden

10th January 2017

Please attempt all questions. There are **5** questions for credit and one for your spiritual growth. You will need to read Chapter 2 of the book for details of the formalism that I skipped over in class. The homework is due in class at the beginning of the class. Question 6 should not be handed in, but discussed privately with me. You will get no extra credit or other benefit related to your grade for doing it.

Question 1[10 points] Consider the following processes, $P_{1,2}$ that share a variable x of type `int` and have their own private *registers*, $R_{1,2}$ respectively. The registers have the operations Load, which takes the value of x and puts it in the local register, Inc which increments the value in the register, Shift which doubles the value in the register and Store which copies the register value back to the shared variable x.

```
P1:                          P2:
LOAD(x,R1);                  LOAD(x,R2);
INC(R1);                     SHIFT(R2);
STORE(R1,x)                  STORE(R2,x)
```

What are the possible values of x at the end when the two processes are run in parallel? If the three actions of P_1 are called $\alpha_1, \alpha_2, \alpha_3$ and those of P_2 are called $\beta_1, \beta_2, \beta_3$ give example interleaved sequences for each possible final value.

Suppose that there is only one register so that both processes share the same register. What are the possible values now? Give an execution sequence for each possible final value.

Question 2[15 points] Question 2.3 from the textbook, pages 82-83.

Question 3[25 points] Here is an attempted solution to the mutual exclusion problem. The key words "parbegin" and "parend" mean that the blocks of code labeled as a process will run in parallel. The variables c_1, c_2 are shared. The idea is that each process sets its own variable to 0 when it enters the critical section; before attempting to enter it checks if the other process is in its critical section, if it is it tried again: this is called a busy-waiting loop.

```
begin integer c1, c2;
      c1:= 1; c2:= 1;
      parbegin
      process1: begin L1: if c2 = 0 then goto L1;
                          c1: = 0;
                          critical section 1;
                          c1:= 1;
                          remainder of cycle 1; goto L1
                end;
      process2: begin L2: if c1 = 0 then goto L2;
                          c2: = 0;
                          critical section 2;
                          c2:= 1;
                          remainder of cycle 2; goto L2
                end
      parend
end .
```

What is wrong with this solution? Explain what atomicity assumptions you have made.
We try to fix the problem by changing the code as follows:

```
begin integer c1, c2;
      c1:= 1; c2:= 1;
      parbegin
      process 1: begin A1: c1:= 0;
                       L1: if c2 = 0 then goto Ll;
                           critical section 1;
                           c1:= 1;
                           remainder of cycle 1; goto A1
                 end;
      process 2: begin A2: c2:= 0;
                       L2: if c1 = 0 then goto L2;
                           critical section 2;
                           c2:= 1;
                           remainder of cycle 2; goto A2
                  end
      parend
end .
```

Does this guarantee mutual exclusion? Argue informally. What is wrong with this solution?

Question 4[25 points] Here is Dekker's solution to the mutual exclusion problem.

```
begin integer c1, c2 turn;
      c1:= 1; c2:= 1; turn = 1;
      parbegin
      process 1: begin A1: c1:= 0;
                       L1: if c2 = 0 then
                               begin if turn = 1 then goto L1;
                                     c1:= 1;
                                 B1: if turn = 2 then goto B1;
                                     goto A1
                               end;
                           critical section 1;
                           turn:= 2; c1:= 1;
                           remainder of cycle 1; goto A1
                 end;
      process 2: begin A2: c2:= 0;
                       L2: if c1 = 0 then
                              begin if turn = 2 then goto L2;
                                    c2:= 1;
                                 B2: if turn = 1 then goto B2;
                                     goto A2
                              end;
                           critical section 2;
                           turn:= 1; c2:= 1;
                           remainder of cycle 2; goto A2
                 end
      parend
end .
```

Give an informal argument that this is correct, deadlock-free and starvation free.

Question 5[25 points] Question 2.9 on pages 85 and 86 of the textbook.

Question 6[0 points] Given a digital circuit with AND, OR and NOT gates we can model the circuit as a propositional formula. Thus the theory of such circuits is just a branch of propositional logic. Now suppose that we want to construct the following circuit. It has three input wires and three output wires. Let the input wires be a, b and c and the output wires be x, y and z. We want the outputs to satisfy

$$x = \neg a, y = \neg b, z = \neg c.$$

Design a circuit that achieves this with only two NOT gates. This is a pure puzzle and does not have anything to do with the class. Can you prove that it is impossible with just one NOT gate? What can you say about the number of NOT gates needed to invert n inputs? With a monotone circuit it is clearly impossible (clearly?).