Matthew Bentz

Mlb0119

1.1

```
P1: {
  shared int x;
  x = 10;
  while (1) {
     x = x - 1;
     x = x + 1;
     if (x != 10)
       printf("x is %d",x)
  }
}
```

```
P2: {
  shared int x;
  x = 10;
  while ( 1 ) {
    x = x - 1;
    x = x + 1;
    if (x!=10)
      printf("x is %d",x)
  }
}
```

Steps of interleaving instructions:

1. P1: x = x - 1; // x = 9
2. P1: x = x + 1; // x = 10
3. P2: x = x – 1; // x = 9
4. P1: if (x != 10) // x = 9
5. P2: x = x + 1; // x = 10
6. P1: printf("x is %d", x) // "x is 10"

1.2

1. P1: x = x – 1; // x = 9
2. P1 & P2: These are concurrent
   a. P1: x = x + 1; Step:
      i. LD R0, X // x = 9 1
      ii. INCR R0 // x = 10 2
      iii. STO R0, X // x = 10 5
   b. P2: x = x – 1;
      i. LD R0, X // x = 9 3

ii. DECR R0 // x = 8 4
iii. STO R0, X // x = 8 6
3. P1: if (x != 10) // x = 8 from step 6
4. P1: printf("x is %d", x) // "x is 8"

2. Binary semaphores - semaphore with only two states. This semaphore allows for mutual exclusion.

General semaphores – semaphore with an ability to have multiple states/values. This semaphore allows multiple processes to "take turns" through something like incrementation.

3. A monitor is a combination of a lock object and a signaling object such as an array. The monitor controls the mutual exclusivity of separate processes. A short example of this would be:

```
while (waiting[i] && key)

key = test_and_set(&lock);
```

Book definition: A high-level language synchronization construct that protects variables from race conditions.

4. Semaphores can have two atomic operations: wait and signal. The wait(s) operation controls the next process to be added in the queue. The signal(s) operation controls the execution of the first in line process in the queue.

Example of a counting semaphore's operations:

```
wait(s) {

while (s ≤ 0);

s--;

}

signal(s) {
```

```
s++;

}
```