HW4

1. Ans
   (a) Since each processor just lock and unlock once as it is test-and-test&set for which processor don't need to do test&set every time released. There are only one transaction.
   (b) 100 cycles; 1600 cycles
   (c) It's better to give priority to the process according to the order, in which way we can ensure one transaction once and saving bus traffic.
   (e) No. Because cache coherence is no more needed.

2. Ans
   (a) (i) F&I
   ticket lock:

```
init ticket=now_s=0;

lock:
  iTicket = F&I (&(ticket));
  while(iTicket != now_s);
unlock:
  now_s++;
```

   array-based lock:

```
init location=0;

lock:
  iLocation = F&I (&(location));
  while(array[iLocation] == lock);
  array[iLocation] = lock;
unlock:
  array[iLocation+1] = unlock;
```

   (ii)LL-SC
   Ticket lock:

```
          ll   reg1 &ticket
          add  ticket ticket 1

lock:     ll   reg2 &now_s
          bnz  reg1 reg2 lock

          ret

unlock:   add  now_s now_s 1
          ret
```

array-based lock:

```
lock:    ll   reg1 array[location]
         add  location location 1
         bnz  reg1 lock

         sc   array[reg1] #1
         ret

unlock: st  array[reg1+1] #0
         ret
```

(b) Yes, the trick is for an arriving process, fetch & store last processor node and check if there is predecessor. If so, wait until it is unlocked. And for an unlock, if there is successor, set it “unlock”.

3. Ans

```
      ll   reg1 location
      bne  reg1 reg2 swap
      ret

swap: sc   location reg2
      ret
```

4. The code

```
void barrier()
{
   pthread_mutex_lock (&mutex);

   waiting--;
   if (waiting > 0)
      pthread_cond_wait (&condition, &mutex);
   else {
      waiting = NumThread;
      pthread_cond_broadcast (&condition);
   }
   pthread_mutex_unlock (&mutex);
}
```