►Solution◄

Question 1: (20 points)

Consider a critical section of the following form where a shared variable `shvar` is updated using a local (non-shared) variable `x` as follows:

```
lock(lk);

if(shvar > 0) {
  shvar = max(shvar, x);
}

unlock(lk);
```

a. (5 points) Write the MIPS assembly code for this critical section, assuming that the address of the `lk` variable is in `$a0`, the address of the `shvar` variable is in `$a1`, and the value of variable `x` is in `$a2`. Your critical section should not contain any function calls, i.e., you should include the MIPS instructions for `lock()`, `unlock()`, `max()`, and `min()`. Use `ll/sc` instructions to implement the `lock()` operation, and the `unlock()` operation is simply an ordinary store instruction.

Solution:

```
try:  addi $t1, $zero, 1
      ll   $t0, 0($a0)       # $t0 <-- lk
      bne  $t0, $zero, try   # while the lock is not free try
      sc   $t1, 0($a0)       # use sc to confirm that got the lock
      beq  $t1, $zero, try   # if lock fail, go try again
      lw   $t3, 0($a1)       # $t3 <-- shvar
      ble  $t3, $zero, done  # if(shvar <= 0) goto done
      bge  $t3, $a2, done    # if(shvar >= x) goto done
      sw   $a2, 0($a1)       # shvar <-- x
done: sw   $zero, 0($a0)     # unlock
```

b. (5 points) Solve the above problem, but this time use `ll/sc` to perform an atomic update of the `shvar` variable directly, without using `lock()` and `unlock()`. Note that in this problem, there is no variable `lk`.

Solution:

```
try:  ll   $t3, 0($a1)       # $t3 <-- shvar
      ble  $t3, $zero, done  # if(shvar <= 0) goto done
      bge  $t3, $a2, done    # if(shvar >= x) goto done
      mov  $t1, $a2
      sc   $t1, 0($a1)       # shvar <-- x
```

```
      beq  $t1, $zero, try
done:
```
if(x >= shvar) goto done, the c code says shvar = max(shvar,x)

c. (5 points) Compare the best-case performance of your code from parts a and b, assuming that each instruction takes one cycle to execute. Note: best-case means that `ll/sc` always succeeds, the lock is always free when we want to `lock()`. Iff there is a branch, we take the path that completes the operation with fewer executed instructions.

Solution: We will list the sequence of instructions that each code has to execute in the best case (assuming that `shvar` has to be updated).
a: `addi, ll, bne, sc, beq, lw, ble, bge, sw, sw` = 10 instructions
b: `ll, ble, bge, mov, sc, beq` = 6 instructions

d. (5 points) Using your code from part b as an example, explain what happens when two processors begin to execute this critical section at the same time, assuming that each processor executes exactly one instruction per cycle.

Solution: We will assume that in both processors, the `sc` instruction has to be executed. The sequence of instruction execution in each cycle is as follows (we assume that `P0` is the processor that completes the `sc` first in each case). If either of the processors does not need to execute a `sc`, then the other processor will succeed in executing it.

Cycle	P0	P1
0	ll	ll
1	ble	ble
2	bge	bge
3	mov	mov
4	sc	sc
5	beq	beq
6		ll
7		bge
8		mov
9		sc
10		beq