2.31 Impliment the following C code in MIPS assembly . what is the total number of MIPS instructions needed to execute the function?

```
Int fib(int n){
        If (n==0)
            Return 0;
        Else if (n==1)
            Return 1;
        Else
            Return fib(n-1) + fib(n-2);
```

```
      .data
n:    .word  12
fibs: .word   0 : 12
      .text
      la   $t0, fibs
      la   $t5, n
      lw   $t5, 0($t5)
      li   $t2, 1
      add $f0, $f2, $f4
      sw   $t2, 0($t0)
      sw   $t2, 4($t0)
      addi $t1, $t5, -2

fib:  lw   $t3, 0($t0)
      lw   $t4, 4($t0)
      add  $t2, $t3, $t4
      sw   $t2, 8($t0)
      addi $t0, $t0, 4
      addi $t1, $t1, -1
```

```
    bgtz $t1, fib

exit:
```

You should only need about 10 instructions for the function by itself.

2.43 Write the MIPS assemble code to implement the following C code:

```
Lock(lk);
Shvar=max(shvar,x);
Unlock(lk);
```

Assume the address of the lk variable is in $a0, the address of the shvar variable is in $a1 and the value of the variable x is in $a2 your critical section should not contain any function calls. Use ll/sc instructions to implement the lock() operation, and the unlock() operation is simply an ordinary store instruction.

```
try:
        addi $a0, $zero, 1
retry:
        ll $t1,0($a0)
        bnez $t1,retry
        lw $t2, 0($a1)
        slt $t1, $t2, $a2
        bne $t1, $t2, skip
        sc $t0,0($a0)
        beq $t0,$zero,try
skip:
        sw $t2, 0($a0)
```

2.45 Using your code from exercise 2.43 as an example explain what happens when two processors begin to execute this critical section at the same time, assuming that each processor execute exactly one instruction per cycle.

One of the processors will write with sc and the other will fail because the first one wrote and changed the state.

A.1 Section A.5 describes how memory is portioned on most MIPS systems. Propose another way of dividing memory that meets the same goals. Explain why you would do it this way.

You could flip the stack and heap allocation, so that the stack comes up and the data goes down. This might make more sense for addressing the stack incrementally, instead of decrementally. The text segment could be put on the top of all of this to cap the memory usage. This would work well on a system with defined memory, sticking the text at the end, and then filling the rest of it with the dynamic stack and data.

A.2 Is it ever safe for a user program to use the registers $k0 or $k1?

It is safe to read values from $k0 and $k1, but not writing to them.