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);
> }
> ```

Answer:

```
la $t1, fib
la $t6, size
lw $t6, 0($t6)
li $t3, 1

sw $t3, 0($t1)
sw $t3, 4($t1)
addi $t2, $t6, -2

loop: lw   $t4, 0($t1)
  lw $t5, 4($t1)
  add $t3, $t4, $t5
  sw $t3, 8($t1)
  addi $t1, $t1, 4
  addi $t2, $t2, -1
  bgtz $t2, loop
```

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.

Answer:

```
try:
    jal max
retry:
    ll $t1, 0($s1) #load linked
    bnez $t1, retry # If lock taken, retry
    sc $t0, 0($s1) #store conditional
    beq $t0, $zero, try #branch if store fails

max:
    bge $t3, $t4, first
    ble $t3, $t4, second

first:
    addi $t2, $t3, $zero
    j $ra
second:
    addi $t2, $t4, $zero
    j $ra
```

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 executes exactly one instruction per cycle.

Answer: One of the processors takes priority and the other has to wait until the first is finished. The way they decide this is a data race if they're not synchronized

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.

Answer: You have registers that hold data. There are some registers with different roles, like holding temporary data vs long term data. You can take this data and move it to memory, via saving, or load data back into registers from memory. The data is not stored in consecutive blocks of memory, it's more like a table with addresses you pull from. Another way that you could store the data is in consecutive blocks, though, if you wanted to pay the over head of reorganizing the memory when you needed to, or having the programmer allocate memory before using it.

A.2

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

Answer: No, they're used for the (k)ernal to make interrupt calls