In More Depth: Tail Recursion

Some recursive procedures can be implemented iteratively without using recursion. Iteration can significantly improve performance by removing the overhead associated with procedure calls. For example, consider a procedure used to accumulate a sum:

```
int sum (int n, int acc) {
  if (n > 0)
      return sum(n - 1, acc + n);
  else
      return acc;
}
```

Consider the procedure call `sum(3,0)`. This will result in recursive calls to `sum(2,3)`, `sum(1,5)`, and `sum(0,6)`, and then the result 6 will be returned four times. This recursive call of `sum` is referred to as a *tail call*, and this example use of tail recursion can be implemented very efficiently (assume `$a0 = n` and `$a1 = acc`):

```
sum: beq$a0, $zero, sum_exit # go to sum_exit if n is 0
     add$a1, $a1, $a0        # add n to acc
     addi$a0, $a0, -1        # subtract 1 from n
     j sum                   # go to sum
sum_exit:
     move$v0, $a1            # return value acc
     jr $ra                  # return to caller
```

2.16 [30] <§2.7> Write a MIPS procedure to compute the *n*th Fibonacci number `F(n)` where

```
F(n) = 0, if n = 0;
       1, if n = 1;
       F(n-1) + F(n-2), otherwise.
```

Base your algorithm on the straightforward but hopelessly inefficient procedure below, which generates a recursive process:

```
int fib(int n){
  if (n == 0)
      return 0;
  else if (n == 1)
      return 1;
  else
return fib(n-1) + fib(n-2);
```

2.17 [30] <§2.7> Write a program as in Exercise 2.15, except this time base your program on the following procedure and optimize the tail call so as to make your implementation efficient:

```
int fib_iter (int a, int b, int count) {
  if (count == 0)
      return b;
  else
      return fib_iter(a + b, a, count - 1);
```

Here, the first two parameters keep track of the previous two Fibonacci numbers computed. To compute `F(n)`, you have to make the procedure call `fib_iter(1, 0, n)`.

2.18 [20] <§2.7> Estimate the difference in performance between your solution to Exercise 2.15 and your solution to Exercise 2.16.