# Recursion

## 1 Recursion in MIPS

Implement the Fibonacci function in MIPS given the following C code.

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

Note that this code contains two recursive calls. Be careful and save the result of the first `fib` before calling it again.

1. *Assign register names to variables and determine which is base case and which is recursive.*

   Only one input, $n$ is passed in register `$a0`. The *base case* is the "then" clause. The *recursive case* is the "else" clause.

2. *Convert the code for the base case.*

   ```
   fib:
        bgt  $a0, 1, recurse
        move $v0, $a0
        jr   $ra
   ```

3. *Save callee- and caller-saved registers on the stack.*

   ```
   recurse:
        sub $sp, $sp, 12 # We need to store 3 registers to stack
        sw  $ra, 0($sp)  # $ra is the first register
        sw  $a0, 4($sp)  # $a0 is the second register, we cannot assume
                         # $a registers will not be overwritten by callee
   ```

4. *Call `fib` recursively.*

   ```
        addi $a0, $a0, -1   # N-1
        jal  fib
        sw   $v0, 8($sp)    # store $v0, the third register to be stored on
                            # the stack so it doesn't get overwritten by callee
   ```

5. *Call `fib` recursively **again**.*

   ```
        lw   $a0, 4($sp)    # retrieve original value of N
        addi $a0, $a0, -2   # N-2
        jal  fib
   ```

   A number of people were tempted to compute $N - 2$ by subtracting 1 from the value in `$a0`, instead of reloading $N$ and subtracting 2. While this is technically correct (assuming that you restore the original value of `$a0` before you return from the procedure), it is error prone and bad coding practice. The MIPS convention dictates that you should make **no assumptions** about what will be returned in any registers other than `$s0-7`, `$sp`, `$gp` and `$ra`, which will have their values preserved.

6. *Clean up the stack and return the result.*

```
lw   $t0, 8($sp)   # retrieve first function result
add  $v0, $v0, $t0
lw   $ra, 0($sp)   # retrieve return address
addi $sp, $sp, 12
jr   $ra
```

## 2 MIPS to C

In the following MIPS assembly code, the value in register `$a0` is an input and the value in register `$v0` is the output.

1. Translate the following MIPS function back to equivalent C code:

```
func:
        addi   $t0, $zero, 1       # i = 1
        addi   $v0, $zero, 1       # v = 1
Loop:   sle    $t1, $t0, $a0       # set $t1 to 1 if (i <= arg)
        beq    $t1, $zero, Exit    # exit loop if (i > arg)
        mul    $v0, $v0, $t0       # v *= i
        addi   $t0, $t0, 1         # i++
        j      Loop                # loop
Exit:
        jr $ra
```

**Translated C function:**

```
int func (int arg){
     int v = 1, i;
     for (i = 1 ; i <= arg ; i++) {
        v = v * i;
     }
     return v;
}
```

2. What mathematical function does this code perform?
   **Factorial for non-negative arguments**