```
#Travis Ritter, Section: 01
.data
Prompt1: .asciiz "Enter N: "
AnswerMsg: .asciiz "Factorial: "
Error: .asciiz "Number must be >= 0"

input: .word 0
output: .word 0

.text

main: #loads prompts, accepts user input
    li $v0, 4 #load string print service
    la $a0, Prompt1 #load string into a0
    syscall # print

    li $v0, 5 #load int reading service
    syscall #read an int

    blt $v0, $0, error #if the user input is < 0, output an error,
               #since we cannot take the factorial of negative numbers
    sw $v0, input #store user input into global variable

    lw $a0, input #stores function parameter in global variable input
    jal factorial #jumps to factorial function
    sw  $v0, output #stores function return in global variable output

    li $v0, 4 #load string print service
    la $a0, AnswerMsg #load message into a0
    syscall #print message

    lw $a0, output #move the factorial total to a0
    li $v0, 1 #load int print service
    syscall #print int

    li $v0, 10 #load system exit service
    syscall #exit program

factorial: #recursive fucntion that uses the stack pointer to store,
       #and then multiply numbers to calculate the factorial of the input (N!)
    addi $sp, $sp, -8 #subtracting from the stack creates space, subtracting 8 bytes
              #makes space for two words
    sw $ra, ($sp) #stores the return address in the first position of the stack
    sw $s0, 4($sp) #stores the local variable (s0) on top of the ra

    li $v0, 1 #when we get to 0, we are done, in that case we want the function return to = 1
          #also 0! = 1, so this takes care of that edge case as well
    beq $a0, 0, endFunction #the loop is over when the parameter is 0, so jump away

    move $s0, $a0 #move the function argument into the local variable s0
    addi $a0, $a0, -1 #decrement the fucntion argument by 1
    jal factorial #loop back to factorial, and save return address

    mul $v0, $s0, $v0 #this is called recursively to multiply from the input down to 1

endFunction: #goes through and grabs the value of where to jump to,
         #and the number we are currently multiplying by.
    lw $ra, ($sp) #grab the first thing off the stack (the retrun address to multiply)
    lw $s0, 4($sp) #grab the next thing off the stack (the local variable)
    addi $sp, $sp, 8 #adding takes space away from stack, we are adding 8 bytes,
             #so we are taking two words.
    jr $ra #jump back to where we need

error: #if the input is negative, output an error message, and exit
    li $v0, 4 #load string print service
    la $a0, Error #load string into a0
    syscall # print

    li $v0, 10 #load system exit service
    syscall #exit program
```