Travis Ritter CMPE 220, 01

Description of the Program:

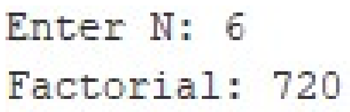
The purpose of this program is to find the factorial of a user inputted number (N) using recursive function calling in MIPS. This is done use $sp register, which allows us to allocate ordered memory in what is called ‘the stack’, and then recursively go back through and multiply the values saved on the stack to find the answer. The stack is a “first in, last out” data structure. Think of a can of pringles. If we want to eat the 7th chip, we must first eat the 6 above it, to reach it. This is how we will be storing our data in this program. With each pass of the factorial function, we are storing the return address, and function argument onto the stack. We then rewind the stack and multiply each number we get until we are out of things to multiply. That number is the result and is printed out.

Pseudocode: *

```
main:
        print "Enter N: "
        read v0
        if (v0 < 0) {
                print "Number must be >= 0"
                exit
        }

        store v0 in global variable 'input'

        loads a0 with 'input'
        jump (and link) to factorial function
        //after the factorial is done being calculated v0, will have the result, and the
        //rest of the program will be run from here
        Store v0 in global variable 'output'

        print "Factorial: "
        print 'output'
        exit

factorial:
        s0 = s0 + (-8) to make space for two words
        first position in stack = return address
        next position = s0 (local variable)

        v0 = 1 to account for base case
        if(a0 == 0) {
                Jump to label 'endFunction'
        } else {
                s0 = a0 to set local variable = current argument
                a0 = a0 -1 to decrement function argument
                Jump (and save ra) to factorial function
        }
        v0 = s0 * v0 called recursively to get factorial
endFunction:
        ra = first pos. of stack (where to jump to)
        s0 = next pos. of stack (actual value to be multiplied)
        sp = sp + 8 to remove 2 words from the stack
        jump to ra (multiply statement)
error:
        print “Number must be >= 0”
        exit
```

Outputs:

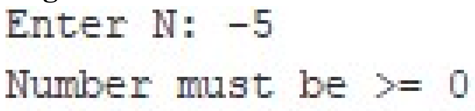
Positive:



```
Enter N: 6
Factorial: 720
```

Proof: 6! = 6 * 5 * 4 * 3 * 2 * 1 =
30 * 4 * 3 * 2 * 1 =
120 * 3 * 2 * 1 =
360 * 2 * 1 =
720 * 1 = **720**

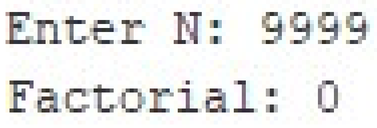
Negative:



```
Enter N: -5
Number must be >= 0
```

You cannot take the factorial of a negative number, by the definition of a factorial, so this is an error case. I display the proper error message, and exit the program

Large Integer:



```
Enter N: 9999
Factorial: 0
```

In reality 9999! = a very large number, but in MIPS we only have a certain number of numbers we can represent, due to the limits of integer overflow. So, this number is outside the bounds that MIPS can represent, and since we cannot represent it, we get 0.

MIPS Source Code on next pg.

```
#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 m
akes 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
```