Procedures and Stack Management

Colt Uhlenhopp and Jack Utzerath

Grand Canyon University

CST-307

Professor Citro

2/26/2023

Procedures and Stack Management

1. The flowchart is attached in a separate submitted document.
2. Complete assembly code:

****DO NOT COPY AND PASTE, .ASM FILE IS ATTACHED****

```
#Jack Utzerath and Colt
#Fib sequence


# Parameters:
# if ( n < 2)
   #return n;
# else
   # return fib (n-1) + fib (n-2)


.text
main:
   la $a0, getInput #ask user for a number
   li $v0, 4
   syscall


   li $v0, 5 #read in the number
   syscall
   move $t1, $v0 #move input to t1
```

```
move $a0, $t1
move $v0, $t1
jal fib # calling fib sequence

move $t2, $v0 #put answer in t2

la $a0, mes #Print result message
li $v0, 4
syscall

move $a0, $t1 #print user input
li $v0, 1
syscall

la $a0, mes2 #Print =
li $v0, 4
syscall

move $a0, $t2 #print answer
li $v0, 1
syscall

li $v0, 10 #exit
```

```
    syscall

fib: #this is the base case
    bgt $a0, 1 , fib_function #comparing the value to one- go into recursive case if true
    move $v0, $a0 # return value gets put into v0
    jr $ra #job register

fib_function: # this is a recursive function
    sub $sp, $sp, 12 #initializing a stack frame storing 3 register on the stack
    sw $ra, 0($sp) #write into ra to save return address
    sw $a0, 4($sp) #saving n in case we need it after

    #fib (n-1)
    addi $a0, $a0, -1 #this is adding -1 and n    (n-1)
    jal fib # call base case with return value
    lw $a0, 4($sp) #restore n
    sw $v0, 8($sp) #saving value from fib(n-1)

    #fib (n-2)
    addi $a0, $a0, -2 #this is adding -2 and n    (n-2)
    jal fib # call base case with return value
    lw $t0, 8($sp) #restore n to register t0 so we dont override return value
```

add $v0, $t0, $v0 #return value (return fib (n-1) + fib (n-2))

lw $ra, 0($sp) #restore $ra

addi $sp, $sp ,12 #deallocate the stack

jr $ra #gets back to the main program

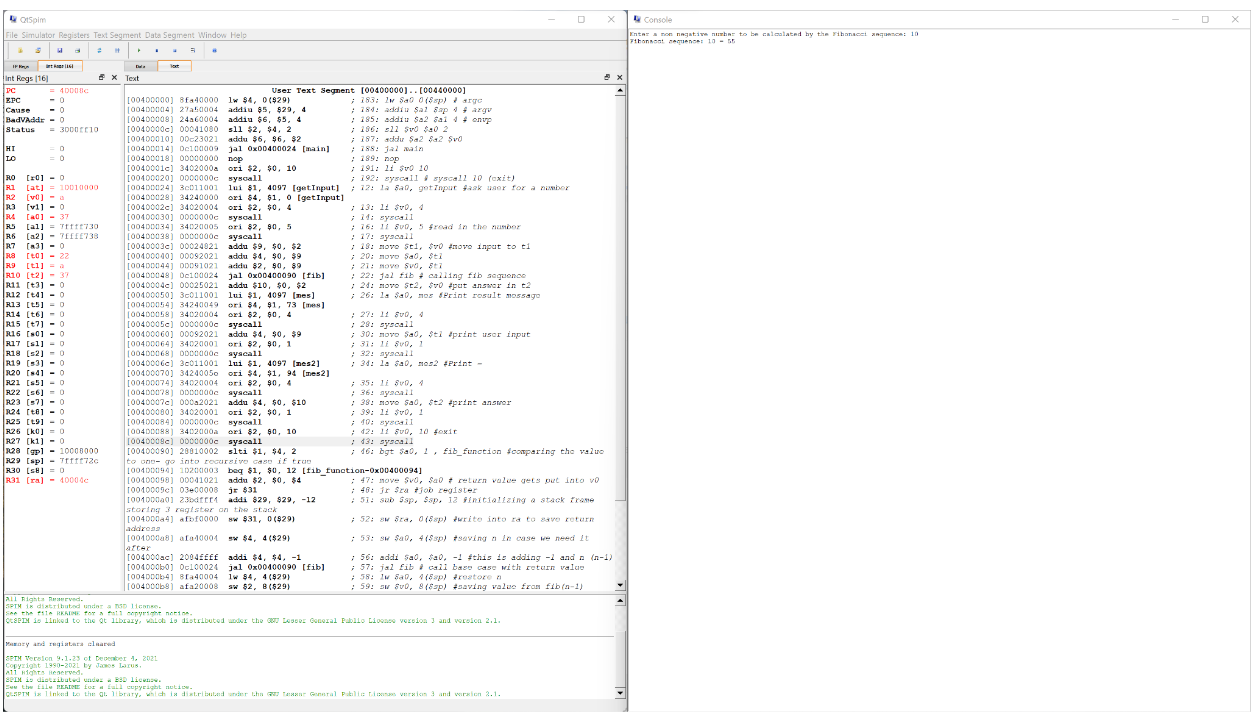
.data

getInput: .asciiz "Enter a non negative number to be calculated by the Fibonacci sequence: "

mes: .asciiz "Fibonacci sequence: "

mes2: .asciiz " = "

3. Execution screenshots



4. **Saved vs. Temporary Registers:**

In our program, we decided to use temporary registers. This choice was made for a few different reasons. Not only are we more familiar with the use of temporary registers, saved registers take a minor performance hit when not used in the proper scenario. Saved registers have to be preserved in the code, and whenever a new saved register is called, the original value needs to be put into the stack memory and then restored before returning it to the caller.

5. **An explanation of why in about half the calls to the recursive function the argument is 1 or 0**

The argument in about half of the calls to the recursive function is 1 or 0 because of the way that the fibonacci sequence works. The Fibonacci sequence starts with 0, then the next two terms are both 1. So in each Fibonacci sequence chosen by the user, if the user wants to see at least three terms then the first three terms will always be the argument 0 or 1. This doesn't necessarily make half of the calls to the recursive function the argument 1 or 0, but a large portion of the arguments will be 1 or 0 because they are the most predominant numbers in the Fibonacci sequence.

6. **6. An explanation of simplifications made related to the stack frame:**

Each function in the program has a local memory that is associated with it. It holds incoming parameters and local variables. This part of the memory is the stack frame and is allocated on the certain process's stack.