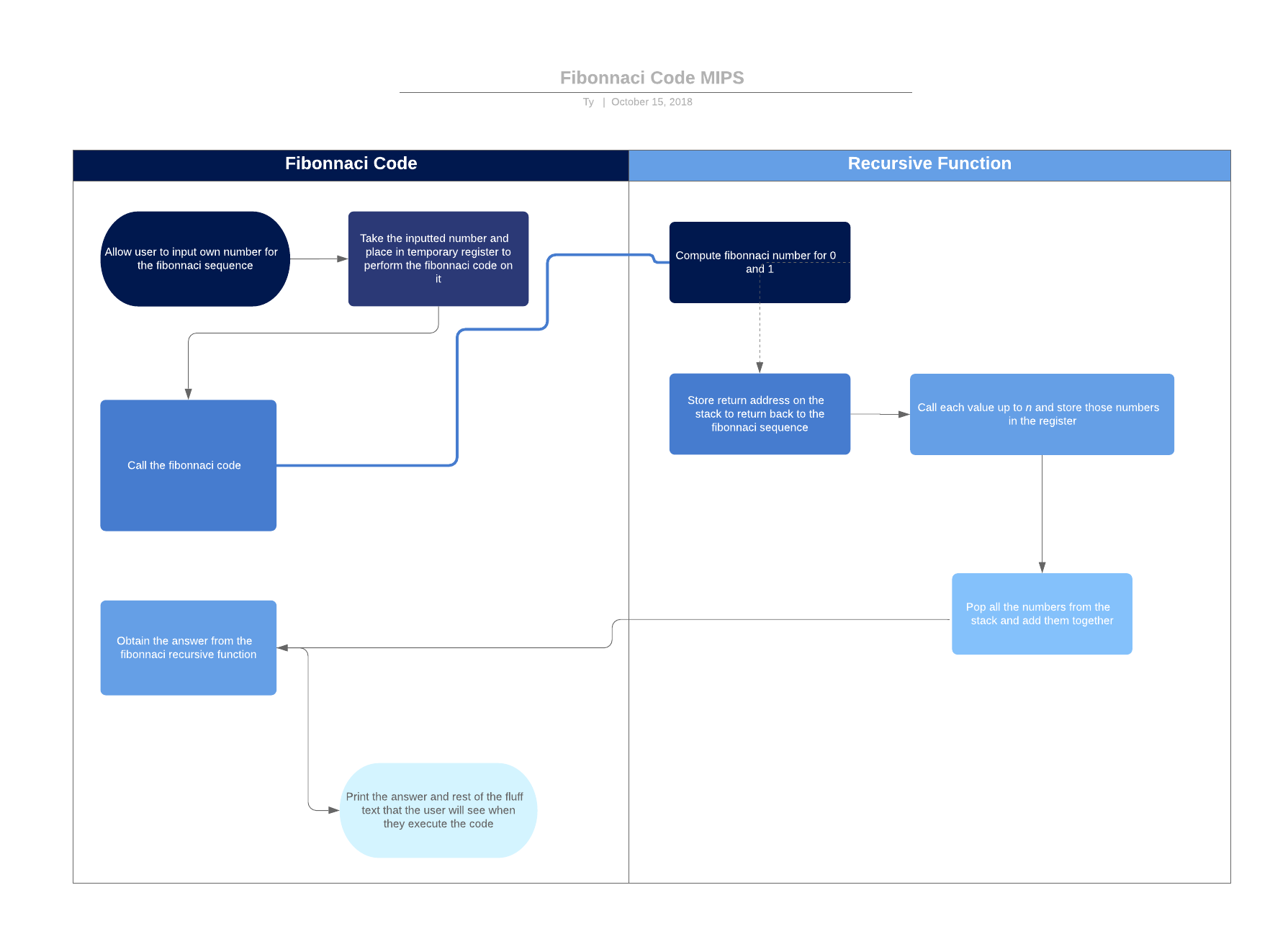
Tyler Nichols

CST-307

10/15/18

Dr. Citro

1. I used a mixture of both saved and temporary registers. The reason for this was because when I would move the number from one register to a second register, or when I was adding the numbers, I would use a temporary register as a place holder for that number to not have to clear a saved register on a number that I would not need in the future. For example, in my code I moved the number that the user inputted to a temporary register $t2 from $v0 to open the register $v0 and then moved the $t2 to $a0 and $v0 to then perform an operation on those saved registers.
2. In about half the calls to the recursive function the argument is 1 or 0 because those are the at the top of the call stack and in order to obtain the numbers from the rest of the stack, it must remove the information at the top of the call stack and work its way down, placing all the numbers on the call stack back on the stack in the same order.
3. In the stack frame I simply store the return address of the Fibonacci code so the program could return back to it, and stored all the values from the Fibonacci code and only popped the values from the stack until the Fibonacci code was finished.



# Tyler Nichols

# The purpose of this program is to show our competency in MIPS programming

# by creating a code that can take the fibonnaci sequence and output the number

# at the user's request

.text

main:

# Prompt user to input non-negative number

la $a0,prompt

li $v0,4

syscall

li $v0,5 #Read the number that the user inputted

syscall

move $t2,$v0 # n to $t2

# Call function to get fibonnacci #n

move $a0,$t2

move $v0,$t2

jal fib #call fib (n)

move $t3,$v0 #result is in $t3

# Output message and n

la $a0,result #Print message 2

li $v0,4

syscall

move $a0,$t2 #Print n

li $v0,1

syscall

la $a0,result2 #Print message 3

li $v0,4

syscall

move $a0,$t3 #Print the answer

li $v0,1

syscall

la $a0,endl #Print '\n'

li $v0,4

syscall

# End program

li $v0,10

syscall

fib:

# Compute and return fibonacci number

beqz $a0,zero #if n=0 return 0

beq $a0,1,one #if n=1 return 1

#Calling fib(n-1)

sub $sp,$sp,4 #storing return address on stack

sw $ra,0($sp)

sub $a0,$a0,1 #n-1

jal fib #fib(n-1)

add $a0,$a0,1

lw $ra,0($sp) #restoring return address from stack

add $sp,$sp,4

sub $sp,$sp,4 #Push return value to stack

sw $v0,0($sp)

#Calling fib(n-2)

sub $sp,$sp,4 #storing return address on stack

sw $ra,0($sp)

sub $a0,$a0,2 #n-2

jal fib #fib(n-2)

add $a0,$a0,2

lw $ra,0($sp) #restoring return address from stack

add $sp,$sp,4

lw $s7,0($sp) #Pop return value from stack

add $sp,$sp,4

add $v0,$v0,$s7 # f(n - 2)+fib(n-1)

jr $ra # decrement/next in stack

zero:

li $v0,0

jr $ra

one:

li $v0,1

jr $ra

.data

prompt: .asciiz "Enter a non-negative number -1 to compute\n(i.e. if you want to compute 13 enter the number 14)"

result: .asciiz "Fibonacci sequence up to number\_"

result2: .asciiz " = "

endl: .asciiz "\n"

