

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

The screenshot displays the QtSpim MIPS simulator interface. The main window shows the assembly code for a Fibonacci sequence calculator. The code includes instructions for loading arguments, calculating the Fibonacci sequence, and restoring the return address. The console window on the right shows the program's execution, including the input prompt and the resulting Fibonacci sequence.

Assembly Code:

```

PC = 10000c
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000c10
HI = 0
LO = 0
R0 (x0) = 0
R1 (x1) = 10010000
R2 (x2) = 0
R3 (x3) = 0
R4 (x4) = 0
R5 (x5) = 0
R6 (x6) = 0
R7 (x7) = 0
R8 (x8) = 0
R9 (x9) = 0
R10 (x10) = 0
R11 (x11) = 0
R12 (x12) = 0
R13 (x13) = 0
R14 (x14) = 0
R15 (x15) = 0
R16 (x16) = 0
R17 (x17) = 0
R18 (x18) = 0
R19 (x19) = 0
R20 (x20) = 0
R21 (x21) = 0
R22 (x22) = 0
R23 (x23) = 0
R24 (x24) = 0
R25 (x25) = 0
R26 (x26) = 0
R27 (x27) = 0
R28 (x28) = 0
R29 (x29) = 0
R30 (x30) = 0
R31 (x31) = 0

```

Console Output:

```

Enter a non negative number to be calculated by the fibonacci sequence: 10
Fibonacci sequence: 10 = 55

```

Assembly Code Details:

```

[00400000] 8fa40000 lw $4, 0($29) ; 103: lw $a0, 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 104: addiu $a1, $sp, 4 # argv
[00400008] 24a40004 addiu $6, $5, 4 ; 105: addiu $a2, $a1, 4 # comp
[0040000c] 00a41080 sll $2, $4, 2 ; 106: sll $v0, $a0, 2
[00400010] 00c23021 addu $6, $6, $2 ; 107: addu $a2, $a2, $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 108: jal main
[00400018] 00000000 nop ; 109: nop
[0040001c] 34a2000a ori $2, $0, 10 ; 110: li $v0, 10
[00400020] 0000000c syscall ; 111: syscall # syscall 10 (exit)
[00400024] 3e111001 lui $1, 4097 [getInput] ; 112: la $a0, getInput #ask user for a number
[00400028] 34a2000a ori $4, $1, 0 [getInput] ; 113: li $v0, 4
[0040002c] 34a20004 ori $2, $0, 4 ; 114: li $v0, 4
[00400030] 0000000c syscall ; 115: syscall
[00400034] 34a20005 ori $2, $0, 5 ; 116: li $v0, 5 #read in the number
[00400038] 0000000c syscall ; 117: syscall
[0040003c] 00024e21 addu $9, $0, $2 ; 118: move $t1, $v0 #move input to t1
[00400040] 00920201 addu $4, $0, $9 ; 119: move $a0, $t1
[00400044] 00910201 addu $2, $0, $9 ; 120: move $v0, $t1
[00400048] 0c100024 jal 0x00400090 [fib] ; 121: jal fib # calling fib sequence
[0040004c] 00000000 nop ; 122: nop
[00400050] 3e111001 lui $1, 4097 [mes] ; 123: la $a0, mes #print result message
[00400054] 34a2000a ori $4, $1, 73 [mes] ; 124: li $v0, 4
[00400058] 34a20004 ori $2, $0, 4 ; 125: li $v0, 4
[0040005c] 0000000c syscall ; 126: syscall
[00400060] 00920201 addu $4, $0, $9 ; 127: move $a0, $t1 #print user input
[00400064] 34a2000a ori $4, $1, 73 [mes2] ; 128: li $v0, 4
[00400068] 0000000c syscall ; 129: syscall
[0040006c] 3e111001 lui $1, 4097 [mes2] ; 130: la $a0, mes2 #print =
[00400070] 34a2000a ori $4, $1, 73 [mes2] ; 131: li $v0, 4
[00400074] 34a20004 ori $2, $0, 4 ; 132: li $v0, 4
[00400078] 0000000c syscall ; 133: syscall
[0040007c] 00920201 addu $4, $0, $9 ; 134: move $a0, $t2 #print answer
[00400080] 34a2000a ori $4, $1, 73 [mes2] ; 135: li $v0, 4
[00400084] 0000000c syscall ; 136: syscall
[00400088] 34a2000a ori $2, $0, 10 ; 137: li $v0, 10 #exit
[0040008c] 00000000 nop ; 138: nop
[00400090] 28810002 sll $1, $4, 2 ; 139: sll $v0, 1, $v0 #comparing the value
[00400094] 00000000 nop ; 140: nop
[00400098] 00910201 addu $2, $0, $4 ; 141: move $v0, $a0 # return value gets put into v0
[0040009c] 00000000 nop ; 142: nop
[004000a0] 23b0ff14 addi $29, $29, -12 ; 143: sub $sp, $sp, 12 #initializing a stack frame
[004000a4] afa40000 sw $31, 0($29) ; 144: sw $ra, 0($sp) #write into ra to save return
[004000a8] afa40001 sw $4, 4($29) ; 145: sw $a0, 4($sp) #saving n in case we need it
[004000ac] 2094ffff addi $4, $4, -1 ; 146: addi $a0, $a0, -1 #this is adding -1 and n (n-1)
[004000b0] 0c100024 jal 0x00400090 [fib] ; 147: jal fib # call base case with return value
[004000b4] 8fa40004 lw $4, 4($29) ; 148: lw $a0, 4($sp) #restore n
[004000b8] afa20008 sw $2, 0($29) ; 149: sw $v0, 0($sp) #saving value from fib(n-1)

```

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.