## **Homework 9: Stack Frame**

## CS 200 • 10 Points Total Due Wednesday, April 19, 2017

## **Assignment**

```
The Fibonacci series is defined as f(x) = \begin{cases} 0, & x = 0 \\ 1, & x = 1 \end{cases}
Thus, the series leader like f(x) = (x + 1) + f(x - 2), & x > 1
```

Thus, the series looks like (0, 1, 1, 2, 3, 5, 8, ...). To further help you envision it, here is a function in a high-level language that returns the  $n^{th}$  value of a Fibonacci series:

```
/*
 * int fib(int n) takes an integer parameter and returns the value of
 * the Fibonacci series at the nth position. To do this, it returns 0
 * if n is 0, 1 if n is one, or recursively calls itself to get the
 * values of fib(n - 1) and fib(n-2) to add together and return
 */
int fib(int n)
{
    // base cases
    if ((n == 1) || (n == 0) return n;

    // if we didn't return in the base cases, we need to recursively
    // call fib() to get the result...
    return (fib(n - 1) + fib(n - 2));
}
```

Write the equivalent function in MIPS assembly. You will need to use a stack frame, of course. I provide the following skeleton code to prompt for an integer. You can use it to test your function. You just need to pay attention to the comments and code in red to call your function; the rest is done for you. Of course, you also have to write the fib() function itself. Submit your code (you can paste it into your submission document) along with a screenshot of your testing.

```
# TITLE fibonacci
                       (fibonacci.s)
# This program handles the I/O for a fibonacci function.
# variables
IntPrompt: .asciiz "Enter a integer between 0 a
OutStr: .asciiz "\nThe Fibonacci value is "
LowError: .asciiz "\nYour input was too small.
                              "Enter a integer between 0 and 25: "
                             "\nYour input was too small. Try again: "
                               "\nYour input was too large. Try again: "
HighError:
               .asciiz
AgainStr:
               .asciiz
.asciiz
                              "\nWould you like to try again? (y/n):"
                              "\n"
NewLine:
                              5
YesNoBuf:
                                      # Plenty of room for 'yes' or 'no'
                .space
                              0
IntIn:
               .word
IntMin:
                .word
                               0
IntMax:
                .word
       .text
               main
       .globl
again: # print a newline on subsequent returns to main
            $a0, NewLine # point to NewLine
        la
       li
               $v0, 4
                                      # print string
       syscall
main: # start of the main procedure
```

```
# Get an integer
       la $a0, IntPrompt  # point to IntPrompt
li $v0, 4  # print string
                                   # print string
       syscall
GetInt: li $v0, 5
                                    # read integer
      syscall
                                   # move input before it gets changed
       move $t0, $v0
       # check if below min
                                   # load our lower bound
       lw $a1, IntMin
              $v0, $a1, BigEnough
                                  # if good, try next check
       bae
            $a0, LowError
       la
                                   # point to Error string
       li
              $v0, 4
                                   # print string
       syscall
       j
               GetInt
       # check if above max
BigEnough:
       ٦w
              $a1, IntMax
                              # load our upper bound
       ble
              $v0, $a1, SmallEnough # if good, try next check
      $a0, HighError # point to Error string
$v0, 4 # print_string
       syscall
             GetInt
SmallEnough:
      # save the input, just in case
       sw $v0, IntIn
       # Print the text to go with the output
       la $a0, OutStr # point to OutStr
li $v0, 4 # print_string
       syscall
# Call your fib routine here. When it returns, put the return value
# in $a0 and print it out. Comment out the next block, which just
# prints out a dummy number.
       # Print a dummy number for output testing. 16 is not a
       # valid return from fib() so I should not see it in anyone's
       # final output.
       li $a0, 16
li $v0, 1
                                   # not a valid number
                                  # print integer
       syscall
       # Print a newline before continuing
       la $a0, NewLine # point to NewLine
li $v0, 4 # print string
                                   # print string
       syscall
       # Prompt to see if the user wants to do it again
       la $a0, AgainStr # point to AgainStr li $v0, 4 # print_string
       syscall
       # Get the input
       la $a0, YesNoBuf
                                  # point to YesNoBuf
       li.
             $a1, 5
                                    # length of buffer
       li
                                    # read string
             $v0, 8
       syscall
            $t0, YesNoBuf
                                   # load the first character into $t0
       1b
       # Test if first character is 'Y'
       li $t1, 89 # ASCII for 'Y'
                                  # equal, so run program again
       beq $t0, $t1, again
       # Test if first character is 'y'
       li $t1, 121 # ASCII for 'y'
       beq $t0, $t1, again
                                  # equal, so run program again
       # Not 'yes', so assume 'no' and end program
       jr $ra
# fib function goes below here:
```

## - SOLUTION -

```
Solution code is in green:
# TITLE fibonacci (fibonacci.s)
# This program handles the I/O for a fibonacci function.
       .data
# variables
IntPrompt:
              .asciiz
                            "Enter a integer between 0 and 25: "
OutStr:
              .asciiz
                            "\nThe Fibonacci value is "
                           "\nYour input was too small. Try again: "
              .asciiz
.asciiz
LowError:
HighError:
                            "\nYour input was too large. Try again: "
                            "\nWould you like to try again? (y/n): "
AgainStr:
              .asciiz
               .asciiz
NewLine:
                            "\n"
                            5
               .space
                                    # Plenty of room for 'yes' or 'no'
YesNoBuf:
              .word
IntIn:
                            0
IntMin:
              .word
                            0
              .word
IntMax:
                            2.5
      .text
       .globl main
again: # print a newline on subsequent returns to main
       la $a0, NewLine # point to NewLine li $v0, 4 # print_string
       syscall
main: # start of the main procedure
       # Get an integer
       la $a0, IntPrompt
li $v0, 4
                                   # point to IntPrompt
                                   # print string
       syscall
             $v0, 5
GetInt: li
                                    # read integer
      syscall
       move $t0, $v0
                                   # move input before it gets changed
       # check if below min
       lw $a1, IntMin
                                   # load our lower bound
              $v0, $a1, BigEnough
                                  # if good, try next check
              $a0, LowError
                                   # point to Error string
       la
                                   # print string
       li
              $v0, 4
       syscall
               GetInt
       # check if above max
BigEnough:
              $a1, IntMax
                                   # load our upper bound
       lw
              $v0, $a1, SmallEnough # if good, try next check
       ble
       la
             $a0, HighError # point to Error string
             $v0, 4
                                   # print string
       1 i
       syscall
       j
              Get.Int.
SmallEnough:
       # save the input, just in case
            $v0, IntIn
       # Print the text to go with the output
            $a0, OutStr # point to OutStr
       la
       li
              $v0, 4
                                   # print string
       syscall
       # I made it so I don't have to worry about any registers
       # but $t0, which holds my parameter, and $ra. So just
       # push $ra and load up $t0.
              $ra, -4($sp)
                                      # push $ra
       SW
             $sp, $sp, -4
       addiu
               $t0, IntIn
       l w
```

```
# put param and return on stack, call, and return
       $sp, $sp, -8
                                      # call the routine
       jal
               fib
               $t1, 4($sp)
                                     # our return into $t1
       1 w
       ٦w
              $t0, 0($sp)
                                      # pop $t0
             $sp, $sp, 8
       addiu
                                      # restore $sp
       # restore the return address
              $ra, 0($sp)
$sp, $sp, 4
       l w
                                       # pop $ra
       addin
       # print the returned value
       move $a0, $t1
                                      # put return in $a0
               $v0, 1
                                      # print integer
       syscall
      la $a0, NewLine # point to NewLine li $v0, 4 # point to NewLine
       # Print a newline before continuing
                                   # print_string
       syscall
       # Prompt to see if the user wants to do it again
      la $a0, AgainStr # point to AgainStr
li $v0, 4 # print_string
       svscall
       # Get the input
       la $a0, YesNoBuf
                                  # point to YesNoBuf
              $a1, 5
                                   # length of buffer
       li
                                   # read_string
            $v0, 8
       1 i
       syscall
       lb $t0, YesNoBuf
                                   # load the first character into $t0
       # Test if first character is 'Y'
       li $t1, 89
                                    # ASCII for 'Y'
                               # ASCII for 1
# equal, so run program again
             $t0, $t1, again
       # Test if first character is 'y'
      li
                                   # ASCII for 'v'
            $t1, 121
                                  # equal, so run program again
           $t0, $t1, again
       # Not 'yes', so assume 'no' and end program
              $ra
# The fibonacci routine. Takes a stack frame with the return at 4($sp) and an
# integer parameter at 0($sp). If the parameter is 0 or 1, simply returns the
# parameter. Otherwise, it recursively calls itself with the parameter - 1 and
# the parameter - 2 and adds the two returns together to make the return for
# the present iteration.
       # Stack frame fixup: standard code
              $fp, -4($sp) # push frame pointer
              $fp, $sp
      move
                                     # point frame at current stack pointer
       addiu $sp, $sp, -4
              $ra, -4($sp)
                                     # push the return address
       SW
       addiu $sp, $sp, -4
      addiu $sp, $sp, -4
                                     # room for a one word local var @ -12($fp)
       SW
              $t0, -4($sp)
                                     # push $t0
       addiu $sp, $sp, -4
sw $t1, -4($sp)
                                     # push $t1
       addiu $sp, $sp, -4
       # test the input
       lw $t0, 0($fp)
                                    # load the parameter into $t0
             $t1, 2
                                    # test value into $t1
       bge $t0, $t1, recurse
sw $t0, 4($fp)
                                    # bigger than base cases so go recurse
# otherwise cleanup and return
```

```
# clean up the stack and return
cleanup:
       lw
               $t1, 0($sp)
                                        # pop $t1
       addiu
              $sp, $sp, 4
               $t0, 0($sp)
                                         # pop $t0
       ٦w
       addiu $sp, $sp, 4
       addiu $sp, $sp, 4
                                        # recover space for local variable(s)
               $ra, 0($sp)
       1 w
                                         # pop $ra
       addiu $sp, $sp, 4
                                         # pop frame pointer
       l w
               $fp, 0($sp)
       addiu
              $sp, $sp, 4
                                         # return to caller
       jr
               $ra
       # do the recursive case
recurse:
       \# subtract 1 from the parameter in $t0 (now n - 1)
       addiu $t0, $t0, -1
       # put param and return on stack, call, and return
                                       # push $t0 (our parameter)
               $t0, -8($sp)
       addiu
                 $sp, $sp, -8
                                         # adjust $sp (-4 is return)
                 fib
                                         # call the routine
       jal
                 $t1, 4($sp)
                                         # our return into $t1
                $t0, 0($sp)
$sp, $sp, 8
       7 w
                                         # pop $t0
       addiu
                                         # restore $sp
       # put return into local variable for temp storage
       sw $t1, -12($fp)
                                   # $t1 into local variable
       \# subtract 1 from the parameter in $t0 (now n - 2)
       addiu $t0, $t0, -1
       \ensuremath{\sharp} put param and return on stack, call, and return
               $t0, -8($sp)  # push $t0 (our parameter)
       addiu
                 $sp, $sp, -8
                                         # adjust $sp (-4 is return)
       jal
                 fib
                                         # call the routine
                $t1, 4($sp)
       lw
                                         # our return into $t1
       lw $t0, 0($sp) addiu $sp, $sp, 8
                                         # pop $t0
                                         # restore $sp
       # load local variable into $t0 and add to $t1
               $t0, -12($fp)
$t0, $t0, $t1
       l w
                                        # get local variable
       addu
                                         # add the returns
               $t0, 4($fp)
                                         # put result in return
       SW
                                         # clean up and return to caller
               cleanup
                                                           _ 🗆 X
                                Console
 Enter a integer between 0 and 25: 0
 The Fibonacci value is 0
 Would you like to try again? (y/n): y
 Enter a integer between 0 and 25: 1
 The Fibonacci value is 1
 Would you like to try again? (y/n): y
 Enter a integer between 0 and 25: 2
 The Fibonacci value is 1
 Would you like to try again? (y/n): y
 Enter a integer between 0 and 25: 6
 The Fibonacci value is 8
 Would you like to try again? (y/n): y
 Enter a integer between 0 and 25: 10
 The Fibonacci value is 55
```

Would you like to try again? (y/n): n