# Homework 9: Stack Frame

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

### Assignment

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

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.

        .data
# variables
IntPrompt:      .asciiz       "Enter a integer between 0 and 25: "
OutStr:         .asciiz       "\nThe Fibonacci value is "
LowError:       .asciiz       "\nYour input was too small.  Try again: "
HighError:      .asciiz       "\nYour input was too large.  Try again: "
AgainStr:       .asciiz       "\nWould you like to try again? (y/n): "
NewLine:        .asciiz       "\n"
YesNoBuf:       .space        5       # Plenty of room for 'yes' or 'no'
IntIn:          .word         0
IntMin:         .word         0
IntMax:         .word         25

        .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          # point to IntPrompt
        li      $v0, 4                  # print_string
        syscall

GetInt: li      $v0, 5                  # read_integer
        syscall
         move   $t0, $v0                # move input before it gets changed

        # check if below min
        lw      $a1, IntMin             # load our lower bound
        bge     $v0, $a1, BigEnough     # if good, try next check
        la      $a0, LowError           # point to Error string
        li      $v0, 4                  # print_string
        syscall
        j         GetInt

        # check if above max
BigEnough:
        lw      $a1, IntMax             # load our upper bound
        ble     $v0, $a1, SmallEnough   # if good, try next check
        la      $a0, HighError          # point to Error string
        li      $v0, 4                  # print_string
        syscall
        j       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                 # not a valid number
        li      $v0, 1                  # print_integer
        syscall

        # Print a newline before continuing
        la      $a0, NewLine            # point to NewLine
        li      $v0, 4                  # 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      $v0, 8                  # read_string
        syscall
        lb      $t0, YesNoBuf           # load the first character into $t0

        # Test if first character is 'Y'
        li      $t1, 89                 # ASCII for 'Y'
        beq     $t0, $t1, again         # equal, so run program 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 "
LowError:       .asciiz       "\nYour input was too small.  Try again: "
HighError:      .asciiz       "\nYour input was too large.  Try again: "
AgainStr:       .asciiz       "\nWould you like to try again? (y/n): "
NewLine:        .asciiz       "\n"
YesNoBuf:       .space        5       # Plenty of room for 'yes' or 'no'
IntIn:          .word         0
IntMin:         .word         0
IntMax:         .word         25

        .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         # point to IntPrompt
        li      $v0, 4                 # print_string
        syscall

GetInt: li      $v0, 5                 # read_integer
        syscall
        move    $t0, $v0               # move input before it gets changed

        # check if below min
        lw      $a1, IntMin            # load our lower bound
        bge     $v0, $a1, BigEnough    # if good, try next check
        la      $a0, LowError          # point to Error string
        li      $v0, 4                 # print_string
        syscall
        j         GetInt

        # check if above max
BigEnough:
        lw      $a1, IntMax            # load our upper bound
        ble     $v0, $a1, SmallEnough  # if good, try next check
        la      $a0, HighError         # point to Error string
        li      $v0, 4                 # print_string
        syscall
        j       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

        # 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.
        sw        $ra, -4($sp)            # push $ra
        addiu     $sp, $sp, -4
        lw        $t0, IntIn
```

```
        # put param and return on stack, call, and return
        sw      $t0, -8($sp)            # push $t0 (our parameter)
        addiu   $sp, $sp, -8            # adjust $sp (-4 is return)
        jal     fib                     # call the routine
        lw      $t1, 4($sp)             # our return into $t1
        lw      $t0, 0($sp)             # pop $t0
        addiu   $sp, $sp, 8             # restore $sp

        # restore the return address
        lw      $ra, 0($sp)             # pop $ra
        addiu   $sp, $sp, 4

        # print the returned value
        move    $a0, $t1                # put return in $a0
        li      $v0, 1                  # print_integer
        syscall

        # Print a newline before continuing
        la      $a0, NewLine            # point to NewLine
        li      $v0, 4                  # 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      $v0, 8                  # read_string
        syscall
        lb      $t0, YesNoBuf           # load the first character into $t0

        # Test if first character is 'Y'
        li      $t1, 89                 # ASCII for 'Y'
        beq     $t0, $t1, again         # equal, so run program 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

# 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.
fib:
        # Stack frame fixup: standard code
        sw      $fp, -4($sp)            # push frame pointer
        move    $fp, $sp                # point frame at current stack pointer
        addiu   $sp, $sp, -4
        sw      $ra, -4($sp)            # push the return address
        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
        li      $t1, 2                  # test value into $t1
        bge     $t0, $t1, recurse       # bigger than base cases so go recurse
        sw      $t0, 4($fp)             # otherwise cleanup and return
```

```
        # clean up the stack and return
cleanup:
        lw      $t1, 0($sp)             # pop $t1
        addiu   $sp, $sp, 4
        lw      $t0, 0($sp)             # pop $t0
        addiu   $sp, $sp, 4
        addiu   $sp, $sp, 4             # recover space for local variable(s)
        lw      $ra, 0($sp)             # pop $ra
        addiu   $sp, $sp, 4
        lw      $fp, 0($sp)             # pop frame pointer
        addiu   $sp, $sp, 4
        jr      $ra                     # return to caller

        # 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
        sw        $t0, -8($sp)            # push $t0 (our parameter)
        addiu     $sp, $sp, -8            # adjust $sp (-4 is return)
        jal       fib                     # call the routine
        lw        $t1, 4($sp)             # our return into $t1
        lw        $t0, 0($sp)             # pop $t0
        addiu     $sp, $sp, 8             # 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

        # put param and return on stack, call, and return
        sw        $t0, -8($sp)            # push $t0 (our parameter)
        addiu     $sp, $sp, -8            # adjust $sp (-4 is return)
        jal       fib                     # call the routine
        lw        $t1, 4($sp)             # our return into $t1
        lw        $t0, 0($sp)             # pop $t0
        addiu     $sp, $sp, 8             # restore $sp

        # load local variable into $t0 and add to $t1
        lw      $t0, -12($fp)             # get local variable
        addu    $t0, $t0, $t1             # add the returns
        sw      $t0, 4($fp)               # put result in return
        j       cleanup                   # clean up and return to caller
```