Iterative Factorial

https://github.com/dashdanw/MIPS-Assembly/blob/master/factorial-iterative.s

.data

nl: .asciiz "\n"

.align 2

name: .asciiz "Casey Bladow\n\n\n"

.align 2

msg1: .asciiz "! is equal to HI:"

.align 2

lomsg: .asciiz " LO:"

.align 2

space: .asciiz " "

.align 2

.text

.globl main

main: li $a3,15 #stores 15 as function parameter

li $t0,1

la $a0,name #system calls use a0 for argument, and v0 for return value to pass to system

li $v0,4

syscall

move $a0,$a3

li $v0,1

syscall

la $a0,msg1

li $v0,4

syscall

ble $a3,1,print

loop: mult $t0,$a3 #uses temp dirs as not to overlap the lo multiplication with the high multiplication

mflo $t0 #preserves the return values of old multiplication to calculate overflow

mfhi $t2

mult $t1,$a3

mflo $t3

add $t1,$t2,$t3 #add into temps

addiu $a3,-1 #decrement function argument for iterative call

bge $a3,2,loop

print: move $a0,$t1

li $v0,1

syscall

la $a0,lomsg

li $v0,4

syscall

move $a0,$t0

li $v0,1

syscall

Exit: li $v0,10

syscall

Iterative Fibonacci

# http://www.cs.usfca.edu/~peter/cs315/code.html

# Program to read a positive integer n, and compute the nth Fibonacci number:

#

# f\_0 = 0

# f\_1 = 1

# f\_n = f\_(n-1) + f\_(n-2), n >= 2

.text

.globl main

main:

subu $sp, $sp, 4 # Make additional stack space.

sw $ra, 0($sp) # Save the return address

# Ask the OS to read a number and put it in a temporary register

li $v0, 5 # Code for read int.

syscall # Ask the system for service.

move $t0, $v0 # Put n in a safe place

# The loop

li $t2, 1 # Initialize f\_old to 1

li $t1, 0 # Initialize f\_older to 0

li $t4, 2 # Initialize counter i to 2

lp\_tst: bgt $t4, $t0, done # If $t4 > $t0 (i > n),

# branch out of loop.

# Otherwise continue.

add $t3, $t2, $t1 # Add f\_old to f\_older

move $t1, $t2 # Replace f\_older with f\_old

move $t2, $t3 # Replace f\_old with f\_new

addi $t4, $t4, 1 # Increment i (i++)

j lp\_tst # Go to the loop test

# Done with the loop, print result

done: li $v0, 1 # Code to print an int

move $a0, $t2 # Put f\_old in $a0

syscall # Print the string

# Restore the values from the stack, and release the stack space.

lw $ra, 0($sp) # Retrieve the return address

addu $sp, $sp, 4 # Make additional stack space.

# Return -- go to the address left by the caller.

# jr $ra

li $v0, 10

syscall

Recursive Factorial

# https://gist.github.com/dcalacci/3747521

.globl main

.data

msgprompt: .word msgprompt\_data

msgres1: .word msgres1\_data

msgres2: .word msgres2\_data

msgprompt\_data: .asciiz "Positive integer: "

msgres1\_data: .asciiz "The value of factorial("

msgres2\_data: .asciiz ") is "

# every function call has a stack segment of 12 bytes, or 3 words.

# the space is reserved as follows:

# 0($sp) is reserved for the initial value given to this call

# 4($sp) is the space reserved for a return value

# 8($sp) is the space reserved for the return address.

# calls may manipulate their parent's data, but parents may not

# manipulate their child's data.

# i.e: if we have a call A who has a child call B:

# B may run:

# sw $t0, 16($sp)

# which would store data from $t0 into the parent's return value register

# A, however, should not(and, in all cases I can think of, cannot) manipulate

# any data that belongs to a child call.

.text

main:

# printing the prompt

#printf("Positive integer: ");

la $t0, msgprompt # load address of msgprompt into $t0

lw $a0, 0($t0) # load data from address in $t0 into $a0

li $v0, 4 # call code for print\_string

syscall # run the print\_string syscall

# reading the input int

# scanf("%d", &number);

li $v0, 5 # call code for read\_int

syscall # run the read\_int syscall

move $t0, $v0 # store input in $t0

move $a0, $t0 # move input to argument register $a0

addi $sp, $sp, -12 # move stackpointer up 3 words

sw $t0, 0($sp) # store input in top of stack

sw $ra, 8($sp) # store counter at bottom of stack

jal factorial # call factorial

# when we get here, we have the final return value in 4($sp)

lw $s0, 4($sp) # load final return val into $s0

# printf("The value of 'factorial(%d)' is: %d\n",

la $t1, msgres1 # load msgres1 address into $t1

lw $a0, 0($t1) # load msgres1\_data value into $a0

li $v0, 4 # system call for print\_string

syscall # print value of msgres1\_data to screen

lw $a0, 0($sp) # load original value into $a0

li $v0, 1 # system call for print\_int

syscall # print original value to screen

la $t2, msgres2 #load msgres2 address into $t1

lw $a0, 0($t2) # load msgres\_data value into $a0

li $v0, 4 # system call for print\_string

syscall # print value of msgres2\_data to screen

move $a0, $s0 # move final return value from $s0 to $a0 for return

li $v0, 1 # system call for print\_int

syscall # print final return value to screen

addi $sp, $sp, 12 # move stack pointer back down where we started

# return 0;

li $v0, 10 # system call for exit

syscall # exit!

.text

factorial:

# base case - still in parent's stack segment

lw $t0, 0($sp) # load input from top of stack into register $t0

#if (x == 0)

beq $t0, 0, returnOne # if $t0 is equal to 0, branch to returnOne

addi $t0, $t0, -1 # subtract 1 from $t0 if not equal to 0

# recursive case - move to this call's stack segment

addi $sp, $sp, -12 # move stack pointer up 3 words

sw $t0, 0($sp) # store current working number into the top of the stack segment

sw $ra, 8($sp) # store counter at bottom of stack segment

jal factorial # recursive call

# if we get here, then we have the child return value in 4($sp)

lw $ra, 8($sp) # load this call's $ra again(we just got back from a jump)

lw $t1, 4($sp) # load child's return value into $t1

lw $t2, 12($sp) # load parent's start value into $t2

# return x \* factorial(x-1); (not the return statement, but the multiplication)

mul $t3, $t1, $t2 # multiply child's return value by parent's working value, store in $t3.

sw $t3, 16($sp) # take result(in $t3), store in parent's return value.

addi $sp, $sp, 12 # move stackpointer back down for the parent call

jr $ra # jump to parent call

.text

#return 1;

returnOne:

li $t0, 1 # load 1 into register $t0

sw $t0, 4($sp) # store 1 into the parent's return value register

jr $ra # jump to parent call

Recursive Fibonacci

# http://www.cs.usfca.edu/~peter/cs315/code.html

# Program to read a positive integer n, and compute the nth Fibonacci number:

#

# f\_0 = 0

# f\_1 = 1

# f\_n = f\_(n-1) + f\_(n-2), n >= 2

#

# This version uses recursion.

#

.text

.globl main

main:

subu $sp, $sp, 8 # Make additional stck sp.

sw $ra, 4($sp) # Save the return address

sw $s0, 0($sp) # Save $s0 = n

# Read n

li $v0, 5 # Code to print an int

syscall # Read n

move $s0, $v0

# Call Fibo function

move $a0, $s0

jal fibo

# Print the result

move $a0, $v0 # Put f\_n in $a0

li $v0, 1 # Code to print an int

syscall # Print the nth Fibonacci no.

# Restore the values from the stack, release stack sp.

sw $s0, 0($sp) # Retrieve $s0

lw $ra, 4($sp) # Retrieve the return address

addu $sp, $sp, 8 # Make additional stack space.

# Return in Spim

# jr $ra

# In Mars exit

li $v0, 10

syscall

#############################################################

# Fibo Function

# $a0 = n

#

fibo:

addi $sp, $sp, -12 # Put $ra, $s0, $s1 on stack

sw $ra, 8($sp)

sw $s0, 4($sp)

sw $s1, 0($sp)

move $s0, $a0 # Put n in $s0

bne $s0, $zero, not\_0 # Go to not\_0 if $s0 != 0

li $v0, 0 # n = 0, f\_n = 0

j done

not\_0: li $t0, 1

bne $s0, $t0, gt\_1 # Go to gt\_1 if $s0 != 1

li $v0, 1

j done

gt\_1: addi $a0, $s0, -1 # Assign $a0 = n-1

jal fibo # Compute f\_(n-1)

move $s1, $v0

addi $a0, $s0, -2 # Assign $a0 = n-2

jal fibo # Compute f\_(n-2)

add $v0, $s1, $v0 # Add f\_(n-1) + f\_(n-2)

done:

# Retrieve $ra, $s0, $s1 from stack and return

lw $ra, 8($sp)

lw $s0, 4($sp)

lw $s1, 0($sp)

addi $sp, $sp, 12

jr $ra