Jingpeng Wu

Iterative vs. recursive Fibonacci

//Instruction cycles count

Summary: Aside from the common code that cancels out, the recursive fib has to load and store values onto the stack which is at least 18 more CPU cycles per n when compared to iterative (2 load and 2 store). Thus, recursive is slower than iterative.

Iterative:

loop:

sub $s2, $s2, 1 //cancels out

blt $s2, 2, return // cancels out

add $t0, $s0, $s1 # compute next fib in t0 // cancels out

move $s0, $s1 # update window // cancels out

move $s1, $t0 // cancels out

move $a0, $s1 # print fib 2 // cancels out

li $v0, 1 // cancels out

syscall // cancels out

la $a0, space // cancels out

li $v0, 4 // cancels out

syscall // cancels out

j loop // cancels out

return:

li $s0, 1 # reset //Present in recursive as well but is

li $s1, 1 //done in main so cancels out

jr $ra //cancels out

Recursive:

fib\_r:

addi $sp, $sp, -8 # make room //

sw $a0, 4($sp) # store $a0 //4 cycles

sw $ra, 0($sp) # store $ra //4 cycles

addi $t0, $0, 3 # loop one less than n // cancels out

slt $t0, $a0, $t0 // cancels out

beq $t0, $0, else // cancels out

addi $sp, $sp, 8 # restore $sp // cancels out

jr $ra # return // cancels out

else: addi $a0, $a0, -1 # n = n – 1 // cancels out

jal fib\_r # recursive call // cancels out

lw $ra, 0($sp) # restore $ra //5 cycles

lw $a0, 4($sp) # restore $a0 //5 cycles

addi $sp, $sp, 8 # restore $sp //

add $t0, $s0, $s1 # compute next fib in t0 // cancels out

move $s0, $s1 # update window // cancels out

move $s1, $t0 // cancels out

move $a0, $s1 # print fib 2 // cancels out

li $v0, 1 // cancels out

syscall // cancels out

la $a0, space // cancels out

li $v0, 4 // cancels out

syscall // cancels out

jr $ra # return // cancels out