Lab 2 – Writing ARM Assembly

Nathan Paternoster

**Factorial**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {lr}

mov r3, r0 @ k is stored in r3

cmp r3, #0 @ check k to see if it is negative or zero

ble zero

mov r4, #1 @ r4 will store the product

b factorial

zero: @ branch here if k is negative

mov r0, #1 @ return 1 because 0! = 1

pop {lr}

mov pc, lr

factorial: @ this loop decrements k down to 1

mul r5,r4,r3 @ multiply the current product (r4) by the current iteration (r4)

sub r3,r3,#1 @ decrement k by 1

mov r4, r5 @ move the product of r4 and r3 back into r4 to collect the total product

cmp r3,#0 @ the ending condition - when the iterator reaches 0

ble end

b factorial

end:

mov r0,r4 @ put the product (r4) into the output register r0

pop {lr}

mov pc, lr

**Fibonacci**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {lr}

mov r3, r0 @ k is put into r3

cmp r3, #0 @ a check to see if k is zero or negative

ble zero

b fibonacci

zero: @ will come here if k is zero or negative

mov r0, #-1 @ return -1 as an error value

pop {lr}

mov pc, lr

fibonacci:

mov r4,#0 @ r4 and r5 will be the two registers representing F(n) and F(n+1)

mov r5,#1 @ they are initialized with the first two numbers of fib sequence: 0 and 1 respectively

add r6,r4,r5 @ the first summation of the series occurss and is put into r6

sub r3,r3,#1 @ if k was the first num in the sequence r3 would be 0 here

cmp r3,#1 @ therefore 1 is the first number of the sequence and the loop doesn't have to be entered

ble end @ so branch to end

b loop

loop: @ otherwise enter recursive loop

mov r4,r5 @ each register (r4 and r5) are moved one number forward in the sequence

mov r5,r6 @ the sum of the previous two (r6) becomes the next number in the sequence

add r6,r4,r5 @ r6 becomes the sum again

sub r3,r3,#1 @ k (r3) is the decremeter

cmp r3,#1

ble end @ loop ends when the decrementer reaches 0

b loop

end:

mov r0,r6 @ r6 (the latest position in the sequence) is output

pop {lr}

mov pc, lr

**Sum Array**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {lr}

cmp r1,#0 @ r0 = k. r1 = n. r2 = first position in array

blt wrong @ if n is negative it is invalid

cmp r0,#0 @ and if k is negative it is invalid

blt wrong

mov r3,#0 @ r3 will be the total summation

sub r0,r0,#1 @ r0 is initially decremented by 1 because our array includes a position 0

b for

wrong:

mov r0,#-1 @ -1 is output to signify an invalid input

pop {lr}

mov pc, lr

for:

mov r4,r0,lsl#2 @ r0 (k) will be the decrementer for this loop

add r5,r2,r4 @ r0 will be an address offset used to access each spot in the array

ldr r6,[r5] @ left shifting by 2 is equivalent to multiplying by 4. instruction addresses are multiples of 4

add r3,r3,r6 @ r6 gets the current position in the array. it is added to the partial sum r3

sub r0,r0,#1 @ the decrementer is decreased by 1

cmp r0,#0

blt exit @ loop ends when r0 is less than 0

bge for

exit:

mov r0,r3 @ the summation in r3 is output

pop {lr}

mov pc, lr

**Find Item**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {lr}

cmp r1,#0 @ r0 = k. r1 = n. r2 = first position in the array

blt wrong @ if n is negative it is invalid (wrong)

mov r6,r1 @ the amount of items in the array (n) is stored in r6

b for

wrong:

mov r0,#-1 @ -1 is returned when input is invalid

pop {lr}

mov pc, lr

for:

mov r5,r1,lsl#2 @ the offset for the array will be r1\*4 because instructions are multiples of 4

add r4,r2,r5 @ the offset is added to the first address of the array

ldr r3,[r4] @ that position in the array is accessed and put in r3

cmp r3,r0 @ if r3 = r0 (k, the number we are looking for) branch to exit

beq exit

sub r1,#1 @ otherwise subtract 1 from our decrementer r1

cmp r1,#0 @ if r1 is now negative the number was not found

blt wrong

b for

exit:

mov r0,r1 @ the position in the array of the answer (r1) is put to the output

pop {lr}

mov pc, lr

**Bubble Sort**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {lr}

cmp r1,#0 @compare if the number of nodes are larger than 0

ble wrong @ if number < or = 0, the return "-1"

sub r1,r1,#1

mov r10,r1

mov r11,r1

b for1

wrong:

mov r0,#-1

pop {lr}

mov pc, lr

for1:

cmp r1,#1

blt case1 @if there is only one node, return the "input"

b for2 @if there is more than 2 inputs, do the bubble function

case1:

ldr r0,[r2]

pop {lr}

mov pc,lr

for2:

mov r3,r1,lsl#2 @r3=index of node \*4

sub r4,r3,#4

ldr r5,[r2,r3] @r5=the value pointed by r2+r3

ldr r6,[r2,r4] @r6=the value of the next node

cmp r5,r6

blt case2 @if r5 < r6,do swap

b case3 @if r5 >= r6, keep the same

case2:

str r5,[r2,r4]

str r6,[r2,r3]

b case3

case3:

sub r1,r1,#1 @in the first round, we need to compare "number of node - 1" times

cmp r1,#0

bgt for2

b case4 @finish the first round, find the maximun value, then go to the second round

case4:

sub r10,r10,#1 @the initial value of r10 is equal to "the number of nodes-1", which is the total number of round

cmp r10,#0

bgt case5

b final @all the inputs are put in order

case5:

mov r1,r11

b for2

final:

pop {lr}

mov pc,lr

**Tree Height**

.globl \_EEN312\_STUDENTMAIN

\_EEN312\_STUDENTMAIN:

push {r1,lr}

mov r0,#0 @r0 is the height

cmp r1,#0 @r1 is the number of the node

beq end @if there is no node, end

bl add @otherwise...

add:

add r0,r0,#1 @height=height+1

mov r1,r1,lsr#1 @r1 = r1/2

cmp r1,#0 @compare r1 with 0

bgt add @if r1>0, then do the add function

b end @otherwise, output the height

end:

pop {r1,lr}

mov pc, lr