UNIVERSITY OF MIAMI, ELECTRICAL AND COMPUTER ENGINEERING

Project 2: Mastering Stack Discipline
Assigned: February 4th, 2014
Due: February 17th, 2014 11:59pm

EEN 312, Spring 2014
Professor Eric W. D. Rozier

Maximum Score: 75pts

1 Pre-Lab

For this assignment you will be implementing several functions while observing stack discipline, and the procedure calling conventions of ARM and Linux. Before the lab, study the following resource:
`http://www.performalumni.org/erozier2/EEN312/arm-call.pdf`
In particular, you will find page 14 useful, as it indicates the registers used to pass and return values when using ARM and Linux. A main object file has been created for the use of this lab, called `project2-main.o`, and an assembly skeleton called `studentmain.s`.

2 Lab Phases

This lab is divided into several phases,

- Recursive functions
  - `factorial(k, -, -)` - **(5pts)** - Your function should find the value of $k! = \Pi_{k=1}^{n} k$, recursively, and return the product.
  - `fibonacci(k, -, -)` - **(5pts)** - Your function should find the k^{th} value of the fibonnaci sequence, recursively, and return it. Recall the fibonacci sequnce begins with $F_0 = 0$ and $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$.

- Array functions
  - `sum_array(k, n, *array)` - **(5pts)** - Your function will be passed a pointer to the start of an array of n items whose first item is at the address pointed to by `*array`. Your function should sum the first k items of the array, and return that value.
  - `find_item(k, n, *array)` - **(10pts)** - Your function will be passed a pointer to the start of an array of n items whose first item is at the address pointed to by `*array`. Your function should find and return the index of the first item in the array with value k, or -1 if the item is not found.
  - `bubble_sort(-, n, *array)` - **(10pts)** - Your function will be passed a pointer to the start of an array of n items whose first item is at the address pointed to by `*array`. Your function should perform a bubble sort on the array (see NIST discussion here: `http://xlinux.nist.gov/dads/HTML/bubblesort.html`) and replace the items in the array with the correctly sorted version.
- Tree functions
  - `tree_height(-, n, *array)` - **(15pts)** - Your function will be passed a pointer to the start of an array which represents a tree at the address pointed to by `*array`. The tree is represented in the array stored in breadth first order such that for some node found in array position i, its children are found at the indicies $2i+1$ for the left child, and $2i+2$ for the right child. The parent of any child (if it exists) is found at index $\lfloor \frac{i-1}{2} \rfloor$. All nodes, if present, will be stored as positive, non-zero, integers. A zero indicates a node that is not present, i.e. node i has both children equal to zero, it is a leaf node. The variable n indicates the size of the array containing the tree. Your function should traverse the tree, find its height, and return that value. It is suggested that you implement your function recursively.
  - `traverse_tree(-, n, *array)` - **(15pts)** Your function will be passed a pointer to the start of an array which represents a tree at the address pointed to by `*array`. The tree is represented in the array stored in breadth first order such that for some node found in array position i, its children are found at the indicies $2i+1$ for the left child, and $2i+2$ for the right child. The parent of any child (if it exists) is found at index $\lfloor \frac{i-1}{2} \rfloor$. All nodes, if present, will be stored as positive, non-zero, integers. A zero indicates a node that is not present, i.e. node i has both children equal to zero, it is a leaf node. The variable n indicates the size of the array containing the tree. Your function should traverse the tree in depth first order, using a preorder traversal, and replace the original array with a depth first ordering of the nodes, followed by filling zeros for the rest of the array.

An additional **10 points** will awarded for proper use comments to document your code.
For each phase, you will make a copy of `studentmain.s` and rename it for the lab phase you are completing. Once completed, you will assembly your solution with the existing code as follows:

```
as -o <solution>.o <solution>.s
gcc -O1 -o <solution> project2-main.o <solution>.o
```

The same main program, `project2-main.o` is used for each of the lab's phases. It reads a series of integers from the command line, the first integer corresponds to the first possible argument in the listed phases, which we will call `k`. The next argument is the size of an optional array, which we will call `n`. After the argument

`n`, there will be a series of n integer arguments, if no array will be passed, n will be zero. When the program terminates, it prints the returned value for your function, and the elements of the array, in order.
This main program does the work of reading in arguments, and will then pass those arguments to your assembly function, which you must look for according to the procedure calling conventions of ARM and Linux. You must then allocate space on the stack for local variables, and any registers that must be saved to implement the procedures.

3 Obtaining the Code Skeleton

The code you will need to begin this lab can be found at the course website:

`http://performalumni.org/erozier2/een312.html`

You will find several important files under the heading for Lab 2:

- `project2-main.cc` - The C-code for the main function for the lab. This is included for your reference only, and should not be used to construct your solution.
- `project2-main.s` - The assembled code for the main function for the lab. This is included for your reference only, and should not be used to construct your solution.
- `project2-main.o` - The object file you will use to build the executable solutions for the lab.
- `studentmain.s` - An assembly code skeleton to use in your solution.

Use the skeleton in `studentmain.s` as a starting point for your solution. The main code in `project2-main.cc` will set up the registers for the call, and branch to the label `_EEN312_STUDENTMAIN`. You should bear in mind that for ARM and Linux, it is the duty of the callee to save registers when writing your solution.

4 Logistics

This project should be completed in your groups. It is recommended you run your program in `gdb` to avoid problems if you inadvertantly violate procedure calling conventions. It is very easy to accidentally create an infinite loop, or fail to properly return. Running your program in `gdb` will help you to debug your program, monitor the stack, and safeguard against errors. To run your program in gdb with command line options, remember to do the following:

```
gdb <my program>
gdb> r <my arguments>
```

Remember to set a breakpoint! A breakpoint on your function name, `_EEN312_STUDENTMAIN` might be particularly helpful.

5 Handin

You will submit your solutions in commented, assembly code to your TAs by e-mail who will assemble and link them with `project2-main.o`, testing them against several sequences of input for correctness.

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
```

<u>Bubble Sort</u>

```
.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
```