Kaiqi Chee 12/09/2020

I pledge my honor that I have abided by the Stevens Honor System.

CS 383: Programming Project 2

Approach:

I started off in my main function where I took in the input from the .data segment and set d1 to be the input double value, x0 to be the input integer value, d0 to be 0 and to hold the result, and d2 to 1. Then I branched to my start function.

```
11
12
13 main:
14     ldr x2, =doubleVal    //set double input value to d1
15     ldr d1, [x2]
16
17     ldr x7, =integerVal   //set integer input value to x0
18     ldr x0, [x7]
19
20     ldr x2, =c
21     ldr d0, [x2]
22     ldr x2, =a
23     ldr d2, [x2]
24
25     bl start
26
```

```
104
105 .data
106
107 //input
108 doubleVal:
109   .double 2
110 integerVal:
111   .quad 2
112
113 //constants
114 c:
115   .double 0.0
116 a:
117   .double 1.0
118
119 //print string
120 string:
121   .ascii "The approximation is %f\n "
122
123 .end
124
125
```

In the start function I began by making space on the stack. After this I initialized x1 to 0 to act as the iterator in my loop. Then I branched to my recursive taylor function, this is the function that calculates the *i*th term of the series. In the taylor function, I set d3 to 1, d3 will hold the result of the numerator. Similarly, I set d4 to 1, d4 will hold the result of the denominator. Then I branched to my power and factorial functions to find the numerator and denominator values for the given input double and integer.

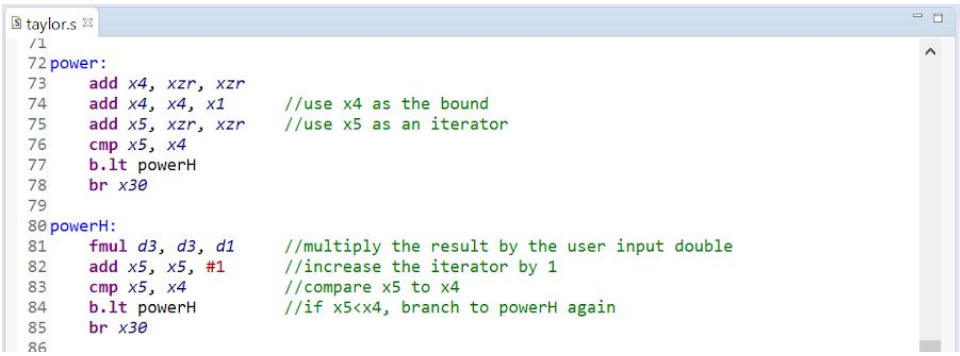
taylor.s

```
27
28 start:
29     sub SP, SP, #48
30     str x0, [SP, #0]
31     str x1, [SP, #8]
32     str x2, [SP, #16]
33     str x3, [SP, #32]
34     str x4, [SP, #40]
35     str x5, [SP, #48]
36
37     add x1, xzr, xzr     //set x1=0, x1 will be the iterator
38
39     cmp x1, x0           //input integer to i
40     b.lt taylor          //if x1 is less than than x0, go to taylor function
41
42     ldr d0, =string
43     add SP, SP, #48      //restore the stack
44     bl printf            //print the result
45     br x30
46
```

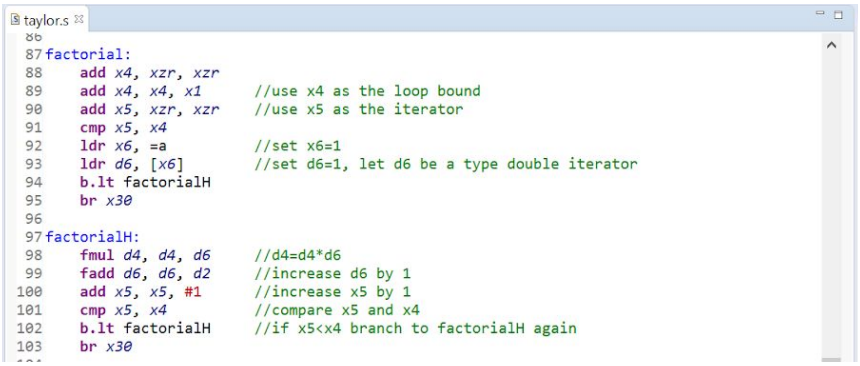
taylor.s

```
47 taylor:
48     ldr d3, [x2]         //set d3=1, d3 represents the numerator
49     ldr d4, [x2]         //set d4=1, d4 represents the denominator
50
51     bl power             //solve for the numerator
52     bl factorial         //solve for the denominator
53
54     fdiv d5, d3, d4      //d5=d3/d4, term=power/factorial
55     fadd d0, d0, d5      //add d5 to the result in d0
56
57     add x1, x1, #1       //increase the iterator x1 by 1
58     cmp x1, x0           //compare x1 to x0
59     b.lt taylor          //if x1<x0 recurse back to taylor function
60
61     add SP, SP, #48      //restore the stack
62     ldr x2, =c           //let x2=0
63     ldr d7, [x2]         //set d7=0
64     fadd d7, d7, d0      //add the result to d7
65     mov x0, #1           //set x=1
66     scvtf d0, x0         //convert x0 from an into into a double, store in d0
67     fmul d0, d0, d7      //d0=d0*d7
68     ldr x0, =string
69     bl printf            //print the result
70     br x30               //branch to link register x30
71
```

In the power function I used x4 as the bound for my loop which I set equal to x1, and x5 as the iterator for my loop. For each loop I multiplied the result, d3, with the user input double held in d1. After this I increased x5 by 1 and checked if x5 was less than x4, if it was I recursively branched to the powerH function again, if not I branched back into the taylor function.



Similarly, in the factorial function I used x4 as the bound for my loop and set it equal to x1, and x5 as the iterator for my loop. Then I declared d6 as an iterator with the type double so that I could iteratively multiply my result, d4, with d6 until the loop ended and x5 was no longer less than x4. This led to the br x30 instruction, so instead of branching to the factorialH function, it would return to the taylor function.



Once the power and factorial functions had completed, the appropriate results were stored in d3 and d4 respectively. After this I used d5 to store the overall result of the fraction by dividing d3 by d4. Once I had the completed term I added it to d0 which held the result of the entire series. After completing this step I increased x1 by 1 and used cmp to compare x1 to x0. If

x1 was less than x0 I would branch back to the taylor function and solve for the next term, if not I continued to restore the stack pointers and print out the result.

```
60
61    add SP, SP, #48      //restore the stack
62    ldr x2, =c           //let x2=0
63    ldr d7, [x2]         //set d7=0
64    fadd d7, d7, d0      //add the result to d7
65    mov x0, #1           //set x=1
66    scvtf d0, x0         //convert x0 from an into into a double, store in d0
67    fmul d0, d0, d7      //d0=d0*d7
68    ldr x0, =string
69    bl printf            //print the result
70    br x30               //branch to link register x30
```

```
EL3:0x0000000080012C4 8B010084 ADD x4,x4,x
App Console  Target Console  Error Log
Linked: taylor configuration
Info: Foundation_AEMv8A: CADI Debug Server starte

CADI server is reported on port 7000
The approximation is 3.000000
```