Systems and Networks 2019 Assessed Exercise

0801323

22/11/2019

Short Status Report

I completed the assignment and tested it fairly rigorously, but there is always room for more testing.

As far as I have been able to test, the program works (it achieves the same output as the Python version in all of my test cases except when I use decimals). I have included some of the sample tests I ran in my code below.

There are no bugs that I encountered.

However, I initially made an error as my JUMPT should have been a JUMPF for the modulus calculation. In the given test data, this provides the same output (4) either way – there are 4 odd and 4 even numbers!

However, I was able to catch it with additional testing.

One major limitation of the program is that it cannot run with decimals (only integers). As such, it cannot work with all numbers.

It seems to work with 0 well enough, and doesn't seem to break in any of the tests I've thrown at it.

It works with empty arrays and arrays of all sizes I tested it with. I think that there will be a limit of around 2^16 (65536) array elements, due to Sigma16 being a 16 bit system.

My code is shown on the next 2 pages.

Code

```
;_____High Level Language (Python) Component_____
;Python Code
;
;n = 12                      #n holds the number of list elements - when testing, make sure to change this to fit
;x = [0] * n                 #initialise empty array of size n
;test_data = [3, -6, 27, 101, 50, 0, -20, -21, 19, 6, 4, -10] #Defined  test array
;alt_test_1 = [-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1] #extra test 1 - more negatives, same number of elements (works in both)
;alt_test_2 = [3,4,5,6]                  #extra test 2 - less elements (works in both)
;alt_test_3 = [-1]                       #extra test 3 - single element, negative (works in both)
;alt_test_4 = [3.4, 5.5, 1, 0]           #extra test 4 - less elements, decimals (works in python, not in sigma16)
;for i in range(n):          #populate x with our test data using loop (essentially equivalent to data statements)
;    x[i] = test_data[i]
;
;#initialise variables to 0
;negcount = 0
;possum = 0
;oddcount = 0
;
;#for loop to populate variables
;for i in x:
;    if i < 0:
;        negcount = negcount + 1        #count if negative
;    else:
;        possum = possum + i            #add if >= 0
;        if i % 2 != 0:
;            oddcount = oddcount + 1    #increment by 1 if odd and positive
;
; #print outputs below
;print("Sum of positive array elements: "+str(possum))
;print("Count of unique negative array elements: "+str(negcount))
;print("Count of odd array elements: "+str(oddcount))

;_____Sigma 16 Section_____
;Register usage
;R1 = constant 1
;R2 = constant 2
;R3 = n
;R4 = i
;R5 = possum (sum of positive elements in the array)
;R6 = negcount (count of negative elements in the array)
;R7 = oddcount (count of odd elements in the array)
;R8 = x[i]
```

```
;R15 = remainder

 LEA R1,1[R0]         ;R1 = constant 1
 LEA R2,2[R0]         ;R2 = constant 2
 LOAD R3,n[R0]         ;R3 = n
 LEA R4,0[R0]         ;R4 = i, initially 0
 LEA R5,0[R0]         ;R5 = possum, initially 0
 LEA R6,0[R0]         ;R6 = negcount, initially 0
 LEA R7,0[R0]         ;R7 = oddcount, initially 0

LOOP CMPLT R10,R4,R3       ;if i < n
    JUMPF R10,DONE[R0]     ;If not (i < n), exit loop
    LOAD R8,x[R4]         ;Load x[i] into R8
    ADD R4,R4,R1          ;increment i by 1
    CMPLT R10,R8,R0       ;if R8 < 0
    JUMPT R10,LESSTHAN[R0] ;if R8 < 0, go to the start of LESSTHAN
    ADD R5,R5,R8          ;possum = possum + x[i]
    DIV R12,R8,R2         ;Division to get the remainder <which goes into R15>. Value in R12
unimportant.
    CMPEQ R10,R15,R0      ;if remainder <R15> is equal to 0
    JUMPT R10,LOOP[R0]     ;if no remainder, go to beginning of loop
    ADD R7,R7,R1          ;increment counter of odds
    JUMP LOOP[R0]         ;go to top of loop

LESSTHAN ADD R6,R6,R1
       JUMP LOOP[R0]

DONE STORE R5,possum[R0]      ;Save R5 into possum
    STORE R6,negcount[R0]     ;Save R6 into negcount
    STORE R7,oddcount[R0]     ;Save R7 into oddcount
    TRAP R0,R0,R0

n DATA 12                  ;initialises n
x DATA 3                  ;initialises x (this is x[0]
  DATA -6                 ;x[1]
  DATA 27                  ;x[2]
  DATA 101                  ;x[3]
  DATA 50                 ;x[4]
  DATA 0                  ;x[5]
  DATA -20                  ;x[6]
  DATA -21                  ;x[7]
  DATA 19                 ;x[8]
  DATA 6                  ;x[9]
  DATA 4                  ;x[10]
  DATA -10                  ;x[11]
possum DATA 0              ;positive sum result, initial value unimportant
negcount DATA 0             ;negative count result, initial value unimportant
oddcount DATA 0             ;odd count result, initial value unimportant
```