Systems & Networks 2019 Assessed Exercise

Wesley Scott, 2460681S

06/11/2019

The status of my coursework submission is such that the program runs as specified by the Assessed Exercise task description. The evidence of this is its correct output values associated with the test data prescribed in the spec; the location of possum store in memory holds value $00d2, the location of oddcount store in memory holds value 4, the location of negcount store in memory holds value 4.

These same tests were carried out with other values in x and n respectively (array values and array length). The Sigma16 code implements the high-level language code (written in Java) in its entirety with functionality being essentially identical.

A potential limitation might be considered depending on the desired design of memory allocation on finishing code execution. The interpretation of spec shaped this solution as such that the registers holding the values of possum, negcount and oddcount at the end of runtime are stored into memory. Included are initialising statements at the beginning of the code to reset all used registers values to 0, for the ease of repeated runs. I was unsure if this is required but left included in my solution. However, I feel that this code represents a complete solution to the assessment specification. If this were to be a consumer product, further input checks could also be added to ensure values are the correct type.

Overleaf is my code, some comments have been shortened to accommodate page width.

; Wesley Scott, 2460681S

; Systems and Networks 2019 Assessed Exercise

; File: SysNetworksAssExWS.asm.txt

; Assessed Coursework Exercise Source Code

; HLL Source Code (written in Java, compiles and runs correctly)

; public class AssessedEXSandN{

; public static void main(String[] args){

; int n = 12;

; int[] x = {3, -6, 27, 101, 50, 0, -20, -21, 19, 6, 4, -10};

; int possum = 0;

; int negcount = 0;

; int oddcount = 0;

;

; for(int i=0;i<n;i++){

; if(x[i]<0){

; negcount++;

; } else {

; possum += x[i];

; if(x[i] % 2 != 0){

; oddcount++;

; }

; }

; }

; }

; }

; Sigma 16 Code (compiles and runs correctly)

; Register assignment:

; R1 = i, set to 0 at runtime (for loop counter)

; R2 = constant 2 (for remainder calculation)

; R3 = n (array length)

; R4 = holds value of x[i] in each loop cycle

; R5 = 1 (constant for incrementing loop and variabke counters)

; R6 = holds data to be stored in possum (stores total sum of all elements in the array (x))

; R7 = holds data to be stored in negcount (stores total number of negative values)

; R8 = holds data to be stored in oddcount (store total number of positive odd numbers)

; R9 = boolean value to provide jump to positive or negative processing

; R10 = holds result of division, required for odd number calculation

; R15 = holds remainder, used in odd number calculation

; Initialise registers

START LEA R1,0[R0] ; instantiate i with value 0

LEA R2,2[R0] ; used in remainder calculation (constant 2)

LOAD R3,n[R0] ; n array size, sets loop condition (constant 12) LEA R5,1[R0] ; 1 constant for loop incrementing

LEA R6,0[R0] ; reset register storing sum (possum) value

LEA R7,0[R0] ; reset register storing negative number count

LEA R8,0[R0] ; reset register storing positive odd number count

LEA R9,0[R0] ; reset register storing boolean for positive or negative

LEA R15,0[R0] ; reset register storing remainder (if oddnumber)

; Main for loop, cycles through x[n-1]

FORLOOP CMPLT R5,R1,R3 ; true if i<n (index lower than value of array length)

JUMPF R5,OUT[R0] ; exit for loop when condition false

LOAD R4,x[R1] ; loads i-th value of array x into R4 from memory

CMPLT R9,R4,R0 ; compare if x[i] is < 0

JUMPT R9,NEGATIVE[R0] ; jump if x[i] is a negative value

JUMPF R9,POSITIVE[R0] ; jump if x[i] is a positive value

; Used to increment the for loop after processing of x[i]

INCREMENT ADD R1,R1,R5 ; loop increment, i = i + 1

JUMP FORLOOP[R0] ; loop again (till R5 is no longer true)

; Processing negative numbers in x

NEGATIVE ADD R7,R7,R5 ; increment the negative number count

JUMP INCREMENT[R0] ; after processing increment loop counter

; Processing positive numbers in x

POSITIVE ADD R6,R6,R4 ; adding x[i] each time around the loop

DIV R10,R4,R2 ; divide x[i] by 2

JUMPT R15,ODDCOUNT[R0] ; jump to ODDCOUNT if there is remainder

JUMPF R15,INCREMENT[R0] ; jump to INCREMENT if there is no remainder

; Processing positive odd numbers in x

ODDCOUNT ADD R8,R8,R5 ; increment the positive odd number count

JUMP INCREMENT[R0] ; after processing increment loop counter

; Program exit statement and store of register holdings

OUT STORE R6,possum[R0] ; stores sum of x

STORE R7,negcount[R0] ; stores number of negative numbers in x

STORE R8,oddcount[R0] ; stores number of positive odd numbers in x

TRAP R0,R0,R0 ; finish code execution

; Static variables and stores

possum DATA 0 ; possum variable holds sum value after runtime

negcount DATA 0 ; holds number of negative numbers encountered

oddcount DATA 0 ; holds number of positive odd numbers encountered

n DATA 12 ; n used to set the size of array (x) and loop condition

x DATA 3 ; x[0]

DATA -6 ; x[1]

DATA 27 ; x[...]

DATA 101

DATA 50

DATA 0

DATA -20

DATA -21

DATA 19

DATA 6

DATA 4

DATA -10