**Homework 8: Subroutines**

**Single (input/output) parameter:** In this part we use the Accumulator (AC) to pass the single parameter to the subroutine as an input parameter and from the subroutine as return parameter.

**Note**: in ALL implementations of a subroutine:

* You are **not** allowed to use the names of variables defined for main() in the subroutine and **vice versa**!
* Write the high-level algorithm first.
* Use the program skeleton where it is given and stick to the given names.
* Comment your programs properly!

**Warm up:** A subroutine with the signature: ***signed short int Absolute (signed short int a)***;   
This subroutine takes one parameter as input – a **signed short integer** (16-bit) and returns its absolute value:

1. Write the high-level algorithm of the **main**() part that calls the subroutine and the subroutine in a pseudo high-level language.
2. Implement the algorithm in Mano CPU Assembly language and test them with different values. .

**main**() {

Main, LDA X

BSA ***Absolute***

STA Y

HLT   
**…**

**short** **signed** **int** x = ?; **short** **signed** **int** y = 0;

y = ***Absolute***(x);

}

**short** **signed** **int** ***Absolute***(**short** **signed** **int** a) {

***Absolute***, HEX 0

**...**

BUN ***Absolute*** **I**

???

}

**Note**: in the following questions, all integers are 16 bit. For the sake of brevity, instead of writing:

* **signed short int** → **signed** **int**
* **unsigned short int** → **unsigned int**

1. Write a subroutine that given the address of an array, it initialises it to 0 and returns the length of the array. The subroutine assumes that the input array has a special negative number as its Terminator = -999;

**The array may be empty** – so that it contains only the terminating special value.

Write the high-level algorithm, do whatever transformations are necessary and then code and test it in Assembler language.

main ( ) {

*signed int* Array1[] = {11,2,-33, …, -999};

*signed int* Array2[] = {11, 1, -2,13,1, 15, 2…, -999};   
 *signed int* **Ar1\_count**, **Ar2\_count**;

**Ar1\_count** = **InitArray**(**@**Array1); // initialise Array1

**Ar2\_count** = **InitArray**(**@**Array2); // initialise Array2

**}**

**@**Array means the address of the array.

The subroutine does the following:

1. Initialises the array elements to 0 and
2. Counts and returns the number of the elements in the array.   
   It then stores the result in an appropriate local variable, for example:  **Ar1\_count** = **InitArray**(**@**Array1);

***Unsigned int* InitArray**( int \*ArrPtr ) {

**?**

**Input Parameter**: Address of the array

}

The parameters:

* **Ar1\_Start** and **Ar2\_Start** are the addresses of the start of the array and are passed in the AC (**NOT as a Global variable**!).

Suggested data skeleton in the next page!

Suggested skeleton

// main data

Ar1\_Start, HEX 100 // pointer to start of Array1

**Ar1\_count, DEC 0 // Array1 size - calculated by subroutine Init\_Array**

Ar2\_Start, HEX 200 // pointer to start of Array2

**Ar2\_count, DEC 0 // Array2 size - calculated by subroutine Init\_Array**

//

ORG 100

Array1, DEC 3 //

DEC 6

DEC 9

DEC -999 // Array Terminating value

ORG 200

Array2, DEC 11 //

DEC 12

DEC 13

DEC 77

DEC 7

DEC -999 // Array Terminating value

// Subroutine InitArray data

ArrPtr, HEX 0 // temporary pointer to Array

**Count, DEC 0 // where to store the return parameter of the subroutine – the size of array**

1. **OddEven**: Write a subroutine **oddEven** that takes a number and determines whether it is odd or even, returning a 1 in the End-carry flag if is odd and 0 if it is even. The subroutine does not disturb the contents of the Accumulator on return (in other words, the output of the subroutine in AC is the same as the input).

Note: in order to ensure that the subroutine does not disturb the working-space (i.e. the Accumulator in this case), the subroutine has to store the value of the AC in a local (temporary) variable and retrieve it before returning from the subroutine.

Write the high-level algorithm before deciding how you are going to code your algorithm in assembly language. Use any transformations you think are necessary…

1. **\*Find the maximum value**: Implement the subroutine **findArrayMax()** which finds the maximum value of a given array:

* Its size is not known, but is terminated with a special value: -999;
* The array cannot be empty!
* The range of values in the array are**: -100 ≤ value ≤ +100**

**signed int findArrayMax(signed int Array[])**;

Maximum value

Address of array

**Write the high-level algorithm before you code the solution!**

**Do the appropriate transformations.**

**Code in assembly with comments.**

**Test the program properly. Make sure it will work if used on 2 different arrays, one after the other.**

The **input** parameter (passed using the AC) is the address of the start of the Array.

The **output** (return) parameter (passed using the AC) is the maximum value of the array.

**// Data local to main()**

**Max**, DEC 0

//

**A\_start**, HEX 100 // pointer to Start of Array

ORG 100

Array, DEC -1

DEC 2

DEC 3

DEC -44

DEC 22

DEC 5

DEC -999

Terminator, DEC -999

**// Data local to subroutine**

temp\_max, DEC 0 // temporary max value

arrayPtr, HEX 0 // temporary array pointer

minus\_1, DEC -1

1. **\*\*\* Multiple (input/output) parameters:**  in this part we pass a pointer to a STRUCT to the subroutine as input parameter, and if more than a single return parameter is needed we also pass it in a STRUCT.

|  |
| --- |
|  |

You are given the following problem:

**main** ( ) {

**signed length** **A\_size** = 6; // The length of the arrays is given

**int** result;

**signed int** **Array**[] = {1, 2, 3, 4, 5, 6};

Address **A\_Start = @Array;**

**result = SumArray**(**A\_Size***,* **A\_Start**);

}

// the subroutine **SumArray** calculates & returns the sum of the elements of the array:

**short** **int** **SumArray** (***signed int*** *L\_Size****, int \**** *L\_TmpStructPtr*) **{**

**unsigned int** *Sum* = 0;

Input Parameters: size of the array & the addresses of the array – passed in struct

…

***FOR( ? )   
DO***

**?**

Output Parameter: sum of the elements of the array. Passed back through the AC

***OD;***

**return** *Sum*;

**}**

The **input parameters** are passed as a pointer to a struct of 2 fields:

* **A\_size** is the length of the array.
* ***A\_Start = @***Array is the addresses of the array.

The **return parameter** is passed through the AC.

Implement the algorithm in Mano's CPU Assembly language AFTER writing the high-level algorithm. Do not forget to comment the code with high-level comments.

**You can use the template for solving exercise 2 in the following page.**

The initialisation of the struct Par\_struct has to be done **dynamically**!  
You can use the following template for solving exercise 4:

// **main**

Main**, LDA** ***ParPTR*** //

// use the pointer to the Struct

// prepare parameter Array\_size

Note – the different style of commenting is for your understanding!

//Par\_struct.size = A\_size;

// prepare parameter address of Array

//Par\_struct.pointer = Ar\_Start;

// AC = @Par\_struct;

**BSA SumArray** //

**STA** Result // **result = SumArray**( ***size, Array[ ]*** );

**HLT**

// **main data**

Result, DEC 0 //

**A\_Start,** HEX ***100*** // pointer to start of Array

**A\_size**, DEC 6 // size of the array

//

ORG ***100***

**Array**, DEC 1 //

DEC 2

DEC 3

DEC 4

DEC 5

DEC 6

TempPtr, HEX 0 // temporary pointer

// struct used to pass 2 parameters to subroutine **SumArray**

***ParPTR***, HEX ***200*** // pointer to Par\_struct

ORG ***200*** // parameter struct

// Struct containing:

Par\_struct, HEX 0 // **size** of arrays

HEX 0 // **pointer** to array

// **end of main data**

ORG 300

**// Subroutine SumArray**

**SumArray**, HEX 0 // signed int **SumArrays*(***signed ***int Size,*** signed ***int Array[]***) **{**

**STA** *L\_TmpStructPtr* //

End\_Loop, **LDA** L\_Sum // return Sum;

**BUN** **SumArray** **I** // **}**

**// local data of Subroutine SumArray**

*L\_Size*, DEC 0 // size of arrays

//

*L\_TmpStructPtr*, HEX 0 // pointer to array

L\_Sum, DEC 0 //