**Homework 9: Subroutines – passing parameters through the Stack**

**Warm up Part I:**

Implement the **Stack** in Mano CPU assembly language:

* Draw the Stack configuration in memory with the Stack pointer.
* ~~Write the high-level algorithm of the Stack operations~~ **~~Push~~** ~~and~~ **~~Pop~~**~~.~~
* Implement the Stack and its two operations as 2 subroutines:
  1. Write the necessary data space allocation and initialisation.
  2. Write the instructions for the Push and Pop subroutines.
  3. Think carefully how the Stack operations can be tested.
  4. Test your implementation before you use it in exercise 2-4!

// Stack data

Minus1, DEC -1

**SPtr**, HEX 400

ORG 400

**Stack**, DEC 0

DEC 0

DEC 0

**…**

DEC 0

DEC 0 // allocate enough space in the Stack for the problems you are solving

**Push\_AC**(AC) {

\*SPtr = AC;

SPtr++;

}

AC = **Pop\_AC**() {

SPtr--;

AC = \*SPtr;

}

**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**

**Warm up Part II:**

A subroutine with the signature: ***signed short int Absolute (signed short int x)*** takes one parameter as input – a **signed short integer** (16-bit) using 2's complement representation and returns its absolute value. **Both input and output parameters are passed on the Stack implemented in the warm up exercise!**

1. Use the high-level algorithm from HW8 and implement in Assembly language using the Stack to pass the parameters.
2. Use the Stack (implemented in the warm up exercise) to pass two input parameters to the subroutine from **main**() and accept two output parameters from the subroutine.

Returning more than 1 parameter: Although not found in most high-level languages (ML has it, for example), this is possible in assembly language

**main**() {

**int** X = 6; **int** Y = -1;

**int** sum, difference;

(sum, difference) = AddSubtract(X, Y);

}

(**int**, **int**) **AddSubtract**(**int** par1, **int** par2) {

**return** (par1+par2, par1**-**par2);

}

1. Implement the subroutine **findArrayMaxMin()** which finds the maximum value and the minimum value of a given array whose size is known. The values in the array are: -100 ≤ value ≤ +100. It is given that the array cannot be empty.

**(int min, int max) = findArrayMaxMin(int Array[], int Size)**;

Address of array

The subroutine uses the **Stack** to pass 2 *input parameters* and uses the Stack to return 2 *output parameters* from the subroutine back to main().

Call the subroutine twice with different arrays – to check that your code really works.

1. Write the high-level algorithm before you code the solution!

No parameter is passed through the Accumulator!

The **input** parameters (passed using the Stack) are:

* The length of the Array = **A\_size**.
* **Array[]** is the address of the start of the Array.

The **output** (return) parameters (passed using the Stack) are:

* The maximum value of the array.
* The minimum value of the array.

1. Write the transformations.
2. Code your solution in assembly language.
3. Check: What will happen with your code if the arrays can be of size = 0? Does your code deal with such a case?
4. Check: What will happen with your code if the arrays can be of size = -1? Does your code deal with such a case?

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

Amax, DEC 0 // the result = maximum value of the array Aarray

Amin, DEC 0 // the result = minimum value of the array Aarray

A\_size, DEC 5 // Aarray length

//

A\_start, HEX 50 // pointer to Start of Aarray

**Array A\_start**

ORG 50

Aarray, DEC 11

DEC 2

DEC 0

DEC -3

DEC 4

Bmax, DEC 0 // the result = maximum value of the array Barray

Bmin, DEC 0 // the result = minimum value of the array Barray

B\_size, DEC 4 // Barray length

//

B\_start, HEX 100 // pointer to Start of Barray

**Array B\_start**

ORG 100

Barray, DEC 99

DEC -2

DEC 3

DEC 33

**// Data local to subroutine**

temp\_max, DEC 0 // temporary Max

temp\_min, DEC 0 // temporary Min

Count, DEC 0 // temporary iteration count

arrayPtr, HEX 0 // temporary array pointer

minus\_1, DEC -1

1. Given:

* An Array of integers (**int** Array[])
* Terminated by a special terminating value of +9999;
* The array is never empty (i.e. ASize > 0).
* The values in the array are: 0 ≤ value ≤ 1000.

The task is to write a main() that calls a subroutine:

**(int found, int location, int size) =   
 findElement\_inArray (int Array[], int Terminator, int Value**);

* **Gets** 3 input parameters (passed using the Stack):

1. The address of the Array.
2. The terminating value of the Array.
3. The number to search for in the Array.

* **Finds** whether a given integer is in the array or not.
* **Returns** 3 values (passed using the Stack):

1. A value of FALSE in case of failure and TRUE if the required number was found.
2. The location of the found number in the array (0 ≤ location < size of Array).
3. The size of the array.

Note: 3 input parameters and 3 return parameters!

You are asked to:

1. Write the High-level algorithm with main() and a subroutine.
2. Write any necessary transformations.
3. Write the Mano CPU assembly language program.

Use the following data structures:

Number, DEC ? // the number searched for

Found, DEC -1 // **Success**: Found = Number, **Failure**: Found = -1

Terminator, DEC 9999

Aptr, HEX 200

//

ORG 200

Array, DEC 1

DEC 2

DEC 77

**…**

DEC 9999