Practical 4
Calculator (Part 1)

Prerequisite: students are required to have read all of the lecture notes.

We want to design a calculator that performs the four basic operations (addition, subtraction, multiplication and division).

Example:
```
Enter an expression:
2+50*4
Result:
208
```

The order of operations will not be taken into account and numbers will be only integers. A number in an expression will be smaller than 32,767 and the result will be a 16-bit signed integer (if the expected result is larger, the result will be truncated).

A single error message (« Error ») will be displayed if the user's expression is erroneous. An expression is erroneous if:

- One of the numbers is greater than 32,767.
- The expression contains a character that is neither a digit nor an operator.
- A division by zero occurs.
- Two operators are side by side.

The user will be free to enter spaces at any time in order to improve the readability of the expression. We assume that he will never key zeros to the left of a number.

This practical is made up of several steps. For each of them, you will write a subroutine. A single source file called "`Calculator.asm`" will contain all of your subroutines and your program will be divided into four distinct parts:

- Vector Initialization
- Main Program
- Subroutines
- Data

Each subroutine will be added to the source file as you go along the different steps. **The main program should be adapted to test the subroutine in progress**. Except for the output registers, none of the data or address registers must be modified when the subroutine returns.

Step 1

Write the **RemoveSpace** subroutine that removes all spaces in a string of characters.

Input : **A0.L** points to a string whose spaces have to be removed.

Output : The string is modified in memory (none of the registers is modified).

Example:											
	Before:	' '	'5'	' '	'+'	' '	' '	'1'	'2'	' '	**0**
	After:	'5'	'+'	'1'	'2'	**0**	' '	'1'	'2'	' '	0

The two strings above are located in the exact same memory space. The first one is the string just before executing the subroutine. The second one is the string just after executing the subroutine (so the first string will be lost). The null character (0) denotes the end of a string. Therefore, the characters that follow the zero in the second string do not belong to the string.

To sum up:

- Before the execution, the string is: `" 5 +  12 "`
- After the execution, the string becomes: `"5+12"`

Tips:

- Two registers should point to the same string.
  - ➔ A0 as a source pointer.
  - ➔ A1 as a destination pointer.
- At the beginning, the two registers should point to the same memory location (the first character of the string).
- Each character should be tested in a loop.
- If a character is null, exit the loop.
- If a character is not a space, copy it from the source to the destination. That is to say, copy the contents of **A0** into the contents of **A1** and increment the two registers.
- If a character is a space, do not copy it (increment only **A0**).
- Be careful not to forget to copy the null character.

Find where the string is located by using the **[Mémoire]** tab and check that all the spaces have been removed after executing your subroutine.

Step 2

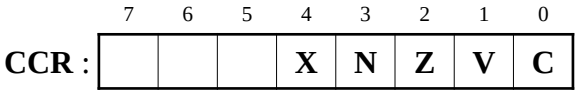
Write the **`IsCharError`** subroutine that determines if a string that is not empty contains only digits.

Input : **A0.L** points to a string that is not empty (i.e. that contains at least one character different from the null character).

Output : **Z** returns true (1) if the string contains at least one character that is not a digit.
Z returns false (0) if the string contains only digits.

Tips:

- If at least one character is lower than the '0' character, you should return true (**Z** = 1).
- If at least one character is higher than the '9' character, you should return true (**Z** = 1).
- To use the **Z** flag as an output value, you can directly modify the **CCR** (Condition Code Register), which is an 8-bit register containing the **X**, **N**, **Z**, **V** and **C** flags.



- To set the **Z** flag to 0 without affecting the other flags, just perform a logical AND between the Z flag and 0, and between the other flags and 1. To set the **Z** flag to 1, perform a logical OR. Here are the two instructions you should use in your source code:

```
    andi.b  #%11111011,ccr  ; Set the Z flag to 0 (false).
    ori.b   #%00000100,ccr  ; Set the Z flag to 1 (true).
```

- Write two distinct outputs. One called `\true` (that returns **Z** = `1`) and one called `\false` (that returns **Z** = `0`).

- The name of the subroutine can be seen as a question: **`IsCharError`** = Is there a character error?
  - True = Yes, there is.
  - False = No, there is not.

- If the name of the subroutine had been **`IsCharOK`**, the output value should have been inverted:
  - True = OK, there is no error.
  - False = Not OK, there is an error.

Step 3

Write the **`IsMaxError`** subroutine that determines if the integer value of a string is lower than or equal to 32,767.

Input : **A0.L** points to a string that is **not empty and contains only digits**.

Output : **Z** returns true (1) if the integer value of the string is higher than 32,767.
Z returns false (0) if the integer value of the string is lower than or equal to 32,767.

Tips:

- As a first step, use **`StrLen`** to compare the length of the string.
- If the length is shorter than five characters, the integer value is correct.
- If the length is longer than five characters, the integer value is too big.
- If the length is equal to five characters, they have to be compared one by one to the ‘3’, ‘2’, ‘7’, ‘6’ and ‘7’ characters.
- Do not use **`Atoui`** to compare the converted value to 32,767.