CS 271 Computer Architecture and Assembly Language
Programming Assignment #6

Objectives:

1) Designing, implementing, and calling low-level I/O procedures
2) Implementing and using a macro

Problem Definition:

- Implement and test your own *ReadVal* and *WriteVal* procedures for unsigned integers.
- Implement macros *getString* and *displayString*. The macros may use Irvine's *ReadString* to get input from the user, and *WriteString* to display output.
  - *getString* should display a prompt, then get the user's keyboard input into a memory location
  - *displayString* should the string stored in a specified memory location.
  - *readVal* should invoke the *getString* macro to get the user's string of digits. It should then convert the digit string to numeric, while validating the user's input.
  - *writeVal* should convert a numeric value to a string of digits, and invoke the *displayString* macro to produce the output.
- Write a small test program that gets 10 valid integers from the user and stores the numeric values in an array. The program then displays the integers, their sum, and their average.

Requirements:

1) User's numeric input must be validated the hard way: Read the user's input as a string, and convert the string to numeric form. If the user enters non-digits or the number is too large for 32-bit registers, an error message should be displayed and the number should be discarded.
2) Conversion routines must appropriately use the **`lodsb`** and/or **`stosb`** operators.
3) All procedure parameters must be passed on the system stack.
4) Addresses of prompts, identifying strings, and other memory locations should be passed by address to the macros.
5) Used registers must be saved and restored by the called procedures and macros.
6) The stack must be "cleaned up" by the called procedure.
7) The usual requirements regarding documentation, readability, user-friendliness, etc., apply.
8) Submit your text code file (*.asm*) to Canvas by the due date.

Notes:

1) For this assignment you are allowed to assume that the total sum of the numbers will fit inside a 32 bit register.
2) When displaying the average, you may round down to the nearest integer. For example if the sum of the 10 numbers is 3568 you may display the average as 356.

Example (user input in *italics*):

```
PROGRAMMING ASSIGNMENT 6: Designing low-level I/O procedures
Written by: Sheperd Cooper

Please provide 10 unsigned decimal integers.
Each number needs to be small enough to fit inside a 32 bit register.
After you have finished inputting the raw numbers I will display a list
of the integers, their sum, and their average value.

Please enter an unsigned number: 156
Please enter an unsigned number: 51d6fd
ERROR: You did not enter an unsigned number or your number was too big.
Please try again: 34
Please enter an unsigned number: 186
Please enter an unsigned number: 15616148561615630
ERROR: You did not enter an unsigned number or your number was too big.
Please try again: -145
ERROR: You did not enter an unsigned number or your number was too big.
Please try again: 345
Please enter an unsigned number: 5
Please enter an unsigned number: 23
Please enter an unsigned number: 51
Please enter an unsigned number: 0
Please enter an unsigned number: 56
Please enter an unsigned number: 11

You entered the following numbers:
156, 34, 186, 345, 5, 23, 51, 0, 56, 11
The sum of these numbers is: 867
The average is: 86

Thanks for playing!
```

Extra Credit:

1) 1 point: number each line of user input and display a running subtotal of the user's numbers.
2) 2 points: Handle signed integers.
3) 3 points: make your *ReadVal* and *WriteVal* procedures recursive.
4) 4 points: implement procedures *ReadVal* and *WriteVal* for floating point values, using the FPU.

To ensure you receive credit for any extra credit options you did, you must add one print statement to your program output PER EXTRA CREDIT which describes the extra credit you chose to work on. You will not receive extra credit points unless you do this. The statement must be formatted as follows...

```
--Program Intro--
**EC: DESCRIPTION

--Program prompts, etc—
```