

Project 6 - String Primitives and Macros

Due Dec 6, 2020 by 11:59pm **Points** 50 **Submitting** a file upload **File Types** asm
Available after Nov 22, 2020 at 12am

Introduction



This program, the portfolio project for the class, is the final step up in difficulty. Once again the Rubric (see below) now has a number of point deductions for not meeting requirements. It is not uncommon for a student to generate a program that meets the **Program Description** but violates several **Program Requirements**, causing a *significant* loss in points. Please carefully review the Rubric to avoid this circumstance.

The purpose of this assignment is to reinforce concepts related to string primitive instructions and macros (CLO 3, 4, 5).

1. Designing, implementing, and calling *low-level I/O procedures*
2. Implementing and using *macros*

What you must do

Program Description


Write and test a MASM program to perform the following tasks (check the Requirements section for specifics on program modularization):

- Implement and test two **macros** for string processing. These macros may use Irvine's `ReadString` to get input from the user, and `WriteString` procedures to display output.
 - `mGetString`: Display a prompt (*input parameter, by reference*), then get the user's keyboard input into a memory location (*output parameter, by reference*). You may also need to provide a count (*input parameter, by value*) for the length of input string you can accommodate and a provide a number of bytes read (*output parameter, by reference*) by the macro.
 - `mDisplayString`: Print the string which is stored in a specified memory location (*input parameter, by reference*).
- Implement and test two **procedures** for signed integers which use string primitive instructions
 - `ReadVal`:
 1. Invoke the `mGetString` macro (see parameter requirements above) to get user input in the form of a string of digits.
 2. Convert (using string primitives) the string of ascii digits to its numeric value representation (SDWORD), validating the user's input is a valid number (no letters, symbols, etc).
 3. Store this value in a memory variable (*output parameter, by reference*).



- `WriteVal`:
 1. Convert a numeric SDWORD value (*input parameter, by value*) to a string of ascii digits
 2. Invoke the `mDisplayString` macro to print the ascii representation of the SDWORD value to the output.
- Write a test program (in `main`) which uses the `ReadVal` and `WriteVal` procedures above to:
 1. Get 10 valid integers from the user.
 2. Stores these numeric values in an array.
 3. Display the integers, their sum, and their average.

Program Requirements

1. User's numeric input **must** be validated the hard way:
 - a. Read the user's input as a string and convert the string to numeric form.
 - b. If the user enters non-digits other than something which will indicate sign (e.g. '+' or '-'), or the number is too large for 32-bit registers, an error message should be displayed and the number should be discarded.
 - c. If the user enters nothing (empty input), display an error and re-prompt.
2. `ReadInt`, `ReadDec`, `WriteInt`, and `WriteDec` are **not allowed** in this program.
3. Conversion routines **must** appropriately use the `LODSB` and/or `STOSB` operators for dealing with strings.
4. All procedure parameters **must** be passed on the runtime stack. Strings **must** be passed by reference
5. Prompts, identifying strings, and other memory locations **must** be passed by address to the macros.
6. Used registers **must** be saved and restored by the called procedures and macros.
7. The stack frame **must** be cleaned up by the **called** procedure.
8. Procedures (except `main`) **must not** reference data segment variables or constants by name.
9. The program **must** use *Register Indirect* addressing for integer (SDWORD) array elements, and *Base+Offset* addressing for accessing parameters on the runtime stack.
10. Procedures **may** use local variables when appropriate.
11. The program **must** be fully documented and laid out according to the [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1784177/files/81565321/download?download_frd=1)  (https://canvas.oregonstate.edu/courses/1784177/files/81565321/download?download_frd=1). This includes a complete header block for identification, description, etc., a comment outline to explain each section of code, and proper procedure headers/documentation.

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. We will be testing this program with positive **and** negative values.
3. When displaying the average, you may round down (floor) to the nearest integer. For example if the sum of the 10 numbers is 3568 you may display the average as 356.



4. Check the [Course Syllabus](https://canvas.oregonstate.edu/courses/1784177/files/82431838/download?download_frd=1) ↓
(https://canvas.oregonstate.edu/courses/1784177/files/82431838/download?download_frd=1) for late submission guidelines.
5. Find the assembly language instruction syntax and help in the [CS271 Instructions Guide](https://canvas.oregonstate.edu/courses/1784177/files/82340729/download?download_frd=1) ↓
(https://canvas.oregonstate.edu/courses/1784177/files/82340729/download?download_frd=1) .
6. To create, assemble, run, and modify your program, follow the instructions on the course [Syllabus Page](#)'s "Tools" tab.

Resources

Additional resources for this assignment

- [Project Shell with Template.asm](https://canvas.oregonstate.edu/courses/1784177/files/81456520/download?download_frd=1) ↓
(https://canvas.oregonstate.edu/courses/1784177/files/81456520/download?download_frd=1)
- [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1784177/files/81565321/download?download_frd=1) ↓ (https://canvas.oregonstate.edu/courses/1784177/files/81565321/download?download_frd=1)
- [CS271 Instructions Reference](https://canvas.oregonstate.edu/courses/1784177/files/82340729/download?download_frd=1) ↓
(https://canvas.oregonstate.edu/courses/1784177/files/82340729/download?download_frd=1)
- [CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1784177/files/82364688/download?download_frd=1) ↓
(https://canvas.oregonstate.edu/courses/1784177/files/82364688/download?download_frd=1)

What to turn in

Turn in a single .asm file (the actual Assembly Language Program file, not the Visual Studio solution file). File must be named "Proj6_ONID.asm" where ONID is your ONID username. Failure to name files according to this convention may result in reduced scores (or ungraded work). When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become Proj6_ONID-1.asm. Don't worry about this name change as no points will be deducted because of this.

Example Execution

User input in this example is shown in ***boldface italics***.

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

```
Please provide 10 signed 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 signed number: 156
Please enter an signed number: 51d6fd
ERROR: You did not enter a signed number or your number was too big.
Please try again: 34
```

```
Please enter a signed number: -186
Please enter a signed number: 115616148561615630
ERROR: You did not enter an signed number or your number was too big.
```

```
Please try again: -145
Please enter a signed number: 5
Please enter a signed number: +23
Please enter a signed number: 51
Please enter a signed number: 0
Please enter a signed number: 56
Please enter a signed number: 11

You entered the following numbers:
156, 34, -186, -145, 5, 23, 51, 0, 56, 11
The sum of these numbers is: 5
The rounded average is: 1

Thanks for playing!
```

Extra Credit (Original Project Definition must be Fulfilled)

To receive points for any extra credit options, 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--
```

Extra Credit Options

1. Number each line of user input and display a running subtotal of the user's valid numbers. These displays must use `writeVal`. (1 pt)
2. Implement procedures `ReadFloatVal` and `WriteFloatVal` for floating point values, using the FPU. These must be in addition to `ReadVal` and `WriteVal` and you must have a separate code block to demo them (one 10-valid entry loop to demo `ReadVal` / `WriteVal` and one 10-valid-entry loop to demo `ReadFloatVal` and `WriteFloatVal`). (4pts)

Grading criteria

Please view the rubric attached to this assignment to understand how your assignment will be graded. If you have any questions please ask on the course discussion board.

Project 6 Rubric



Criteria	Ratings		Pts
Files Correctly Submitted Submitted file is correct assignment and is an individual .asm file.	1 pts Full Marks	0 pts No Marks	1 pts
Program Assembles & Links Submitted program assembles and links without need for clarifying work for TA and/or messages to the student. This assumes the program is actually an attempt at the assignment. Non-attempts which compile/link earn no points.	1 pts Full Marks	0 pts No Marks	1 pts
Documentation - Identification Block - Header Name, Date, Program number, etc as per CS271 Style Guide are included in Identification Block	1 pts Full Marks	0 pts No Marks	1 pts
Documentation - Identification Block - Program Description Description of functionality and purpose of program is included in identification block, in student's own words.	1 pts Full Marks	0 pts No Marks	1 pts
Documentation - Procedure/Macro Headers Procedure and Macro headers describe functionality and implementation of program flow in student's own words. See CS271 Style Guide for detailed requirements.	2 pts Full Marks	0 pts No Marks	2 pts
Documentation - Section and In-line Comments Section and In-line comments contribute to understanding of program flow where necessary, but are not line-by-line descriptions of moving memory to registers. See CS271 Style Guide.	2 pts Full Marks	0 pts No Marks	2 pts
Verification - Program Executes Program executes and makes some attempt at the assigned functionality.	2 pts Full Marks	0 pts No Marks	2 pts
Completeness - Test program gets 10 validated integers from user and displays them later	3 pts Full Marks	0 pts No Marks	3 pts



Criteria	Ratings				Pts
Completeness - Displays Correct Sum	1 pts Full Marks	0 pts No Marks			1 pts
Completeness - Displays Correct Average	1 pts Full Marks	0 pts No Marks			1 pts
Correctness - String to Number Conversion All instances of conversion from String to Number generate correct output.	4 pts Full Marks	0 pts No Marks			4 pts
Correctness - Number to String Conversion All instances of conversion from Number to String generate correct output.	4 pts Full Marks	0 pts No Marks			4 pts
Correctness - mGetString macro works Correctly displays prompt, gets user input, and stores user input in memory.	2 pts Full Marks	0 pts No Marks			2 pts
Correctness - mDisplayString macro works Correctly prints the string at the passed address.	2 pts Full Marks	0 pts No Marks			2 pts
Correctness - mGetString and mDisplayString are used to get/display strings	2 pts Full Marks	0 pts No Marks			2 pts
Requirements - User Input validated as per Assignment Document User input validated by checking string for invalid characters, or value too large for 32-bit SDWORD.	4 pts Full Marks	2 pts Invalid characters handled, Overflow unhandled	2 pts Overflow handled, but Invalid Characters unhandled	0 pts No Marks	4 pts
Requirements - Conversions use String Primitives Conversions use LODSB and/or STOSB These points are only awarded if the conversions function correctly.	2 pts Full Marks	0 pts No Marks			2 pts



Criteria	Ratings		Pts
Requirements - Procedures implemented with logical hierarchy	2 pts Full Marks	0 pts No Marks	2 pts
Requirements - Properly uses WriteVal Numeric output is converted from numeric to string using WriteVal Must be used for both for SUM and AVERAGE	4 pts Full Marks	0 pts No Marks	4 pts
Requirements - Properly uses ReadVal All number input is obtained as string input and converted to numeric values, stored in memory, using ReadVal	4 pts Full Marks	0 pts No Marks	4 pts
Requirements - Used registers saved/restored by procedures and macros	2 pts Full Marks	0 pts No Marks	2 pts
Requirements - No Globals Outside of Main, STDCALL calling convention (pass parameters on stack) No global variable references outside of main. Parameters are passed on runtime stack, cleaned up by called procedure using "ret n" -5 points per procedure (other than main) which uses globals -5 points (per procedure) for parameters not properly passed via STDCALL calling convention	0 pts Full Marks	0 pts No Marks	0 pts
Coding Style - Uses appropriately named identifiers Identifiers named so that a person reading the code can intuit the purpose of a variable, constant, or label just by reading its name.	1 pts Full Marks	0 pts No Marks	1 pts
Coding Style - Readability Program uses readable white-space, indentation, and spacing as per the Indentation Style Guide. Logical sections are separated by white space.	1 pts Full Marks	0 pts No Marks	1 pts
Output Style - Readability Program output is easy to read	1 pts Full Marks	0 pts No Marks	1 pts



Criteria	Ratings		Pts
Extra Credit Numbers each line of user input / displays running subtotal of user's numbers (1) Implements ReadVal / WriteVal for floating point values, using the FPU (4) handles input / output with interrupts instead of ReadString and WriteString (obsolete: 4)	0 pts Full Marks	0 pts No Marks	0 pts
Late Penalty Remove points here for late assignments. (Enter negative point value)	0 pts Full Marks	0 pts No Marks	0 pts
Total Points: 50			

