AUBURN UNIVERSITY CPSC 3350 Programming Project 4

Instructions For this project, you must write three **ROBUST** procedures in **assembly language using ONLY at most 32-bit registers**. The three **ROBUST** procedures will be included in **ONE** file inside **ONE** project and called by your main to test them. These procedures should be implemented and tested using the *Visual Studio* IDE. For all these exercises, you cannot use built-in functions that perform these operations. In case of a doubt, check with your instructor.

For this project, you will build simple strings functions. You will use basic **string** input/output functions provided by the author of the textbook.

Objectives of this assignment:

- **to learn how to perform 64-bit operations using only 32-bit registers**.
- to get familiar with *Visual Studio*
- to stress on the fact that the memory (variables) ultimately are 0s and 1s
- to use some basic **string** input/output functions
- to use some basic **byte** input/output functions

Warning:

The executable you will build may get **blocked** by an Antivirus scanner on your system. In this case, read this from the Textbook author: "*Antivirus scanner software has improved greatly in recent years, and so have the number of viruses (one website reports 50,000 at present). Because of this, your computer's antivirus scanner may report a false positive when you build your program, and refuse to let you run it. There are a few workarounds: (1) You can add your project's bin/debug folder into an exclusion list in your antivirus configuration. This is my preferred approach, and it nearly always works. (2) You can suspend your realtime antivirus scanner software, but this will leave you open to malware for a short time. If you choose this option, be sure to temporarily disconnect your computer from the Internet. (3) You can send a copy of your program's EXE file to the antivirus software vendor, labeling it as a false positive. The virus scanner I use automatically uploads any questionable EXE file to their website and either quarantines or releases the file within about 30 minutes.*" *I used the first workaround, i.e., I added my project's /debug folder to the exclusion list in my antivirus configuration.*

Input/Output Built-in Functions Allowed:

The textbook author wrote some helpful functions. For this project, you **can** use these functions:

1) `ReadString`: the `ReadString` procedure reads a string from the keyboard, stopping when the user presses the *Enter* key. You must pass two parameters before calling this function:

   a. **Offset** of the buffer where to store the entered string: this offset must be passed in Register EDX

   b. **Maximal** number of characters: this number is the maximal number of characters a user can enter. This number must be passed in Register ECX.

   As a result, `ReadString` will store in the buffer the string entered by the user and will return in the register EAX the number of characters entered by the user. See excerpt from the textbook:

ReadString The ReadString procedure reads a string from the keyboard, stopping when the user presses the Enter key. Pass the offset of a buffer in EDX and set ECX to the maximum number of characters the user can enter, plus 1 (to save space for the terminating null byte). The procedure returns the count of the number of characters typed by the user in EAX. Sample call:

```
.data
buffer BYTE 21 DUP(0)          ; input buffer
byteCount DWORD ?              ; holds counter
.code
mov   edx,OFFSET buffer        ; point to the buffer
mov   ecx,SIZEOF buffer        ; specify max characters
call  ReadString               ; input the string
mov   byteCount,eax            ; number of characters
```

ReadString automatically inserts a null terminator in memory at the end of the string. The following is a hexadecimal and ASCII dump of the first 8 bytes of **buffer** after the user has entered the string "ABCDEFG":

```
41 42 43 44 45 46 47 00        ABCDEFG
```

2) `WriteString`: (from the textbook) the `WriteString` procedure writes a null-terminated string to the console window. You must pass the string's offset in EDX. Below is a sample call from the textbook:

WriteString The WriteString procedure writes a null-terminated string to the console window. Pass the string's offset in EDX. Sample call:

```
.data
prompt BYTE "Enter your name: ",0
.code
mov   edx,OFFSET prompt
call  WriteString
```

3) `ReadChar` : this function will allow you to read a character from the keyboard and receive its ASCII code in the register AL. This function is described in the textbook Page 165 (beginning/page).

4) `WriteChar` : this function will allow you to display on the console window (command window) a character whose ASCII code is stored in the register AL. See in the textbook Page 169 (start/page).

5) `WriteDec` : (from the textbook page 169) the procedure `WriteDec` procedure writes a 32-bit unsigned integer to the console window in **decimal** format with no leading zeros. You must pass the integer (to display) in the register EAX.

6) `WriteHex` : (from the textbook page 169) the procedure `WriteHex` procedure writes a 32-bit unsigned integer to the console window in **hexadecimal** format with leading zeros. You must pass the integer (to display) in the register EAX.

7) `ReadHex` : this function reads a 32-bit hexadecimal integer from the keyboard and returns it in EAX. This function is described in the textbook Page 166 (middle/page).

==========

Note: the three procedures must be included in ONE project using ONE assembly file. Try to create one procedure for each exercise. **Write a main program calling the three procedures**. **If your exercises are not set as PROCEDURES that can be CALLED from the main or any other procedure, you will loose half of the MAX credit.**

Efficient programming is expected.

Programming Exercise 1 (25 points): Reminder (Use at most 32-bit registers)

The objective of this exercise is to write a **ROBUST procedure** that

1) Prompts the user with the message: "Please enter a 64 bit in hexadecimal (8 bytes) "
2) Reads the eight bytes and store them in the memory as a 64 bit unsigned integer N_1.
3) Prompts the user with the message: "Please enter a 64 bit in hexadecimal (8 bytes) "
4) Reads the eight bytes and store them in the memory as a 64 bit unsigned integer N_2.
5) Computes the sum $S = N_1+N_2$.
6) Displays in hexadecimal the number S. Note that S could be a 65 bit number.

Programming Exercise 2 (55 points): Reminder (Use at most 32-bit registers)

The objective of this exercise is to write a **ROBUST procedure** that

7) Prompts the user with the message: "Please enter a 64 bit in hexadecimal (8 bytes) "
8) Reads the eight bytes and store them in the memory as a 64 bit unsigned integer N_1.
9) Prompts the user with the message: "Please enter a 64 bit in hexadecimal (8 bytes) "
10) Reads the eight bytes and store them in the memory as a 64 bit unsigned integer N_2.
11) Computes the product $P = N_1*N_2$.
12) Displays in hexadecimal the product P .

Programming Exercise 3 (30 points):

The objective of this exercise is to write a **ROBUST procedure** that

1) Prompts the user with the message: "Please enter a 32 bit in hexadecimal (4 bytes) "
2) Reads the four bytes and store them in the memory as a 32 bit unsigned integer N.
3) Displays the number N in Base 8. (Do not use division to "covert")

AUBURN UNIVERSITY CPSC 3350 Programming Project 4

What you need to turn in:

- **Post on Canvas** the following files:
  1) Electronic copy of your report (standalone)
  2) The source code ***main.asm*** (standalone)
  3) Your full **project** folder that will include your modified ***main.asm***. This project folder must be named **first** named m8-name where name is your last name. Do not have spaces in the file names. **After** you correctly name the folder (m6-name), zip it and post it on Canvas.

Your report must:

- State whether your code works (this should not take more than a sentence). Failing to make this statement can be interpreted that your code does not work correctly and does not meet the requirements
- Good writing and presentation are expected.

How this assignment will be graded:

1. The program compiles but does not produce any meaningful result (5%).
2. The program compiles and executes with major bugs (40% credit).
3. The program compiles and executes with minor bugs. The code is well designed and commented (90% credit).
4. The program compiles and executes correctly without any apparent bugs. The code is well designed and commented (100%).
5. If the instructor needs additional communications/actions to compile and execute your code, then a 30% penalty will be applied.
6. If the turn-in instructions are not correctly followed, 10 points will be deducted.