CS3BMASM2S17Arithmetic Input strings Inputting Numerical Values and Performing Arithmetic

Due: Tuesday, October 11, 2016 by 9:00 AM **Edited: 9/28/2016 at 1:00 PM**

Write a program named **MASM2.asm** which will input numeric information from the keyboard, add, subtract, multiply, and divide, as well as check for overflow and/or invalid numeric information. You will use methods from the **.obj** files in your MASM folder (**convutil** and **utility**)

**New Commands**

For this assignment you may need the following NEW commands: **CBW, CWD, CWDE, CDQ,** **JMP, CMP, LOOP, IDIV,IMUL,INC,DEC** along with several of the mnemonic jumps:**jne,je,jg,jge,jc,jo, etc.**

and new methods: **getch, getche, putch,**

**Documentation standards for Assembly – please remember to adhere to the below standards**

In addition to a comment block at the top of your code (as per examples in class), you must follow **one** of the below coding requirements:

**COMMENT #**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*File name: MASM2.asm**

**\*Project name: MASM2**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*Creator’s name & email: *yourname and email address***

**\*Course-Section: CS3B**

**\*Creation Date: February 12, 2017 at 8:00 AM**

**\*Purpose:**

**\* Input numeric information from the keyboard, perform addition, subtraction,  
\* multiplication, and division. Check for overflow upon all operations.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#**

OR

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
;Name: Your Name  
;Program: MASM2.asm  
;Class: CS3B  
;Lab: MASM2  
;Date: February 12, 2017 at 8:00 AM  
;Purpose:  
; Input numeric information from the keyboard, perform addition, subtraction,   
; multiplication, and division. Check for overflow upon all operations. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

.486

.model flat

.stack 100h

1) a) All of the “dot” directives (.486, .model, .data,.code, etc) MUST be indented at least one TAB  
 b) the three directives, .486, .model,.stack should be single-spaced with ONE space following .stack  
 c) double-space BEFORE each of .data and .code  
2) a)ALL labels (including data identifiers and labels on assembly instructions) MUST begin in column 1  
 b) ALL code instructions MUST begin at least one TAB over  
3) ALL **PROTO** statements  **need to start at least one TAB** and the PROTO’s all line up

.stack 100h

ExitProcess PROTO Near32 stdcall, dwExitCode:dword

ascint32 PROTO Near32 stdcall, lpStringOfNumericChars:dword

intasc32 PROTO Near32 stdcall, lpStringToHold:dword, dVal:dword

putstring PROTO Near32 stdcall, lpStringToPrint:dword

getche PROTO Near32 stdcall ;returns character in the AL register  
 getch PROTO Near32 stdcall ;returns character in the AL register

putch PROTO Near32 stdcall, bChar:byte

**You may NOT use the getstring method!!!**

4) All string identifiers for input/output MUST begin with **str**   
5) All **dwords** must begin with either ‘d’ or the letter ‘i’ to indicate that is the size of an **int**  
6) All **words** must begin with the letter ‘s’ or ‘w’ to indicate the size of a **short**  
7) All **qwords** must begin with the letter ‘q’ to indicate the size of a qword  
8) All **byte** identifiers must begin with the letter ‘b’ to indicate a **byte** OR the letter ‘c’ to indicate a **char**  
9) Every identifier (except string constants) MUST have a comment explaining its use in the program  
10) a) Identifiers and labels MUST be camel-cased and begin with a lower-case letter  
 b) label identifiers MUST be on a line by itself. Do not write any code on a line that contains a label identifier  
11) The only label in your “driver” program that begins with an underscore ‘\_’ is the **\_start**. This convention is typically used in other methods for any identifiers that might be used (labels and/or data identifiers – in our class you will never define data storage within any method except your “driver” (unless otherwise instructed)  
12) Line comments MUST be used on EVERY line of code in your assembly program. Only on rare occasions will you not put a comment.   
13) Comments MUST line up within the data segment and within the code segment where at all possible. There will be instances when a few instructions are so long that it would confuse things.  
14) No line-wrapping permitted! Set the edge to 91 with Notepad++.  
15) The directives (byte, word, dword, qword) MUST line up within the data segment, as closely as possible  
16) Use as the first 2 lines **.code** the following (When using **Ollydbg** you can only “step into” methods where you have the source (.asm) for that method. Because I do not provide you the code for methods that I have written, you must “step over” those instructions that INVOKE one of my methods. The instruction below gives you a chance to step over an INVOKE. Without it, the debugger will crash.  
 **\_start:  
 MOV EAX,0 ;ensures first instruction can be executed in Ollybg**

**Converting an ASCII string of numeric characters to its equivalent binary value**

**ascint32 PROTO stdcall, lpStringToConvert:dword**

This method accepts the address of a null-terminated string and attempts to convert that string to a signed 4-byte value. If the conversion is successful, the resulting integer value is returned to the calling program in the **EAX** register. However, if there is an invalid character in the string which does not permit the conversion, then the Carry flag is set and the value 0 is returned in the EAX register. If the conversion would produce an overflow, then the overflow flag is set and the vale 0 is returned in the EAX register. Thus, you will need to test these flags upon return and display an appropriate message. You should test the carry flag first, and if it is not set, then test the overflow flag. If it is not set, then it is ok to store the returned value . For example:

**INVOKE ascint32, ADDR strNumber ;convert string to a number  
 jc displayInvalidMsg ;not a valid numeric string  
 jo displayOvflMsg ;too big a number to convert**

**Program Requirements**

Write your program to continue asking for values until the user hits just the ENTER key (in which case the very first character in the numeric string is the NULL character). If the user hits just the ENTER key for **any** requested input the program terminates.

Below, I have listed “sample runs” to show you what should appear on the screen. Underlined values indicate what the user entered and how you are to respond.

**Name: *Your name*  
Program: MASM2.asm  
Class: CS3B  
Date: February 12, 2017**

**Enter your first number: 100  
Enter your second number: 200**  
**The sum is 300  
The difference is -100** (the result of the first number – the second number) **The product is 20000**  
**The quotient is 0** (the resulting quotient of dividing the first number by the second)  
**The remainder is 100**  (the resulting remainder upon division of the first number by the second)

**Enter your first number: -152  
Enter your second number: 6**  
**The sum is -146  
The difference is -158   
The product is -912**  
**The quotient is -25   
The remainder is -2**

**Enter your first number: 146  
Enter your second number: 0**  
**The sum is 146  
The difference is 146   
The product is 0**  
**You cannot divide by 0. Thus, there is NO quotient or remainder**

**Enter your first number: 1B0  
INVALID NUMERIC STRING. RE-ENTER VALUE  
Enter your first number: 1400  
Enter your second number: 4000000000  
OVERFLOW OCCURRED. RE-ENTER VALUE  
Enter your Second number: 400**  
**The sum is 1800  
The difference is 1000   
The product is 560000**  
**The quotient is 3   
The remainder is 200**

**Enter your first number: 100  
Enter your second number: 1B0  
INVALID NUMERIC STRING. RE-ENTER VALUE  
Enter your second number: 1400  
The sum is 1500  
The difference is -1300** (the result of the first number – the second number) **The product is 140000**  
**The quotient is 0   
The remainder is 100**

**Enter your first number: 2147483647  
Enter your Second number: 10**  
**OVERFLOW OCCURRED WHEN ADDING  
The difference is 2177483637   
OVERFLOW OCCURRED WHEN MULTIPLYING  
The quotient is 214748364   
The remainder is 7**

If you are ever prompted for a number and you just hit the **ENTER** key, terminate your program. For example,

**Enter your first number: 1400  
Enter your second number: 1B0  
INVALID NUMERIC STRING. RE-ENTER VALUE  
Enter your second number:** (here user just hits the ENTER key)

**Thanks for using my program!! Good Day!**(leave a blank line after your program ends to make it easier to read)

**C:\Temp> \_**

**Deliverables- DO NOT SUBMIT A ZIPPED FILE**

Upload **proj2.asm** to Bb by the listed due date. Make sure that you have full documentation at the beginning of your program, that your purpose is descriptive and specific, that you display the header information when the program is run. Make sure that every line is commented with a brief, but descriptive, comment. DO NOT OVER COMMENT. Where possible line up your comments to make them more readable. You must comment every identifier that is not a literal constant (e.g. it is not necessary to comment something like   
  
**strOverflowAdd byte 10,13,”OVERFLOW occurred when ADDING”,0  
strOverflowSub byte 10,13,”OVERFLOW occurred when SUBTRACTING”,0  
strOverflowMul byte 10,13,”OVERFLOW occurred when MULTIPLYING”,0  
strOverflowConv byte 10,13,”OVERFLOW occurred when CONVERTING”,0  
strInvalidString byte 10,13,”INVALID character in numeric string”,0**

**ALSO: Save your project as MASM2.txt and upload it to Bb so I can run it through TurnItIn. When you finish your project remember to upload BOTH the .asm and .txt files. If you do NOT submit a .txt file you will receive a 20% penalty.**

**Hint:** If the user is prompted to enter something and the user only hits the ENTER key without typing anything, a NULL string is created, which means the very 1st character in that string is 0 (the null character). So, to see if the user is ending the program, just compare the 1st character in the input string to 0. If it is equal to 0, then jump to displaying the final output line(s).

**Restrictions**

You may NOT use the getstring method to input a string. Instead you will use **getch** which gets a character but does NOT echo it back on the screen. Your normal input from the keyboard is really made up of 2 commands: get the character AND display it. The method **getche** will get the character AND displays it.

You will have to set up loop that ends ONLY when the user hits the ENTER key (ASCII value **0dh**) As the user enters a character, store the character the user entered into memory. Every character the user types is acceptable and will be stored in memory. The backspace key, **08h**, when entered, actually moves the cursor on the screen to the left one position. So, to erase a character, you need to do 3 steps: move cursor to the left, display a blank, and then move the cursor to the left again (when you display a blank, you move the character to the right).

**Caution**: you have to be careful when to allow backspacing as you do NOT want to backspace over anything that may have been displayed on the screen before starting to enter the number. To make sure that your program does not permit this, you are required to enter your values right after the input prompt and NOT on the next line underneath the prompt. In the below example you do NOT want to allow the user to backspace over any part of the prompt.

**Enter a numeric value: 123**

In your program declare the below 2 identifiers. When requesting a numeric value, use dLimitNum to restrict the number of characters the user can enter. When requesting any other string, restrict the number of permitted characters to dLimitAlpha. Once the user has reached the limit do NOT simply continue on because they may decide to hit the backspace key. **Your program does NOT continue on until the user presses the ENTER key. At that point, append the value 0 to the string you were building in memory.**

dLimitNum dword 11; the limit for entering numeric strings  
dLimtitAlpha dword 79; the limit for entering alphameric strings

In essence you are writing the same instructions that would be found in getstring. You have to limit the number of characters the user is allowed to enter to the number specified. In our case, when you are requested to enter a numeric string, limit the number of characters to 11. Once the user types the 11th character, no other character is acceptable except the backspace key. When you are requesting input you do not need to worry about it being a valid numeric string as the ascint32 method will test it for you.