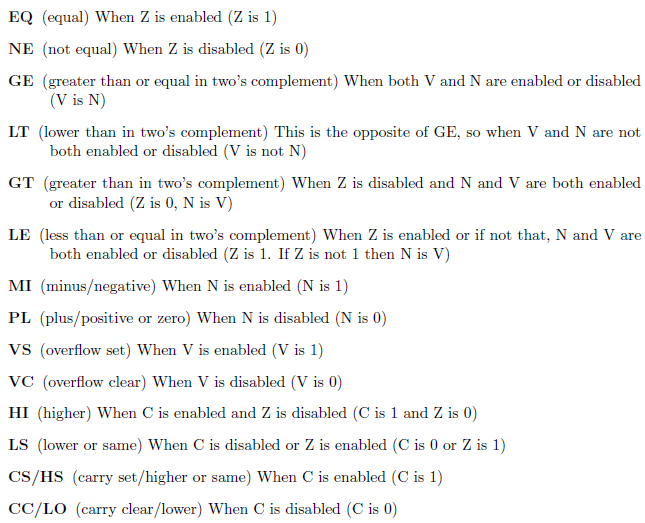
CS 3Bpi Arithmetic Input strings Inputting Numerical Values and Performing Arithmetic

Write a program named **rasm2.s** 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 may only use methods from external files that I approve.

**New Commands**

For this assignment you may need the following NEW external macro: **IDIV,**along with several of the mnemonic ‘B’ jumps:

****

**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:

**@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
@Name: Your Name  
@Program: RASM2.s  
@Class: CS 3B  
@Lab: RASM2  
@Date: October 4, 2018 at 8:00 AM  
@Purpose:  
@ Input numeric information from the keyboard, perform addition, subtraction,   
@ multiplication, and division. Check for overflow upon all operations. @\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1) a) All of the “dot” directives (.data, .text, etc) MUST be indented at least one TAB  
 b) ALL labels (including data identifiers and labels on assembly instructions) MUST begin in column 1  
 c) ALL code instructions MUST begin at least one TAB over

.data

iNum .word 0

.text

2) All string identifiers for input/output MUST begin with **str**   
3) All **words** must begin with the letter ‘i’ to indicate that is the size of an **int**  
4) All **byte** identifiers must begin with the letter ‘b’ to indicate a **byte** OR the letter ‘c’ to indicate a **char**  
5) Every identifier (except string constants) MUST have a comment explaining its use in the program  
6) 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  
7) 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)  
8) Line comments MUST be used on EVERY line of code in your assembly program. Only on rare occasions will you not put a comment.   
9) 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.  
10) No line-wrapping permitted! Set the edge to 91 with Notepad++.  
11) The directives (byte, hword, word, ascii, etc.) MUST line up within the data segment, as closely as possible

**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: rasm2.asm  
Class: CS 3B  
Date: October 4, 2018**

**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 2147483637   
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 **rasm2.s** to canvas. 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.  
**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**

**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**

ALL user input must go into a keyboard buffer that can hold up to 512 bytes. Reuse it for all of your inputs. If you need to save the input into another string, then write a loop that will copy from the keyboard buffer into your string variable. If you want to store in your program declare the below 2 identifiers. When requesting a numeric value, use iLimitNum to restrict the number of characters the user can enter.

iLimitNum .word 12; the limit for entering numeric strings

Handle the case if the user types more than iLimitNum allows (i.e. type 12345678901234567890). Your program should not break. 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.