CS/ECE 3280 LAB Assignment #2
Due date: Wednesday, September 25, before 12:00 pm.

You are to design, write, assemble, and simulate an assembly language program which will divide one integer (dividend) by another (divisor) by subtracting the divisor from the dividend the correct number of times. Both numbers, dividend and divisor, are **UNSIGNED** byte variables.

In general: quotient = floor(dividend/divisor)
remainder = dividend mod divisor = dividend - quotient*divisor

For example, if dividend = 14 and divisor = 3, then your program should do the following steps:

1. 14 – 3 = 11
2. 11 – 3 = 8
3. 8 – 3 = 5
4. 5 – 3 = **2** ; **2< 3** => stop

=> quotient =4; remainder = 2

Your program should compute and store both the quotient and the remainder, if the divisor is not zero; if **the divisor is zero, your program should store $FF in both the quotient and remainder locations**. Your program should also handle the case where the divisor is larger than the dividend. The dividend, divisor, quotient, and remainder are all 8-bit unsigned integers.

PLEASE NOTE:

1. Your program should work for any DIVIDEND and DIVISOR values, not just for the ones given.
2. You are NOT allowed to use the DIVL instruction anywhere in this program; you need to use the above-mentioned division algorithm instead.
3. Your program is NOT allowed to change the numbers stored in DIVIDEND and DIVISOR.
4. You have to use the program skeleton provided for Lab2. Do not change the data section or you will lose points! This means: do not change the 'ORG $B000' and 'ORG $B010' statements or the variable names 'DIVIDEND', DIVISOR', 'QUOTIENT, and 'REMAINDER'. You are allowed to change the values assigned to DIVIDEND and DIVISOR to simulate different number pairs. If you need to define additional variables, please add them after the 'REMAINDER RMB 1' statement.
5. You must terminate your program correctly using the infinite loop structure as shown in class (for easier simulation).
6. You do not have to optimize your algorithm or your assembly program for speed.
7. You have to provide a pseudo-code solution. In your pseudo code, do NOT use a for loop, but either a while or a do-until structure to implement a loop. Also, do NOT use any “goto”, “break”, “exit”, or “return” statements in your pseudocode.
8. The structure of your assembly program should match the structure of your pseudo code 1-to-1 (e.g., if the pseudo code shows a while structure, your assembly program should also have a while structure).
9. Make sure that you implement any if-then, if-then-else, while, or do-until structures the way you learned it in class. Incorrectly implemented structures will result in points lost.
10. Any assembler or simulator error/warning messages appearing when assembling/simulating your submitted program will result in up to 50 points lost.

You should test your program with at least five sets of decimal data:

A. Dividend: 255 Divisor: 6
B. Dividend: 20 Divisor: 4
C. Dividend: 98 Divisor: 255
D. Dividend: 59 Divisor: 0
E. Dividend 0 Divisor: 25

Figure out the answers (in hex) to be sure that your program is executing correctly.
PLEASE NOTE: Your program will be tested by us using random number pairs. If your program does not produce correct results for those random pairs, you will lose up to 25 points (see grading guidelines below).

Your program should include a header containing your name, student number, the date you wrote the program, and the lab assignment number. Furthermore, the header should include the purpose of the program and the pseudocode solution of the problem. At least 85% of the instructions should have meaningful comments included - not just what the instruction does; e.g., don't say "increment the register A" which is obvious from the INCA instruction; say something like "increment the loop counter" or whatever this incrementing does in your program. You can ONLY use branch labels related to structured programming, i.e., labels like IF, IF1, THEN, ELSE, ENDIF, WHILE, ENDWHL, DOUNTL, DONE, etc. DO NOT use labels like LOOP, JOE, etc. **Remember:** labels need to be unique. So, for example, if you have two if-then structures, you should use the labels IF,THEN, ENDIF for the first structure and IF1,THEN1, ENDIF1 for the second one.

YOU ARE TO DO YOUR OWN WORK IN WRITING THIS PROGRAM!!! While you can discuss the problem in general terms with the instructor and fellow students, when you get to the specifics of designing and writing the code, **you are to do it yourself**. Re-read the section on academic dishonesty in the class syllabus. If there is any question, consult with the instructor.

Submission:

Electronically submit your .ASM file on Canvas by 12:00pm on the due date. Late submissions or re-submissions (with a 10% grade penalty) are allowed for up to 24 hours (please see the policy on late submission in the course syllabus).

Note:

Because of some inherent lack of reliability designed into computers, and Murphy's law by which this design feature manifests itself in the least convenient moment, you should start your work early. Excuses of the form:

"my memory stick went bad,"
"I could not submit my program,"
"my computer froze up, and I lost all my work;"

should be directed to the memory stick manufacturer, Canvas system administrator, and your local Microsoft vendor respectively.

Grade Requirements and Breakdown

The assignment is worth 100 points. The following deductions will be made (maximum deduction of 100 pts):

Correct Pseudocode

- Pseudocode incomplete or missing: -50 points
- wrong algorithm implemented: -50 points
- for-loop used in pseudocode: -10 points
- "break/exit/return/goto" used in pseudo-code: -10 points

Program must produce correct results

- program does not produce correct result for certain DIVIDEND/DIVISOR pairs: up to -25 points

Program must have good structure

- program does not assemble or is incomplete: -50 points
- assembler or simulator error/warning messages during assembly/simulation: -25 points
- DIVL instruction used in program: -50 points
- program changes DIVIDEND/DIVISOR variables: -5 points
- structure X incorrectly implemented: -5 points for each structure
- hardcoded addresses used in program: -5 points

Program must match pseudo code 1-to-1

- program structure does not match pseudo-code (program implements structure X, but different or no structure shown in pseudo code; or vice versa): -5 points for each structure
- branch for signed numbers used but variables unsigned; or vise versa: -5 points for each branch

Good Commenting

- no program comments at all: -20 points
- program not commented enough: -10 points
- program description missing: -5 points
- incomplete program header: -5 points
- incorrect branch labels used: -5 points

Note: This list is by no means comprehensive but only lists the most common deductions.