CS/ECE 3280
LAB Assignment #5
Due date: November 6, before 12:00 pm.

You are to design, write, assemble, and simulate an assembly language program which will calculate the Euclid's Greatest Common Divisor of 1-byte unsigned numbers stored in two TABLES (the sentinel of both tables is $FF). The calculated GCD of each number pair will be stored in an array RESULT. The actual GCD calculation has to be implemented in a subroutine.

The difference between this lab and lab 4 is that only local DYNAMIC variables implemented on the stack are allowed in the subroutine. Please see item 13 below for further info.

PLEASE NOTE:

1. The complete GCD calculation has to be done using the algorithm defined in Lab3, resulting in two nested loops, **inside a single subroutine**.
2. Your program should work for any number values, not just the ones given in the tables.
3. Your program is NOT allowed to change the numbers stored in TABLE 1 and TABLE2.
4. You have to use the program skeleton provided for Lab4. 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 'TABLE1'. 'TABLE2' and 'RESULT'. Do NOT change the values assigned to the tables. If you need to define additional variables, please add them in the appropriate places.
5. You are allowed to use parts of your LAB3 or parts of the official LAB3 solution.
6. You are allowed to **declare static local variables** in your subroutine (through RMB).
7. You must terminate your main program correctly using an infinite loop structure.
8. You do not have to optimize your program or algorithm for speed
9. You have to provide a pseudo-code solution for your main program AND your subroutine. In your pseudo codes, 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", or "exit" statements in your pseudo codes. The structure of your assembly program should match the structure of your pseudo codes 1-to-1.
10. Any assembler or simulator error/warning messages appearing when assembling/simulating your submitted program will result in up to 25 points lost.
11. The main program should be a WHILE structure which goes through the TABLE1 and TABLE2 tables and sends two values to the subroutine during each iteration. The while structure will also check for the Sentinel (which is the $FF at the end of the tables) at each iteration. The Sentinel is NOT one of the data items and it should NOT be processed by the subroutine. The main program must end the while loop when the $FF is encountered. For each subroutine call, the subroutine will send back a 1-byte result that has to be stored consecutively in the RESULT table.

    - You can assume that TABLE1 and TABLE2 will always have the same length.
    - You are not allowed to just manually count the number of elements in the table and set up a fixed number in memory as a count variable.
    - Your program should still work if the arrays are moved to different places in memory (do not use any fixed offsets).
    - You don't have to copy the sentinel to the end of the RESULT array.
    - Your program should work for any number of elements in the table. Thus, there could be more than 255 elements in the table. Using the B-register as an offset and the ABX/ABY instructions to point into the array will therefore not work.

12. For each iteration, the main program should take one number from each table and pass it to the subroutine **in a register (call-by-value in register)**. The subroutine performs the GCD calculation and produces a 1-byte result. This result byte must be passed back to the main program **OVER THE STACK (call-by-value over the stack)**. The main program then retrieves the bytes from the stack and stores it in the RESULT table.
    - Make sure that your program will not generate a stack underflow or overflow.
13. **Only local DYNAMIC variables can be used in the subroutine**. Thus, local variables have to be implemented on the stack and have to be accessed using indexed addressing mode (as shown in class).
14. Any assembler or simulator error/warning messages appearing when assembling/simulating your submitted program will result in up to 50 points lost.

!!! NOTE !!! – ONLY LOCAL VARIABLES ARE ALLOWED (LOCAL TO THE MAIN PROGRAM AND THE SUBROUTINE). INSIDE THE SUBROUTINE YOU CAN ONLY ACCESS LOCAL VARIABLES AND ITEMS PASSED IN FROM THE MAIN PROGRAM!!! YOU MUST NOT ACCESS MAIN PROGRAM VARIABLES (SUCH AS TABLE1, TABLE2, RESULT, OR ANY OTHER VARIABLE DECLARED IN THE MAIN PROGRAM) FROM WITHIN THE SUBROUTINE!!! ALSO, THE MAIN PROGRAM IS NOT ALLOWED TO ACCESS ANY LOCAL SUBROUTINE VARIABLES!!!

-> IF YOU HAVE A QUESTION AS TO WHETHER YOUR PROGRAM OR SUBROUTINE VIOLATES ANY OF THE SPECIFIC REQUIREMENTS, ASK THE INSTRUCTOR.

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).

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
- index used instead of pointers: -10 points
- for-loop used in pseudocode: -10 points
- "break/exit/goto" used in pseudo-code: -10 points

Program must produce correct results

- program does not produce correct result for certain TABLE1/TABLE2 values: up to -25 points
- program does not work for more than 255 table elements: -10
- program does not work if tables moved to different locations: -10
- program does not work with an empty table: -5 points

Correct Parameter passing implemented

- main program variables accessed in subroutine (or vice versa): -20 points
- incorrect parameter passing from subroutine ("over the stack" not implemented): -10 points
- incorrect parameter passing from subroutine ("call-by-value " not implemented): -10 points
- incorrect parameter passing to subroutine ("in register" not implemented): -10 points
- incorrect parameter passing to subroutine ("call-by-value" not implemented): -10 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
- static variables used in subroutine: -50 points
- program changes TABLE1/TABLE2 elements: -5 points
- multiple subroutines used: -10 points
- part of GCD calculation done in main program: -10 points
- excess bytes left on stack or stack underflow: -5 points
- does not work when stack is moved to a different memory section: -10 points
- multiple RTS in subroutine: -5 points
- program does not work for any sentinel != $FF: -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.