CS/ECE 3280
LAB Assignment #4
Due date: Wednesday, October 26, before 2:00 pm.

You are to design, write, assemble, and simulate an assembly language program which will generate sum of integer squares numbers. Giving is an array NARR of byte-long numbers (with a $FF sentinel). Each element in the table corresponds to the end value N of the sum of squares to be generated (as defined in Lab3). The actual calculation of the corresponding 4-byte sum of integer squares has to be implemented in a subroutine. The 4-byte sum has to be passed back to the main program, which stores it consecutively in the RESARR array.

PLEASE NOTE:

1. The complete sum calculation has to be done in the subroutine using the algorithm defined in Lab3, resulting in two nested loops, **inside a single subroutine**. Using any other algorithm, implementation, multiplication, or the MUL instruction instead is NOT allowed.
2. Your program should work for any table values, not just the ones given.
3. Your program is NOT allowed to change the numbers stored in NARR.
4. All multi-byte data items are in Big Endian format (including all program variables)
5. 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 memory locations variables are stored (i.e., $B000 for NARR and $B010 for RESARR). If you need to define additional variables, please add them in the appropriate places.
6. You are allowed to use parts of your LAB3 or parts of the official LAB3 solution.
7. You are allowed to **declare static variables** in your subroutine (through RMB, FCB, FDB).
8. Your subroutine should only have one exit point. This means that only **a single RTS instruction at the end of the subroutine is allowed**.
9. You must terminate your program correctly using the STOP instruction.
10. The main program must only have one exit point (i.e., only one STOP instruction at the end of the main program is allowed).
11. You do not have to optimize your program or algorithm for speed.
12. 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.
13. The main program should be a WHILE structure which goes through the NARR table and sends the value 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 4-byte result that has to be stored consecutively in the RESULT table.

    - 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 tables. Using the B-register as an array index and the ABX/ABY instructions add this index to the pointers will therefore not work.
- Your program should work with for any sentinel value, not just $FF.

14. For each iteration, the main program should take a number from the NARR table and pass it to the subroutine **in a register (call-by-value in register)**. The subroutine performs the sum calculation and produces a 4-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 four bytes from the stack and stores it in the RESULT array.

    - Make sure that your program will not generate a stack underflow or overflow.
    - ALL of the number processing must be done inside **a single subroutine** using the algorithm from Lab3.

15. Preparation of parameter passing back to the main program (i.e., opening hole on the stack for the parameter) needs to be done in the subroutine, not in the main program.

16. Any assembler or simulator error/warning messages appearing when assembling/simulating your submitted program will result in up to 25 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 2:00pm 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
- re-submitting an older lab: -100 points
- submitted program not related to lab assignment: -100 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 NARR elements: up to -25 points
- program does not work if arrays moved to different locations: -10
- program does not work for more than 255 table elements: -10
- program does not work with an empty table: -5 points
- program does not work for any sentinel != $FF: -5 points

Correct subroutine implementation

- multiple subroutines used: -10 points
- part of 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

Correct Parameter passing implemented

- main program variables accessed in subroutine (or vice versa): -20 points
- incorrect parameter passing to subroutine ("in register" not implemented): -10 points
- incorrect parameter passing to subroutine ("call-by-value" not implemented): -10 points
- incorrect parameter passing from subroutine ("over the stack" not implemented): -20 points
- incorrect parameter passing from subroutine ("call-by-value " not implemented): -10 points
- parameter passing hole on stack opened in main program: -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
- multiple STOP instructions used: -10 points
- program changes NARR elements: -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
- conditional branch used does not match pseudo-code condition: -5 points

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.