CS 3280
LAB Assignment #5
Due date: Friday, April 13, before 1:00 pm.

You are to design, write, assemble, and simulate an assembly language program which will generate Fibonacci sequence numbers. Giving is an array NARR of byte-long numbers (with a $FF sentinel). Each element in the table corresponds to the sequence number of a Fibonacci number to be generated. The actual calculation of the corresponding 4-byte Fibonacci numbers has to be implemented in a subroutine. The 4-byte Fibonacci numbers have to be passed back to the main program, which stores them in the RESARR array.

The difference between this lab and lab 4 is that the subroutine has to be TRANSPARENT to the main program and only local DYNAMIC variables implemented in registers or on the stack are allowed in the subroutine. Please see items 15, 16, and 17 below for further info.

Also, because of the more complex nature of this assignment, it is worth *200* points.

PLEASE NOTE:

1. Your program should work for any N value, not just the ones given in the table.
2. Do NOT use the X or Y registers for storing or manipulating DATA.
   Only use the X and Y registers for storing/manipulating ADDRESSES.
3. All multi-byte data items are in Big Endian format (including all program variables)
4. Your program is NOT allowed to change the numbers stored in the NARR table.
5. You have to use the program skeleton provided for Lab5. 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 ‘NARR’ and ‘RESARR’. Do NOT change the values assigned to the NARR table. If you need to define additional variables, please add them in the appropriate places.
6. Your subroutine should only have one exit point. This means that only a single RTS instruction at the end of the subroutine is allowed.
7. Initialize any additional variables that your program (main program and subroutine) needs within the program, NOT with a FCB or FDB in the data section
8. You must terminate your main program correctly using an infinite loop structure.
9. You do not have to optimize your algorithm or your assembly program for speed.
10. You have to provide a pseudo-code solution for your main program AND your subroutine. In your pseudo code, do NOT use a for loop, but either a while or a do-until construct to implement a loop. Also, do NOT use any “goto”, “break”, or “exit” statements in your pseudocode.
11. The structure of your assembly program should match the structure of your pseudo code 1-to-1.
12. You are allowed to use parts of your LAB4 or parts of the official LAB4 solution.
13. The main program should be a WHILE structure which goes through the NARR table and sends an N 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 table) 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 $00 is encountered. For each subroutine call, the subroutine will send back a 4-byte result that has to be stored consecutively in the RESARR array **in Big-Endian format**.

- 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 (NARR and RESARR) are moved to different places in memory (do not use any fixed offsets).
- 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.
- You don’t have to copy the sentinel to the end of the RESARR array.

14. For each iteration, the main program should take one number from the NARR table and pass it to the subroutine **in a REGISTER (call-by-value in register)**. The subroutine performs the calculation and produces the corresponding 4-byte Fibonacci number. The resulting 4-byte number must be passed back to the main program **OVER THE STACK (call-by-value over the stack) in Big Endian format**. The main program then retrieves the 4 bytes from the stack and stores them in the RESARR array in **Big Endian** format. **Thus, if the NARR table has 8 data items (excluding the sentinel), the RESARR array should consist of 32 bytes (8 4-byte Fibonacci numbers) after program execution.**

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

15. **The subroutine MUST BE transparent to the main program**. This means that **any registers used in the subroutine** (including the CC register and the register used for passing the current N value to the subroutine) must be pushed onto the stack at the beginning of the subroutine and pulled off the stack at the end of the subroutine.

16. **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 only (as shown in class) – it is NOT allowed to access these variables through PUSH/PULL operations. Also, it is NOT allowed to temporarily store register variables on the stack using PUSH/PULL operations or temporarily free registers by pushing them onto the stack. The only places **in your subroutine** where PUSH/PULL operations are allowed are at the beginning of the subroutine to store registers on the stack and at the end of the subroutine to retrieve the register contents to obtain a transparent subroutine and for passing the result back to the main program. Furthermore, **INS and DES instructions** are only to be used in the subroutine to open/close holes in the stack for the local variables and the return value. IF YOU HAVE A QUESTION AS TO WHETHER YOUR SUBROUTINE VIOLATES ANY OF THESE REQUIREMENTS, ASK THE INSTRUCTOR.

17. Your program will be using three separate stack regions: local variable region, transparency region, and parameter passing region. Make sure that these regions stay separate: **do not use the transparency or the parameter passing region for local subroutine variables** (e.g., do not use the parameter passing hole for a local subroutine variable).

18. You do not have to check for overflow when calculating the Fibonacci numbers.

19. Any assembler or simulator error/warning messages appearing when assembling/simulating your submitted program will result in 100 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 ANY GIVEN MAIN PROGRAM VARIABLES, 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.

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 1:00pm 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

Requirements	Point Value
Program must produce correct answers	100 pts
Program implements correct parameter passing	50 pts
Program must have good structure	50 pts
Total Points	**200 pts**
Penalties:	
Program does not assemble or is incomplete	-100 pts (No partial credit)
Any assembler or simulator error/warning messages	-100 pts (No partial credit)
No comments at all	-40 pts (No partial credit)
Wrong algorithm implemented	-100 pts (No partial credit)
Main program variables accessed directly in subroutine	-100 pts (No partial credit)
Insufficient program comments (including incomplete program header) or incorrect/incomplete pseudo code	-40 pts
Use of static variables in subroutine	-100 pts
Accessing dynamic variables in subroutine not using indexed addressing mode	-100 pts (No partial credit)
Subroutine not transparent to main program	-100 pts
Late Submission/Resubmission	-10% for up to 24 hours late

Please Note: Submitted programs that won't assemble, produce assembler or simulator warnings/errors, or are incomplete lose 100 points. Be sure to check/assemble/simulate your code one last time before you submit your assignment!!!!