14. Modify the code for `larg.s` presented in class to use the `loop` and/or `jcxz` opcodes.

15. Write a program that
   - Contains two unsigned integers in the `.data` section,
   - Pushes these two numbers onto the stack,
   - Reads them from the stack,
   - Adds them, and
   - Uses the result as the exit code from the program.

   What happens if the numbers are negative? What happens if the addition overflows?

16. Repeat the previous, but return the product.

17. Repeat the previous, but return the quotient; ignore the remainder.

18. Write an assembly language program that contains at least three function calls to the C standard library. Compile and run the code.

19. Write a C program that contains a function call. [The program is otherwise arbitrary.]
   (a) Compile and run the program.
   (b) Use `gcc` to obtain the assembly language source of the program.
   (c) Compile the assembly language program, and run it. Check that it runs in the same fashion as the original.
   (d) Debug the program, and stop the program before the function call.
      i. What is the program’s EIP?
      ii. What is the program’s ESP?
      iii. What is the program’s EBP?
      iv. What is the current state of the program stack?
      v. Identify all of the local variables for the function `main()` on the stack. Verify that they hold the correct values.
   (e) Debug the program, and stop the program immediately after the function call.
      i. What is the program’s EIP?
      ii. What is the program’s ESP?
      iii. What is the program’s EBP?
      iv. What is the current state of the program stack?
      v. Identify all of the local variables for the function on the stack. Verify that they hold the correct values.
      vi. Verify that the correct return address for the function has been placed on the stack.