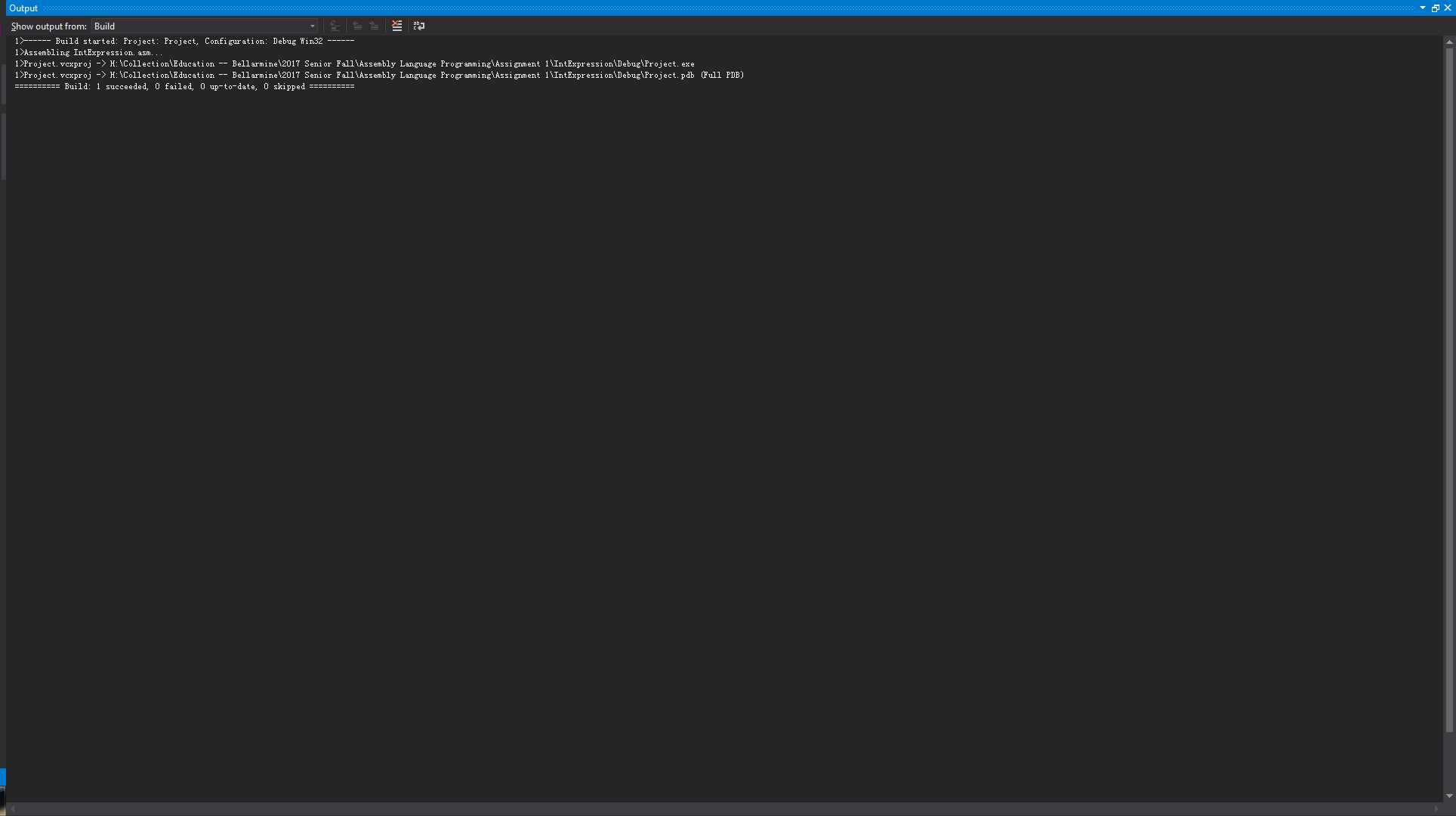
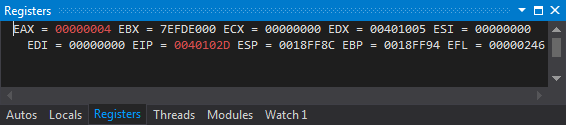
Chap 3 Programming Exercises Tyler Paulley

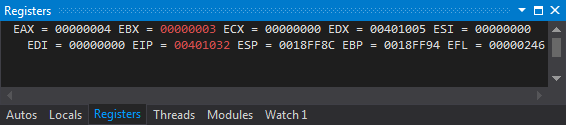
Exercise 1: Integer Expression Calculation

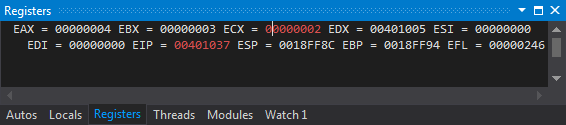
Output from build

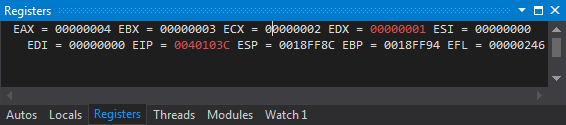


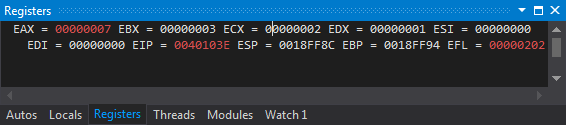
Register Values during Run

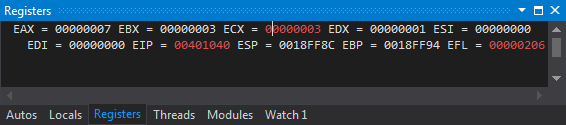


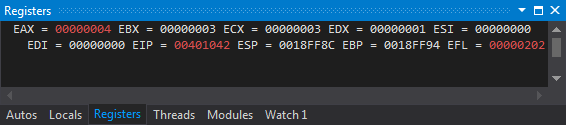






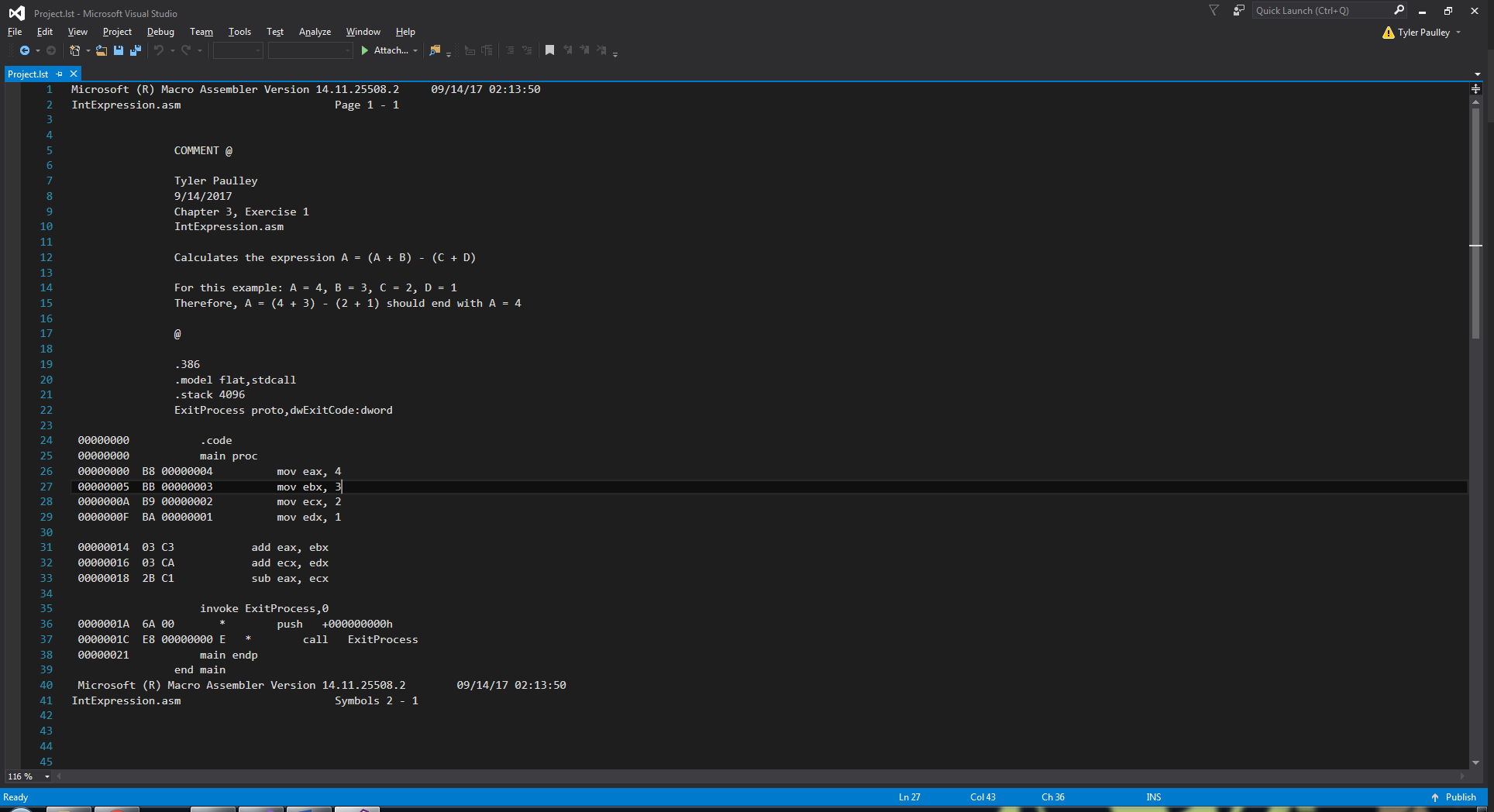






What I learned from the .lst file:

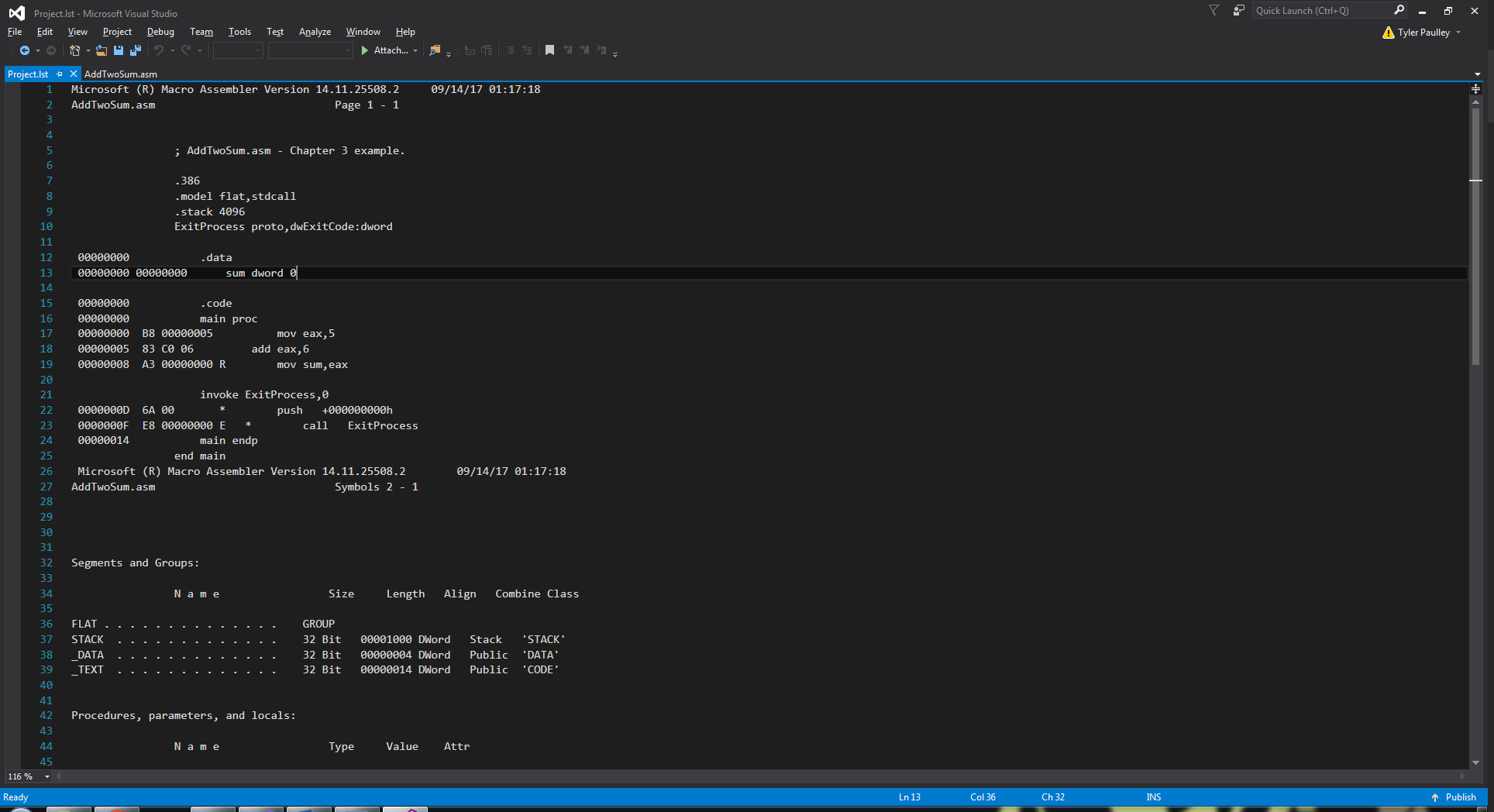
The .lst file contains the instructions in the source code, the memory location of each instruction, the opcode for each instruction, and the constant values used for each instruction. For example, the line:



States that the instruction is stored at location 00000005H, that the instruction for moving a constant to ebx has the opcode BB, and that the immediate value (3) to store in the register has a 32-bit hex value represented by 00000003H. The memory location for this instruction is 00000005H since the previous instruction was 5 bytes long: 4 bytes for the immediate value and 1 byte for the opcode.

Exercise 5: Listing File for AddTwoSum

Here is an exerpt from the .lst file that shows the machine code for each instruction.



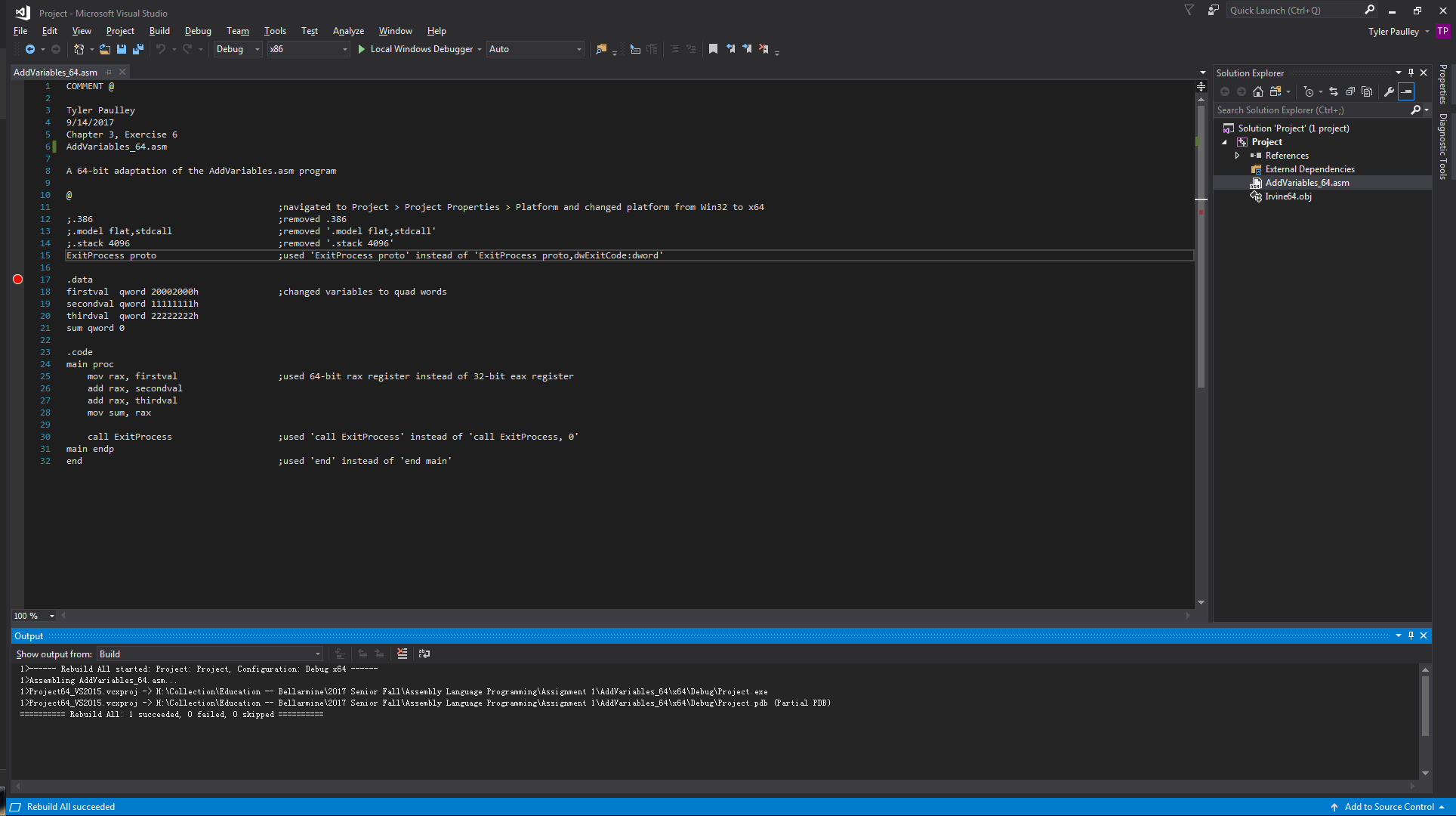
Description of Machine Code for each instruction:

As mentioned in the previous exercise, the first hexadecimal value in each line indicates the first byte in memory that the instruction is stored at, the second hex number is the 8-bit opcode, and the third hex number is the value to use in the instruction.

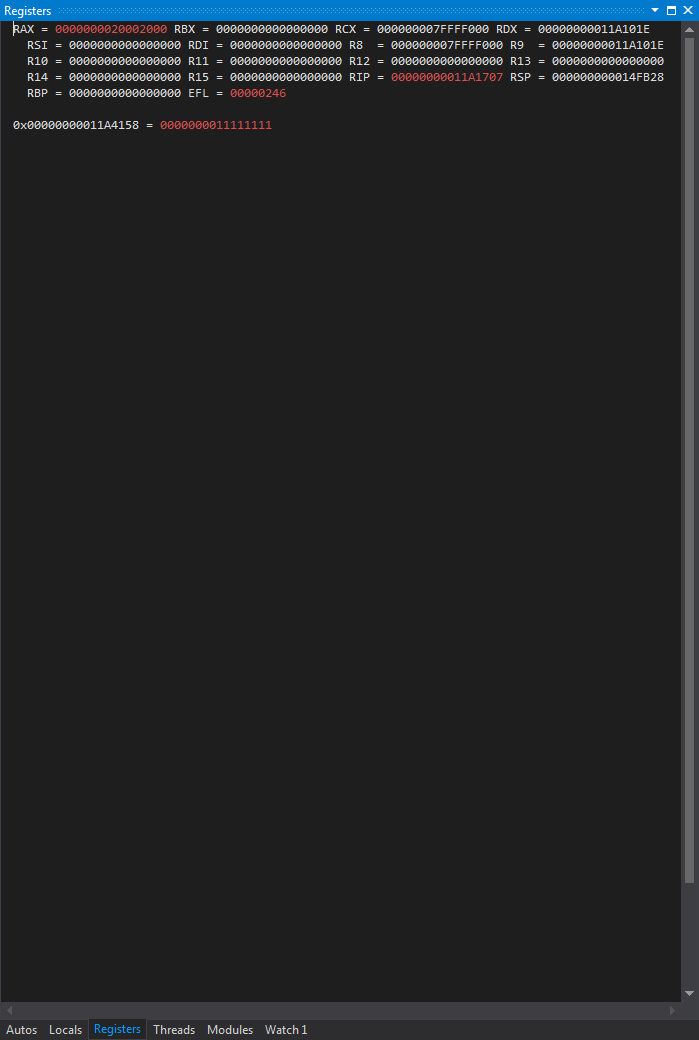
The add instruction seems to be an exception. On line 18, I believe the ‘83 C0’ means that this instruction has a 2 byte opcode instead of a 1 byte opcode such as for mov. It is worth noting that the immediate value of 6 in ‘add eax, 6’ is stored in 1 byte instead of 4 bytes such as in ‘mov eax, 5’. As a result, the offset of the next instruction is only 3 bytes higher from the location of ‘add eax, 6’ since 2 bytes are used for the opcode and 1 byte for the immediate value.

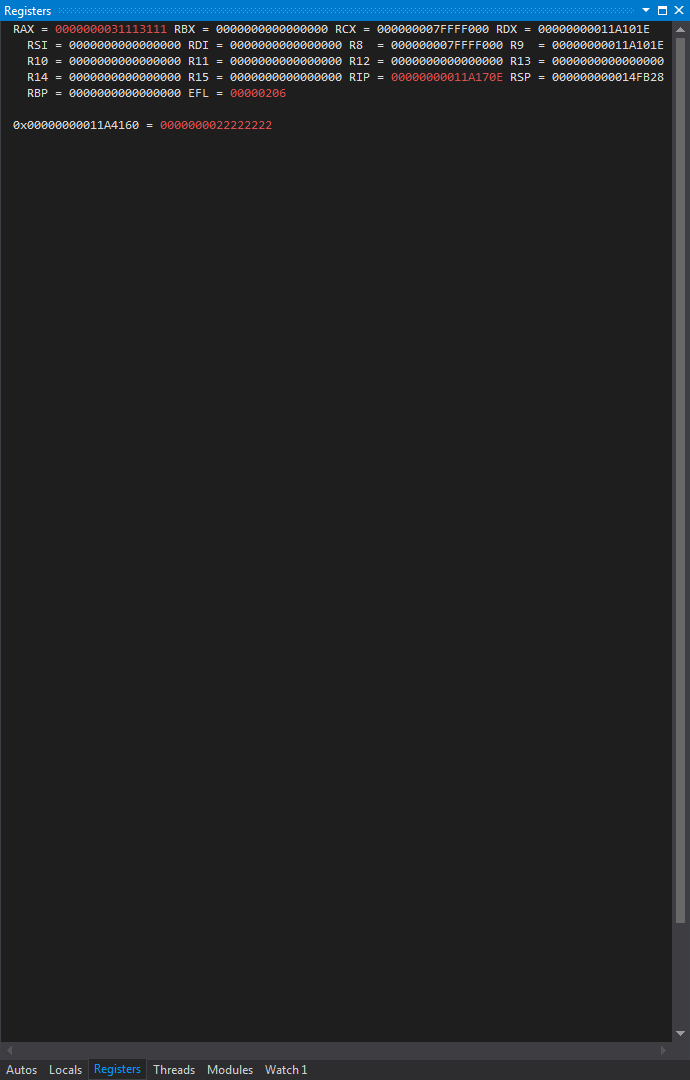
Exercise 6: AddVariables Program

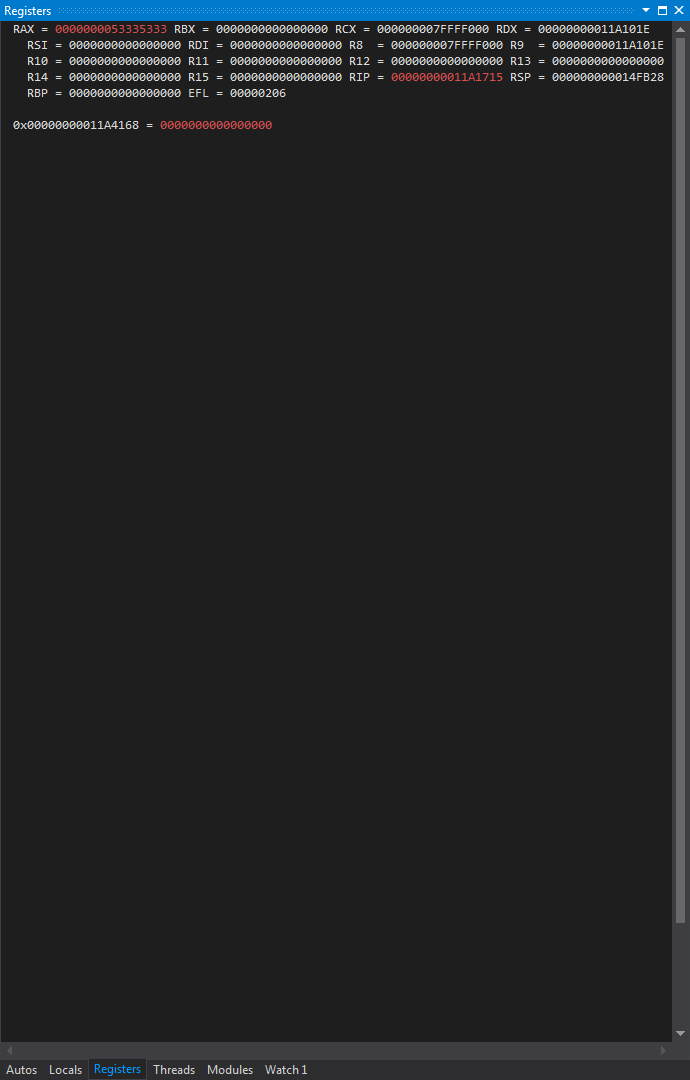
Output from build



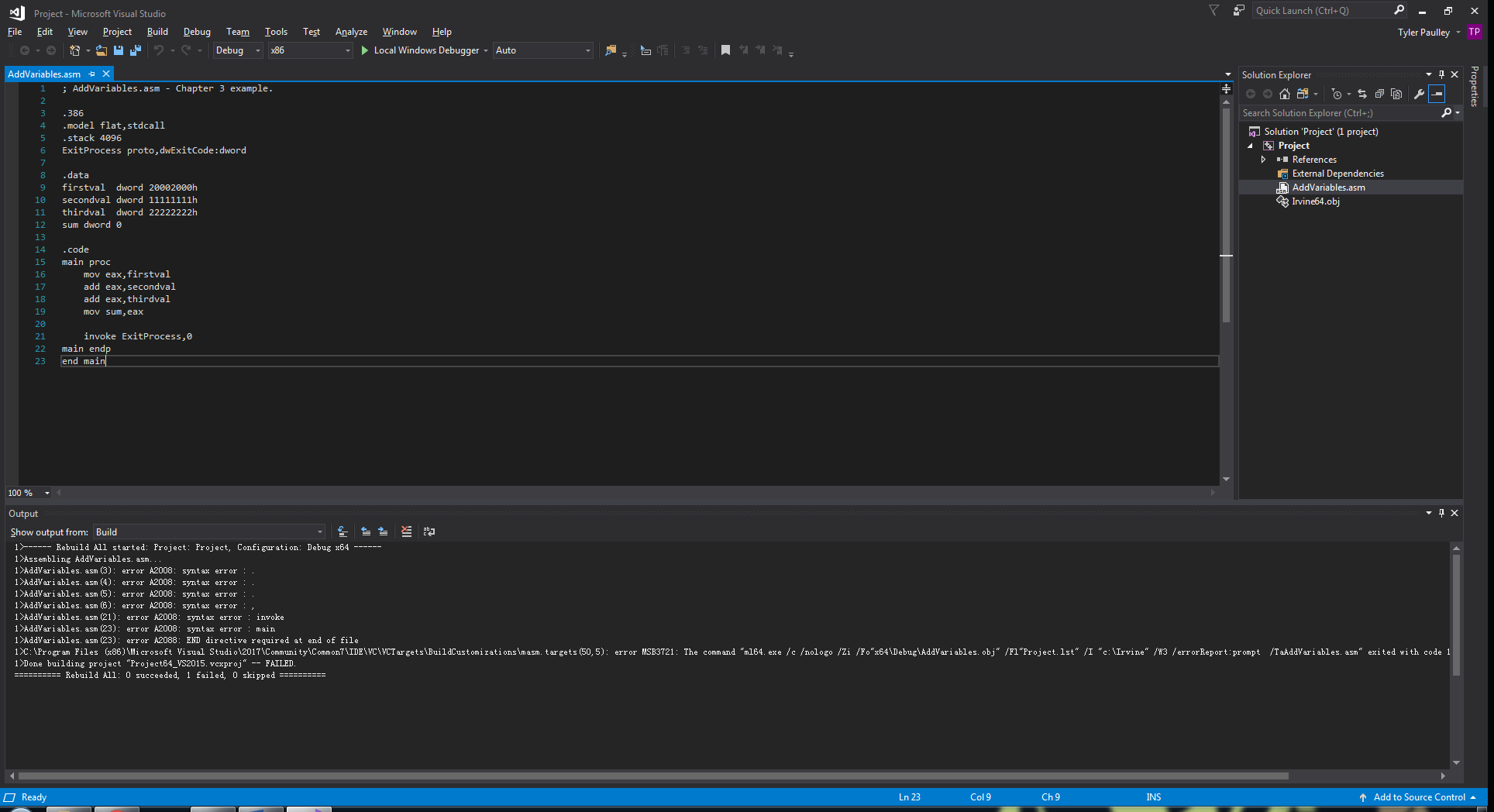
Register Values during Run







Syntax Errors generated by assembler (before changes to make program 64-bit):

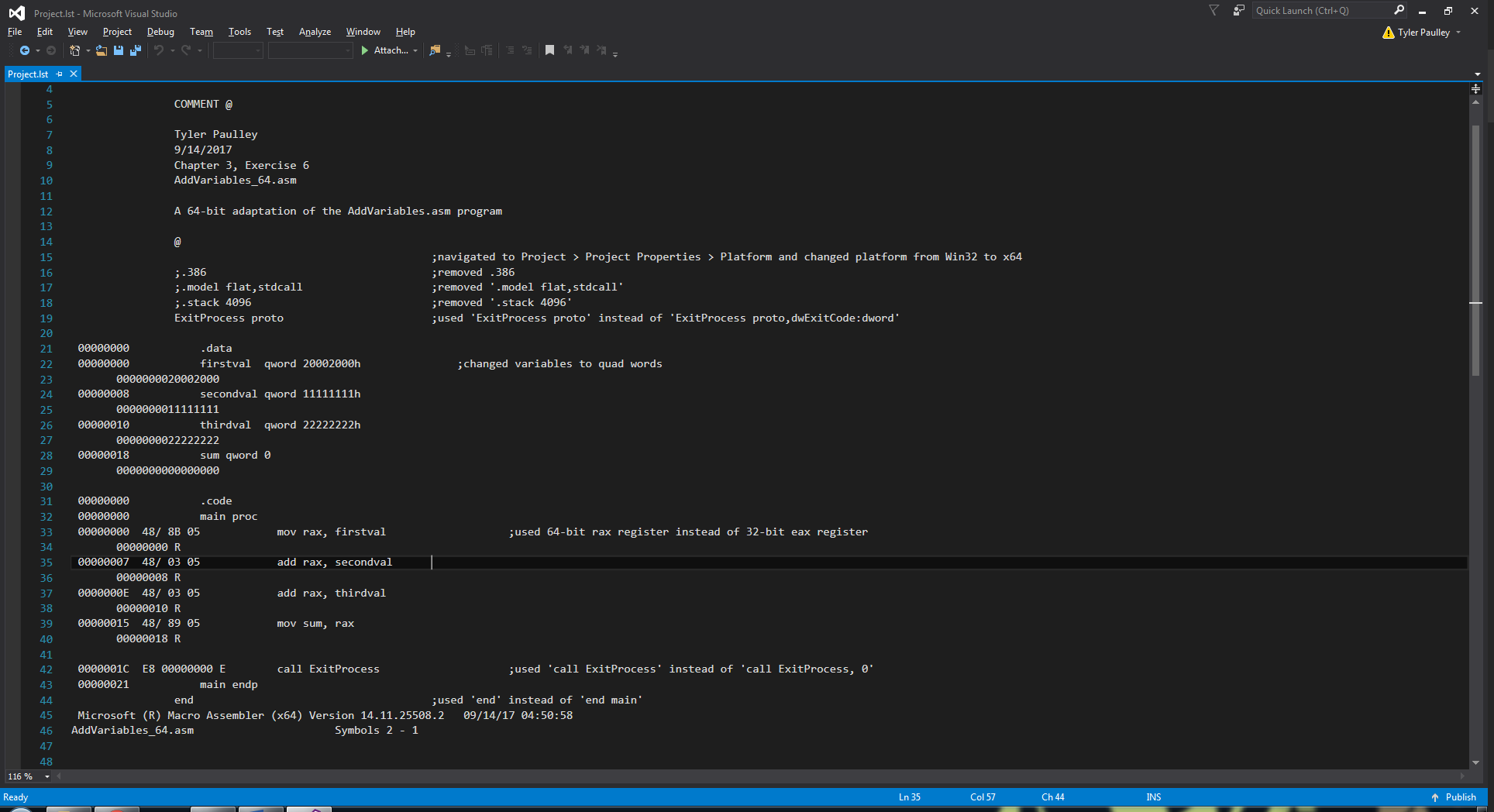


When I tried to build the 32-bit version of the AddVariables.asm program as a 64-bit program, the majority of the errors seemed to be syntactical. This is because many of the lines needed in the 32-bit program are not used in 64-bit programming. Each change I made to the file is listed in the AddVariable\_64.asm print out as comments. The changes can be summarized as:

* Changing doubleword variables to quadwords
* Using 64-bit registers instead of 32-bit registers
* Removing unnecessary lines such as ‘.386’ or ‘.model flat,stdcall’ or syntactical changes such as using 'ExitProcess proto' instead of 'ExitProcess proto,dwExitCode:dword'
* Navigating to the project platform settings and changing the value from Win32 to x64.

What I learned from list (.lst) file:

After looking at the list file for this program, I noticed that the memory locations for the instructions were farther apart. For example, instead of using 5 bytes for a mov instruction as in a 32-bit assembly instruction (4 bytes for the value and 1 byte for the opcode) the 64-bit mov used 7 bytes presumably since it is moving a value that uses more bytes of memory. Since the instruction is 7 bytes in length and not 9 bytes (64-bits for the value, 1 byte for the opcode) we can know that the instruction is not ‘true’ 64-bit since it uses less than 8 bytes for the value.



Looking at the instruction above, we also notice that there are more hex numbers in the opcode indicating that 64-bit opcodes may be longer than 1 byte. We also see 00000008 which seems to correspond to the memory location where the start of the value secondval is stored.