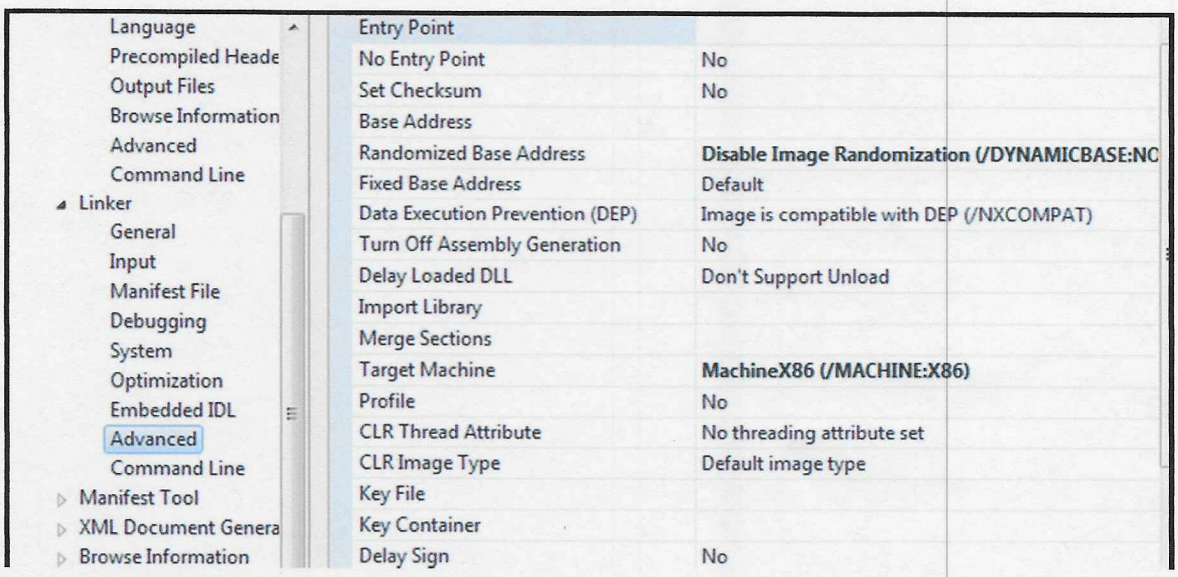
Ariel Guerrero

(5 pts) CS 3844 Computer Organization - Lab #03 Name/abc123: @aeg604
Due Fri, Sep 18th 2020 11:59 pm

This lab continues introducing you to the x86 Intel instruction set. Set up a new project in Visual Studio using the Lab3.cpp file. Create an Empty Windows Console application and "Add Existing" to make Lab3.cpp part of the project. In fact, you can reuse Lab2 as long as you rename the main from Lab #2.

Also we can disable address space randomization byte going to project properties and disabling it. This should make everyone's EIP the same.



Once you have the project compiled, go ahead and run it and observe what it does. Now, set breakpoints at the two "No Operation" (nop) instructions. Run the program in the Visual Studio debugger and set up the memory/register windows as before. You can also go to Debug\Windows\Disassembly if you wish to see the disassembled code intermixed with the C code.

1. What is the value of EIP? (It could still vary due to different compiler optimizations.) Whatever it is, it should be the same every time you re-run the program.

   EIP = 00434C2D

2. In the memory window, type gString and press enter. See the hexadecimal values for the ASCII characters?

   a. What is the hex value of a lowercase 'o'? 6F

   b. What is the address of gString? (May vary)

   0x004AE000

   c. Compare the address of gString with esp. What is the difference in values? (Use a calculator and subtract.) Are they close? Why or why not? Diff → 31E1AC is not close because of segmented memory.

   d. What is the address of localString? (May vary)

   localString = 0x0018FF24

   e. Is it close in value to esp? Yes, it is! Why is that? because the string is local and pushed onto the stack

f. Type gString in the memory window again. Click "step into" until the yellow arrow is pointing at the "lea edx,gString" instruction. Before looking, take a guess at the value of edx. What is the value of edx? edx has the value at 0x00 4A E0 00

edx = 276F7754

g. Where does that number come from?

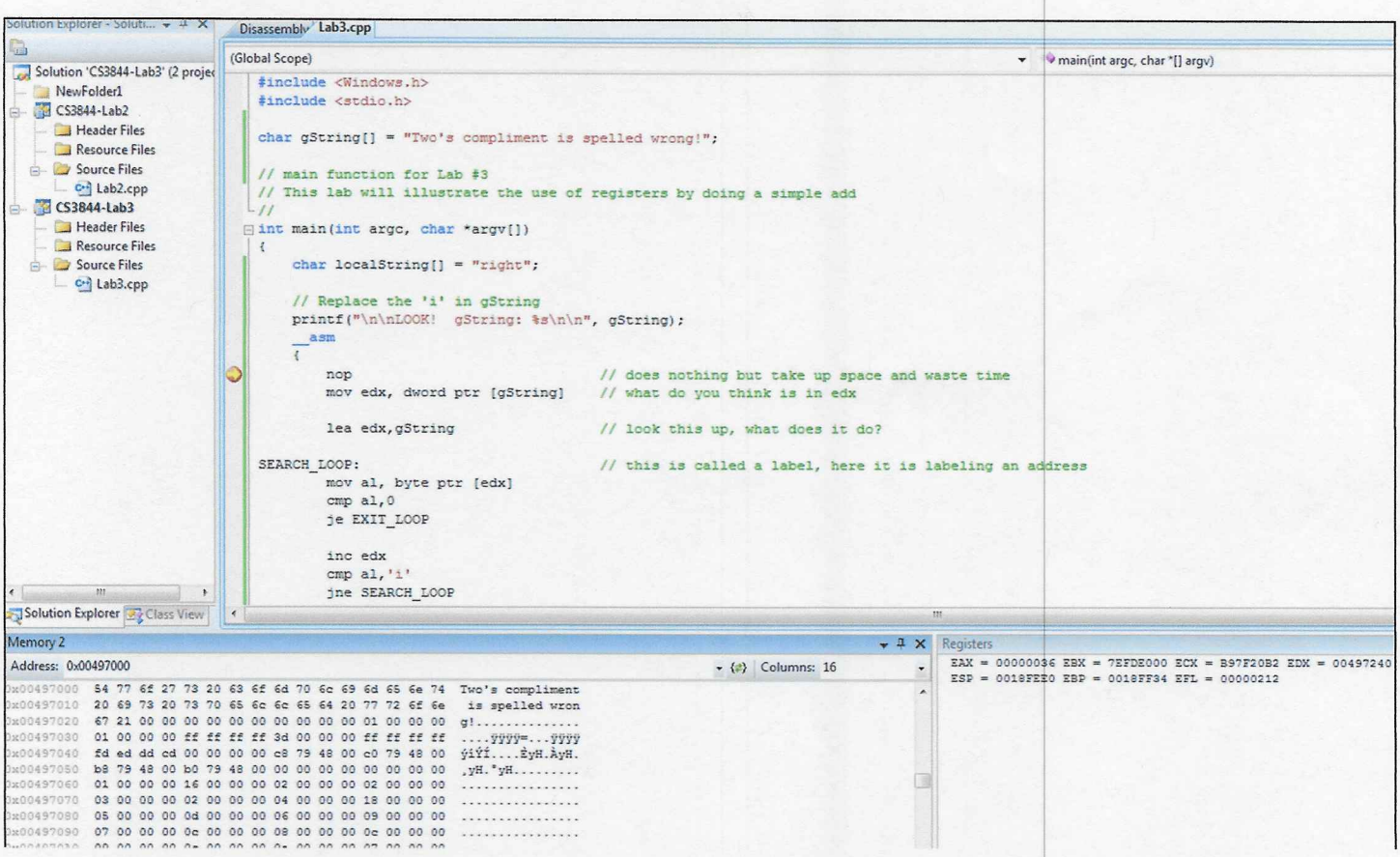
comes from gString

h. Why is the order of the hex digits in edx backwards from that in memory?

read in Little Endian.

i. Step to the next instruction inside SEARCH_LOOP. Look up the instruction you just executed (lea) to get an idea of what it does. What is in edx now? Put edx in the memory window, what do you see?

edx is Address of gString



Sample Screenshot

3. Step through the following code until you understand what it does. Then comment each line such that a novice computer person would be able to read and understand too. Stop when you get to SEARCH_LOOP2.

```
mov al, byte ptr [edx]
```
store byte (char) at edx in al

```
cmp al,0
```
Ex: BAD "check al for zero" GOOD: Check string for NULL

```
je EXIT_LOOP
```
if end of the string jump to exit loop

```
inc edx                    move over to next letter to check it.

cmp al,'i'                 check if al contains 'i'

jne SEARCH_LOOP            re loop if al does not equal 'i'

dec edx                    move back a char (because we inc edx) since the
                           current char is i

mov byte ptr [edx],'e'     replace i with e. i is at the first
                           byte at edX
```

4. Briefly describe the overall function of that piece of code. Look up any instructions that you don't know. You can Google something like "x86 intel dec" for example.

5. Inside SEARCH_LOOP2 you see the instruction "cmp byte ptr [edx],0." In a few sentences, describe what that instruction is doing. Don't just say, "It's comparing something to zero," but rather dig a little deeper. Consider how it accomplishes a comparison and why there is a "je" (jump if equal, i.e. jump if the zero flag is set to one) after it.

6. In the code you see the "byte ptr" notation – what do you think that does? If you changed it to WORD PTR, what register would you use in place of AL? And the same question for a DWORD PTR.

7. Briefly describe what the COPY_LOOP does. Step through the loop and determine under what conditions the jne doesn't loop back.

4.) Search loop goes through a string and replaces the first i with an e by storing its address in edx and cycling byte by byte looking for the 'i' until it is found

5.) Because after the inc we want to see if we are at the end of the string. It would be pointless to continue if we were.

6.) it means the byte and a certain location [b location]

Word → AX Dword → EAX

7.) Copy_loop replaces the word "wrong" with the word "right" in gString.

ecx = 5, because that's the length of the word.

we move char by char (byte by byte) starting at edx, which is where we found "w"
we replace the byte at edx with al
next we inc ebx to get the next char of the word "right"
and we inc edx to move to the next char to replace in the word "wrong".
then finally decrement ecx since we need 1 less char.

when ecx = 0 we are done and exit the loop.