Assembly Language and System Programming Lab4 Report

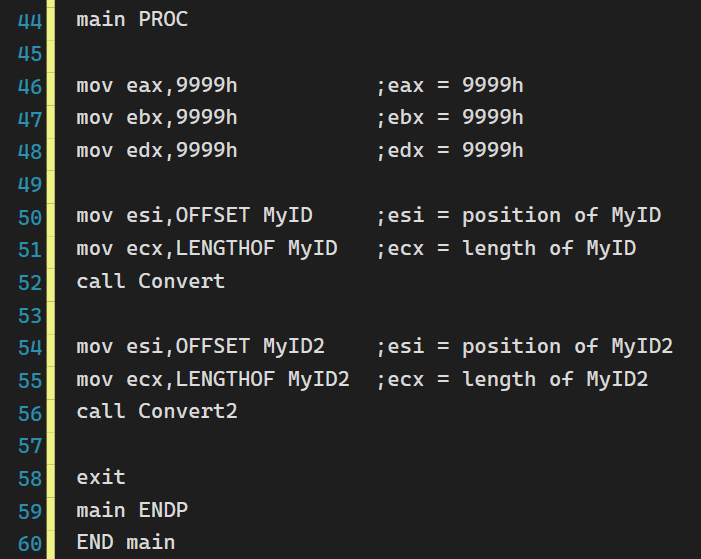
Group: 2

Names: 林柏廷(Brian)、蔡淵丞(Vincent)

Student ID: 110502531、110502567

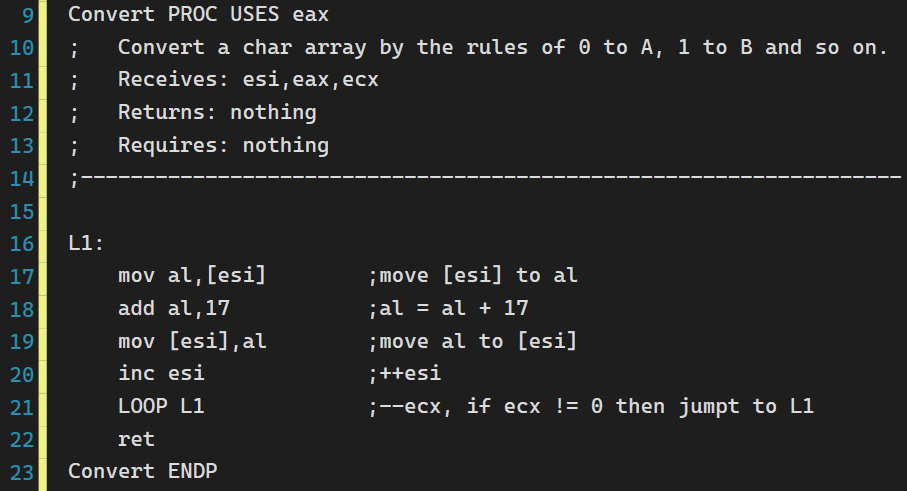
**Objective:** complete two PROC: CONVERT, CONVERT2. ( convert numbers to letters: 0->A, 1->B and so on…)

**MAIN PROC:**



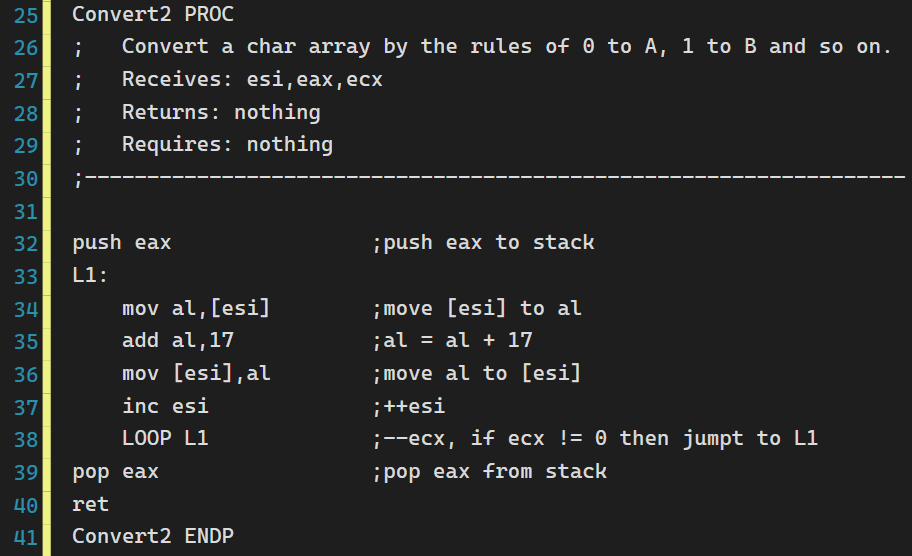
**CONVERT:**

Convert is a procedure that convert a char array by the rules of 0 to A, 1 to B and so on. To accomplish this, we first store the position of the array in ***esi***, and store the length of the array in ***ecx***. Convert also uses an operator **USES** since we want ***eax*** remain unchanged after the procedure call. We use a loop to iterate through the array, as the following code shows.



**CONVERT2:**

Instead of using USES, we directly use push and pop. This has the same effect as Convert.



**LOOP(L1):**

* Move the element MyID[***esi***] to ***al***.
* Increase the value of ***al*** by 17(the difference between ‘A’ and ‘0’ in ASCII), which completes the conversion.
* Move the increased number back to MyID[***esi***]***.***
* Increase ***esi*** by 1, makes it point to next element.
* Subtract ***ecx*** by 1, meaning that we have iterated once. If ***ecx*** is not equal to 0 then jump to L1, otherwise, continue the program.
* The loop in Convert2 is the same as Convert.

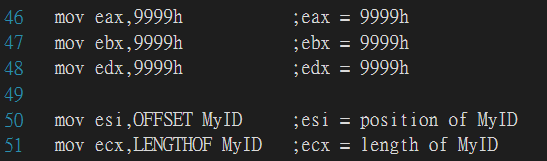
**Memory and registers before calling Convert (line 46-51):**

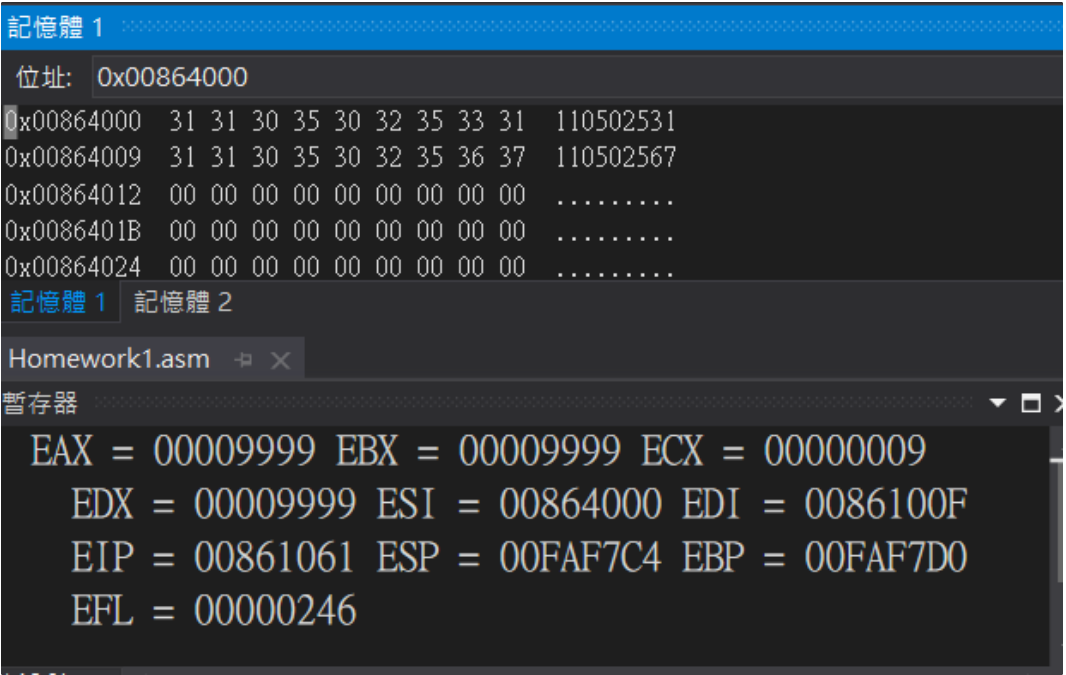
***Eax, ebx, edx:***Some value we stored.

***Esi***: the position of MyID.

***Ecx:*** thelength of MyID.

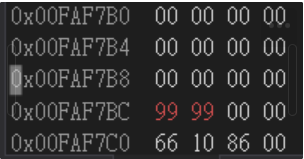
***Esp***: currently empty.





**Stack status after entering Convert (before L1):**

The address of the next instruction is pushed into the stack after the instruction “call Convert”. ***Eax*** is also pushed into the stack because of the instruction “USES ***eax***”.

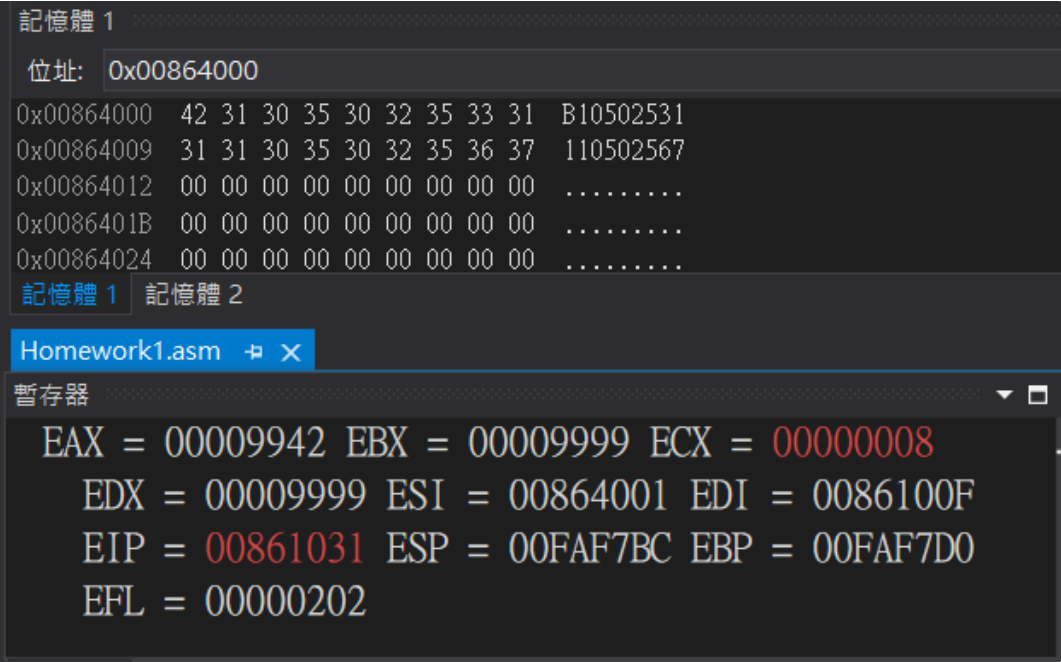
***esp*** before L1

**Memory and registers after first iteration:**

First character of the array is changed from ‘1’ to ‘B’ as required.

***Ecx*** decreased by 1 since we’ve iterated once.

***Al*** changed to 42(hex), which is the ASCII code of ‘B’.



**Memory and registers after 9 iterations (jumped out of Convert):**

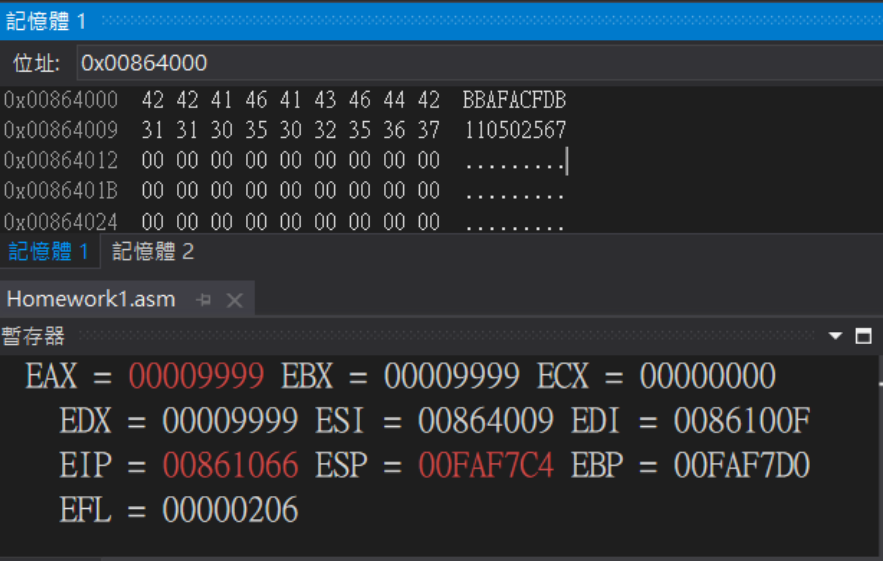
All of the characters in the array are converted as required.

***Eax*** same as initial state since we uses the operator USES.

***Eip*** points to the instruction previously stored in the stack.

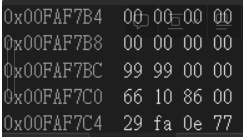
***Esp*** increased by 8 since we popped two elements out of the stack.

***Ecx*** decreased to zero since we’ve iterated nine times.



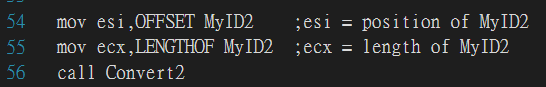
**Stack status after returning from Convert:**

No longer contains ***eax*** and the instruction we stored.

***esp***

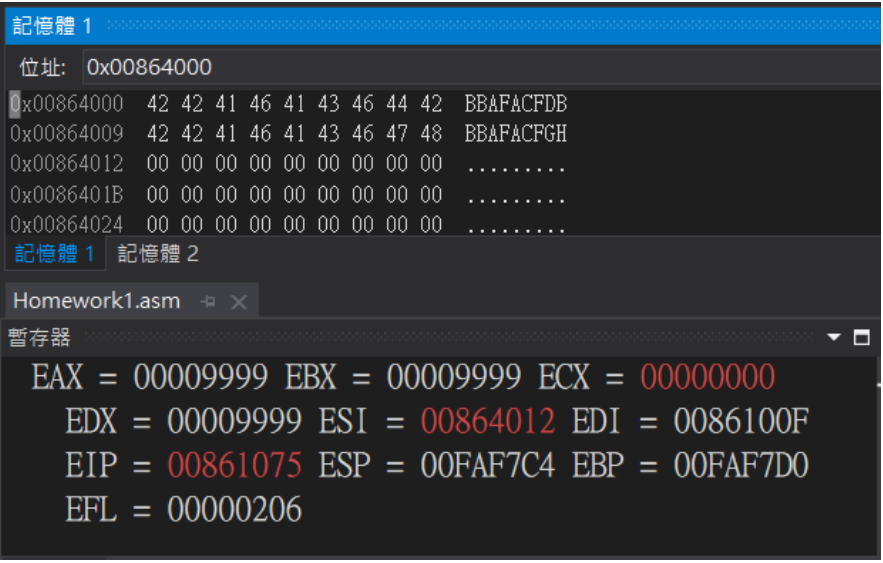
**Calling Convert2(line 54-56):**

Convert2 does the same thing as Convert, except this time we change the elements in MyID2 instead of MyID. Also, we explicitly push and pop ***eax*** from the stack, instead of using USES operator and let the assembler do the job.



**The memory and registers after calling Convert2:**

Both MyID and MyID2 are converted by the rules of ‘0’ to ‘A’, ‘1’ to ‘B’and so on. We’ve made it!



**Review:**

Working through the lab, we not only practiced the usage of procedure but also understand how the operator USES works. By inspecting the data stored in the stack, we’ve fully understood what actually happened after popping and pushing data to the stack. However, it’s a bit difficult to describe all of the things happening before and after calling the procedures.