

Assembly Programming - Addressing, Branching and Looping

Objectives.

- To explain how addressing works in assembly programming.
- To construct arrays.
- To create control and decision structures for branching.
- To create repeatable code blocks.

Aim

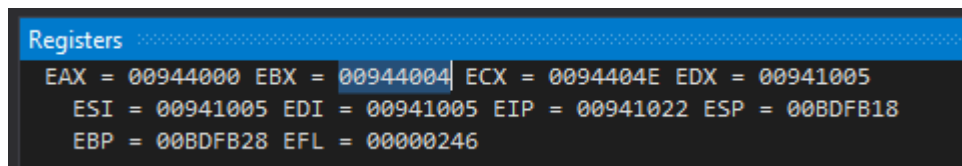
The aim of this session is to learn how to write assembly programs using arrays and loops

Tasks.

Inspecting memory.

In this task you will examine how variables and arrays are stored into main memory. Download the 'example1.asm' file. In this example, we define two arrays and one variable. These are defined in the '**.data**' segment. To see where these arrays/variables are stored in memory, we need to follow the steps below.

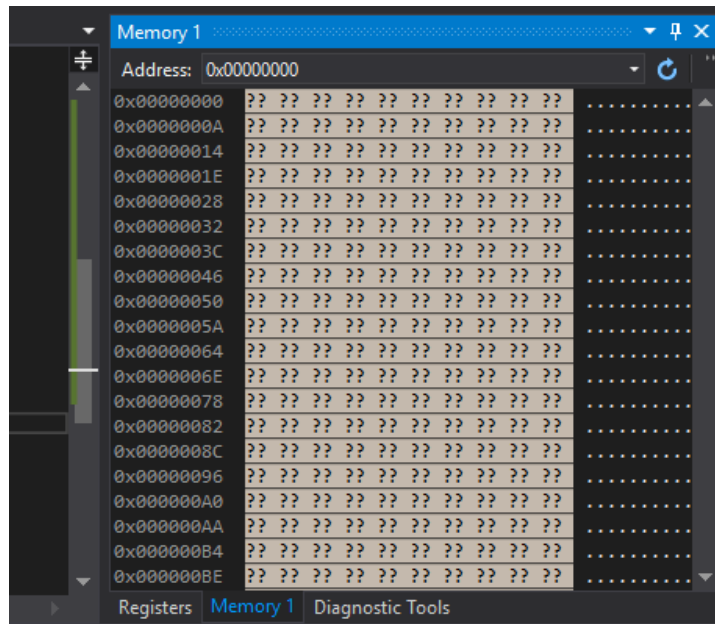
- (this is done already)* Load the address of the variable/array. To this end, 'lea' instruction is used (load effective address)
- Insert a breakpoint into the first lea instruction. Click start debugging. Now the program has started and stopped just before this instruction. Click on the 'step into' option or F11 to run this instruction.
- In the Register view, you should now see the memory addresses of the first array.



```
Registers
EAX = 00944000 EBX = 00944004 ECX = 0094404E EDX = 00941005
ESI = 00941005 EDI = 00941005 EIP = 00941022 ESP = 00BDFB18
EBP = 00BDFB28 EFL = 00000246
```

Please note that these addresses may be different every time you compile and run the program. This is because the addresses here were determined when we compiled the code.

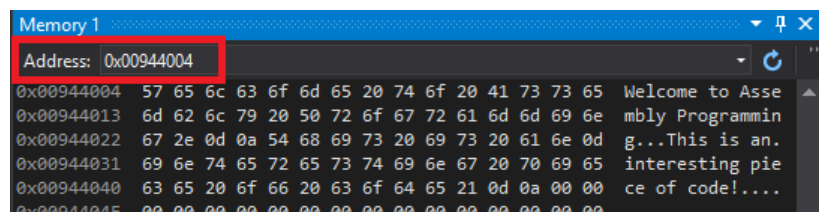
- Now, you can also see the Memory 1 view option beside Registers view option:



If you do not see this, then use the following menu options to enable this view:

Debug > Windows > Memory > Memory 1

- V. Now, the address for the first array is in the `eax` register. Copy and paste this address in the address textbox (shown in red below) in the Memory 1 view. Do not forget the `0x` offset. You should be able to see the array's contents now.



So far, we have seen the memory contents of the 1st array. Let's repeat the above process to see the contents of the second array and the contents of the variable. The contents of the 2nd array include zeros, why?

Task: Define the following hexadecimal array and see how its contents are being allocated in memory. Note that each hex digit is 1 byte. Revise the slide 'little endian vs Big Endian'.

My_array_2 DWORD 12345678h, 23456789h, 3456789Ah, 456789ABh

Playing with arrays.

In the lecture slides, we have seen how to declare, read and write an integer array. Now, do the following tasks.

Task1

Define an integer DWORD (32-bit) array with the elements [4, 3, 5, 6]. Read the second element (i.e. 3) into register ecx, and replace this with the value 10h (16 in decimal). Inspect the memory and confirm that the change has been made.

Solution: The array is defined as follows '*my_array DWORD 4, 3, 5, 6*'. When working with arrays, we need the starting memory address of the array. This can be found using '*lea ebx, my_array*' instruction; this will store the starting memory address of 'my_array' into 'ebx' register. Then to read the ith element we can use the following command:

```
lea ebx, my_array  
mov ecx, [ebx + TYPE array * ith.element]
```

The 'mov' instruction above will copy the contents of the memory address inside the brackets into ecx. This is different from '*mov ecx, whatever*'. '*mov ecx, whatever*' will store the value of 'whatever' to ecx, but '*mov ecx, [whatever]*' will store the contents of the memory address in 'whatever' to ecx. The '*[ebx + TYPE array * ith.element]*' contains the memory address of the array element we need to fetch. 'ebx' contains the starting memory address while 'TYPE' will give the size of each array element (e.g., 4 bytes). So, the memory address of the 2nd element will be '*[ebx + 4*2]*'. Remember the first element is the 0th element. The code follows

```
lea ebx, array ; move the address of the first element in ebx  
mov ecx, [ebx + TYPE array * 1] ; get the value stored at the 2nd element: ecx=3  
mov edx, 10h ; store 10h in edx  
mov [ebx + TYPE array * 1], edx ; put 10h into the second element of the array
```

Task2

Write a program that multiplies each element of a DWORD array by the value of 8 and stores the result in the same element of the array you defined. You are not supposed to use branching or looping here, so please follow the steps below.

- Create an array with the values 5, 10, 15, and 20.
- Load the first array element and multiply it by 8
- Store the result back into the first array element
- Repeat for the other 3 elements

The result array should contain 40, 80, 120, and 160.

Solution

The code that creates the array is '*array DWORD 5, 10, 15, 20*'.

The code that loads the first element, multiplies it by 8 and stores the result back to memory follows

lea ebx, array ; load the memory address of the first element

mov ecx, 8 ; load the multiplier

*mov eax, [ebx + TYPE array * 0]* ; load the multiplicand

mul ecx ; multiply, do $eax \times ecx$ and store the result into eax. The product fits in the 32bit eax register, so ignore the contents of the most significant register of the product

*mov [ebx + TYPE array * 0], eax* ; store the product into the 1st array's element

Try to repeat the above process for the next 3 array elements. The solution is provided on github.

Task3 - Branching and Loops.

Write a program that performs task2, but using a loop. See how a loop is implemented in assembly in the lecture slides. You must use a register to control the loop index ('esi' register).

Solution. The solution is explained line by line below.

```
.data ; data segment
    array DWORD 5, 10, 15, 20

.code ; code segment
main PROC ; main procedure
    ; write your assembly code here
    lea ebx, array ; load address of the first element
    mov ecx, 8 ; store the multiplier
    mov esi, 0 ; initialize the loop index
update: ; introduce a label for looping
    mov eax, [ebx + TYPE array * esi] ; store to eax the appropriate array
    element. If esi=0 store the 0th element, if esi=1 store the 1st element.
    Remember the first element is the 0th. The 0th element is stored into the starting
    memory address of the array (given by 'lea ebx, array'). The 1st element is plus 4
    bytes, the 2nd element plus 8 bytes etc
    mul ecx ; multiply the value of eax by ecx. The result will be in eax.
    Remember, in order to multiply two 32-bit values, the one must always be in eax
    register. See the table in the lecture slides.
    mov [ebx + TYPE array * esi], eax ; store the new value back to the array
    inc esi ; increment esi to show the next array element
    cmp esi, 4 ; compare to see if we have processed all the array elements
    jne update ; if not, go to the start of the loop

    INVOKE ExitProcess, 0 ; otherwise call exit function
```

Task4 (This task is optional – not assessed)

Consider a string with the contents "This is very exciting.". In this string, the number of times the letter 'i' appears is 4. Write a program that counts the number of times 'i' appears in the given string using branching and loops. You should write down the pseudo code first and then code accordingly.

Hint: To move a BYTE type data, you would need to use appropriate register. You may find the following webpage helpful.

https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture

Further Reading

Chapter 1 and Chapter 2 in 'Modern X86 Assembly Language Programming', available at <https://www.pdfdrive.com/download.pdf?id=185772000&h=3dfb070c1742f50b500f07a63a30c86a&u=cache&ext=pdf>