Lab Report

COEN311- LAB 3

Submitted to:
Amirreza Mousavi

Date performed: March 18th, 2021
Lab section: YL-X

By:

Bayan Alsalem
40105034

Objectives

- To learn the basic arithmetic operations (multiplication and addition)
- To write an x86 assembly language program which accesses the elements of a two-dimensional array of integers
- To learn about the two-dimensional array address translation formula and apply it

Theory

Two Dimensional array is an array that consists of more than one rows and more than one column. In 2-D array each element is refer by two indexes. Elements stored in these Arrays in the form of matrices. The first index shows a row of the matrix and the second index shows the column of the matrix.

Syntax of Two-Dimensional Array:

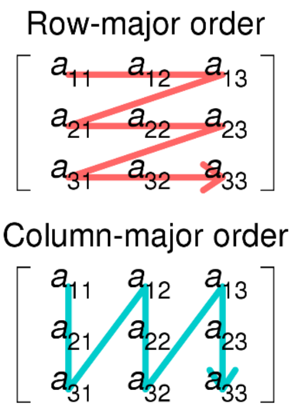
(Data type) (Name of array) [Number of rows] [Number of columns];

For example:

Int matrix [7] [7];

Nested loop is used to enter data in 2-D arrays. The outer loop acts as the number of rows of a matrix and the inner loop acts as the number of columns of a matrix. [1]

row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory. The difference between the orders lies in which elements of an array are contiguous in memory. In row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in column-major order. [2]

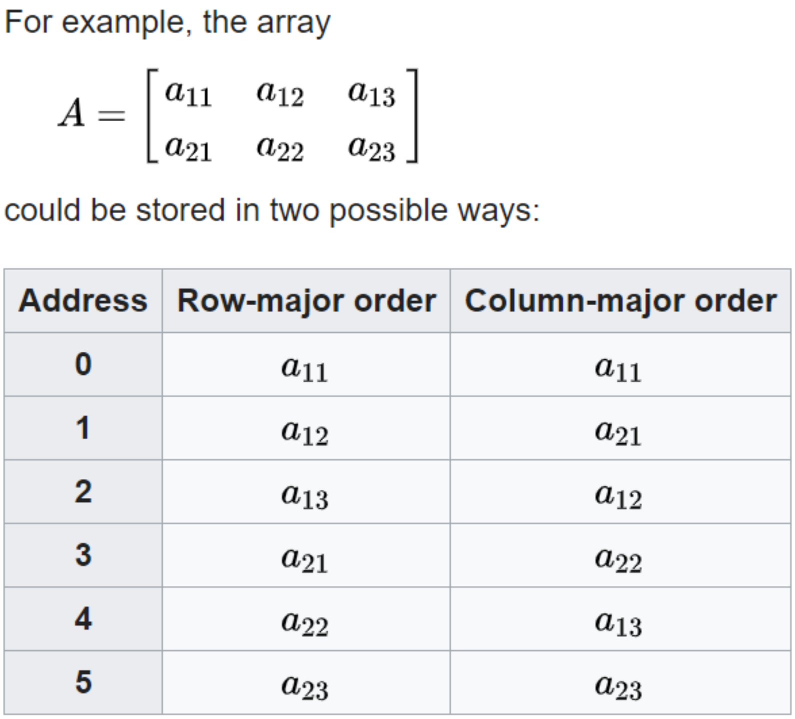


Row-major order

Column-major order

a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

[1] https://www.hellgeeks.com/two-dimensional-arrays/
[2] https://en.wikipedia.org/wiki/Row-_and_column-major_order



For example, the array

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

could be stored in two possible ways:

Address	Row-major order	Column-major order
0	a_{11}	a_{11}
1	a_{12}	a_{21}
2	a_{13}	a_{12}
3	a_{21}	a_{22}
4	a_{22}	a_{13}
5	a_{23}	a_{23}

The two-dimensional array address translation formula is:

Location of a[i][j] = x + (((i × # of columns) + j) × sizeof(data type))

where x is the starting address in main memory where the first array element is stored. In C++, the name of an array is synonymous with the starting address in main memory where the first element of the array is stored. Thus, the above formula can be succinctly stated as a[i][j] is found at address:

a + (((i × # of columns) + j) × sizeof(data type)) [3]

Conclusion

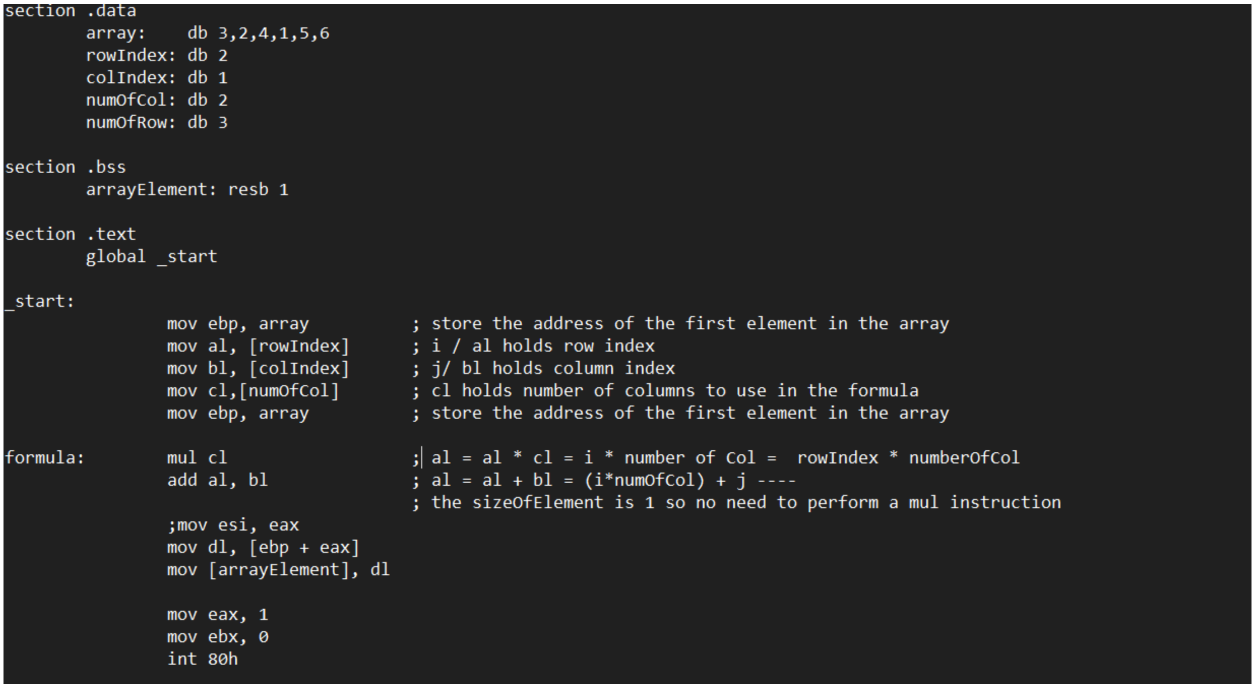
The two-dimensional array address transition formula was introduced and explained. We learned how to access an element of the array using this formula with intel 86 assembly language.

[3] COEN311 Lab manual

Appendix

- **The .asm text file containing the Intel x86 assembly language program**

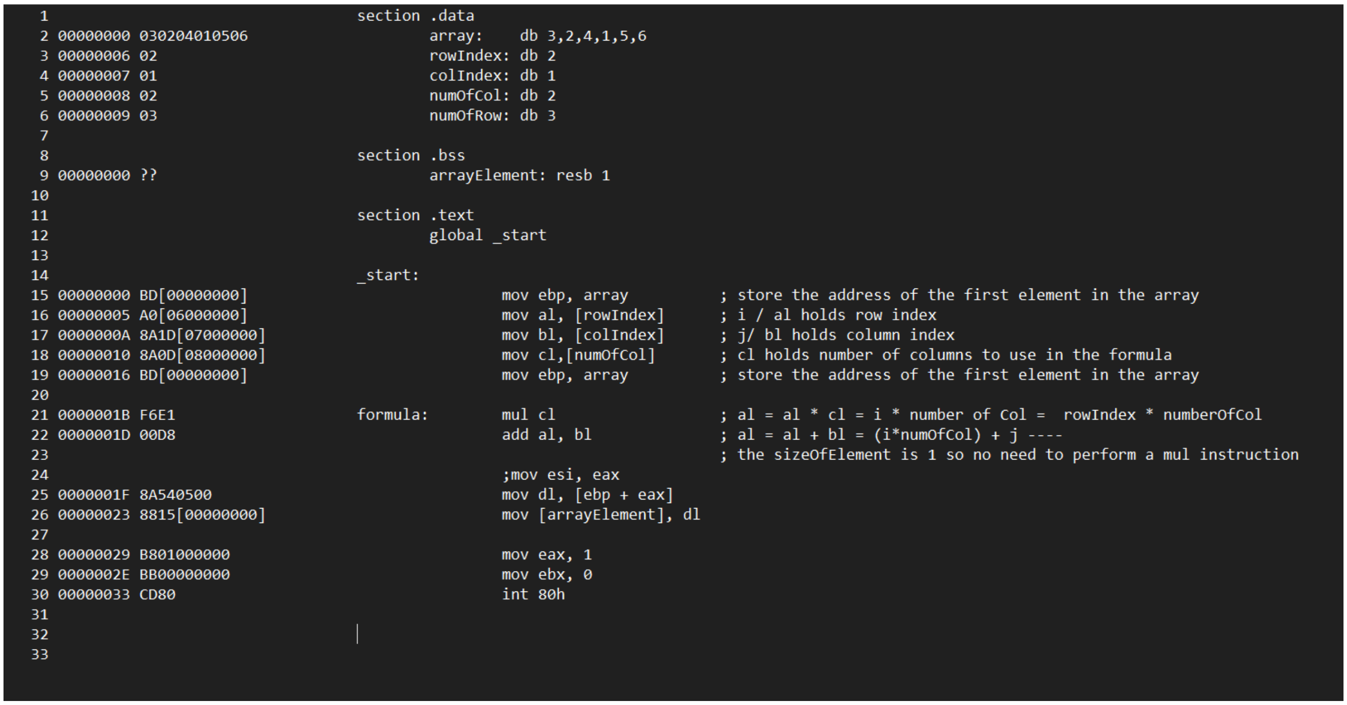
```
section .data
        array:    db 3,2,4,1,5,6
        rowIndex: db 2
        colIndex: db 1
        numOfCol: db 2
        numOfRow: db 3

section .bss
        arrayElement: resb 1

section .text
        global _start

_start:
                mov ebp, array          ; store the address of the first
element in the array
                mov al, [rowIndex]      ; i / al holds row index
                mov bl, [colIndex]      ; j/ bl holds column index
                mov cl,[numOfCol]       ; cl holds number of columns to use
in the formula
                mov ebp, array          ; store the address of the first
element in the array

formula:        mul cl                  ; al = al * cl = i * number of Col =
rowIndex * numberOfCol
                add al, bl              ; al = al + bl = (i*numOfCol) + j ---
-
                                        ; the sizeOfElement is 1 so no need
to perform a mul instruction
                ;mov esi, eax
                mov dl, [ebp + eax]
                mov [arrayElement], dl

                mov eax, 1
                mov ebx, 0
                int 80h
```



```
section .data
        array:    db 3,2,4,1,5,6
        rowIndex: db 2
        colIndex: db 1
        numOfCol: db 2
        numOfRow: db 3

section .bss
        arrayElement: resb 1

section .text
        global _start

_start:
                mov ebp, array          ; store the address of the first element in the array
                mov al, [rowIndex]      ; i / al holds row index
                mov bl, [colIndex]      ; j/ bl holds column index
                mov cl,[numOfCol]       ; cl holds number of columns to use in the formula
                mov ebp, array          ; store the address of the first element in the array

formula:        mul cl                  ; al = al * cl = i * number of Col =  rowIndex * numberOfCol
                add al, bl              ; al = al + bl = (i*numOfCol) + j ----
                                        ; the sizeOfElement is 1 so no need to perform a mul instruction
                ;mov esi, eax
                mov dl, [ebp + eax]
                mov [arrayElement], dl

                mov eax, 1
                mov ebx, 0
                int 80h
```

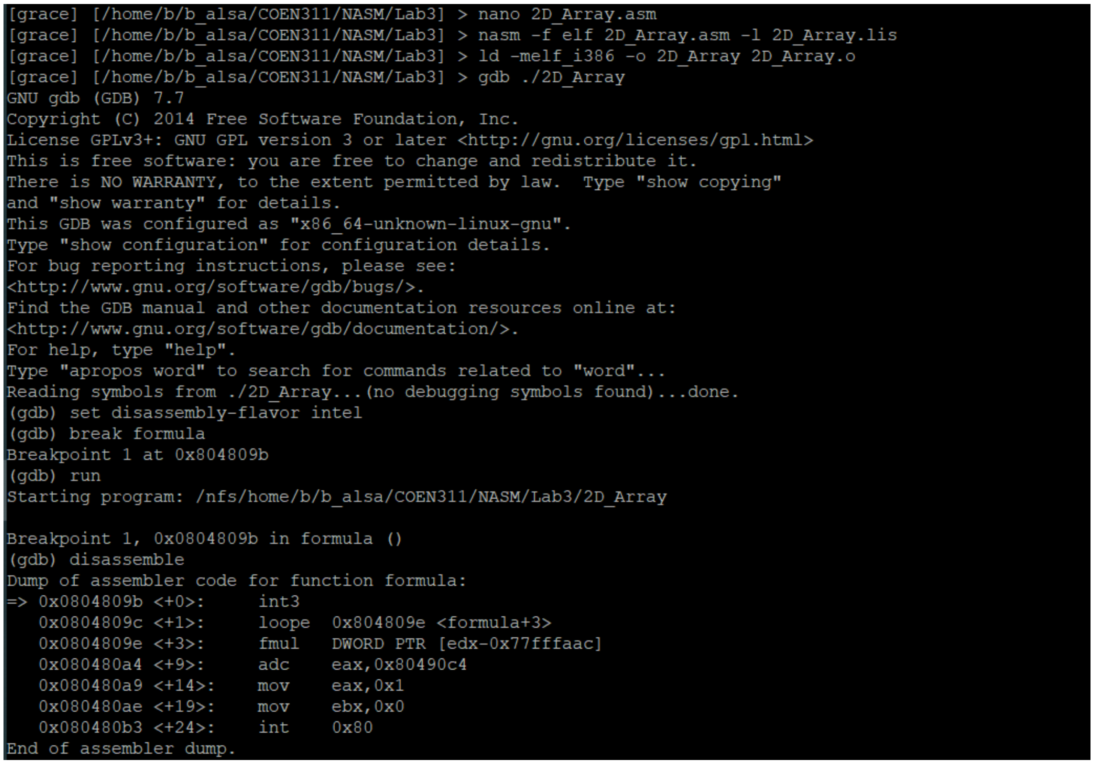
- **The corresponding listing file**

```
     1                                  section .data
     2 00000000 030204010506                array:    db 3,2,4,1,5,6
     3 00000006 02                          rowIndex: db 2
     4 00000007 01                          colIndex: db 1
     5 00000008 02                          numOfCol: db 2
     6 00000009 03                          numOfRow: db 3
     7                                  
     8                                  section .bss
     9 00000000 ??                          arrayElement: resb 1
    10                                  
    11                                  section .text
    12                                      global _start
    13                                  
    14                                  _start:
    15 00000000 BD[00000000]                         mov ebp, array
; store the address of the first element in the array
    16 00000005 A0[06000000]                         mov al, [rowIndex]
     ; i / al holds row index
    17 0000000A 8A1D[07000000]                       mov bl, [colIndex]
     ; j/ bl holds column index
    18 00000010 8A0D[08000000]                       mov cl,[numOfCol]
     ; cl holds number of columns to use in the formula
    19 00000016 BD[00000000]                         mov ebp, array
     ; store the address of the first element in the array
    20                                  
    21 0000001B F6E1                    formula:     mul cl
     ; al = al * cl = i * number of Col =  rowIndex * numberOfCol
    22 0000001D 00D8                                 add al, bl
     ; al = al + bl = (i*numOfCol) + j ----
    23                                  
     ; the sizeOfElement is 1 so no need to perform a mul instruction
    24                                               ;mov esi, eax
    25 0000001F 8A540500                             mov dl, [ebp + eax]
    26 00000023 8815[00000000]                       mov [arrayElement],
dl
    27                                  
    28 00000029 B801000000                           mov eax, 1
    29 0000002E BB00000000                           mov ebx, 0
    30 00000033 CD80                                 int 80h
    31                                  
    32                                  
    33
```

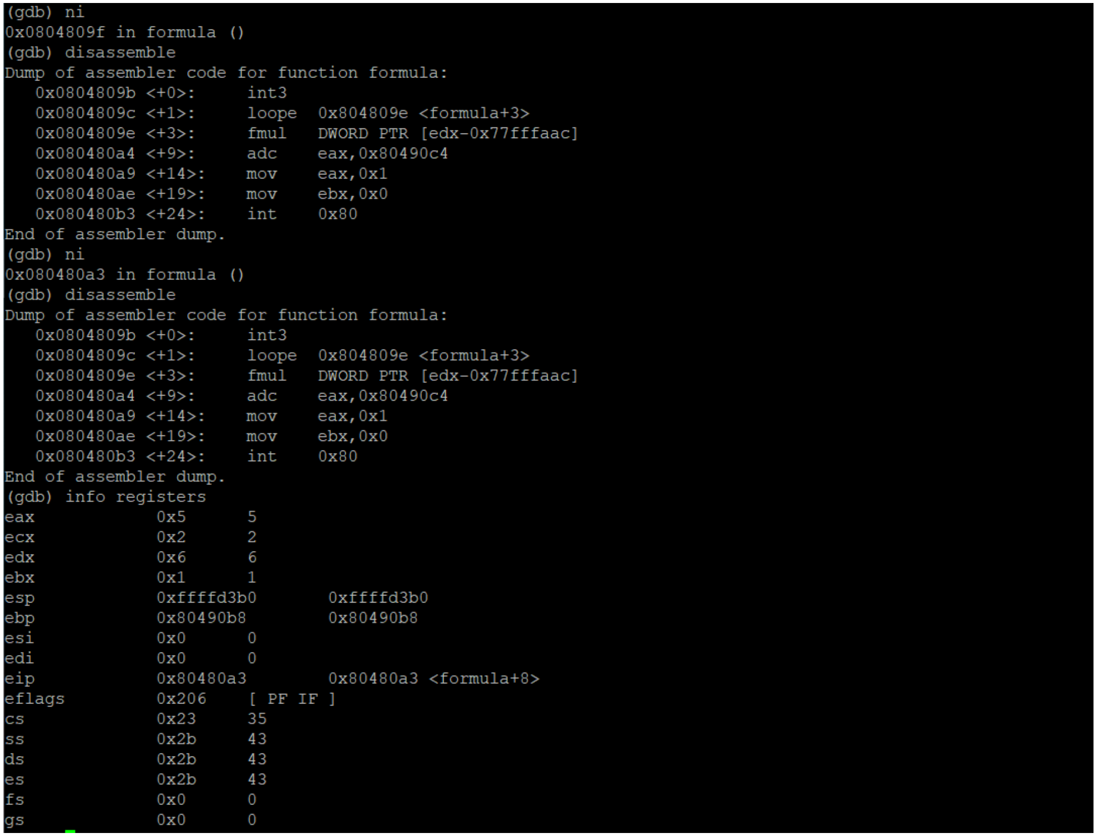


```
                                 section .data
00000000 030204010506                    array:    db 3,2,4,1,5,6
00000006 02                              rowIndex: db 2
00000007 01                              colIndex: db 1
00000008 02                              numOfCol: db 2
00000009 03                              numOfRow: db 3

                                 section .bss
00000000 ??                              arrayElement: resb 1

                                 section .text
                                         global _start

                                 _start:
00000000 BD[00000000]                            mov ebp, array          ; store the address of the first element in the array
00000005 A0[06000000]                            mov al, [rowIndex]      ; i / al holds row index
0000000A 8A1D[07000000]                          mov bl, [colIndex]      ; j/ bl holds column index
00000010 8A0D[08000000]                          mov cl,[numOfCol]       ; cl holds number of columns to use in the formula
00000016 BD[00000000]                            mov ebp, array          ; store the address of the first element in the array

0000001B F6E1                    formula:        mul cl                  ; al = al * cl = i * number of Col =  rowIndex * numberOfCol
0000001D 00D8                                    add al, bl              ; al = al + bl = (i*numOfCol) + j ----
                                                                         ; the sizeOfElement is 1 so no need to perform a mul instruction

                                                 ;mov esi, eax
0000001F 8A540500                                mov dl, [ebp + eax]
00000023 8815[00000000]                          mov [arrayElement], dl

00000029 B801000000                              mov eax, 1
0000002E BB00000000                              mov ebx, 0
00000033 CD80                                    int 80h
```

- **gdb debugging session**



```
[grace] [/home/b/b_alsa/COEN311/NASM/Lab3] > nano 2D_Array.asm
[grace] [/home/b/b_alsa/COEN311/NASM/Lab3] > nasm -f elf 2D_Array.asm -l 2D_Array.lis
[grace] [/home/b/b_alsa/COEN311/NASM/Lab3] > ld -melf_i386 -o 2D_Array 2D_Array.o
[grace] [/home/b/b_alsa/COEN311/NASM/Lab3] > gdb ./2D_Array
GNU gdb (GDB) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./2D_Array...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb) break formula
Breakpoint 1 at 0x804809b
(gdb) run
Starting program: /nfs/home/b/b_alsa/COEN311/NASM/Lab3/2D_Array

Breakpoint 1, 0x0804809b in formula ()
(gdb) disassemble
Dump of assembler code for function formula:
=> 0x0804809b <+0>:     int3
   0x0804809c <+1>:     loope  0x804809e <formula+3>
   0x0804809e <+3>:     fmul   DWORD PTR [edx-0x77fffaac]
   0x080480a4 <+9>:     adc    eax,0x80490c4
   0x080480a9 <+14>:    mov    eax,0x1
   0x080480ae <+19>:    mov    ebx,0x0
   0x080480b3 <+24>:    int    0x80
End of assembler dump.
```



```
(gdb) ni
0x0804809f in formula ()
(gdb) disassemble
Dump of assembler code for function formula:
   0x0804809b <+0>:     int3
   0x0804809c <+1>:     loope  0x804809e <formula+3>
   0x0804809e <+3>:     fmul   DWORD PTR [edx-0x77fffaac]
   0x080480a4 <+9>:     adc    eax,0x80490c4
   0x080480a9 <+14>:    mov    eax,0x1
   0x080480ae <+19>:    mov    ebx,0x0
   0x080480b3 <+24>:    int    0x80
End of assembler dump.
(gdb) ni
0x080480a3 in formula ()
(gdb) disassemble
Dump of assembler code for function formula:
   0x0804809b <+0>:     int3
   0x0804809c <+1>:     loope  0x804809e <formula+3>
   0x0804809e <+3>:     fmul   DWORD PTR [edx-0x77fffaac]
   0x080480a4 <+9>:     adc    eax,0x80490c4
   0x080480a9 <+14>:    mov    eax,0x1
   0x080480ae <+19>:    mov    ebx,0x0
   0x080480b3 <+24>:    int    0x80
End of assembler dump.
(gdb) info registers
eax            0x5      5
ecx            0x2      2
edx            0x6      6
ebx            0x1      1
esp            0xffffd3b0       0xffffd3b0
ebp            0x80490b8        0x80490b8
esi            0x0      0
edi            0x0      0
eip            0x80480a3        0x80480a3 <formula+8>
eflags         0x206    [ PF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x0      0
```