1) Being aware of the c02-01.asm

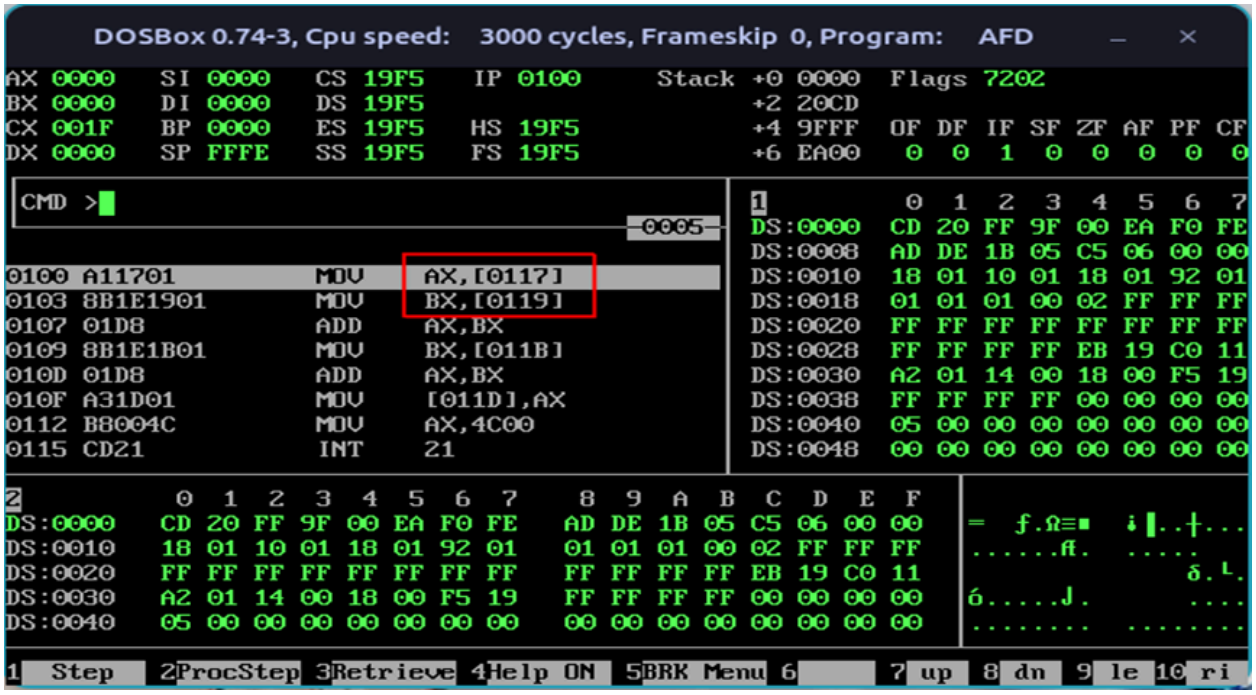
● First, use the previous commands we read in lab to run the file c02-01.asm into the dosbox.Following the file's execution, the Dosbox emulator interface appears on our screen. At this point, we evaluate any modifications made from our earlier codes and highlight them.

● To see the code and the modifications from the preceding code, we first open c02-01.asm.

```
; a program to add three numbers using memory variables
[org 0x0100]

    mov  ax, [num1]          ; load first number in ax
    ; mov  [num1], [num2]      ; illegal
    mov  bx, [num2]
    add  ax, bx
    mov  bx, [num3]
    add  ax, bx
    mov  [num4], ax
    mov  ax, 0x4c00
    int  0x21


num1: dw   5
num2: dw   10
num3: dw   15
num4: dw   0
```

● This code demonstrates how we move the value in Axe using square brackets; this technique is known as sending value utilising the labels. The value of num1 in the brackets is derived from the label at the conclusion of the code, which indicates that the value is 5.

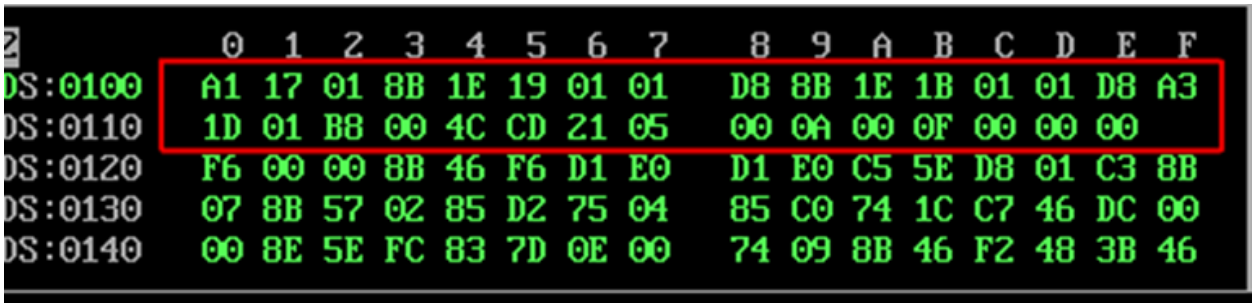
● Now test this code into the debugger and check the changes.



● As we can see, there is an object in the brackets that is the num1 value's address. It takes that value from the end of the address, which is stored in memory, places it there (between the braces), and then moves it to the Axe register.

*That's the only thing that separates the new one from our earlier ones. Instead of using the addressing approach like we did in the past, we shift direct value into the Axe.

● Now, we can observe how much memory this code uses. Count starting from the right side and confirm using the file's attributes.



● same 31 bytes this code consumes in the memory, count that values from 0100 to selected region they equal to 31 also.

2) Being aware of the c02-02.asm:

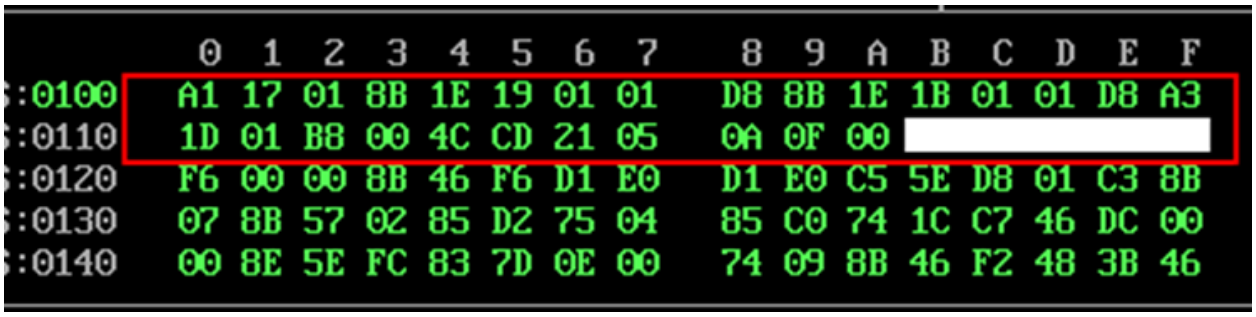
● Let's open the code file now and start by identifying the differences.

Thus, we can observe the modification in the final lines where the values for num1 were given in the earlier code.

```
; a program to add three numbers accessed using a single label
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]     ; notice how we can do arithmetic here
    add  ax, bx             ; also, why +2 and not +1?
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax     ; store sum at num1+6
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5
        dw  10
        dw  15
        dw  0
```

As you can see, we used to use three or four variables for this, but now we only use one variable that has a range of values, functioning similarly to an array.
●However, the difficulty is how to feed numbers like 10, 15, thus using the same technique, we do it in an array, making leaps based on the bit widths. As you can see, the word "dw" is written with values that define it. We should be aware that this word requires two bytes of memory. Therefore, we always leap by two bytes for the subsequent value. Launch it in the debugger to verify.

Since dw was also used in the earlier code and I changed it to db, you can see that there is no difference.



● Observe the distinction The values are now set to 1 bit.Naturally, this will have an impact on memory capacity, resulting in 27 bytes instead of 31 bytes. Let's see

"Observe the dimensions"

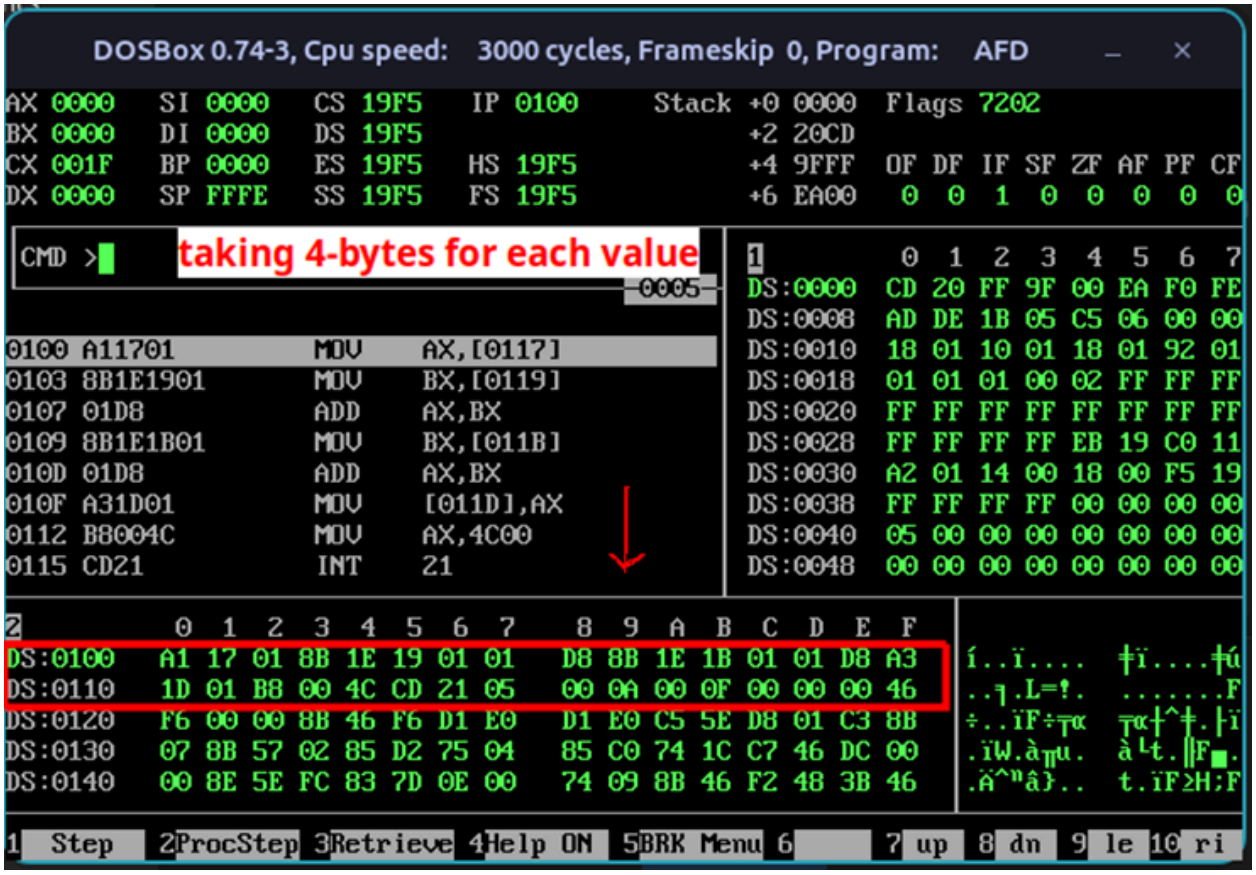
3) Understanding the c02-03.asm:

- Again lets check the new file

```
; a program to add three numbers accessed
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]
    add  ax, bx
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5,  10,  15,  0
```

● As of right now, there is a slight variation in that every value is of the same data type. Thus, we may quickly retrieve the appropriate values.

● Launch the dosbox debugger and examine it.



● As a result, dw causes the stored values in memory to now require two bytes.

4) Being Aware of the c02-04.asm

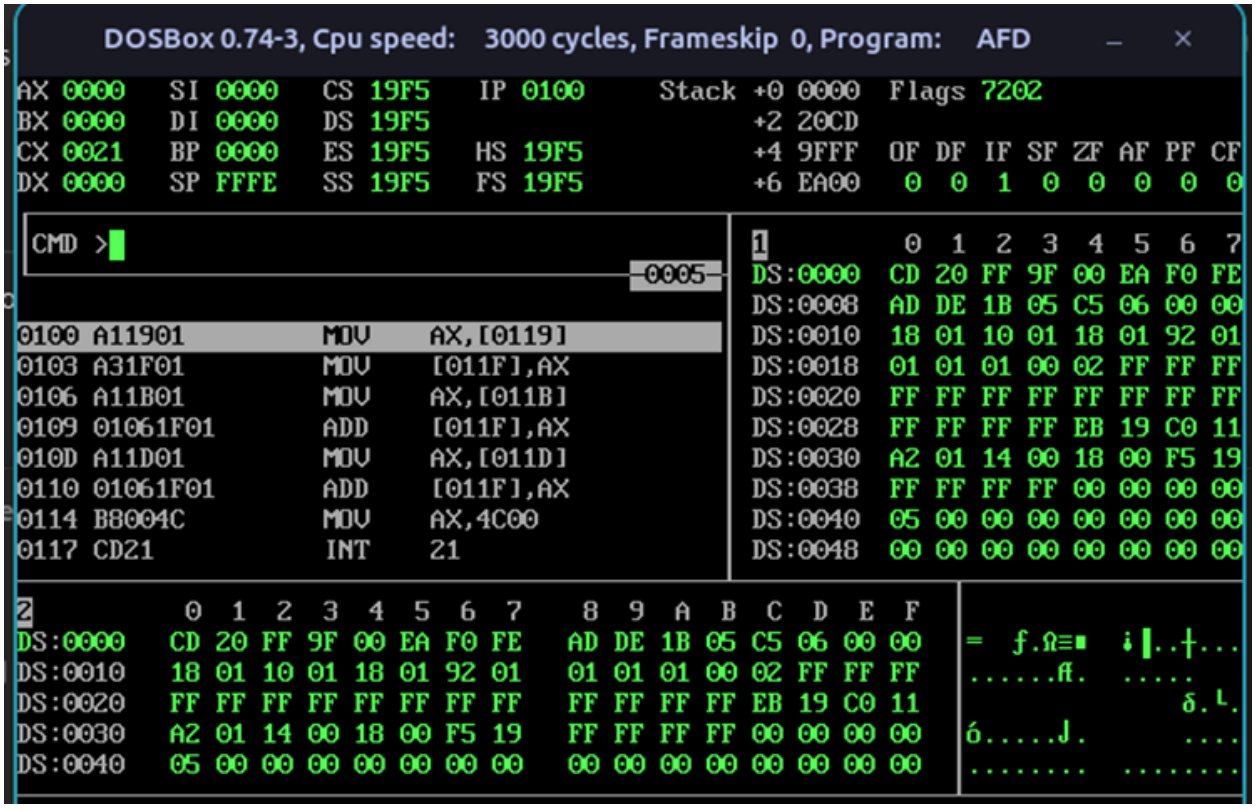
● Open that file first, then look for any changes from the preceding one's code.

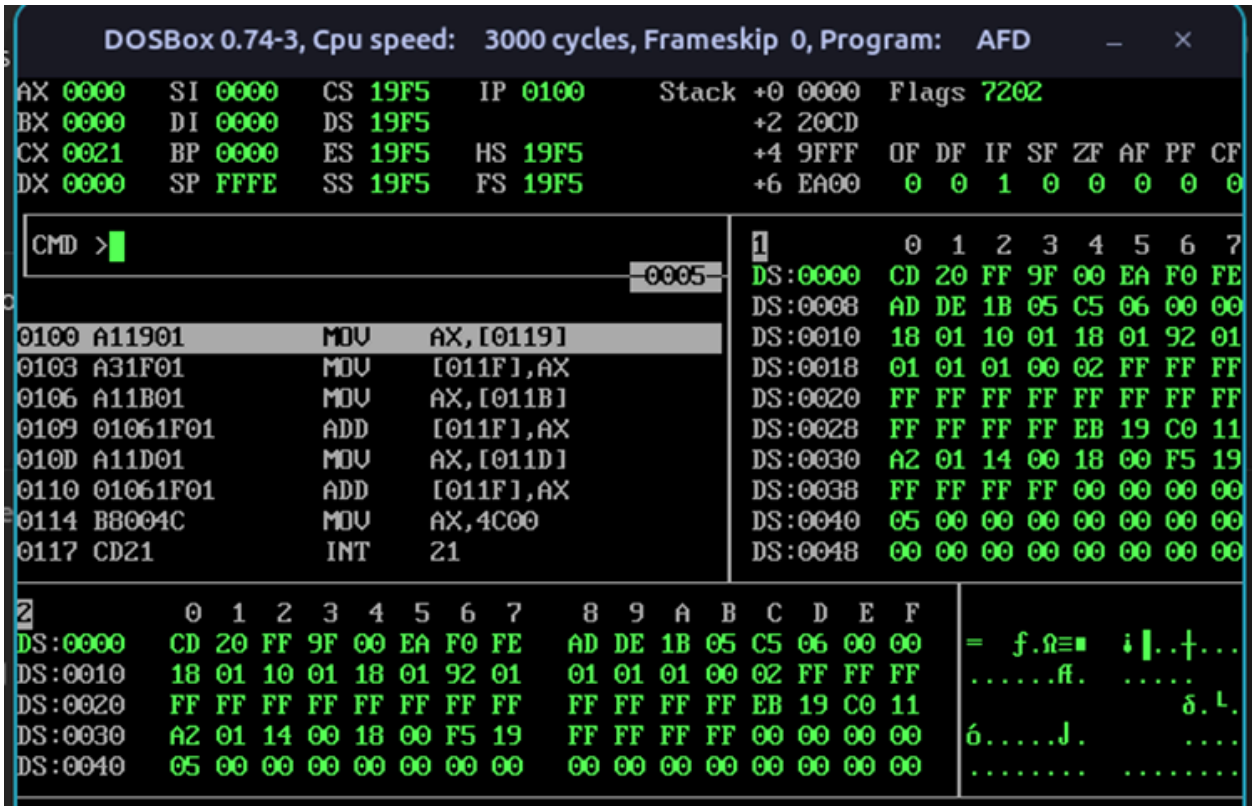
```
[org 0x0100]

    mov  ax, [num1]
    mov  [num1 + 6], ax     ; add this value to result

    mov  ax, [num1 + 2]
    add  [num1 + 6], ax

    mov  ax, [num1 + 4]
    add  [num1+6], ax

    mov  ax, 0x4c00
    int  0x21


num1:   dw  5, 10, 15, 0


; should have the result separate!
; let's change that!
```

Therefore, instead of saving a trash value in this file, we choose to save a value of zero for the addition's result.

As you can see, we moved the value of axe into [num1+6], which implies at the zero location, in the second line. We add [num + 6] because we are using the word type to describe our label and we need to leap three values. Since every value comes after two bytes, we know that 2(bytes) * 3(values to jump) = 6.

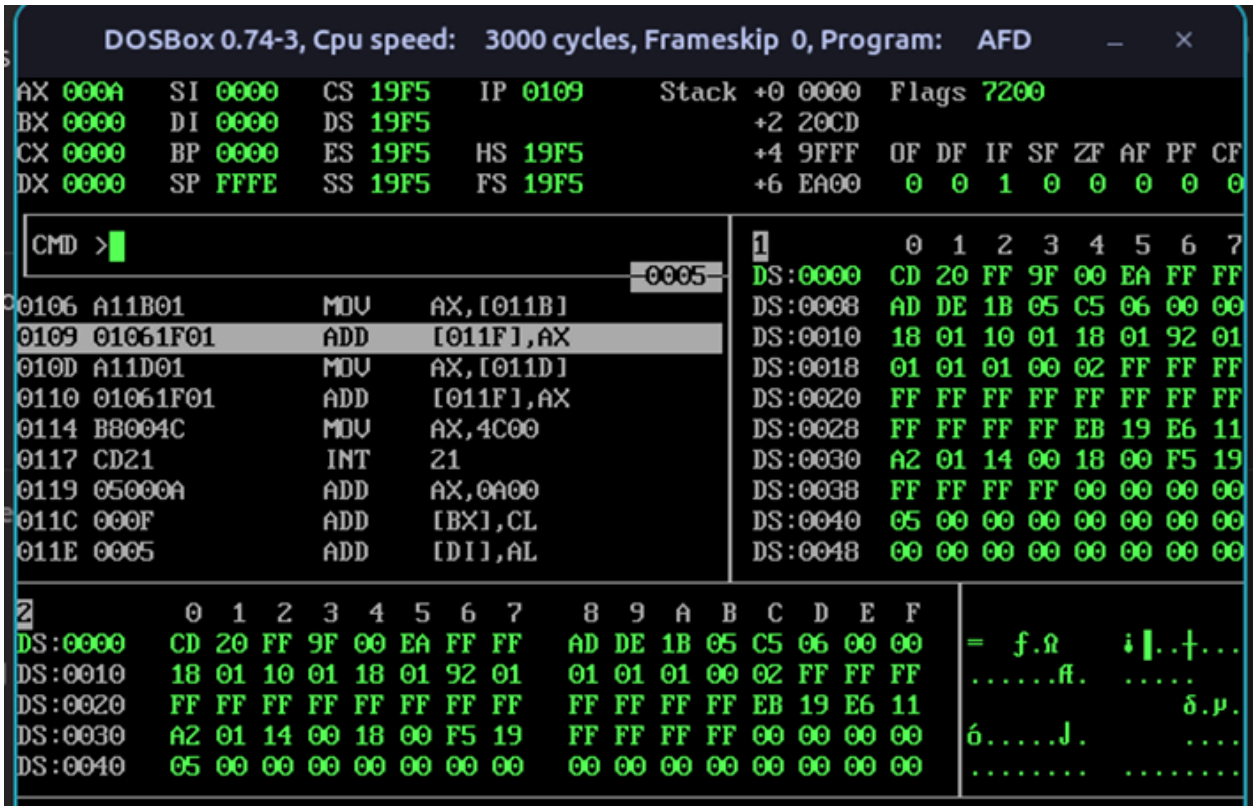
●Let's now try this in the debugger.


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 0000 SI 0000 CS 19F5 IP 0100 Stack +0 0000 Flags 7202
BX 0000 DI 0000 DS 19F5 +2 20CD
CX 0021 BP 0000 ES 19F5 HS 19F5 +4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5 +6 EA00 0 0 1 0 0 0 0 0
CMD >
0005
0100 A11901 MOV AX,[0119]
0103 A31F01 MOV [011F],AX
0106 A11B01 MOV AX,[011B]
0109 01061F01 ADD [011F],AX
010D A11D01 MOV AX,[011D]
0110 01061F01 ADD [011F],AX
0114 B8004C MOV AX,4C00
0117 CD21 INT 21
1 0 1 2 3 4 5 6 7
DS:0000 CD 20 FF 9F 00 EA F0 FE
DS:0008 AD DE 1B 05 C5 06 00 00
DS:0010 18 01 10 01 18 01 92 01
DS:0018 01 01 01 00 02 FF FF FF
DS:0020 FF FF FF FF FF FF FF FF
DS:0028 FF FF FF FF EB 19 C0 11
DS:0030 A2 01 14 00 18 00 F5 19
DS:0038 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00
DS:0048 00 00 00 00 00 00 00 00
2 0 1 2 3 4 5 6 7 8 9 A B C D E F
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 0000 SI 0000 CS 19F5 IP 0100 Stack +0 0000 Flags 7202
BX 0000 DI 0000 DS 19F5 +2 20CD
CX 0021 BP 0000 ES 19F5 HS 19F5 +4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5 +6 EA00 0 0 1 0 0 0 0 0
CMD >
0005
0100 A11901 MOV AX,[0119]
0103 A31F01 MOV [011F],AX
0106 A11B01 MOV AX,[011B]
0109 01061F01 ADD [011F],AX
010D A11D01 MOV AX,[011D]
0110 01061F01 ADD [011F],AX
0114 B8004C MOV AX,4C00
0117 CD21 INT 21
1 0 1 2 3 4 5 6 7
DS:0000 CD 20 FF 9F 00 EA F0 FE
DS:0008 AD DE 1B 05 C5 06 00 00
DS:0010 18 01 10 01 18 01 92 01
DS:0018 01 01 01 00 02 FF FF FF
DS:0020 FF FF FF FF FF FF FF FF
DS:0028 FF FF FF FF EB 19 C0 11
DS:0030 A2 01 14 00 18 00 F5 19
DS:0038 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00
DS:0048 00 00 00 00 00 00 00 00
2 0 1 2 3 4 5 6 7 8 9 A B C D E F
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- After pressing F1

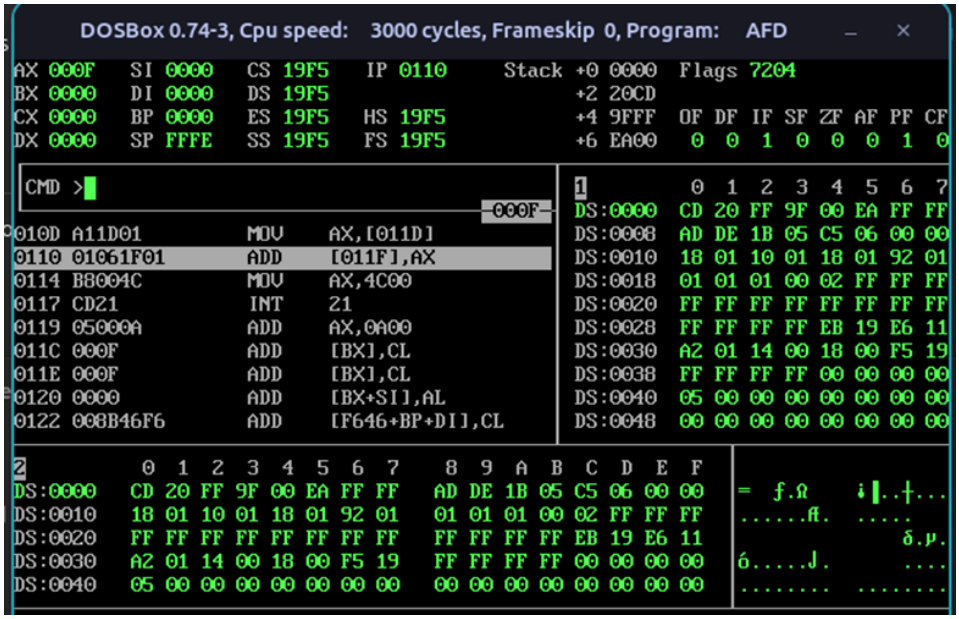
- See changing in value of Ax,also in IP register,i mean to say some general changes which we see in every code. Again…



now the value is added in Ax and become 000A

- On next step more value is added in Ax.

- At next the interept code mov



to ax and code will terminate.

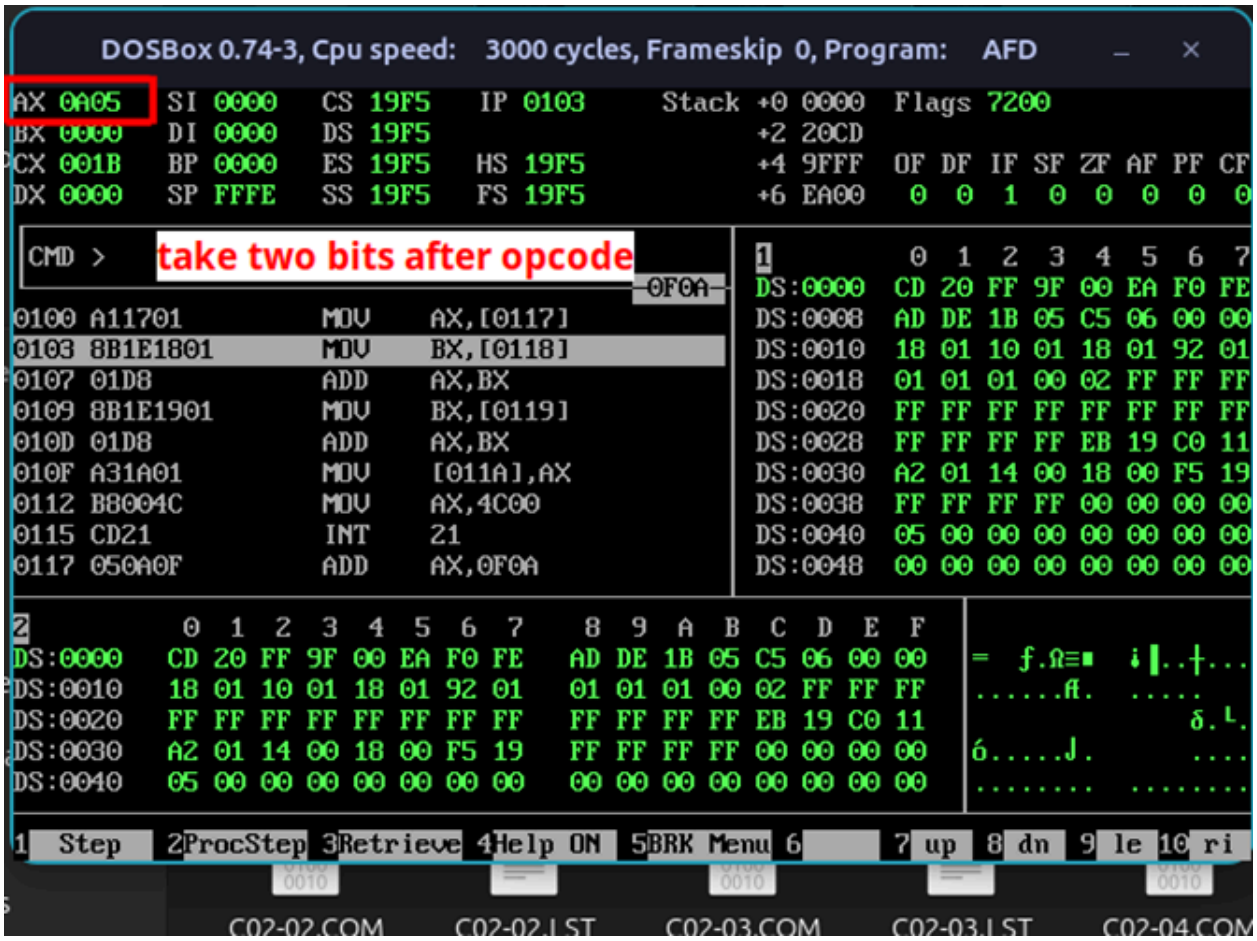
5) Understanding the c02-05.asm:

- This is the code inside the file of c02-05.asm

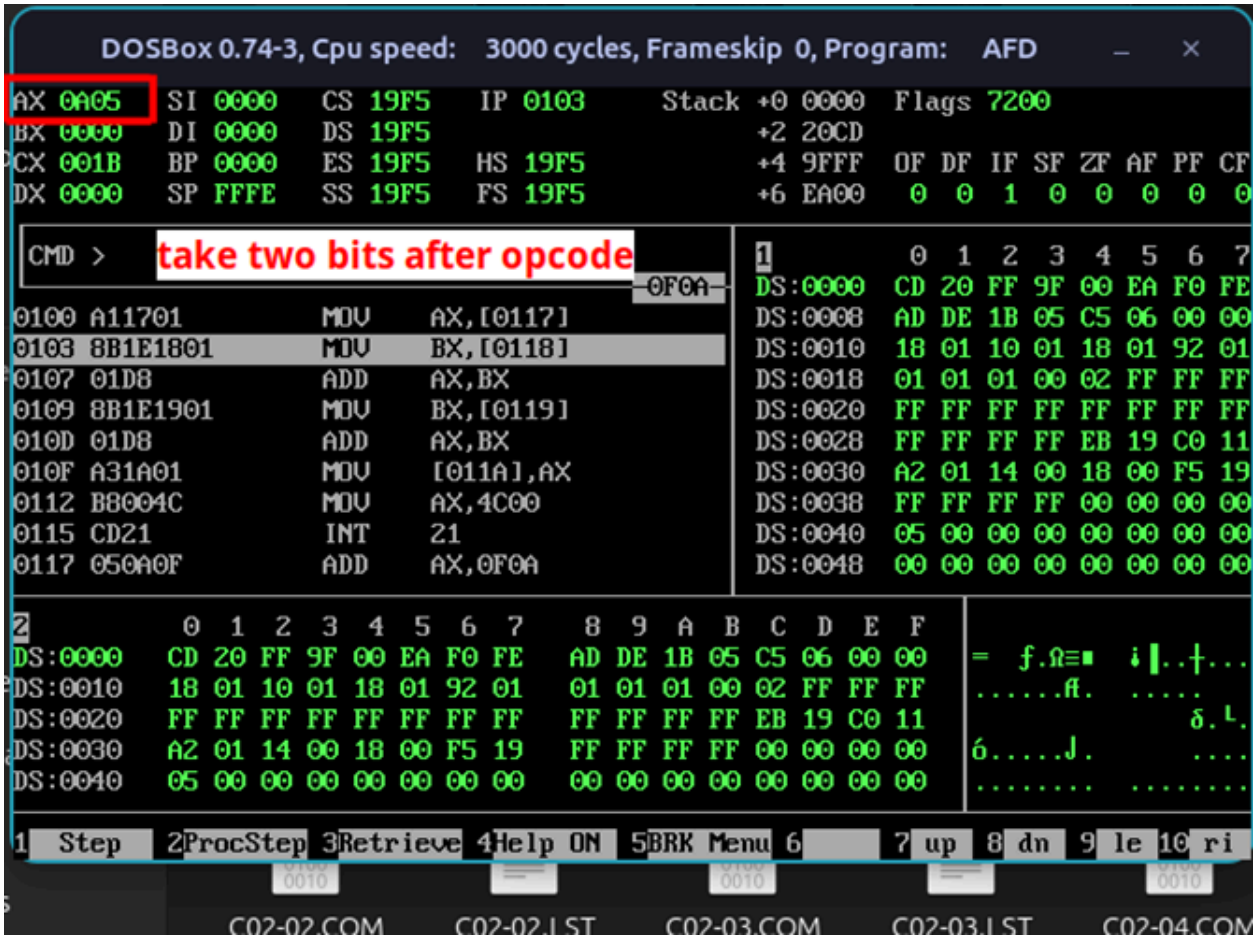
```
; a program to add three numbers using byte variables
[org 0x0100]

    mov   ax, [num1]

    mov   bx, [num1+1]
    add   ax, bx

    mov   bx, [num1+2]
    add   ax, bx

    mov   [num1+3], ax

    mov   ax, 0x4c00
    int   0x21

num1: db  5, 10, 15, 0

; something's wrong with this code.
; let's figure out what that is!
```

● We can add values in this code one at a time. First, save the value in axe, then bx, then add both, and so on. Let's now examine it in the debugger.

● However, there is a problem since, as we know, db stores values in 1 bit, whereas axe stores values in 16 bits. As a result, you can see that there is an error when storing values in Axe, as shown in the figure.



- It can store two different values instead of storing the just value 10. This is the issue in that code.



6) Understanding the c02-06.asm:

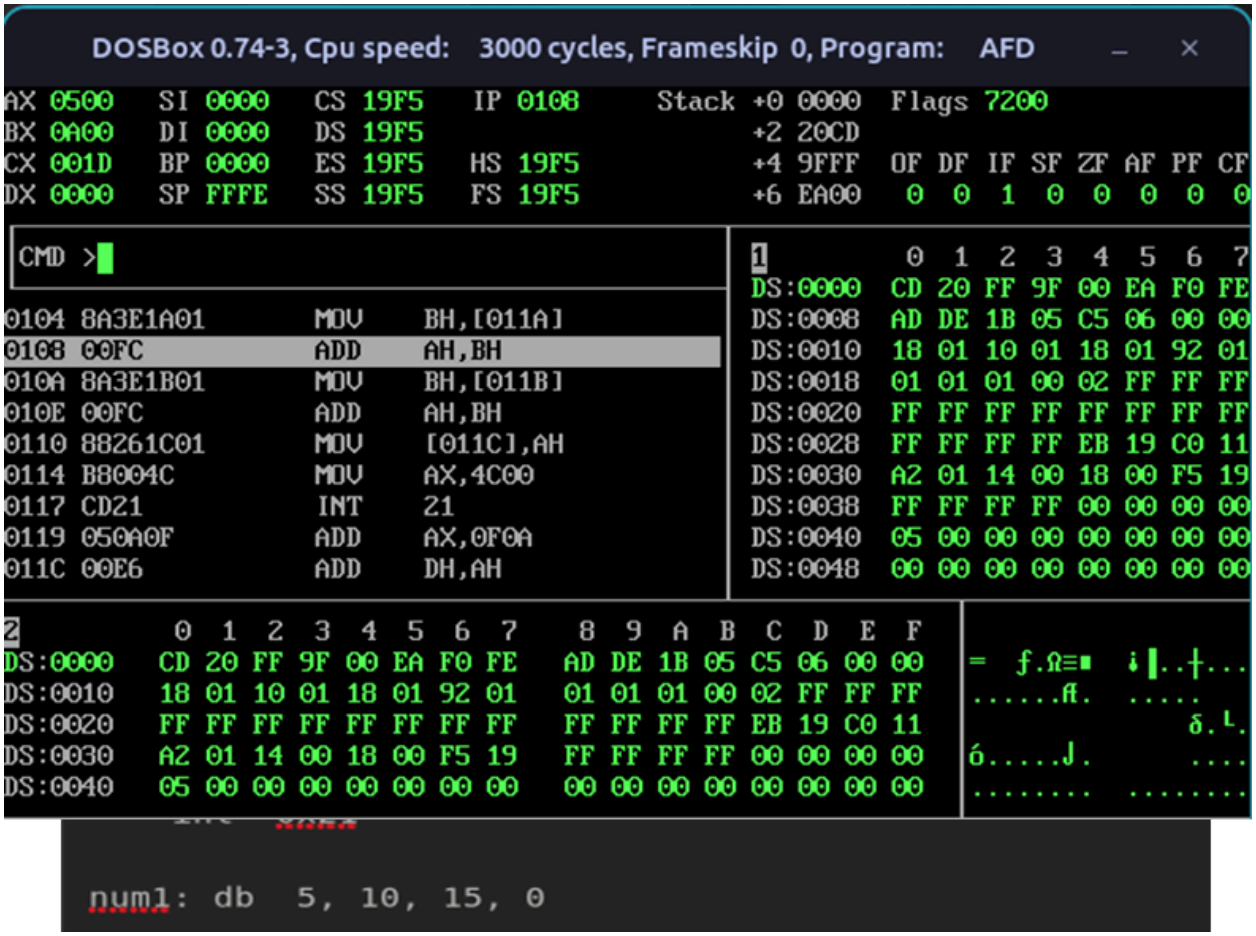
- So now we see the solution of that previous one. Lets see the code of c02-06
- Here is the code of this file

```
; a program to add three numbers using byte variables
[org 0x0100]

    mov  ah, [num1]          ; Intel Sotware Developer

    mov  bh, [num1+1]
    add  ah, bh

    mov  bh, [num1+2]
    add  ah, bh

    mov  [num1+3], ah

    mov  ax, 0x4c00
    int  0x21

num1: db  5, 10, 15, 0
```

● Therefore, we can put values in the al or ah register rather than the axe register in order to avoid making this mistake. This is because the al and ah registers are 8-bit registers. Likewise for bl in place of bx. And this will address the issues we had with the earlier code.

● Let's test this in Dosbox to see whether it solves our issue.

Proceed to look for Bl.



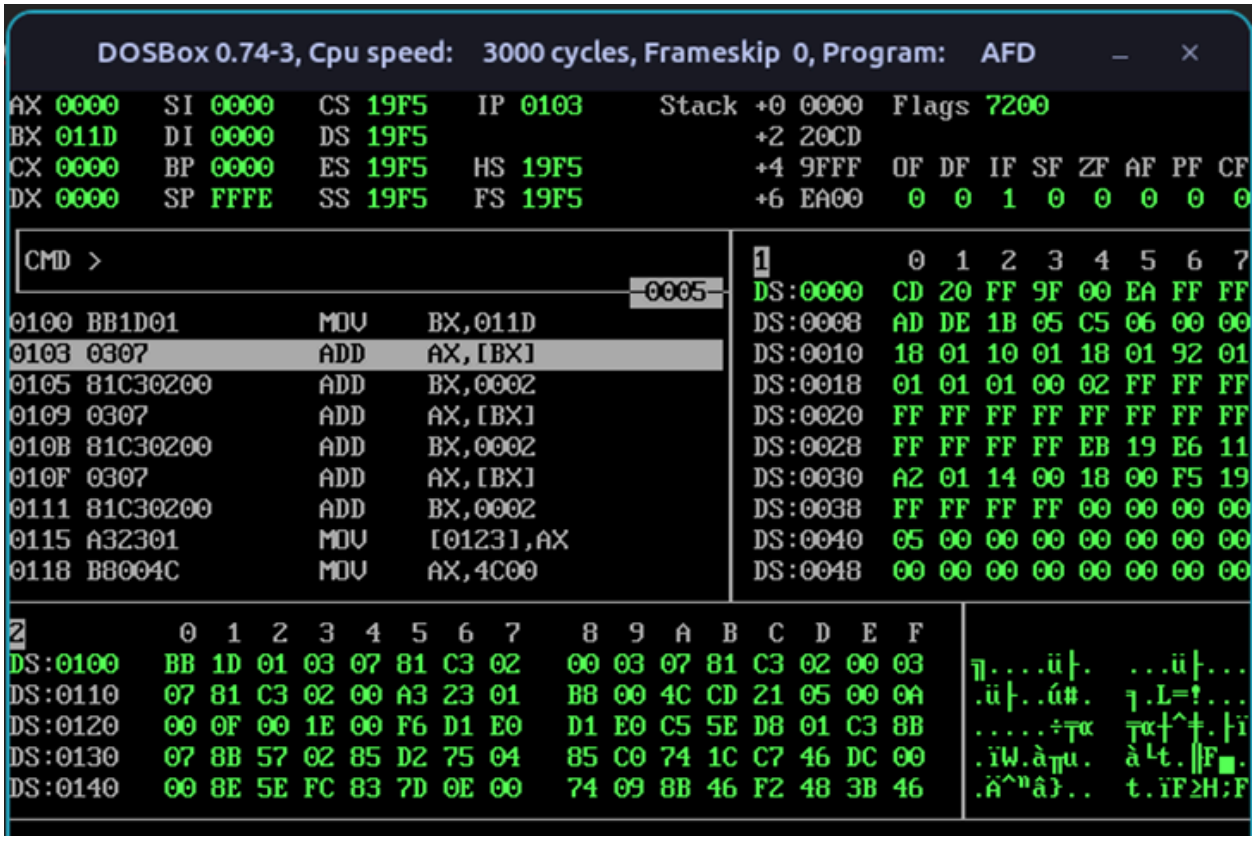
Value 10 appears to be perfectly stored in the Bx register. The remainder of the code operates flawlessly as well because all addition and subtraction will function properly if the value is entered correctly.AL REGISTER AND WE USE THE SAME CODE.

7) Being aware of the c02-07.asm:

This is the C02-07.asm code. This code uses add values in bx to jump, and we know that the [] pick value comes from the location enclosed in curly brackets. We now write bx, move it into axe, and leap by simply adding bx's address.

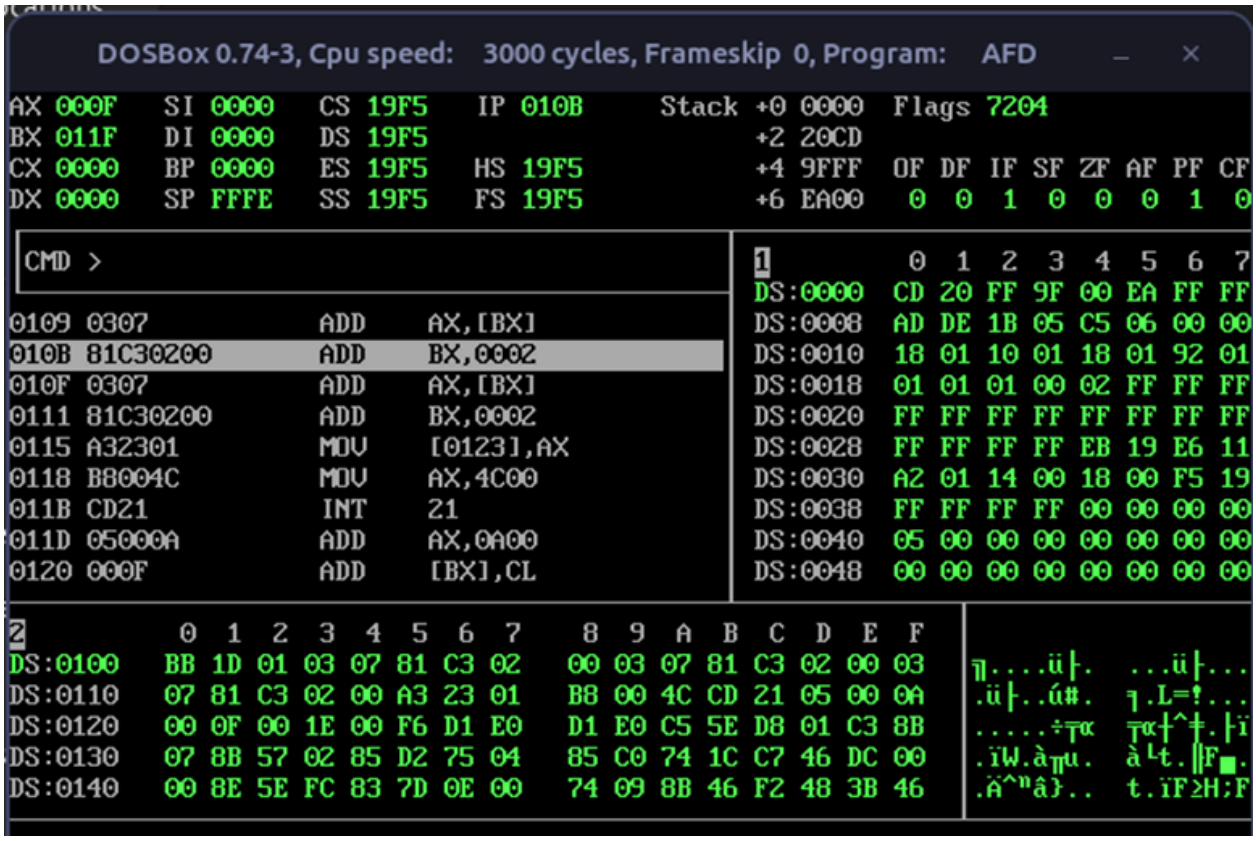
```
[org 0x0100]

    mov   bx, num1
    add   ax, [bx]
    add   bx, 2
    add   ax, [bx]
    add   bx, 2
    add   ax, [bx]
    add   bx, 2

    mov   [result], ax
    mov   ax, 0x4c00
    int   0x21

num1: dw  5, 10, 15
result: dw 0
```

- Bx has the address of num1 value 5.
- we can increase the value according to the type and get the next value store in num1.



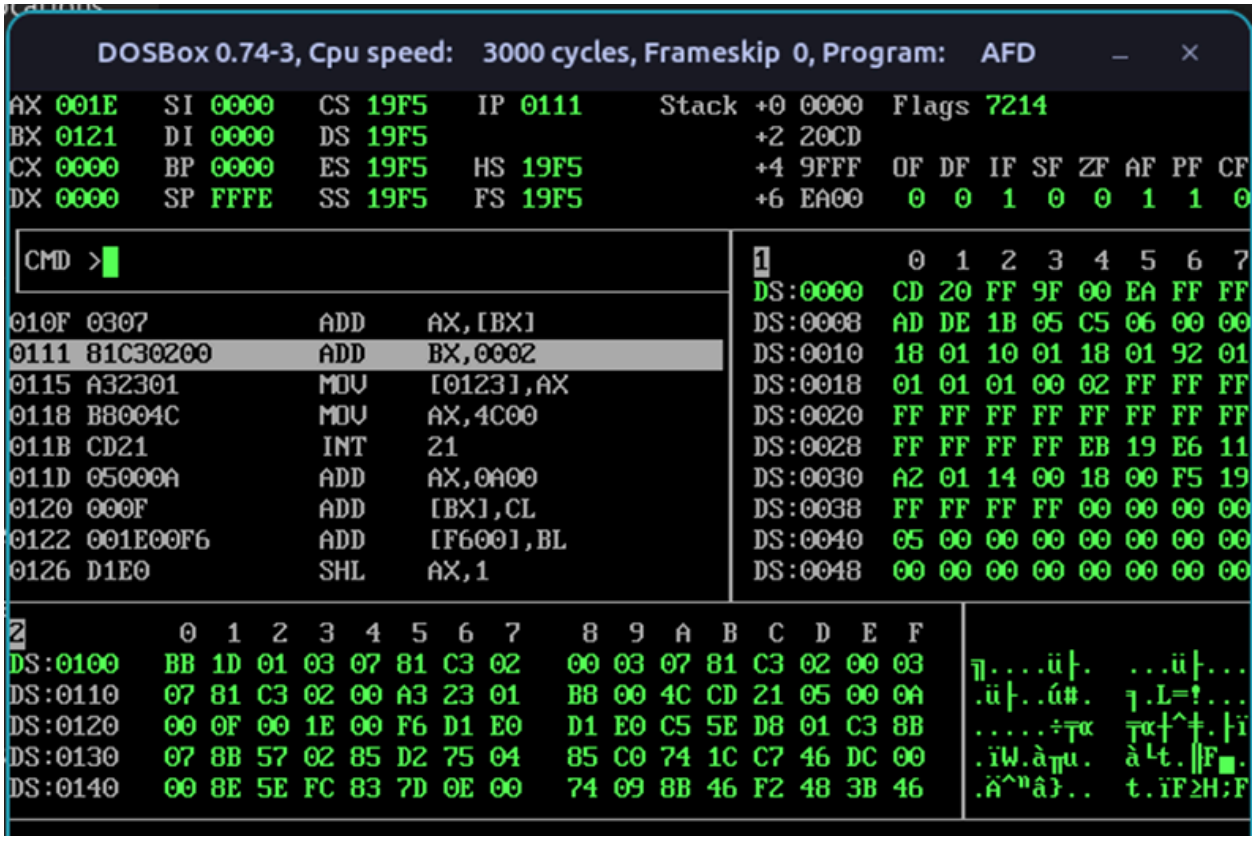
- Address of num1 save in bx and we see value of bx changed.

- Now after apply braces its take the value on this address and store it into ax.
- Now add that bx value into ax



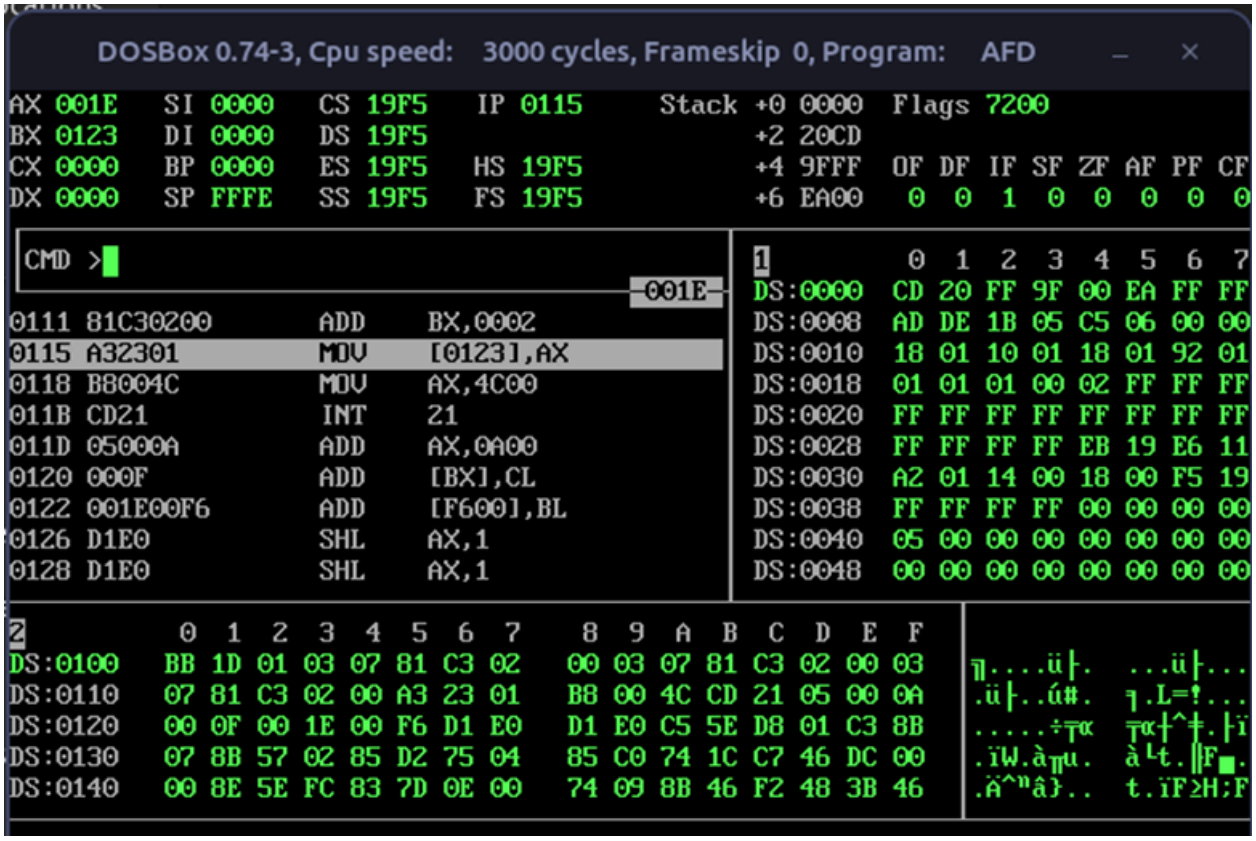
- Result will now see at Ax.

- Now next address in bx and its value is store in ax.



- Result after store into Ax.

- Next store the ax address into another variable.



- Then at last the interept command and the program will terminate.