Lab 3

Execution Of Tasks Lab 3

Task 1

D:\Assembly\labwork3.asm - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

labwork3.asm

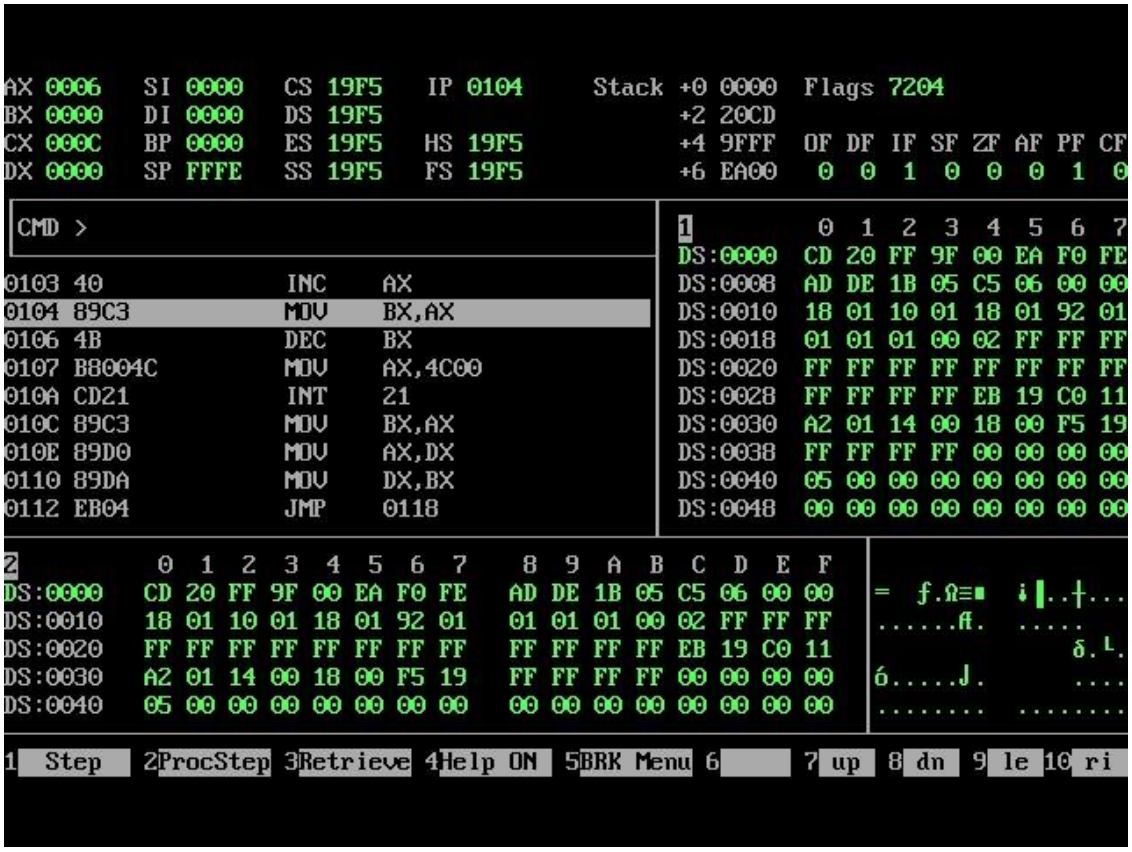
```
[org 0x0100]

; start of code

mov  ax, 5            ; move the constant 5 into register ax
inc  ax,

mov  bx, ax           ; move the value of ax into the value of bx

dec  bx,              ;decrement the value of bx


mov  ax, 0x4c00         ; exit ..
int  0x21               ; .. is what the OS should do for me
```

Explanation:

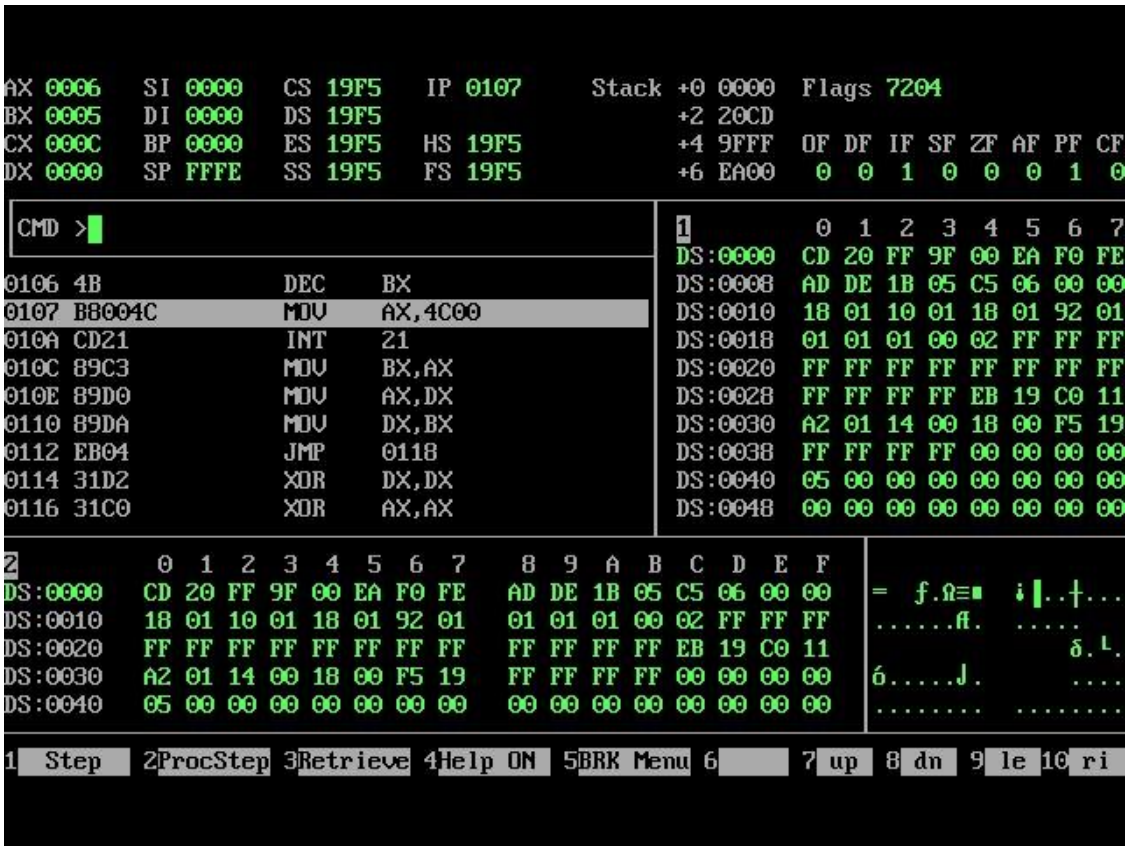
Here in this code, we first add 5 to register ax, and then using **inc** command we increment the value stored in ax by 1, after that we are moving the value from ax to bx register and then using **dec**, we decrement the value in bx by 1 as per question requirement.

Following are the screenshots of execution of code:

```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount x: D:\Assembly
Drive X is mounted as local directory D:\Assembly\

Z:\>X:\

X:\>nasm labwork3.asm -o labwork.com

X:\>afd labwork3.com_
```



The value is added and incremented in ax



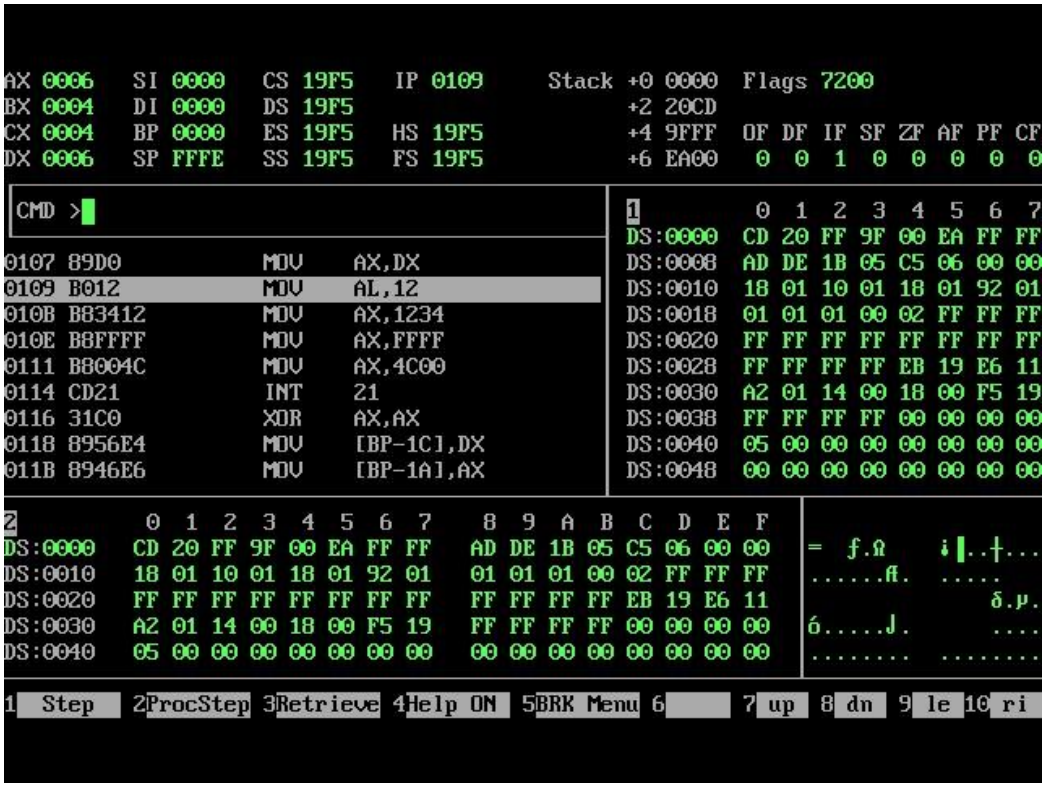
The value is moved to bx and decremented.

Task 2 Following

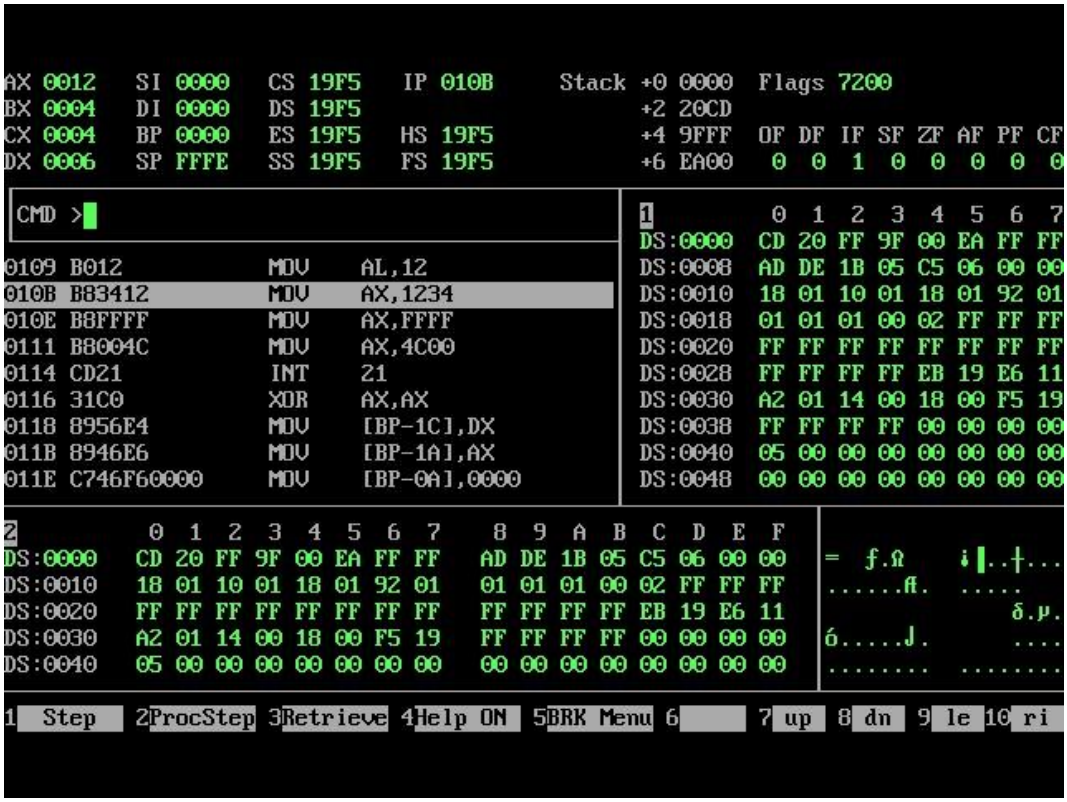
is the code of the task 2:

```
[org 0x0100]

; start of code

mov  bl, 4              ; move the constant 5 into register bl
mov  dx, 6              ;move the constant 6 to dx

mov cl,bl               ;move dl to cl
mov ax,dx               ; move dx to ax

mov al,0x12       ; move 0x12 to al
mov ax,0x1234     ; move 0x1234 to ax
mov ax,0xffff     ; move 0xffff to ax

mov ax,0x4c00
int 0x21
```

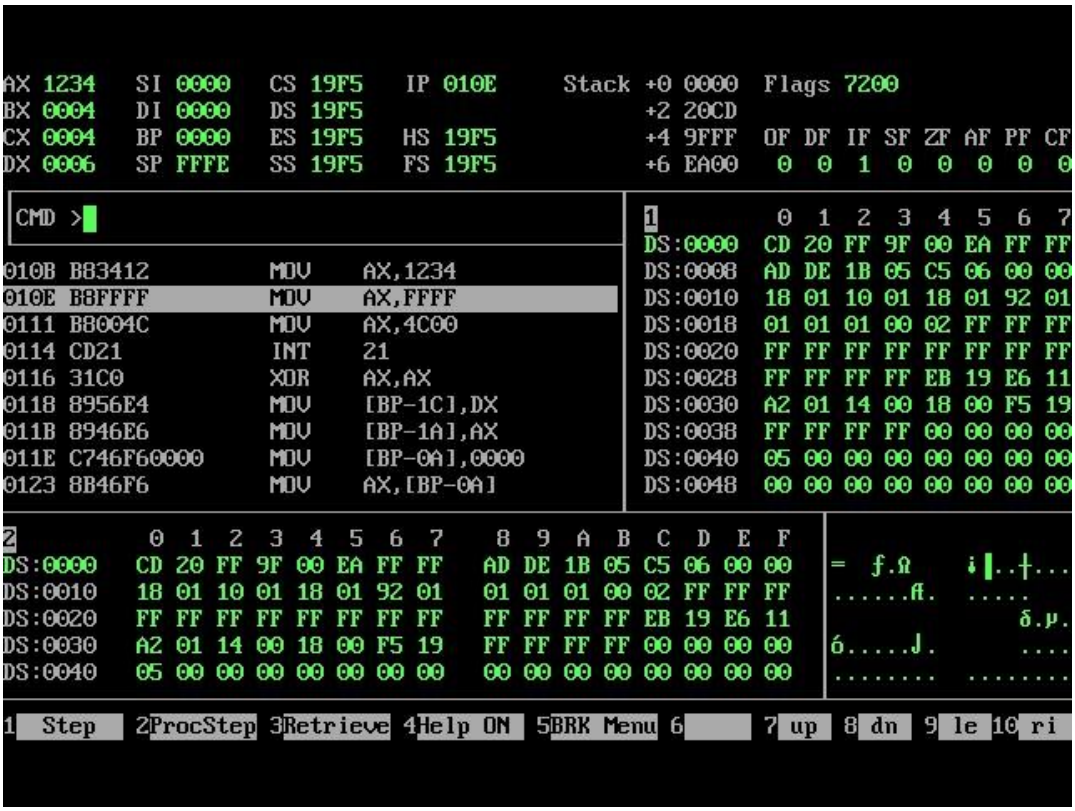
Following are the screenshots of code execution and their explanation:



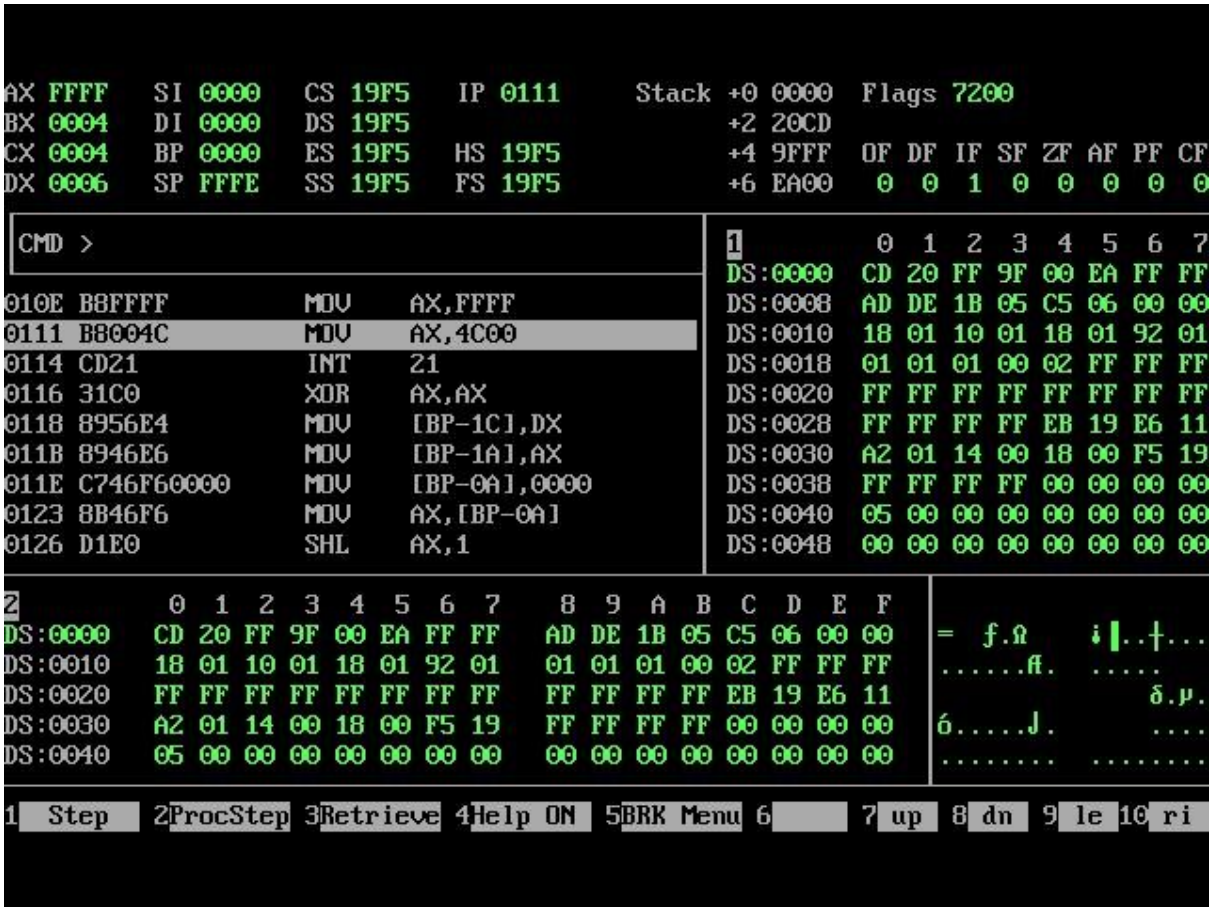
- The value is added to bl ,4 (bl has half of memory size as of bx
- The value is added to bx, 6
- The value is moved from bl to cl (same as bl, cl also is a register containing half of memory as cx)
- The value is moved from dx to ax



The value 0x12 is moved to al.

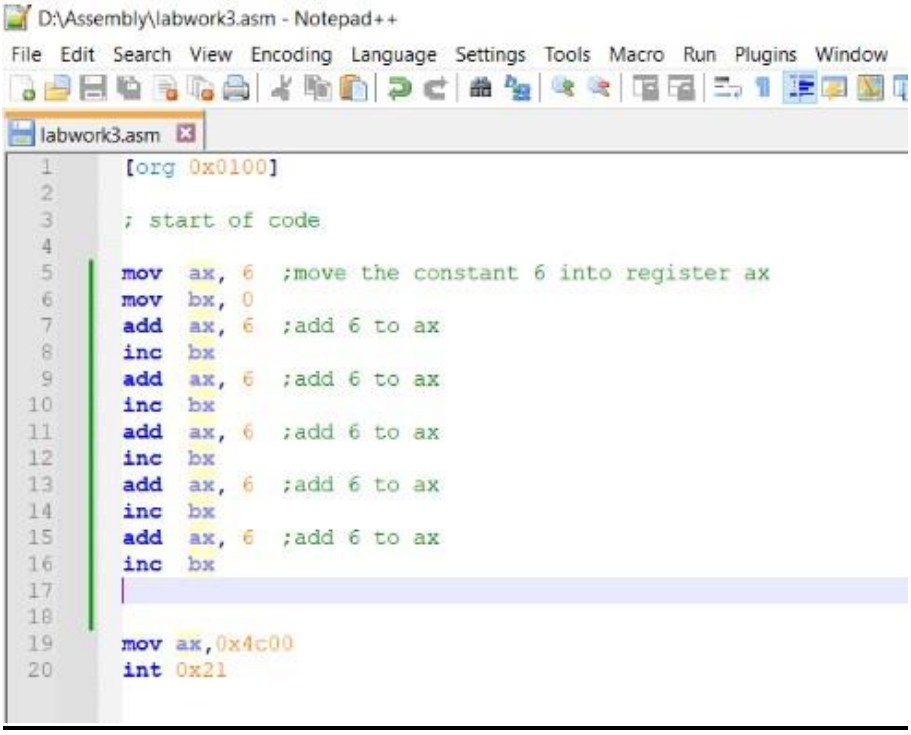


The value 0x1234 is moved to al.

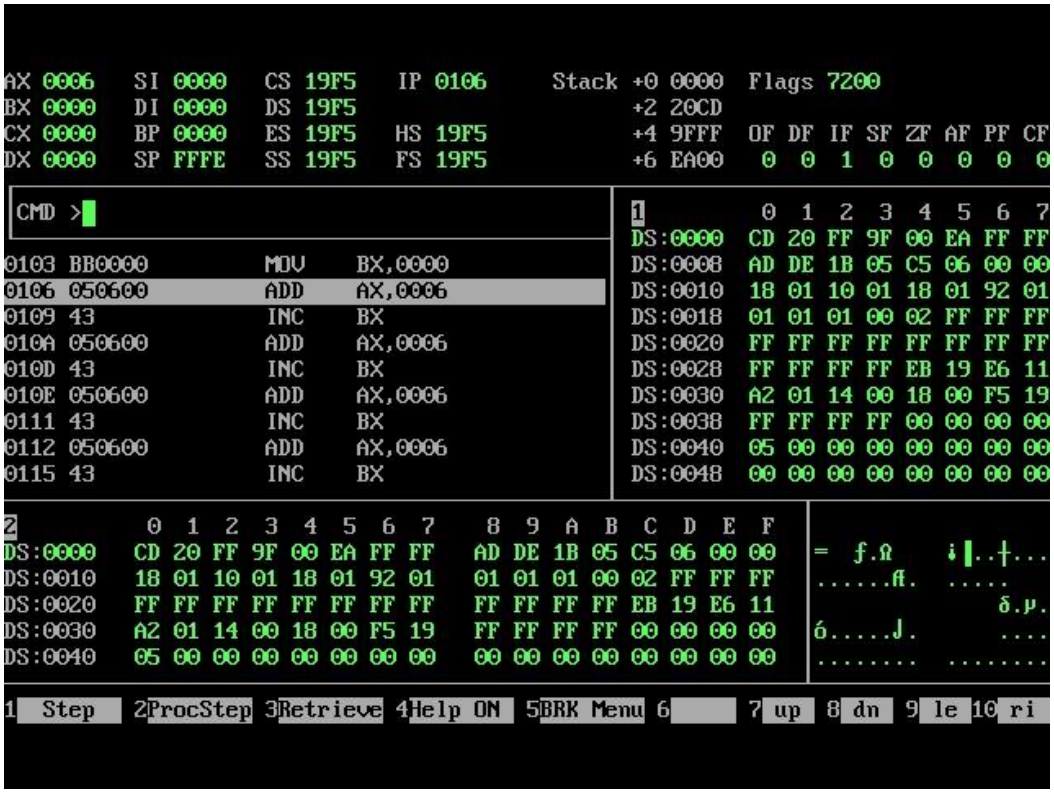


The value 0xffff is moved to al register.

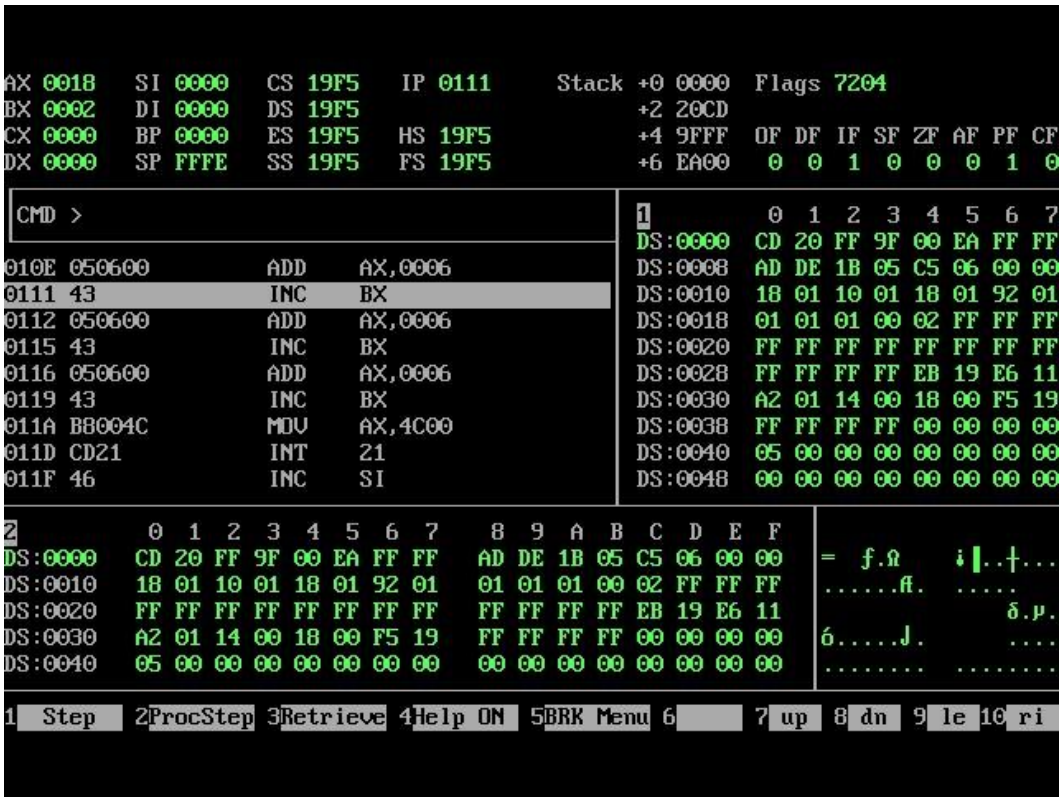
Task 3



Following are the screenshots of execution of code followed by explanation.



The value of 6 is moved to the register ax.

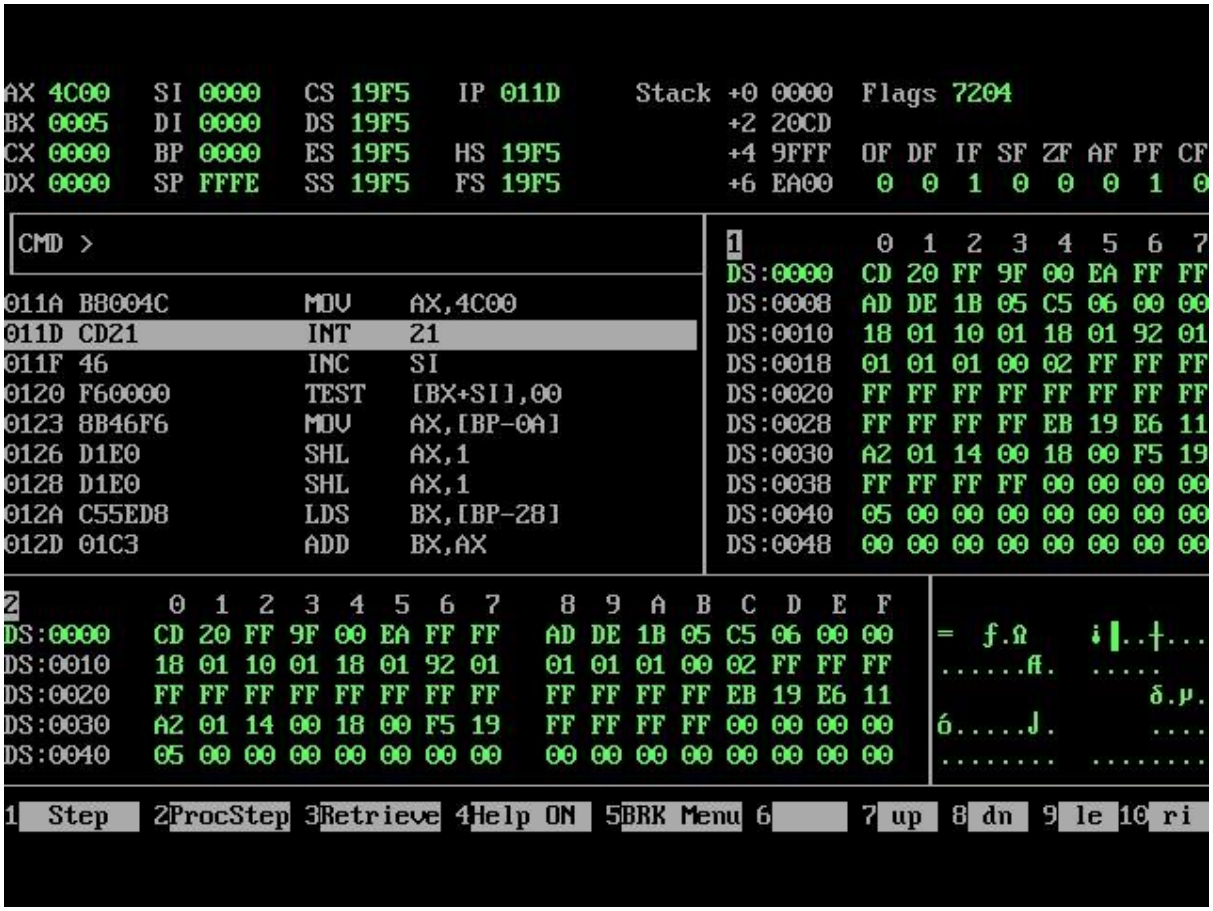


The value is added again followed the incrementation to bx, bx is 2 here and ax is 18, means the process of addition is being repeated 3 times until now…

```
AX 001E   SI 0000   CS 19F5   IP 0115     Stack +0 0000  Flags 7204
BX 0003   DI 0000   DS 19F5                   +2 20CD
CX 0000   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  1  0

CMD >

0112 050600         ADD    AX,0006            1        0  1  2  3  4  5  6  7
0115 43             INC    BX                 DS:0000  CD 20 FF 9F 00 EA FF FF
0116 050600         ADD    AX,0006            DS:0008  AD DE 1B 05 C5 06 00 00
0119 43             INC    BX                 DS:0010  18 01 10 01 18 01 92 01
011A B8004C         MOV    AX,4C00            DS:0018  01 01 01 00 02 FF FF FF
011D CD21           INT    21                 DS:0020  FF FF FF FF FF FF FF FF
011F 46             INC    SI                 DS:0028  FF FF FF FF EB 19 E6 11
0120 F60000         TEST   [BX+SI],00         DS:0030  A2 01 14 00 18 00 F5 19
0123 8B46F6         MOV    AX,[BP-0A]         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 FF FF  AD DE 1B 05 C5 06 00 00   = ƒ.Ω   ¡▌..┼...
DS:0010   18 01 10 01 18 01 92 01  01 01 01 00 02 FF FF FF   ......ª.  .....
DS:0020   FF FF FF FF FF FF FF FF  FF FF FF FF EB 19 E6 11              δ.µ.
DS:0030   A2 01 14 00 18 00 F5 19  FF FF FF FF 00 00 00 00   ó.....J.     ....
DS:0040   05 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........  ........

1  Step   2ProcStep 3Retrieve 4Help ON  5BRK Menu 6      7 up  8 dn  9 le 10 ri
```

```
AX 0024   SI 0000   CS 19F5   IP 0119     Stack +0 0000  Flags 7214
BX 0004   DI 0000   DS 19F5                   +2 20CD
CX 0000   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  1  1  0

CMD >

0116 050600         ADD    AX,0006            1        0  1  2  3  4  5  6  7
0119 43             INC    BX                 DS:0000  CD 20 FF 9F 00 EA FF FF
011A B8004C         MOV    AX,4C00            DS:0008  AD DE 1B 05 C5 06 00 00
011D CD21           INT    21                 DS:0010  18 01 10 01 18 01 92 01
011F 46             INC    SI                 DS:0018  01 01 01 00 02 FF FF FF
0120 F60000         TEST   [BX+SI],00         DS:0020  FF FF FF FF FF FF FF FF
0123 8B46F6         MOV    AX,[BP-0A]         DS:0028  FF FF FF FF EB 19 E6 11
0126 D1E0           SHL    AX,1               DS:0030  A2 01 14 00 18 00 F5 19
0128 D1E0           SHL    AX,1               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 FF FF  AD DE 1B 05 C5 06 00 00   = ƒ.Ω   ¡▌..┼...
DS:0010   18 01 10 01 18 01 92 01  01 01 01 00 02 FF FF FF   ......ª.  .....
DS:0020   FF FF FF FF FF FF FF FF  FF FF FF FF EB 19 E6 11              δ.µ.
DS:0030   A2 01 14 00 18 00 F5 19  FF FF FF FF 00 00 00 00   ó.....J.     ....
DS:0040   05 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........  ........

1  Step   2ProcStep 3Retrieve 4Help ON  5BRK Menu 6      7 up  8 dn  9 le 10 ri
```

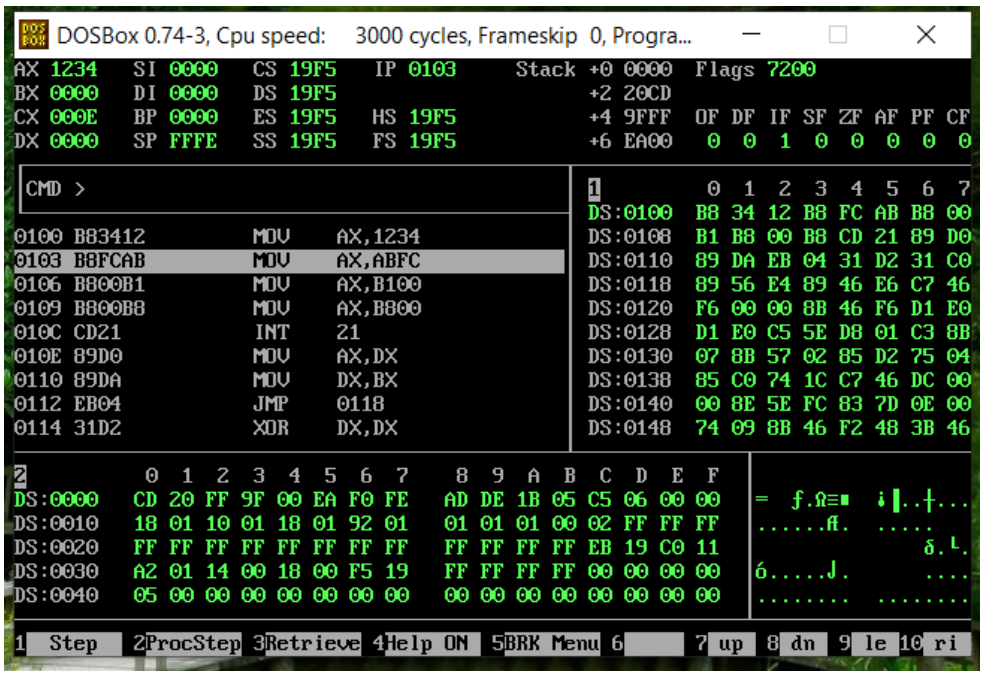
Until now the process is repeated by 6 times giving us the value in bx 4, as we initiated with 0, ax has value of 24 and process continues till we get 36…



Ax has value 4c00, for exit as we have coded in code…

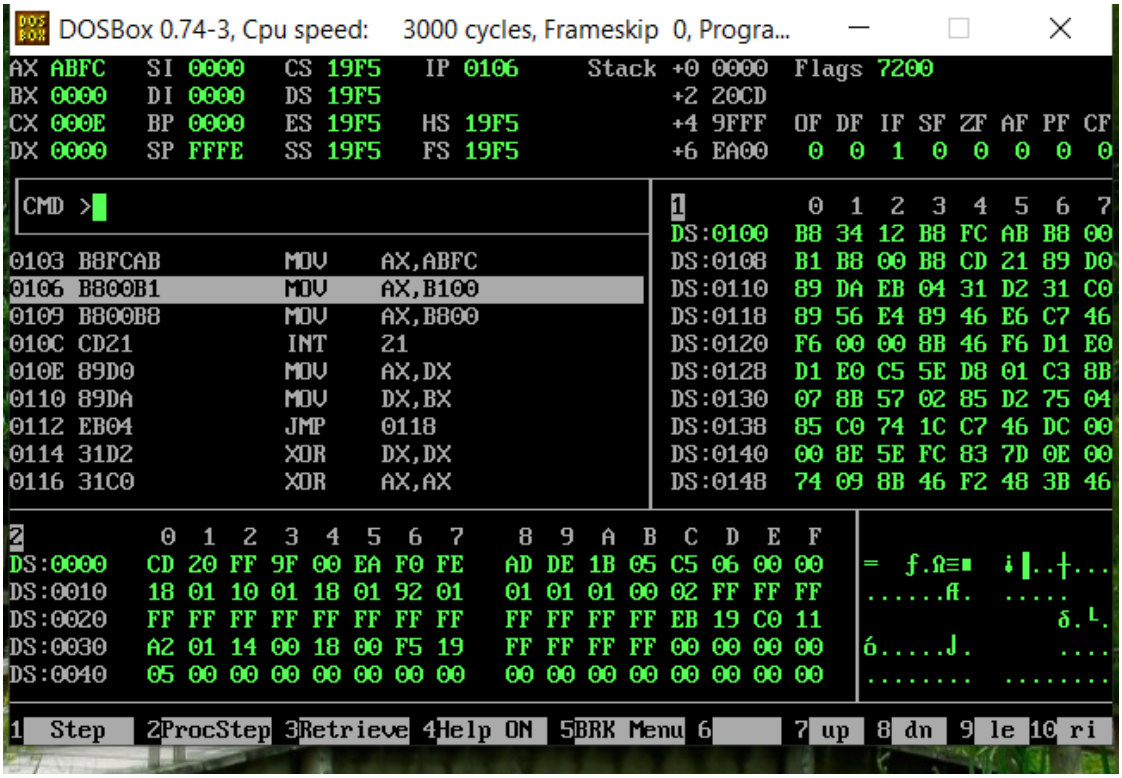
Task 4

Following is the screenshot of how the given values are stored in the registers and explained:

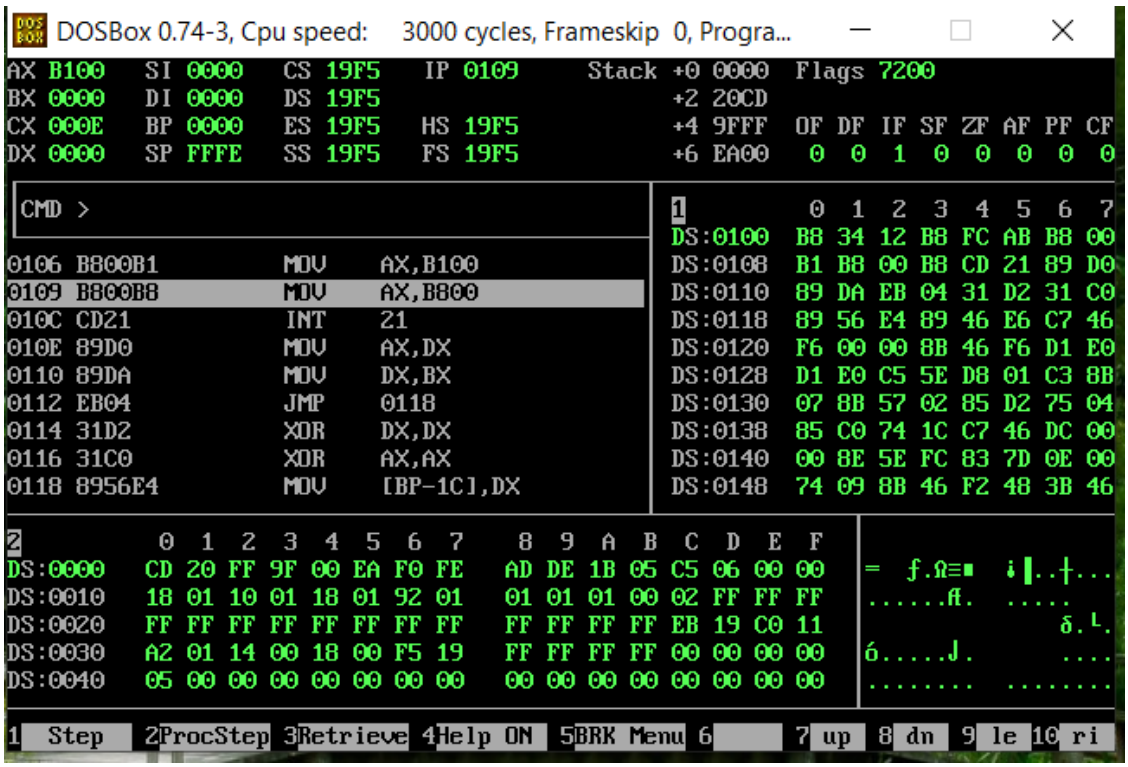


Here the value 34 is stored at address 0101 and value 12 is stored at address 0102

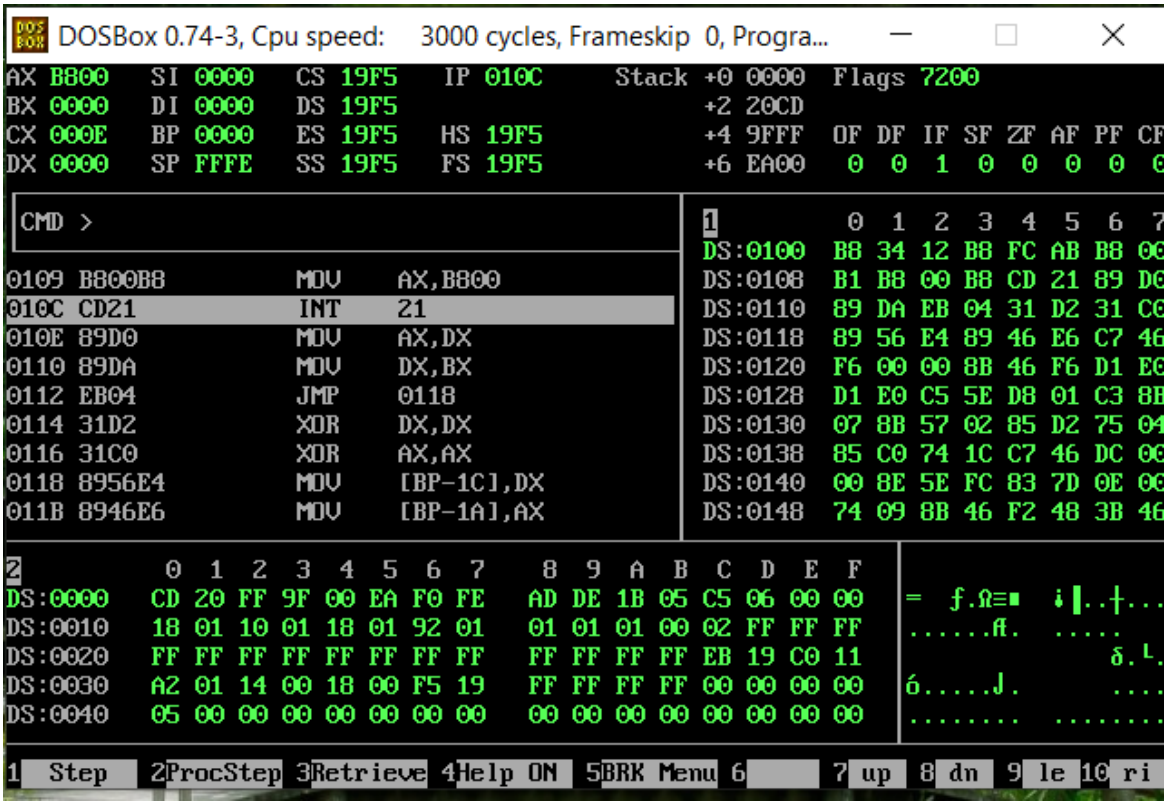
Because intel architecture follows little-Endian method to store value



Here value FC is stored at address 0104 and AB is stored at address 0105 following little-endian method.



Here value 00 is stored at address 0107 and B1 is stored at address 0108



Here 00 is stored at address 0108,2 and B1 is stored at 0108,1 , all these values are stored in little endian method since it's a intel architecture.

Little-Endian: Means that the least significant byte is stored at the lower memory address, while the most significant byte.

Big-Endian is the opposite; the most significant byte is stored at the lower memory address, and the least significant byte is stored at the higher memory address.

a. **1234**

- In Little-Endian: Memory Address: 34 (lower), 12 (higher)
- In Big-Endian: Memory Address: 12 (lower), 34 (higher)

b. **ABFC**

- In Little-Endian: Memory Address: FC (lower), AB (higher)
- In Big-Endian: Memory Address: AB (lower), FC (higher)

c. **B100**

- In Little-Endian: Memory Address: 00 (lower), B1 (higher)
- In Big-Endian: Memory Address: B1 (lower), 00 (higher)

d.**B800**

- In Little-Endian: Memory Address: 00 (lower), B8 (higher)

- In Big-Endian: Memory Address: B8 (lower), 00 (higher)