

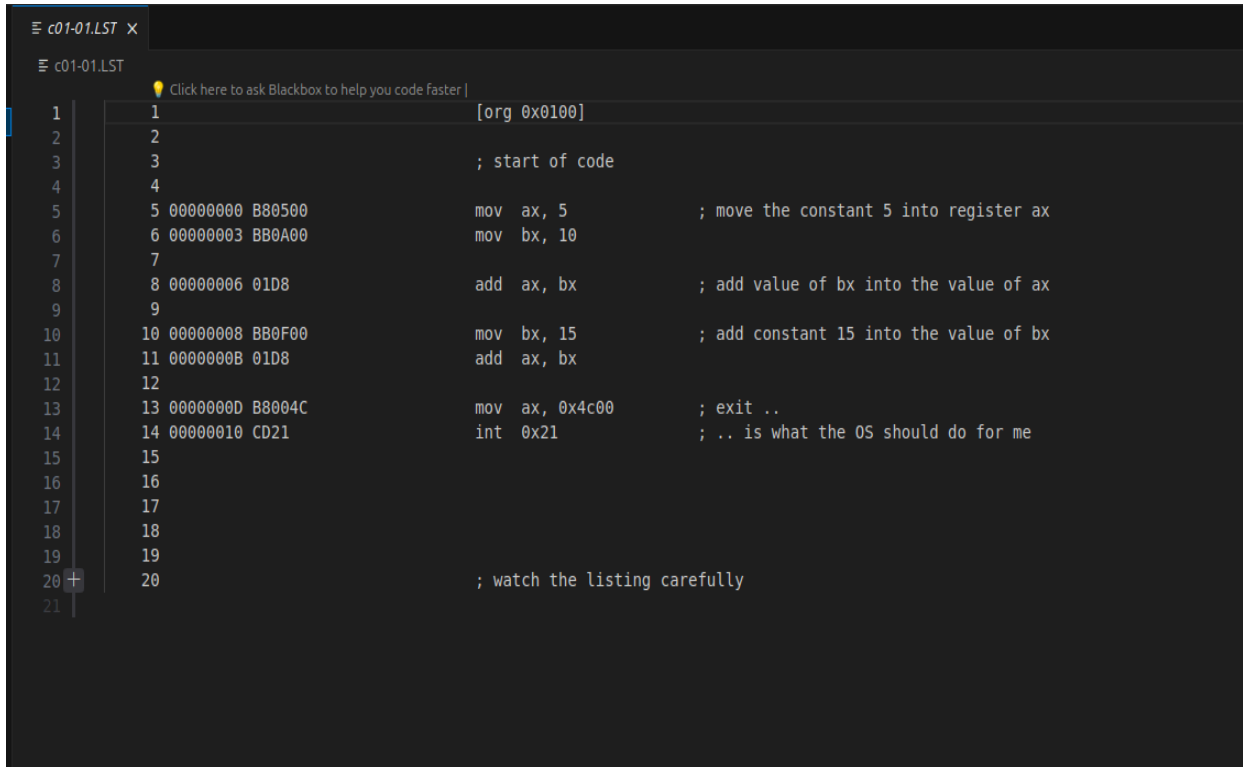
# Lab # 3

**Name : M.Shafeen**

**Roll No : 22P-9278**

**File c02-01 :**

**Step : 1**



```
c01-01.LST x
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00 ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

This above is the listing file of the **C02-01.asm** file and it is the machine code that tells us step by step how much memory it is consuming in our memory

**Debugger 1 :**

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD

AX 0005

SI 0000

CS 19F5

IP 0103

Stack +0 0000

Flags 7200

BX 0000

DI 0000

DS 19F5

+2 20CD

CX 001F

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 >

000A

0100 A11701

MOV

AX, [0117]

0103 8B1E1901

MOV

BX, [0119]

0107 01D8

ADD

AX, BX

0109 8B1E1B01

MOV

BX, [011B]

010D 01D8

ADD

AX, BX

010F A31D01

MOV

[011D], AX

0112 B8004C

MOV

AX, 4C00

0115 CD21

INT

21

0117 05000A

ADD

AX, 0A00

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

= f.Ω■ i |..†...

.....ff. ....

δ. L.

ó.....J. ....

.....

1 Step

2ProcStep

3Retrieve

4Help ON

5BRK Menu

6

7 up

8 dn

9 le

10 ri

```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

## Explanation :

This instruction is taking a number (value : 5) stored in a variable called "**num1**" with the address **0117** and putting it into a special spot in the computer's brain called the "**ax**" register.

## Step : 2

## Debugger 2 :

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD																	
AX 0005	SI 0000	CS 19F5	IP 0107	Stack +0 0000	Flags 7200												
BX 000A	DI 0000	DS 19F5		+2 20CD													
CX 001F	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 >																	
0103 8B1E1901 MOV BX,[0119]				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													
0107 01D8 ADD AX,BX																	
0109 8B1E1B01 MOV BX,[011B]																	
010D 01D8 ADD AX,BX																	
010F A31D01 MOV [011D],AX																	
0112 B8004C MOV AX,4C00																	
0115 CD21 INT 21																	
0117 05000A ADD AX,0A00																	
011A 000F ADD [BX],CL																	

```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21     21
```

## Explanation :

This instruction tells the computer to take the value ( value : 10 ) stored in the "**num2**" with the address **0119** variable and put it into a special place called the "**bx**" register. Just like before, it's like moving a number from one mental drawer (memory) to another (register) so the computer can work with it.

Step : 3

Debugger 3 :

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 000F SI 0000 CS 19F5 IP 010D Stack +0 0000 Flags 7204
BX 000F DI 0000 DS 19F5 +2 20CD
CX 001F 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 >

0109 8B1E1B01 MOV BX,[011B]
010D 01D8 ADD AX,BX
010F A31D01 MOV [011D],AX
0112 B8004C MOV AX,4C00
0115 CD21 INT 21
0117 05000A ADD AX,0A00
011A 000F ADD [BX],CL
011C 0000 ADD [BX+SI],AL
011E 0000 ADD [BX+SI],AL

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

= f.Ω≡ i |..†...
.....ff. ....
δ. L.
ó.....J. ....
.....

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

```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

## Explanation :

This instruction is fetching the value stored in the memory location labeled "**num3**" having address **011B** and putting it into the **BX** register

Step : 4

Debugger 4 :

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 001E SI 0000 CS 19F5 IP 010F Stack +0 0000 Flags 7214
BX 000F DI 0000 DS 19F5 +2 20CD
CX 001F 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 > 0000
010D 01D8 ADD AX,BX
010F A31D01 MOV [011D],AX
0112 B8004C MOV AX,4C00
0115 CD21 INT 21
0117 05000A ADD AX,0A00
011A 000F ADD [BX],CL
011C 0000 ADD [BX+SI],AL
011E 0000 ADD [BX+SI],AL
0120 0000 ADD [BX+SI],AL

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 = f.Ω≡ i |..†...
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF .....ff. ....
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11 .....δ.L.
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00 6.....J. ....
DS:0040 05 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
```



```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

## Explanation :

The instruction "**add ax, bx**" adds the value in the "**bx**" register to the value in the "**ax**" register, storing the result back in the "**ax**" register. In assembly language, this operation is represented by the opcode "**01D8**".

Step : 5

Debugger 5 :

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 001E SI 0000 CS 19F5 IP 0112 Stack +0 0000 Flags 7214
BX 000F DI 0000 DS 19F5 +2 20CD
CX 001F 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 >

010F A31D01 MOV [011D],AX
0112 B8004C MOV AX,4C00
0115 CD21 INT 21
0117 05000A ADD AX,0A00
011A 000F ADD [BX],CL
011C 001E0000 ADD [0000],BL
0120 0000 ADD [BX+SI],AL
0122 0000 ADD [BX+SI],AL
0124 0000 ADD [BX+SI],AL

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 = f.Ω= i |..+...
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF .....ff. ....
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11 .....δ.L.
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
```

```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

## Explanation :

The instruction "**mov [num4], ax**" moves the value in the "**ax**" register into the memory location labeled "num4". In assembly language, this operation is represented by the opcode "**A3**" followed by the memory address where the value should be stored.

Step : 6

Debugger 6 :

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
AX 4C00 SI 0000 CS 19F5 IP 0115 Stack +0 0000 Flags 7214
BX 000F DI 0000 DS 19F5 +2 20CD
CX 001F 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 >

0112 B8004C MOV AX,4C00
0115 CD21 INT 21
0117 05000A ADD AX,0A00
011A 000F ADD [BX],CL
011C 001E0000 ADD [0000],BL
0120 0000 ADD [BX+SI],AL
0122 0000 ADD [BX+SI],AL
0124 0000 ADD [BX+SI],AL
0126 0000 ADD [BX+SI],AL

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 = f.Ω≡ i |..†...
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF .....ff. ....
DS:0020 FF FF FF FF FF FF FF FF FF FF FF FF EB 19 C0 11 .....δ.L.
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
```

```
c01-01.LST X
c01-01.LST
Click here to ask Blackbox to help you code faster |
1      1      [org 0x0100]
2      2
3      3      ; start of code
4      4
5      5 00000000 B80500      mov ax, 5      ; move the constant 5 into register ax
6      6 00000003 BB0A00      mov bx, 10
7      7
8      8 00000006 01D8      add ax, bx      ; add value of bx into the value of ax
9      9
10     10 00000008 BB0F00      mov bx, 15      ; add constant 15 into the value of bx
11     11 0000000B 01D8      add ax, bx
12     12
13     13 0000000D B8004C      mov ax, 0x4c00      ; exit ..
14     14 00000010 CD21      int 0x21      ; .. is what the OS should do for me
15     15
16     16
17     17
18     18
19     19
20     20      ; watch the listing carefully
21
```

## Explanation :

The instruction "**mov ax, 0x4c00**" moves the hexadecimal value "**4c00**" into the "**ax**" register. In assembly language, this operation is represented by the opcode "**B8**" followed by the value to be moved.

# File c02-02 :

```
c02-01.LST x
c02-01.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers using memory variables
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1700] mov ax, [num1] ; load first number in ax
5 5 ; mov [num1], [num2] ; illegal
6 6 00000003 8B1E[1900] mov bx, [num2]
7 7 00000007 01D8 add ax, bx
8 8 00000009 8B1E[1B00] mov bx, [num3]
9 9 0000000D 01D8 add ax, bx
10 10 0000000F A3[1D00] mov [num4], ax
11 11 00000012 B8004C mov ax, 0x4c00
12 12 00000015 CD21 int 0x21
13 13
14 14
15 15 00000017 0500 num1: dw 5
16 16 00000019 0A00 num2: dw 10
17 17 0000001B 0F00 num3: dw 15
18 18 0000001D 0000 num4: dw 0
19 19
20 20
21 21 ; watch the listing carefully
22 22
```

```
c02-02.LST x
c02-02.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers accessed using a single label
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1700] mov ax, [num1]
5 5 00000003 8B1E[1900] mov bx, [num1 + 2] ; notice how we can do arithmetic here
6 6 00000007 01D8 add ax, bx ; also, why +2 and not +1?
7 7 00000009 8B1E[1B00] mov bx, [num1 + 4]
8 8 0000000D 01D8 add ax, bx
9 9 0000000F A3[1D00] mov [num1 + 6], ax ; store sum at num1+6
10 10 00000012 B8004C mov ax, 0x4c00
11 11 00000015 CD21 int 0x21
12 12
13 13 00000017 0500 num1: dw 5
14 14 00000019 0A00 dw 10
15 15 0000001B 0F00 dw 15
16 16 0000001D 0000 dw 0
17 17
```

## Difference :

The instruction "**mov bx, [num1 + 2]**" implies an arithmetic operation. It's fetching the value stored in the memory location labeled "**num1**", but it's also adding 2 to the address before

fetching the value. So, it's effectively accessing the memory location two bytes after "**num1**" and moving its value into the "**bx**" register.

Similarly the the other instructions on other lines as arrowed above works the same way

Here in file 1 each line defines a separate memory location labeled "**num1**", "**num2**", "**num3**", "**num4**", followed by values (5, 10, 15, 0) respectively .

The values **5**, **10**, **15**, and **0** are all defined under a single memory variable , labeled as "**num1**". And every addition of 2 to the memory addition points to the next value

As you can see there is no effect on the machine code, just the syntax is a bit different...

# File c02-03 :

```
c02-02.LST x c02-03.LST
c02-02.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers accessed using a single label
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1700] mov ax, [num1]
5 5 00000003 8B1E[1900] mov bx, [num1 + 2] ; notice how we can do arithmetic here
6 6 00000007 01D8 add ax, bx ; also, why +2 and not +1?
7 7 00000009 8B1E[1B00] mov bx, [num1 + 4]
8 8 0000000D 01D8 add ax, bx
9 9 0000000F A3[1D00] mov [num1 + 6], ax ; store sum at num1+6
10 10 00000012 B8004C mov ax, 0x4c00
11 11 00000015 CD21 int 0x21
12 12
13 13 00000017 0500 num1: dw 5
14 14 00000019 0A00 dw 10
15 15 0000001B 0F00 dw 15
16 16 0000001D 0000 dw 0
17
```

```
c02-02.LST c02-03.LST x
c02-03.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers accessed using a single label
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1700] mov ax, [num1]
5 5 00000003 8B1E[1900] mov bx, [num1 + 2]
6 6 00000007 01D8 add ax, bx
7 7 00000009 8B1E[1B00] mov bx, [num1 + 4]
8 8 0000000D 01D8 add ax, bx
9 9 0000000F A3[1D00] mov [num1 + 6], ax
10 10 00000012 B8004C mov ax, 0x4c00
11 11 00000015 CD21 int 0x21
12 12
13 13 00000017 05000A000F000000 num1: dw 5, 10, 15, 0
14 +
```



## Difference :

Here in **file 1** the variable/label num1 points to the **address** of the value **5** followed by **10,15** and **0** respectively whereas **0** is the result of all the arithmetic operations performed during the program.

The values **5, 10, 15**, and **0** are all defined under a single memory variable, labeled as "**num1**". And every addition of 2 to the memory address points to the next value

As you can see there is no effect on the machine code, just the syntax is a bit different...

**File c02-04 :**

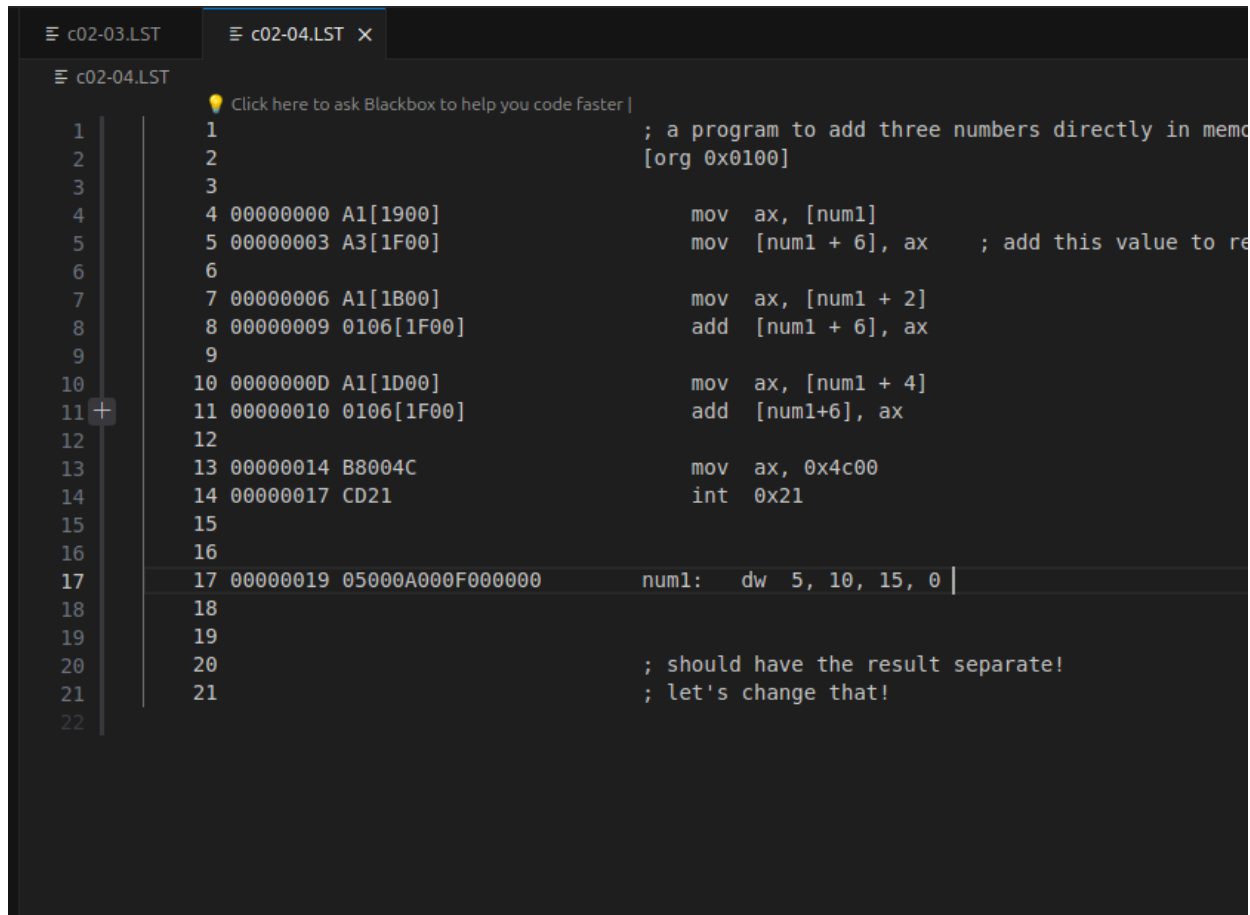
c02-03.LST X

c02-04.LST

c02-03.LST

Click here to ask Blackbox to help you code faster |

```
1      1      ; a program to add three numbers accessed using a  
2      2      [org 0x0100]  
3      3  
4      4 00000000 A1[1700]      mov ax, [num1]  
5      5 00000003 8B1E[1900]    mov bx, [num1 + 2]  
6      6 00000007 01D8          add ax, bx  
7      7 00000009 8B1E[1B00]    mov bx, [num1 + 4]  
8      8 0000000D 01D8          add ax, bx  
9      9 0000000F A3[1D00]      mov [num1 + 6], ax  
10     10 00000012 B8004C        mov ax, 0x4c00  
11     11 00000015 CD21          int 0x21  
12     12  
13     13 00000017 05000A000F000000 num1: dw 5, 10, 15, 0  
14
```



```
c02-04.LST x
c02-04.LST
Click here to ask Blackbox to help you code faster |
1      1      ; a program to add three numbers directly in memory
2      2      [org 0x0100]
3      3
4      4 00000000 A1[1900]      mov ax, [num1]
5      5 00000003 A3[1F00]      mov [num1 + 6], ax      ; add this value to result
6      6
7      7 00000006 A1[1B00]      mov ax, [num1 + 2]
8      8 00000009 0106[1F00]    add [num1 + 6], ax
9      9
10     10 0000000D A1[1D00]      mov ax, [num1 + 4]
11     11 00000010 0106[1F00]    add [num1+6], ax
12     12
13     13 00000014 B8004C      mov ax, 0x4c00
14     14 00000017 CD21      int 0x21
15     15
16     16
17     17 00000019 05000A000F000000 num1: dw 5, 10, 15, 0 |
18     18
19     19
20     20      ; should have the result separate!
21     21      ; let's change that!
22     22
```

## Difference :

- The main difference between the two files is how they handle the storage of the result:
  - **File 1** stores the result in the same memory area as the input numbers.
  - **File 2** stores the result separately from the input numbers.
- **File 1** modifies the original memory area containing the input numbers to hold the result, which may not be suitable if you want to preserve the original data.
- **File 2** keeps the input numbers intact and stores the result in a separate memory location, which is generally a cleaner approach.

### File 1 :

- **File 1** : uses registers (ax and bx) to perform arithmetic operations.
- **File 1** : uses the same memory location (num1) for both input numbers and the result.
- **File 1** : is more register-centric, with explicit loading and storing of values from and to memory.

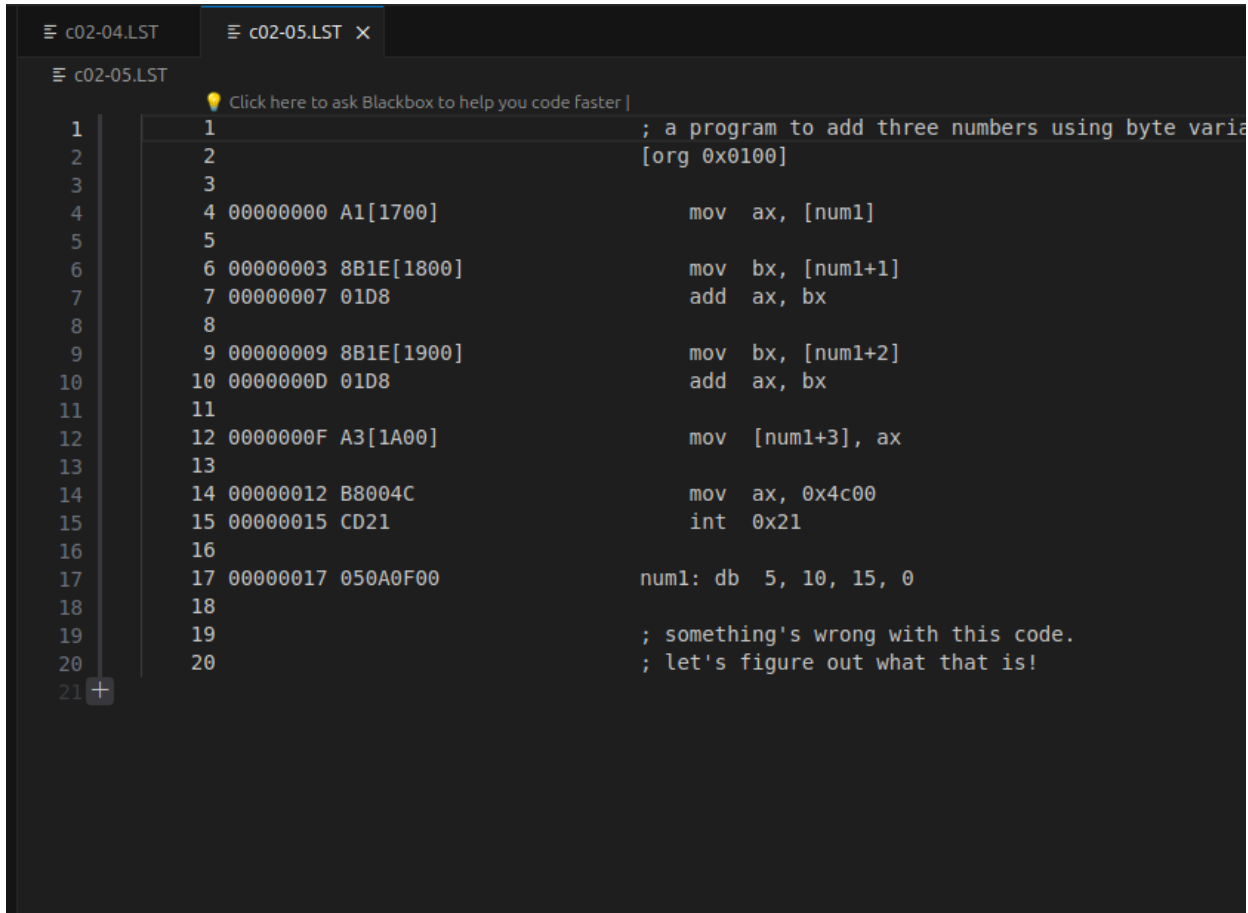
### File 2 :

- **File 2** : performs arithmetic operations directly in memory without using registers.

- **File 2** : separates the input numbers (num1) from the result (stored at num1 + 6).
- **File 2** : operates more directly in memory, with fewer register operations.

# File c02-05 :

```
c02-04.LST x c02-05.LST
c02-04.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers directly in memory
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1900] mov ax, [num1]
5 5 00000003 A3[1F00] mov [num1 + 6], ax ; add this value to register
6 6
7 7 00000006 A1[1B00] mov ax, [num1 + 2]
8 8 00000009 0106[1F00] add [num1 + 6], ax
9 9
10 10 0000000D A1[1D00] mov ax, [num1 + 4]
11 11 00000010 0106[1F00] add [num1+6], ax
12 12
13 13 00000014 B8004C mov ax, 0x4c00
14 14 00000017 CD21 int 0x21
15 15
16 16
17 + 17 00000019 05000A000F000000 num1: dw 5, 10, 15, 0 |
18 18
19 19
20 20 ; should have the result separate!
21 21 ; let's change that!
22 22
```



```
1 1 ; a program to add three numbers using byte variables
2 2 [org 0x0100]
3 3
4 4 00000000 A1[1700] mov ax, [num1]
5 5
6 6 00000003 8B1E[1800] mov bx, [num1+1]
7 7 00000007 01D8 add ax, bx
8 8
9 9 00000009 8B1E[1900] mov bx, [num1+2]
10 10 0000000D 01D8 add ax, bx
11 11
12 12 0000000F A3[1A00] mov [num1+3], ax
13 13
14 14 00000012 B8004C mov ax, 0x4c00
15 15 00000015 CD21 int 0x21
16 16
17 17 00000017 050A0F00 num1: db 5, 10, 15, 0
18 18
19 19 ; something's wrong with this code.
20 20 ; let's figure out what that is!
21 +
```

## Difference :

### File 1 :

- **File 1** : This program adds three numbers using memory directly.
- **File 1** : It loads each number from the num1 array into the ax register, performs addition operations, and then stores the result back into memory.
- **File 1**: Uses words (16 bits) for each number in the num1 array (dw directive).
- **File 1**: Uses only the ax register to perform arithmetic operations.
- Performs addition directly between numbers loaded into **AX** register and the memory location holding the result.
- **File 1**: Treats each number as a 16-bit word (2 bytes).
- **File 1**: Accesses memory in 16-bit chunks (word-by-word).

### File 2 :

- **File 2**: Uses bytes (8 bits) for each number in the num1 array (db directive).
- **File 2**: Uses both ax and bx registers; ax to load numbers, bx to hold the intermediate sum during addition.

- **File 2:** Performs addition using bx register to accumulate the sum before storing the result back into memory.
- **File 2:** Treats each number as an 8-bit byte.
- **File 2:** Accesses memory in 8-bit chunks (byte-by-byte).

## File c02-06 :

```

c02-05.LST x  c02-06.LST  c02-06B.lst  c02-07.LST
c02-05.LST
Click here to ask Blackbox to help you code faster |
1      1      ; a program to add three numbers using byte variables
2      2      [org 0x0100]
3      3
4      4 00000000 A1[1700]      mov ax, [num1]
5      5
6      6 00000003 8B1E[1800]     mov bx, [num1+1]
7      7 00000007 01D8          add ax, bx
8      8
9      9 00000009 8B1E[1900]     mov bx, [num1+2]
10     10 0000000D 01D8          add ax, bx
11     11
12     12 0000000F A3[1A00]      mov [num1+3], ax
13     13
14     14 00000012 B8004C        mov ax, 0x4c00
15     15 00000015 CD21          int 0x21
16     16
17     17 00000017 050A0F00      num1: db 5, 10, 15, 0
18     18
19     19      ; something's wrong with this code.
20     20      ; let's figure out what that is!
21

```

```

c02-05.LST  c02-06.LST x  c02-06B.lst  c02-07.LST
c02-06.LST
Click here to ask Blackbox to help you code faster |
1      1      ; a program to add three numbers using byte variables
2      2      [org 0x0100]
3      3      ; mov ax, 0x8787
4      4      ; xor ax, ax      ; We need to make sure AX is empty! Or do we?
5      5
6      6 00000000 8A26[1900]     mov ah, [num1]      ; Intel Software Developer Manual - Figure 3-5 (Page 76)
7      7
8      8 00000004 8A1E[1A00]     mov bl, [num1+1]
9      9 00000008 00FC          add ah, bh
10     10
11     11 0000000A 8A3E[1B00]     mov bh, [num1+2]
12     12 0000000E 00FC          add ah, bh
13     13
14     14 00000010 8B26[1C00]     mov [num1+3], ah
15     15
16     16 00000014 B8004C        mov ax, 0x4c00
17     17 00000017 CD21          int 0x21
18     18
19     19 00000019 050A0F00      num1: db 5, 10, 15, 0
20     20

```



## **Difference :**

### **File 1 :**

- File 1: Uses ax register to hold the accumulated sum and bx register to load each number from memory.
- File 1: Adds the numbers directly using the add instruction with the ax register.

### **File 2 :**

- File 2: Uses ah, bl, and bh registers to hold each byte of the numbers and accumulate the sum in ah.
- File 2: Adds the numbers using the add instruction with ah register and temporary bh register for each byte.

Both programs access memory using byte-by-byte chunks (db directive).

# File c02-06b :

```
c02-05.LST  c02-06.LST x  c02-06B.lst  c02-07.LST
c02-06.LST
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers using byte variables
2 2 [org 0x0100]
3 3 ; mov ax, 0x8787
4 4 ; xor ax, ax ; We need to make sure AX is empty! Or do we?
5 5
6 6 00000000 8A26[1900] mov ah, [num1] ; Intel Sotware Developer Manual - Figure 3-5 (Page 76)
7 7
8 8 00000004 8A1E[1A00] mov bl, [num1+1]
9 9 00000008 00FC add ah, bh
10 10
11 11 0000000A 8A3E[1B00] mov bh, [num1+2]
12 12 0000000E 00FC add ah, bh
13 13
14 14 00000010 8826[1C00] mov [num1+3], ah
15 15
16 16 00000014 B8004C mov ax, 0x4c00
17 17 00000017 CD21 int 0x21
18 18
19 19 00000019 050A0F00 num1: db 5, 10, 15, 0
20 +
```

```
c02-05.LST  c02-06.LST  c02-06B.lst x  c02-07.LST
c02-06B.lst
Click here to ask Blackbox to help you code faster |
1 1 ; a program to add three numbers using byte variables
2 2 [org 0x0100]
3 3
4 4 00000000 B80787 mov ax, 0x8787
5 5 00000003 31C0 xor ax, ax ; we need to make sure AX is empty!
6 6
7 7 00000005 A0[1C00] mov al, [num1]
8 8
9 9 00000008 8A1E[1D00] mov bl, [num1+1]
10 10 0000000C 00D8 add al, bl
11 11
12 12 0000000E 8A1E[1E00] mov bl, [num1+2]
13 13 00000012 00D8 add al, bl
14 14
15 15 00000014 A2[1F00] mov [num1+3], al
16 16
17 17
18 18
19 19 ; mov ax, bl ; ... assemble time error. Make sure you understand the error!
20 20
21 21 00000017 B8004C mov ax, 0x4c00
22 22 0000001A CD21 int 0x21
23 23
24 24 0000001C 050A0F00 num1: db 5, 10, 15, 0
25 25
```

## Difference :

In x86 assembly language:

- AH and AL are the higher and lower halves, respectively, of the AX register, a 16-bit general-purpose register.
- BH and BL are the higher and lower halves, respectively, of the BX register, another 16-bit general-purpose register.

So, AH and AL represent the high and low bytes of the AX register, while BH and BL represent the high and low bytes of the BX register.

## File 1:

Line 6:

- Loads the first number from the num1 array into the ah register.

Line 8:

- Loads the second number from the num1 array into the bl register.

Line 9:

- Incorrect operation. It tries to add the values in ah and bh registers, which are uninitialized and hold garbage values. It should be adding ah and bl.

Line 11:

- Loads the third number from the num1 array into the bh register.

Line 12:

- Incorrect operation. It again tries to add the values in ah and bh registers, which are uninitialized and hold garbage values. It should be adding ah and bh.

Line 14:

- Stores the result, which is the sum in ah, into the memory location num1 + 3.

## File 2:

Lines 4-5:

- Initializes ax with the value 0x8787 and then clears it to make sure it's empty.

Line 7:

- Loads the first number from the num1 array into the al register.

Line 9:

- Loads the second number from the num1 array into the bl register.

Line 10:

- Adds the values in al and bl registers.

Line 12:

- Loads the third number from the num1 array into the bl register.

Line 13:

- Adds the values in al and bl registers.

Line 15:

- Stores the result, which is the sum in al, into the memory location num1 + 3.

## File c02-07 :

```
c02-05.LST  c02-06.LST  c02-06B.lst x  c02-07.LST
c02-06B.lst
1 1 ; a program to add three numbers using byte variables
2 2 [org 0x0100]
3 3
4 4 00000000 B88787 mov ax, 0x8787
5 5 00000003 31C0 xor ax, ax ; we need to make sure AX is empty!
6 6
7 7 00000005 A0[1C00] mov al, [num1]
8 8
9 9 00000008 8A1E[1D00] mov bl, [num1+1]
10 10 0000000C 09D8 add al, bl
11 11
12 12 0000000E 8A1E[1E00] mov bl, [num1+2]
13 13 00000012 09D8 add al, bl
14 14
15 15 00000014 A2[1F00] mov [num1+3], al
16 16
17 17
18 18
19 19 ; mov ax, bl ; ... assemble time error. Make sure you understand the error!
20 20
21 21 00000017 B8004C mov ax, 0x4c00
22 22 0000001A CD21 int 0x21
23 23
24 24 0000001C 050A0F00 num1: db 5, 10, 15, 0
25 25
```

```
c02-05.LST  c02-06.LST  c02-06B.lst  c02-07.LST x
c02-07.LST
1 1 ; a program to add three numbers using byte variables
2 2 [org 0x0100]
3 3 00000000 31C0 xor ax, ax ; check effect on ZF
4 4
5 5 00000002 BB[1F00] mov bx, num1
6 6
7 7 00000005 0307 add ax, [bx]
8 8 00000007 81C30200 add bx, 2
9 9
10 10 00000008 0307 add ax, [bx]
11 11 0000000D 81C30200 add bx, 2
12 12
13 13 00000011 0307 add ax, [bx]
14 14 00000013 81C30200 add bx, 2
15 15
16 16
17 17 00000017 A3[2500] mov [result], ax
18 18
19 19 0000001A B8004C mov ax, 0x4c00
20 20 0000001D CD21 int 0x21
21 21
22 22
23 23 ; to turn this into an iteration, we need a couple of things:
24 24 ; - branching instruction
25 25 ; - checking constraints -- e.g. c > 0 ; Intel Software Developer Manual - Figure 3-8 (Page 80)
26 26
27 27
28 28 0000001F 0500A000F00 num1: dw 5, 10, 15
29 29 00000025 0000 result: dw 0
30 30
```

Difference :

File 1:

- AX: Initialized with 0x8787, then cleared to ensure it's empty.
- AL and BL: Used to sequentially add byte values from memory.

## File 2:

- AX: Cleared using xor operation for subsequent addition operations.
- BX: Used as a pointer to access byte values in memory sequentially.
- Result: Stores the final sum computed in AX.