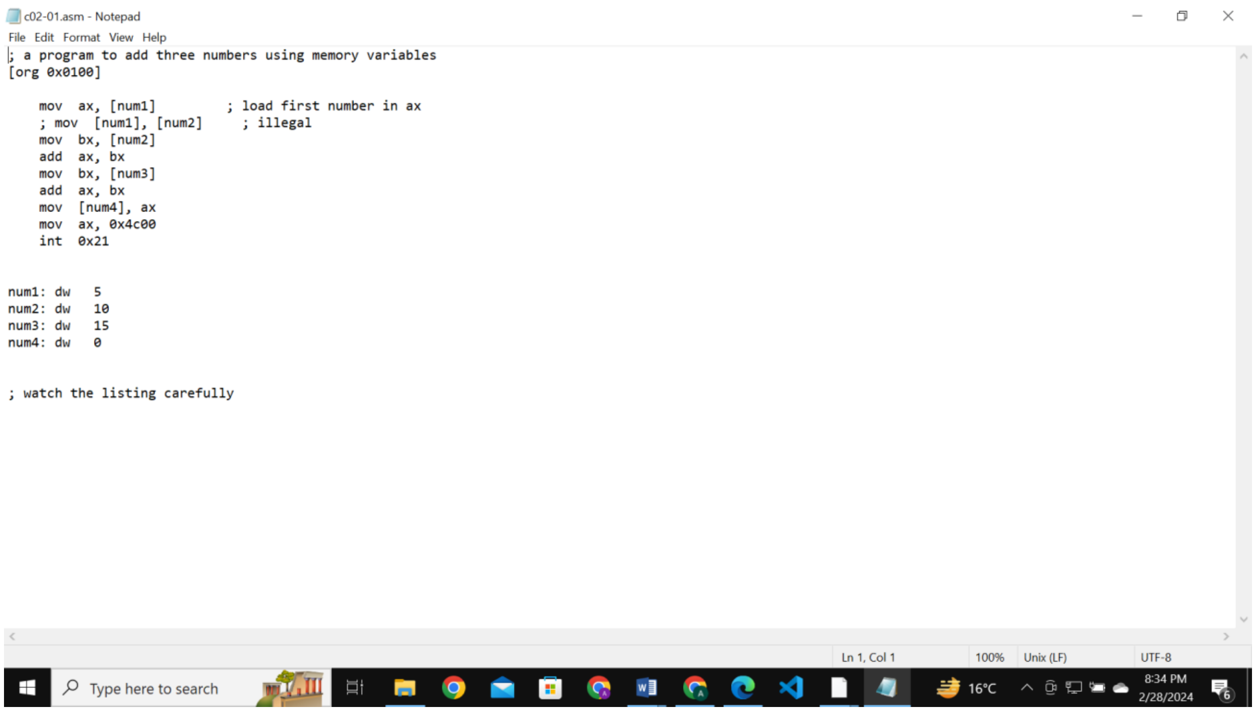
Name: Amber khurshid

Section: BAI-4A

Roll No: 22P-9295

COAL LAB TASK #03

This programs adds three numbers using memory variables.



```
; 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


; watch the listing carefully
```

Step **1:**

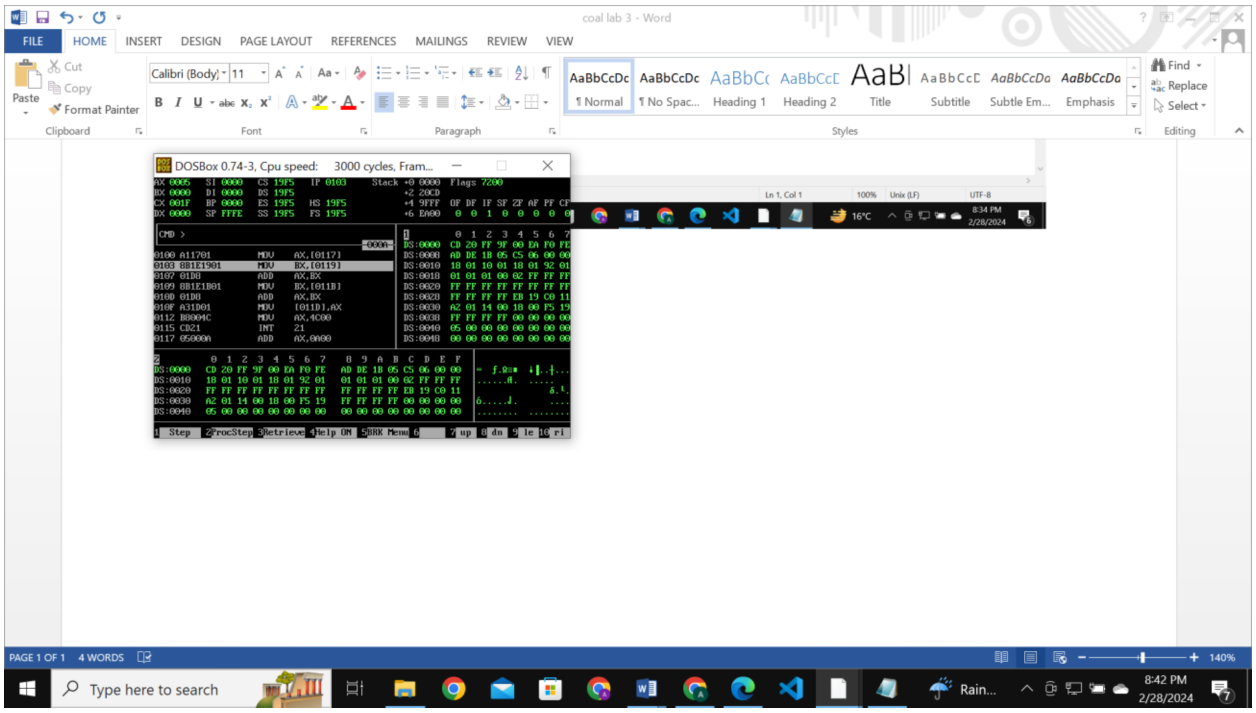
mov ax, [num1]

In assembly language it is given as:

0100 A11701

0100 shows the memory address of the instruction.

A11701 here A1 is opcode for mov instruction and 1701 is the address from where the value is to be moved in ax register.



```
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    1          0  1  2  3  4  5  6  7
0100 A11701         MOV    AX,[0117]            DS:0000  CD 20 FF 9F 00 EA F0 FE
0103 8B1E1901       MOV    BX,[0119]            DS:0008  AD DE 1B 05 C5 06 00 00
0107 01D8           ADD    AX,BX                DS:0010  18 01 10 01 18 01 92 01
0109 8B1E1B01       MOV    BX,[011B]            DS:0018  01 01 01 00 02 FF FF FF
010D 01D8           ADD    AX,BX                DS:0020  FF FF FF FF FF FF FF FF
010F A31D01         MOV    [011D],AX            DS:0028  FF FF FF FF EB 19 C0 11
0112 B8004C         MOV    AX,4C00              DS:0030  A2 01 14 00 18 00 F5 19
0115 CD21           INT    21                   DS:0038  FF FF FF FF 00 00 00 00
0117 05000A         ADD    AX,0A00              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

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

Step 2:

mov bx, [num2]

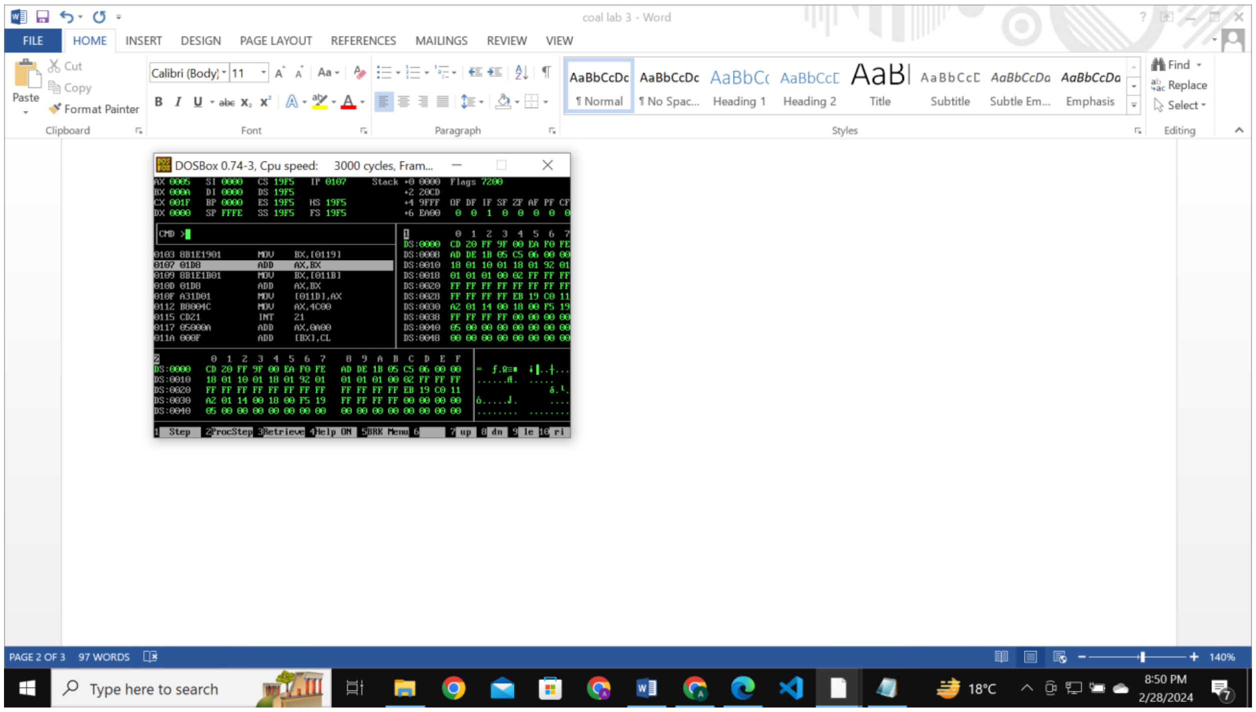
In assembly language it is given as:

0103 8B1E1901

0103 shows the memory address of the instruction.

8B1E1901 here 8B is opcode for mov instruction and 1E1901 is the address from where the value is to be moved in ax register.

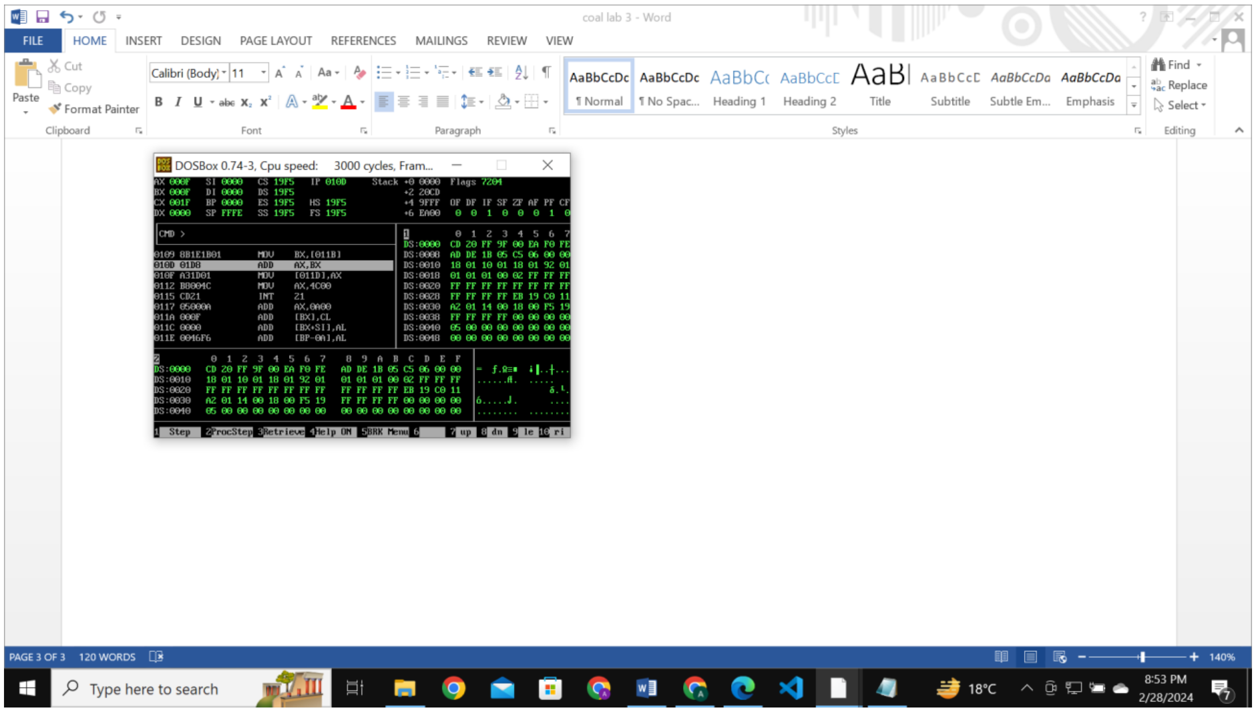
Here we are adding the value stored in the memory location "num3" having address 011B into the BX register.



```
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]
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

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

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

Step 3:

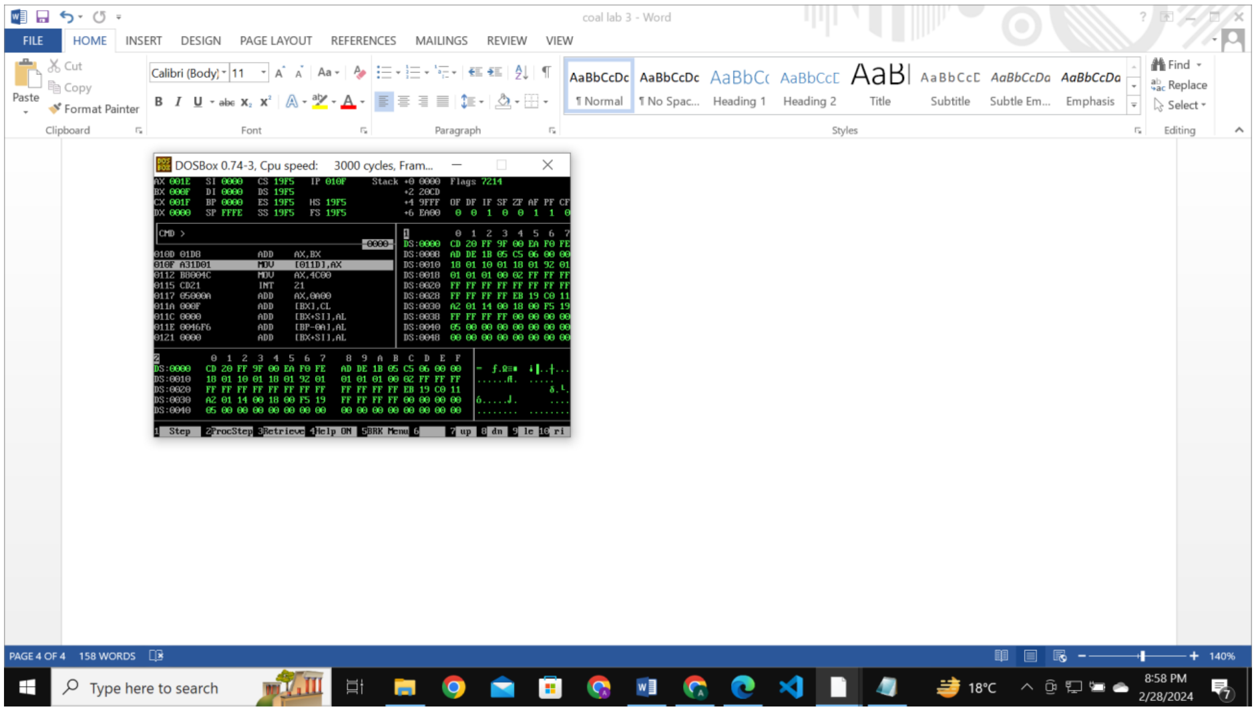
Here we are adding the value from bx register into ax and result is then stored in ax register.



```
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 0046F6           ADD     [BP-0A],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

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

Step 4:

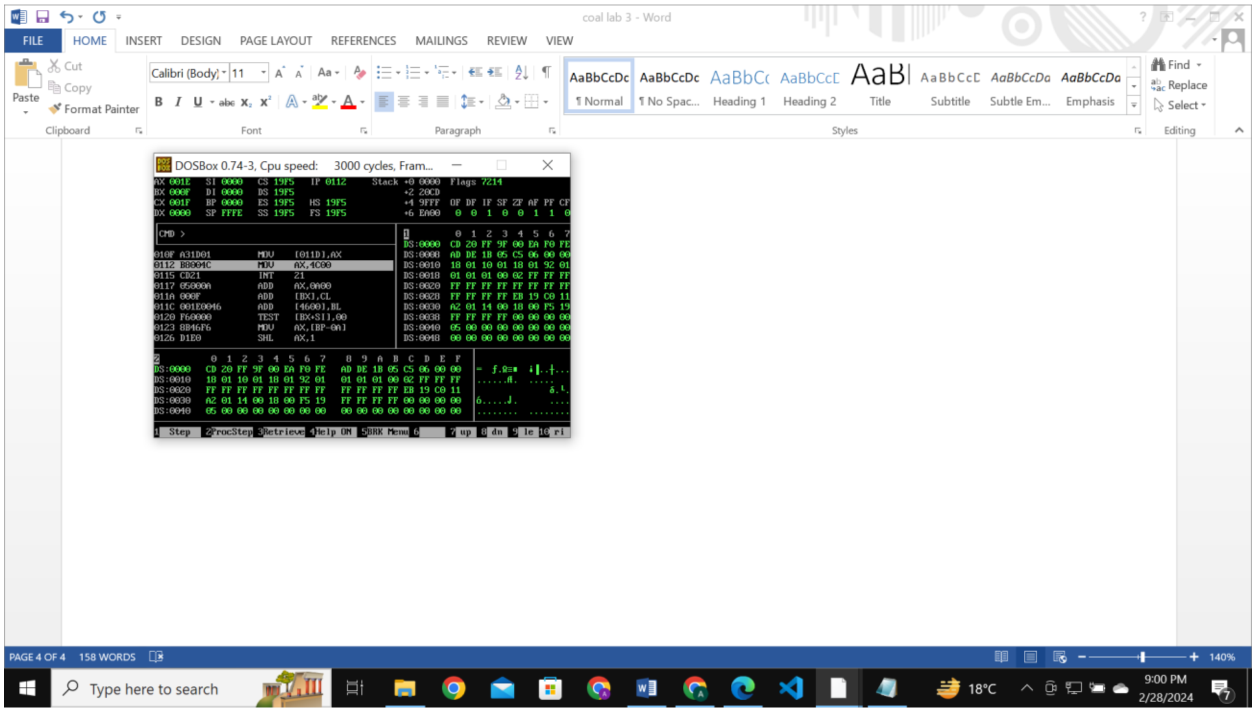
Here the value stored in ax register is moved into the memory location num4. This operation is represented by the opcode ‘a3’ with the address where the value should be stored.



```
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 >                                        1          0  1  2  3  4  5  6  7
                                      0000   DS:0000  CD 20 FF 9F 00 EA F0 FE
010D 01D8           ADD    AX,BX             DS:0008  AD DE 1B 05 C5 06 00 00
010F A31D01         MOV    [011D],AX         DS:0010  18 01 10 01 18 01 92 01
0112 B8004C         MOV    AX,4C00           DS:0018  01 01 01 00 02 FF FF FF
0115 CD21           INT    21                DS:0020  FF FF FF FF FF FF FF FF
0117 05000A         ADD    AX,0A00           DS:0028  FF FF FF FF EB 19 C0 11
011A 000F           ADD    [BX],CL           DS:0030  A2 01 14 00 18 00 F5 19
011C 0000           ADD    [BX+SI],AL        DS:0038  FF FF FF FF 00 00 00 00
011E 0046F6         ADD    [BP-0A],AL        DS:0040  05 00 00 00 00 00 00 00
0121 0000           ADD    [BX+SI],AL        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

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

Step 5:

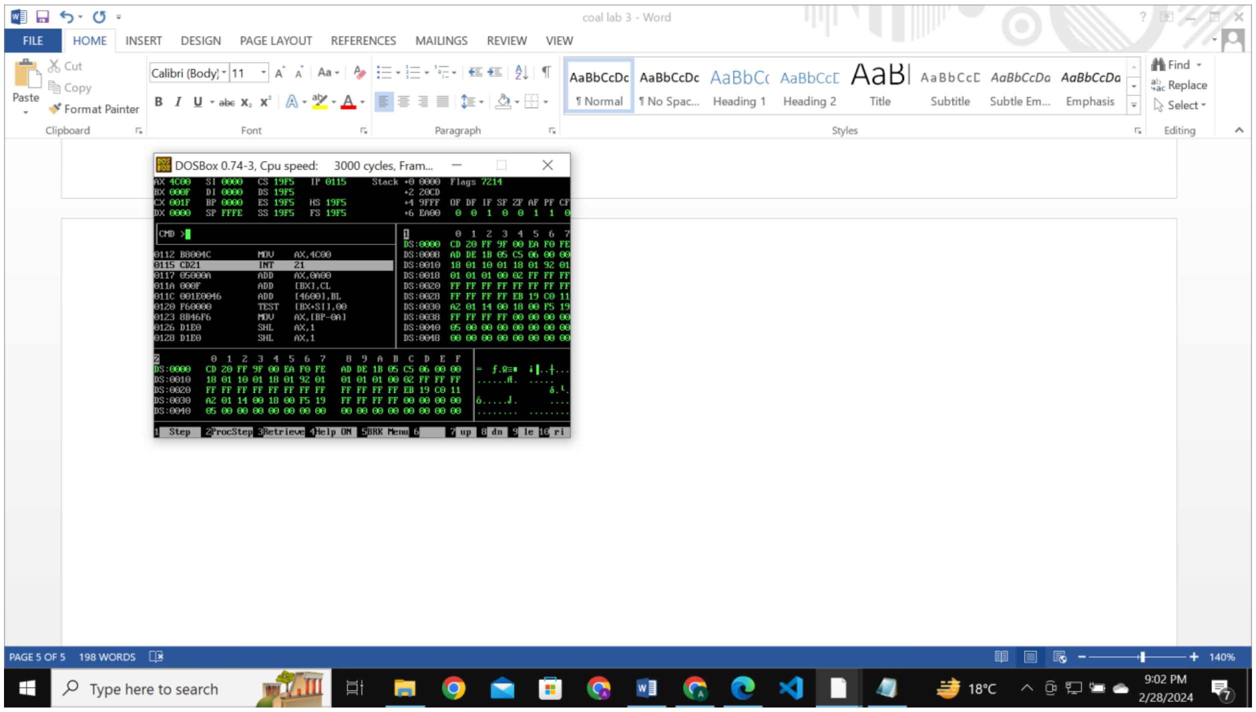
The instruction "mov ax, 0x4c00" moves the hexadecimal value "4c00" into the "ax" register. In assembly language, this operation is represented by the opcode "B8" followed by the value to be moved.



```
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 >                                        1        0  1  2  3  4  5  6  7
                                             DS:0000  CD 20 FF 9F 00 EA F0 FE
010F A31D01         MOV     [011D],AX        DS:0008  AD DE 1B 05 C5 06 00 00
0112 B8004C         MOV     AX,4C00          DS:0010  18 01 10 01 18 01 92 01
0115 CD21           INT     21               DS:0018  01 01 01 00 02 FF FF FF
0117 05000A         ADD     AX,0A00          DS:0020  FF FF FF FF FF FF FF FF
011A 000F           ADD     [BX],CL          DS:0028  FF FF FF FF EB 19 C0 11
011C 001E0046       ADD     [4600],BL        DS:0030  A2 01 14 00 18 00 F5 19
0120 F60000         TEST    [BX+SI],00       DS:0038  FF FF FF FF 00 00 00 00
0123 8B46F6         MOV     AX,[BP-0A]       DS:0040  05 00 00 00 00 00 00 00
0126 D1E0           SHL     AX,1             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

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

Step 6:

Here we have moved the hexadecimal value "4c00" into the "ax" register and is represented by the opcode "B8" with the value to be moved.



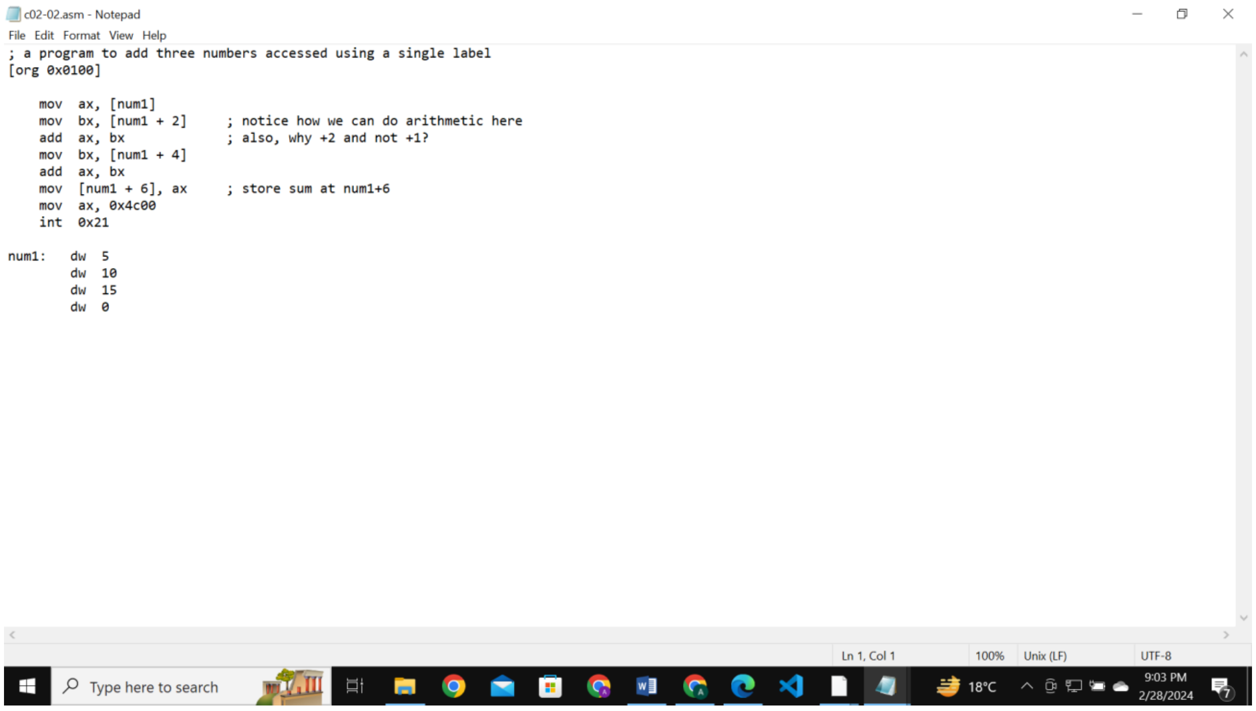
```
AX 4C00   SI 0000   CS 19F5   IP 0115      Stack +0 0000  Flags 7214
BX 000F   DI 0000   DS 19F5                      +2 20CD
CX 001F   BP 0000   ES 19F5   HS 19F5            +4 9FFF  OF DF IF SF ZF AF PF CF
DX 0000   SP FFFE   SS 19F5   FS 19F5            +6 EA00   0  0  1  0  0  1  1  0

CMD >                                        1        0  1  2  3  4  5  6  7
                                             DS:0000  CD 20 FF 9F 00 EA F0 FE
0112 B8004C         MOV     AX,4C00          DS:0008  AD DE 1B 05 C5 06 00 00
0115 CD21           INT     21               DS:0010  18 01 10 01 18 01 92 01
0117 05000A         ADD     AX,0A00          DS:0018  01 01 01 00 02 FF FF FF
011A 000F           ADD     [BX],CL          DS:0020  FF FF FF FF FF FF FF FF
011C 001E0046       ADD     [4600],BL        DS:0028  FF FF FF FF EB 19 C0 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
0126 D1E0           SHL     AX,1             DS:0040  05 00 00 00 00 00 00 00
0128 D1E0           SHL     AX,1             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

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

C02-02.asm

The instruction "mov bx, [num1 + 2]" is an arithmetic operation.

It is adding 2 to the address and getting the value stored there and moving it into “bx” register.

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

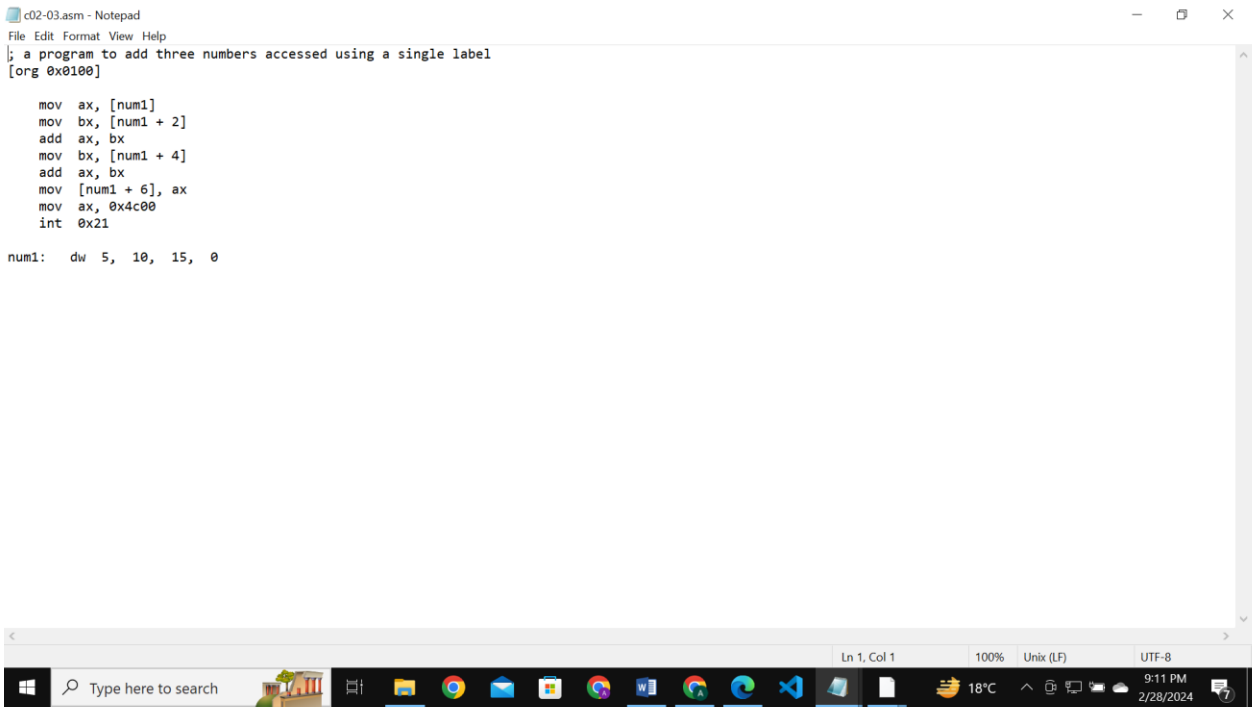


```
; a program to add three numbers accessed using a single label
[org 0x0100]

    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
```

C02-03.asm

C02-02.asm and c02-03.asm

Here the label num1 points to the address of the value 5 ,10,15 and 0 . 0 contains the result of all the arithmetic operations performed. And every addition of 2 to the memory points to the next value.

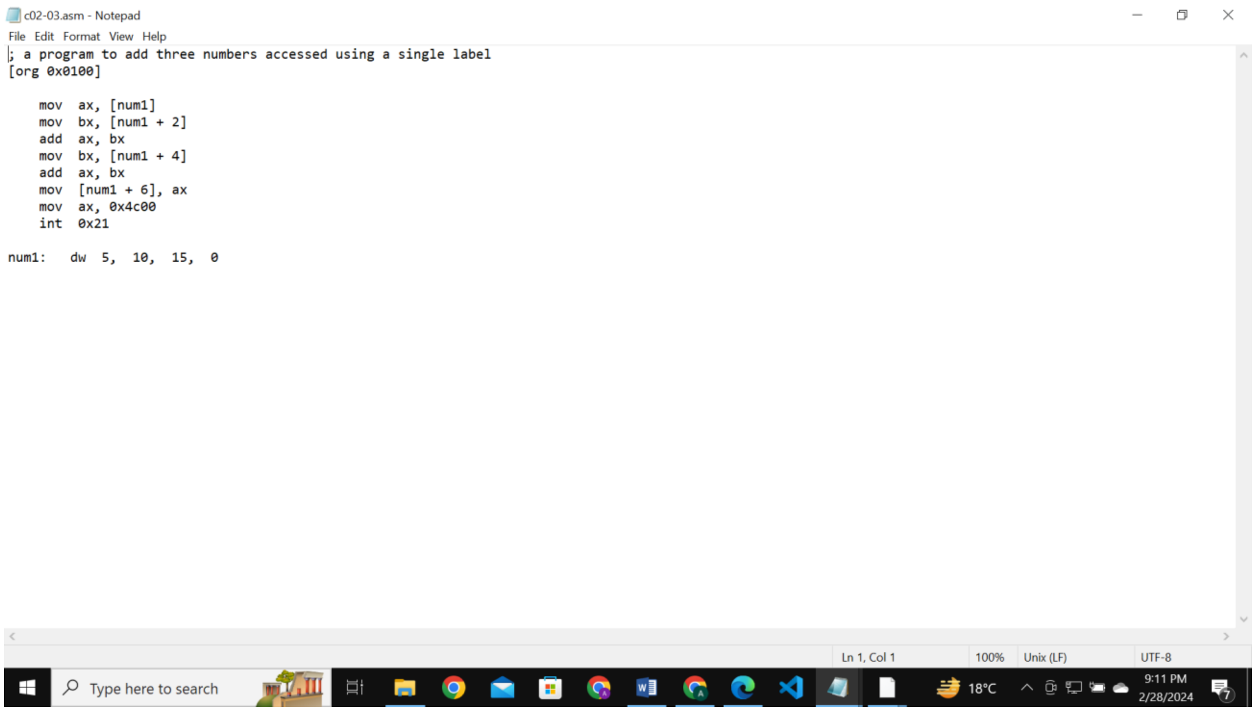


```
; a program to add three numbers accessed using a single label
[org 0x0100]

    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
```

C02-04.asm

C02-03.asm and c02-04.asm

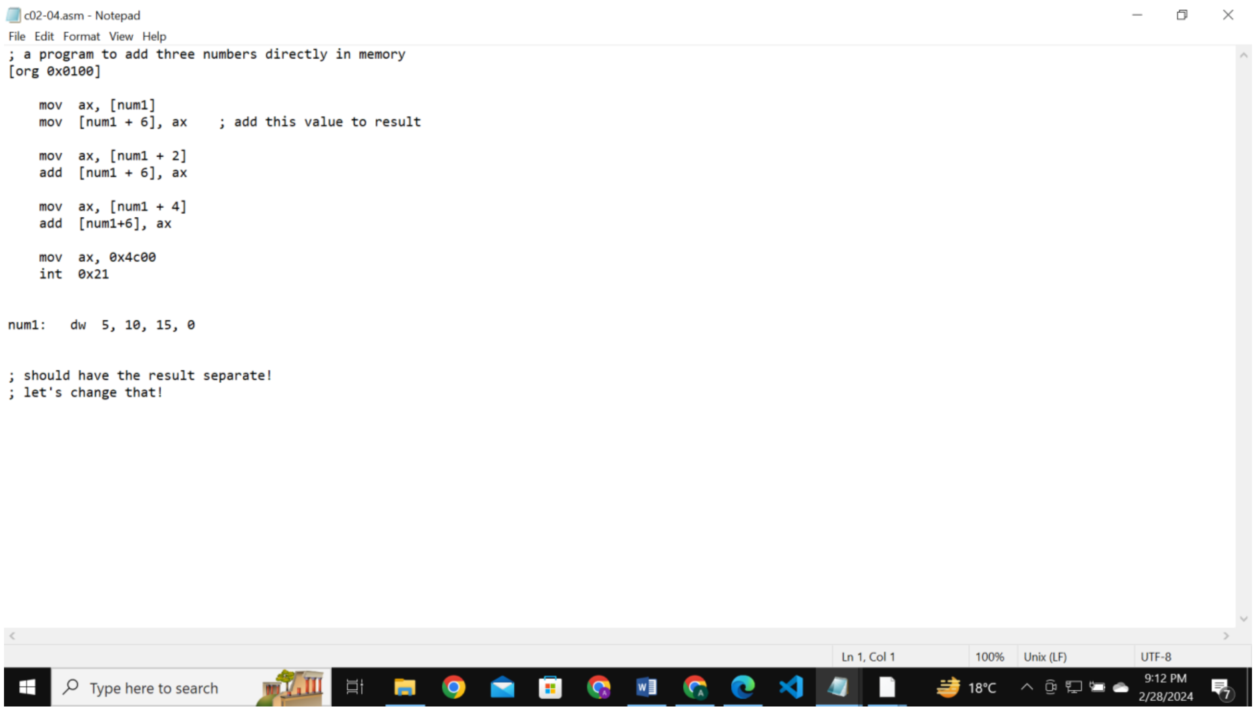
The two provided code files differ primarily in their handling of result storage. In File 1, the result is stored in the same memory space as the input numbers, potentially altering the original data. On the other hand, File 2 adopts a cleaner approach by keeping the input numbers intact and storing the result separately. File 1 utilizes registers (ax and bx) for arithmetic operations and employs the same memory location (num1) for both input and result, emphasizing a register-centric methodology. In contrast, File 2 directly performs arithmetic operations in memory without extensive register use, separating input numbers (num1) from the result, which is stored at num1 + 6. This approach operates more straightforwardly in memory with fewer register operations, contributing to a more streamlined code structure.


c02-03.asm - Notepad

File Edit Format View Help

```
; a program to add three numbers accessed using a single label
[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
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8


c02-04.asm - Notepad

File Edit Format View Help

```
; a program to add three numbers directly in memory
[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!
```

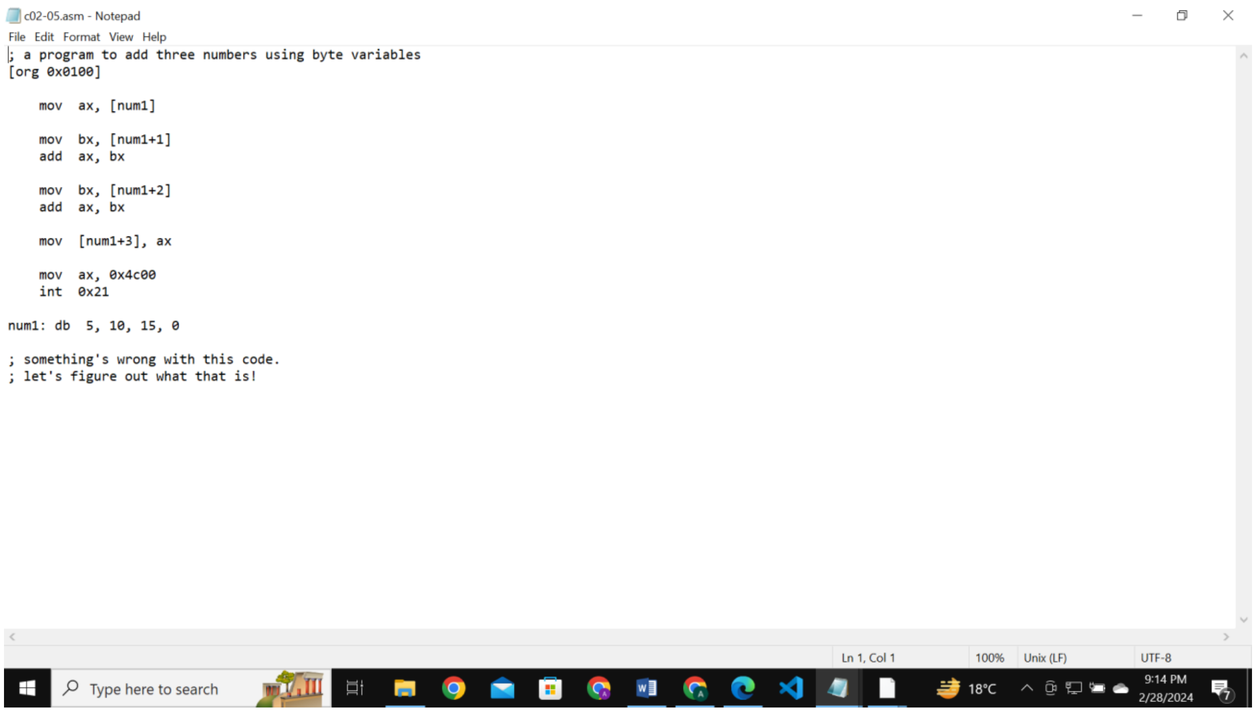
Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

C02-05.asm

C02-04.asm and c02-05.asm

The provided code represents a program designed to add three numbers using byte variables. However, there appears to be an issue in the code. In File 1, the program loads each number from the num1 array into the ax register, performs addition operations, and then stores the result back into memory. It utilizes words (16 bits) for each number in the num1 array and employs only the ax register to execute arithmetic operations. The code treats each number as a 16-bit word and accesses memory in 16-bit chunks (word-by-word).

In contrast, File 2 uses bytes (8 bits) for each number in the num1 array, employing both ax and bx registers. Ax is used to load numbers, while bx holds the intermediate sum during addition. The program performs addition using the bx register to accumulate the sum before storing the result back into memory. File 2 treats each number as an 8-bit byte and accesses memory in 8-bit chunks (byte-by-byte). The distinction in data size between the two files (words in File 1 and bytes in File 2) might be the source of the issue. Ensure consistency in data size and adjust the code accordingly for a successful execution.



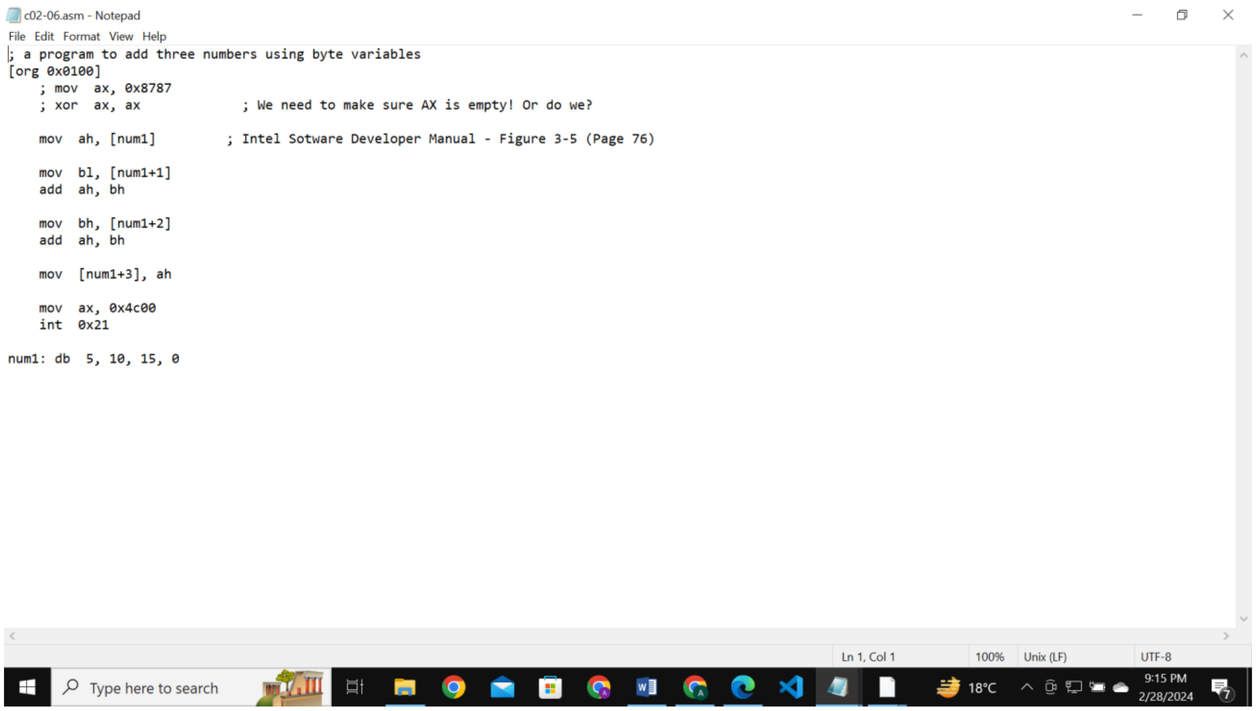
```
; a program to add three numbers using byte variables
[org 0x0100]

    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!
```

C02-06.asm

C02-05.asm and c02-06.asm

The provided code constitutes a program aimed at adding three numbers using byte variables. In File 1, the ax register is utilized to accumulate the sum, and the bx register is employed to load each number from memory. The addition operation is directly performed using the add instruction with the ax register.

Contrastingly, File 2 utilizes the ah, bl, and bh registers to manage each byte of the numbers and accumulate the sum in the ah register. The addition of numbers is executed using the add instruction with the ah register, along with a temporary bh register for each byte. Both programs access memory using byte-by-byte chunks (db directive). The commented-out code suggests a consideration for ensuring the ax register is empty, although the necessity for this is left open-ended. The provided code demonstrates a byte-oriented approach, emphasizing the manipulation of individual bytes in the registers for the arithmetic operations.



```
; a program to add three numbers using byte variables
[org 0x0100]
    ; mov  ax, 0x8787
    ; xor  ax, ax            ; We need to make sure AX is empty! Or do we?

    mov  ah, [num1]        ; Intel Sotware Developer Manual - Figure 3-5 (Page 76)

    mov  bl, [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
```

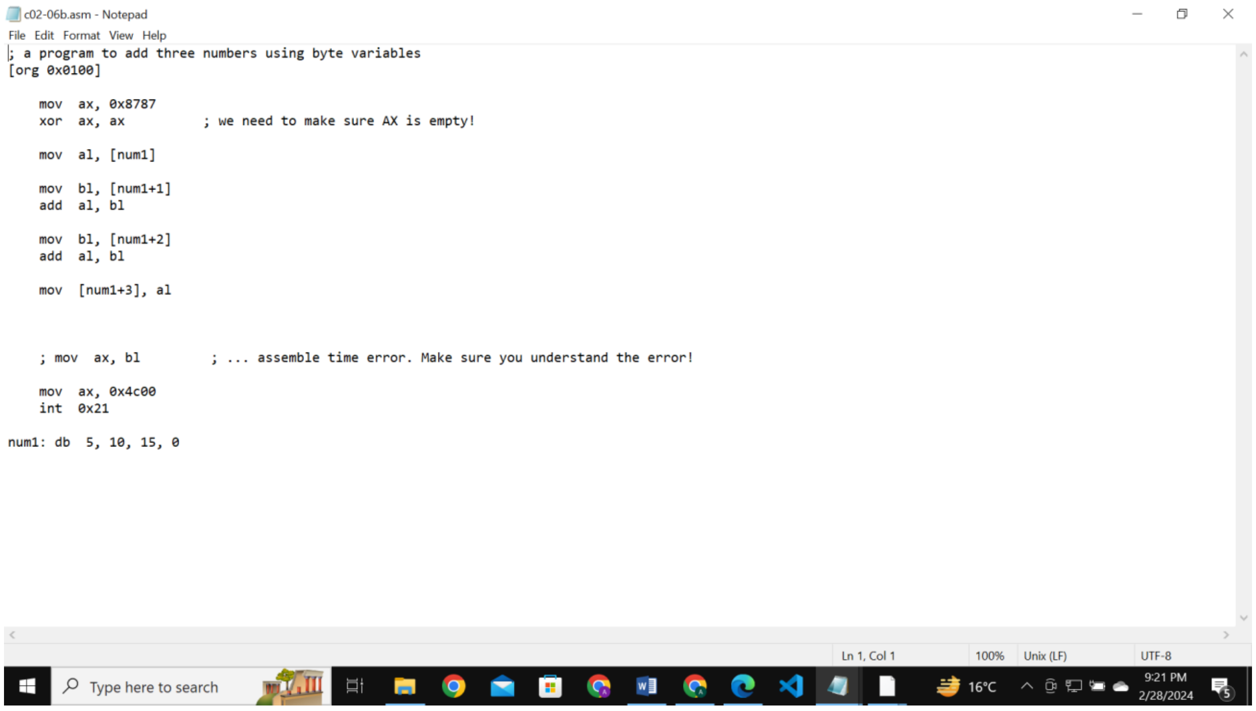
C02-06b.asm

C02-06.asm and c02-06b.asm

In x86 assembly language, AH and AL constitute the higher and lower halves, respectively, of the AX register, which is a 16-bit general-purpose register. Similarly, BH and BL represent the higher and lower halves of the BX register, another 16-bit general-purpose register. Therefore, AH and AL signify the high and low bytes of the AX register, while BH and BL represent the high and low bytes of the BX register.

Regarding File 1, on Line 6, it loads the first number from the num1 array into the ah register. On Line 8, it loads the second number into the bl register. However, Line 9 contains an error as it attempts to add the uninitialized values in ah and bh registers, which hold garbage values. The correct operation should involve adding ah and bl. On Line 11, the third number is loaded into the bh register, and on Line 12, there's a similar error in trying to add the uninitialized values in ah and bh registers instead of ah and bh. Finally, Line 14 stores the result, the sum in ah, into the memory location num1 + 3.

In File 2, Lines 4-5 initialize ax with the value 0x8787 and then clear it to ensure it's empty. Line 7 loads the first number from the num1 array into the al register, and Line 9 loads the second number into the bl register. Lines 10 and 13 correctly add the values in al and bl registers. Lines 12 and 15 handle the third number and store the result, the sum in al, into the memory location num1 + 3.



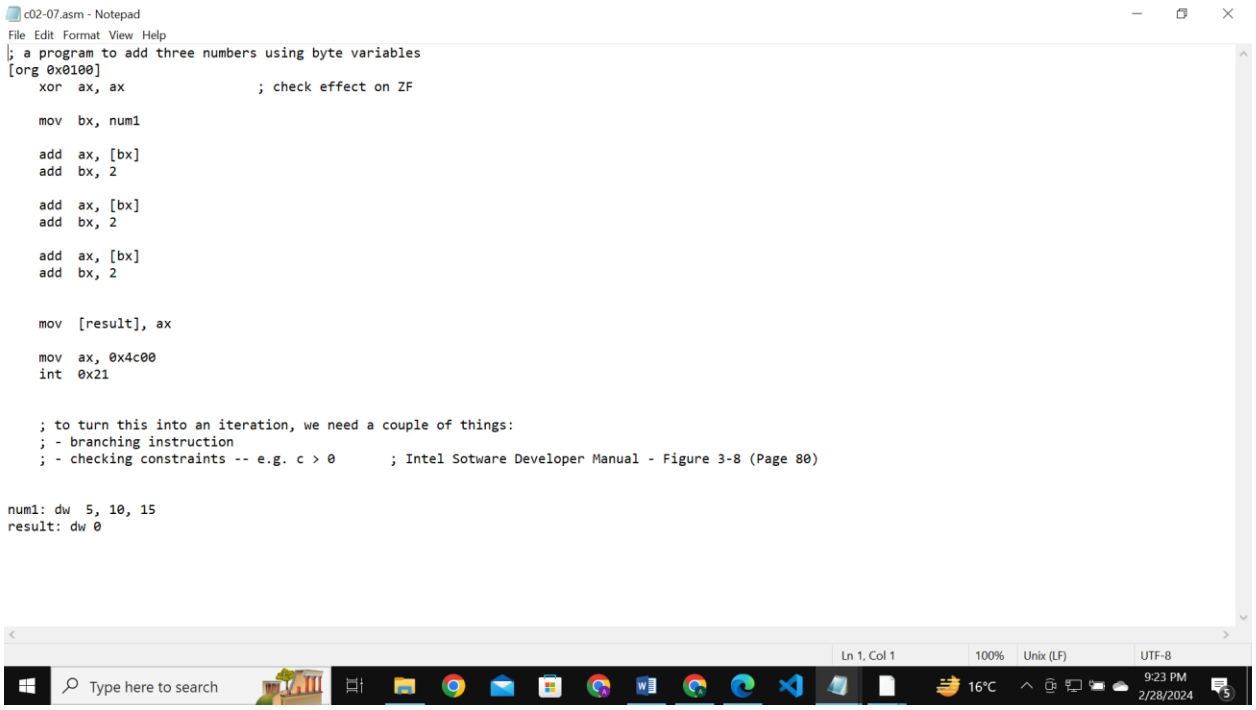
```
; a program to add three numbers using byte variables
[org 0x0100]

    mov  ax, 0x8787
    xor  ax, ax          ; we need to make sure AX is empty!

    mov  al, [num1]

    mov  bl, [num1+1]
    add  al, bl

    mov  bl, [num1+2]
    add  al, bl

    mov  [num1+3], al


    ; mov  ax, bl         ; ... assemble time error. Make sure you understand the error!

    mov  ax, 0x4c00
    int  0x21

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

C02-07.asm

C02-06b and C02-07.asm

In File 1, AX is first initialized with 0x8787 and then cleared to ensure it is empty. Subsequently, AL and BL are employed to sequentially add byte values from memory.

On the other hand, in File 2, AX is cleared using the XOR operation in preparation for subsequent addition operatins. Meanwhile, BX serves as a pointer to access byte values in memory sequentially. The result of the computations is stored in AX, representing the final sum.



```
; a program to add three numbers using byte variables
[org 0x0100]
    xor  ax, ax                 ; check effect on ZF

    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


    ; to turn this into an iteration, we need a couple of things:
    ; - branching instruction
    ; - checking constraints -- e.g. c > 0       ; Intel Sotware Developer Manual - Figure 3-8 (Page 80)


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