Name : Tazmeen Afroz
Roll No: 22p-9252
Section: BAI-4A

COAL LAB TASK 3

file 1 : 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


; watch the listing carefully
```

1. **Data Section:**

- The lines num1: dw 5, num2: dw 10, num3: dw 15, and num4: dw 0 define four **data words** (16 Bits)named num1, num2, num3, and num4. These words **reserve memory locations** and **initialize them with the given values** (5, 10, 15, and 0, respectively).

2. **Code Section:**

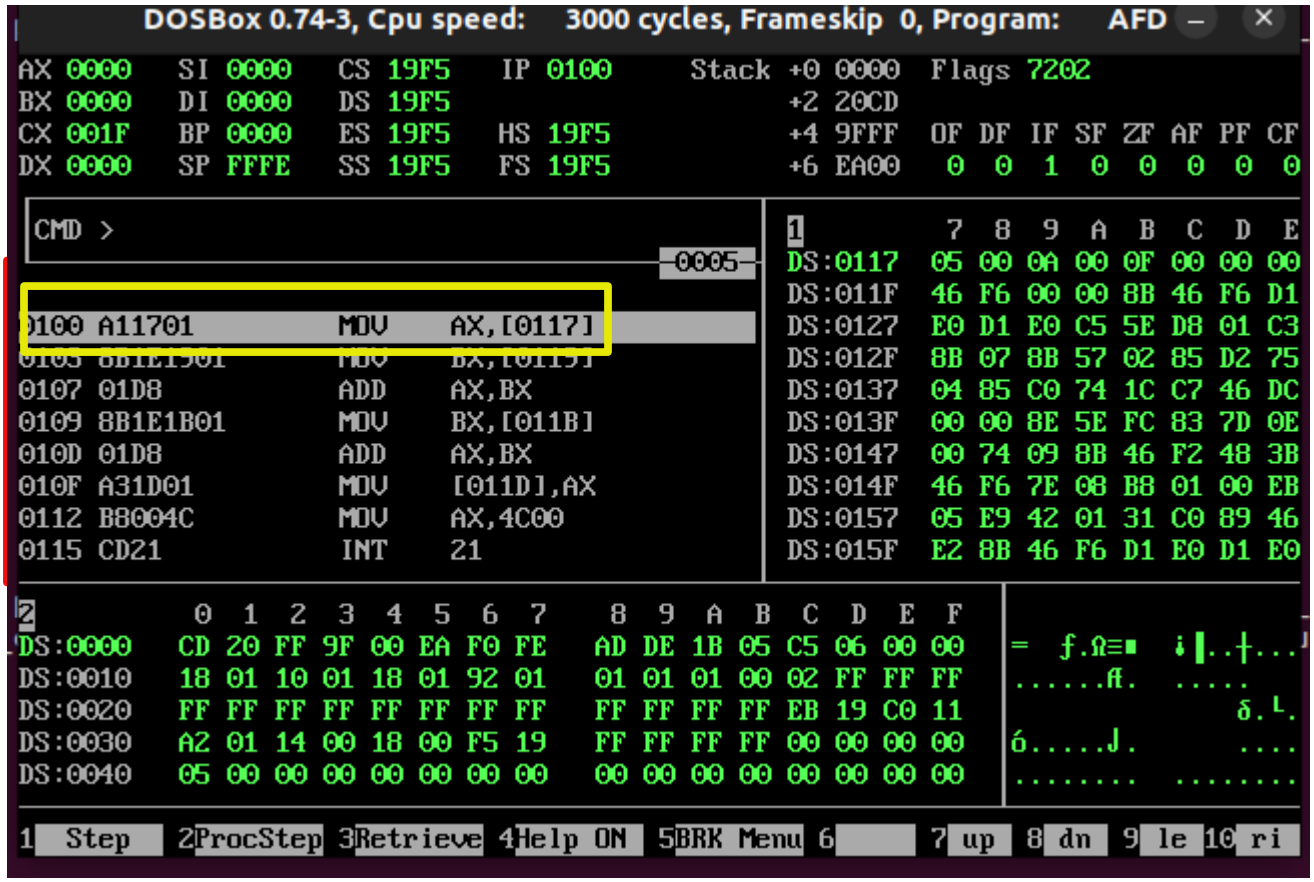
- **Instruction 1:** mov ax, [num1]

- This instruction **loads the value stored at the memory address of num1 into the ax register.** Since num1 holds 5, this instruction effectively moves 5 into ax.
- The [] brackets indicate **memory operands** (accessing data at a specific memory location).

- **Instruction 2:** mov [num1], [num2] (illegal)

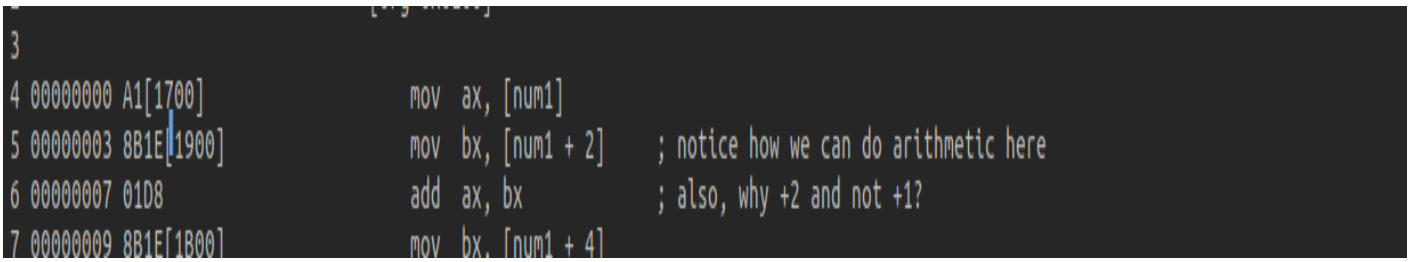
  In older computer architectures, directly moving data between memory locations without going through registers was generally not possible due to limitations in the bus system.

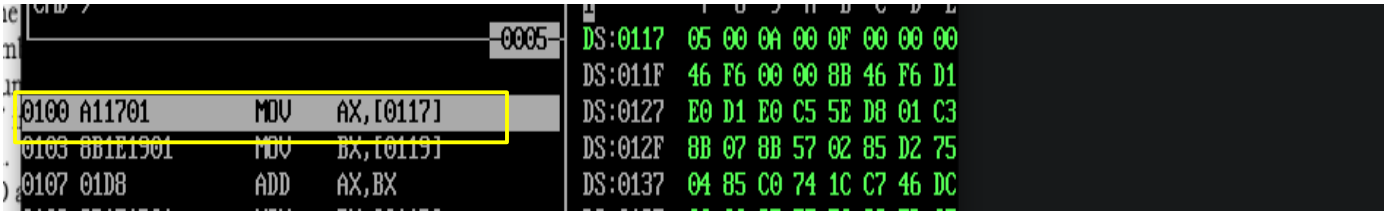
- **Instruction 7:** mov [num4], ax
- This instruction **stores the value in ax (30) into the memory location of num4.**



- **The first seven instructions in your assembly code occupy a total of 23 bytes (0x17 in hexadecimal).**

- **The label num1 serves as a symbolic reference for the memory address where the value 5 is stored. During assembly, this label is replaced with its actual address, which is 0x0117 in this case.**
- **Since the program starts at memory address 0x0100, num1 resides at an offset of 23 bytes (0x17) from the program's base. This translates to an absolute memory address of 0x0117.**
-





- **1. Instruction Change:**
- The original instruction (B80500) directly loaded a value into the register.
- The assembler replaced it with A11701 to load the value from the memory location labeled num1.
- **2. Why the Change?**
- num1 is a variable, and its actual address depends on the program's location in memory.
- Using the memory address ensures the correct value is loaded, regardless of where the program is loaded.
- **3. Address Adjustment:**
- The number 117 in the instruction might seem wrong because the program starts at 100.

- This is because the assembler adds the program's base address (100) to the offset (17) to get the absolute memory address (117).

```
14
15 num1: dw   5
16 num2: dw   10
17 num3: dw   15
18 num4: dw   0
```

File: c02-02.asm and c02-03.asm

We used dw(word/2bytes/16 bits) because ax is a 16 bits register and it stores 16 bits.

```
13 num1:    dw  5
14          dw  10
15          dw  15
16          dw  0
```

The labels num2, num3, and num4 are removed and the data there will be accessed with reference to num1.
as we used dw(word/2bytes/16 bits) because ax is a 16 bits register and it stores 16 bits so to access the next num we use num1+2
which will be 0117+ 2 = 0119. and num1 + 4 to access next one and so on.

```
5       mov  bx, [num1 + 2]
6       add  ax, bx
```

```
    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
```

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

we don't need different variables to store multiple values,we can save space and declare all data on a single line separated by commas.

```
13 00000017 0500                num1:   dw 5
14 00000019 0A00                        dw 10
15 0000001B 0F00                        dw 15
16 0000001D 0000
```

c02-02.lst

```
12
3 00000017 05000A000F000000     num1:   dw 5, 10, 15, 0
```

c02-03.lst

see the difference here the operands are stored in different variables (there addresses are given).

File : c02-04.asm

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

Storing the value of ax in result.

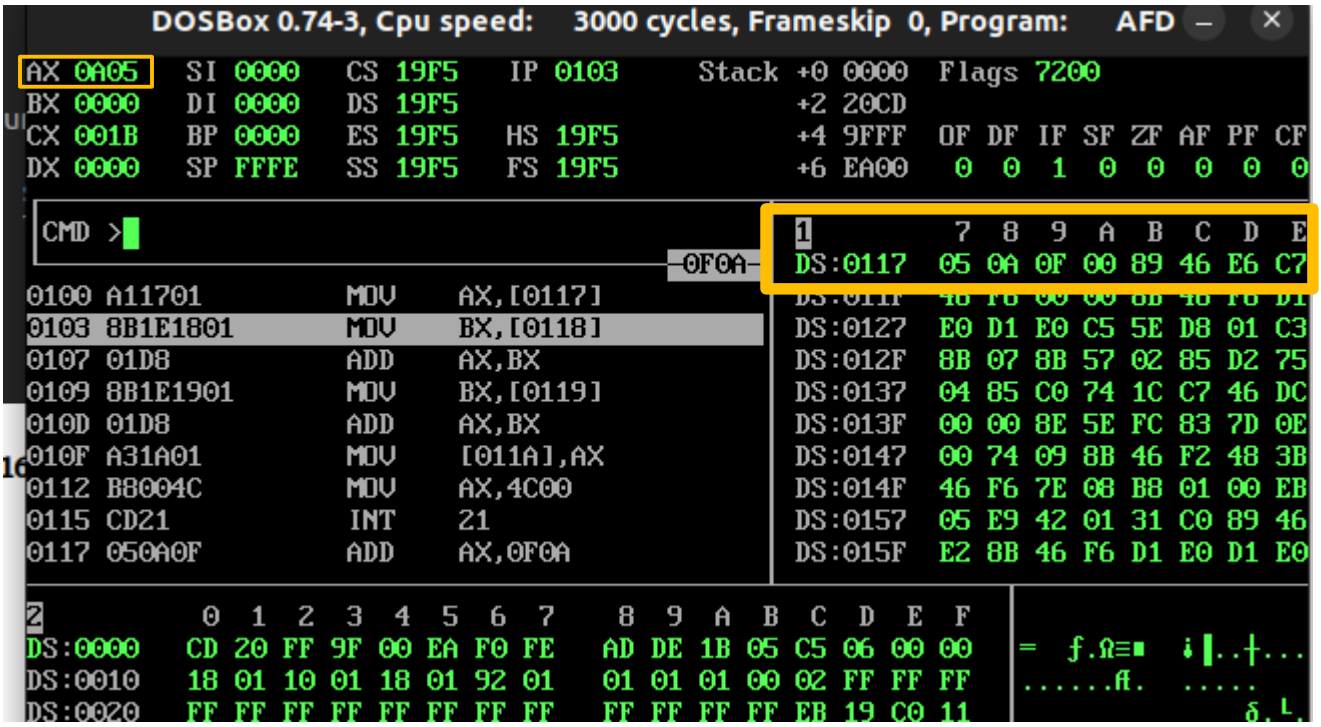
```
add  [num1 + 6], ax

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

mov  ax, 0x4c00
int  0x21
```

```
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
```

The opcode of add is changed because the destination is now a memory location instead of a register.

File : 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!
```



Problem: The code mixes byte variables (defined with db) and a 16-bit register (ax). This leads to incorrect addition because ax stores extra garbage bits when loading single bytes.

Solution:

1. **Use al (lower 8 bits of ax) with db:**
    - Load each byte value into al: mov al, [num1], mov al, [num1+1], mov al, [num1+2]
    - Perform additions using al: add al, [num1+1], add al, [num1+2]
    - Store the result (in al) back to memory: mov [num1+3], al
2. **Use ax with dw :**
    - Change num1 to dw (defines word variables): num1: dw 5, 10, 15, 0

file : c02-06.asm

```
; 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  ah, [num1]          ; Intel Sotware Developer Manual - F

    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
```

db signifies that each element in num1 is a single byte (8 bits) of data.

The instructions mov al, [num1], mov al, [num1+1], and mov al, [num1+2] load the byte values from memory locations num1, num1+1, and num1+2, respectively, into the lower 8 bits (AL) of the ax register.

Adding 1 to the base address (num1) moves to the next byte location (num1+1).

The instruction mov [num1+3], al stores the final result (the sum) from AL back into the memory location num1+3.

File c02-07.asm

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

     ; for (int c = 3    c > 0     c--) {
     ;   result += data[c];
     ;}



    ; initialize stuff
    mov  ax, 0              ; reset the accumulator
    mov  cx, 3              ; set the iterator count
    mov  bx, num1           ; set the base

    outerloop:
        add  ax, [bx]
        add  bx, 2

        sub  cx, 1
        jnz  outerloop


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21


    ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)

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

The loop (outerloop) continues as long as the counter (cx) is greater than 0.

- The jnz (jump if not zero) instruction checks the **Zero Flag (ZF)** after the sub cx, 1 instruction.
- If cx becomes **zero** after the subtraction, the Zero Flag is set (ZF = 1).
- If cx is still **positive** (greater than zero), the Zero Flag remains **cleared** (ZF = 0).