******

**JK Lakshmipat University**

[**Computer**](https://canvas.instructure.com/courses/8496533) **Organization and Architecture**

**ASSIGNMENT 4**

**Name – Indrajit Roy**

**Section – B**

**Roll No - 2023BTECH037**

**Date of Submission – 18th November, 2024**

**Submitted To – Dr. Divyanshu Jain Sir**

**Software used – Emu8086**

**Question 1**

**Write an Assemble Language Program (ALP) for 8-bit addition**

**Theory : The program performs the addition of two 8-bit hexadecimal numbers using assembly language. In the data segment, the inputs a and b are initialized with hexadecimal values 0C1h (C1 in hex, 193 in decimal) and 0B1h (B1 in hex, 177 in decimal), respectively, while sum is left uninitialized to store the result, and carry is set to 0 to track any overflow during addition. In the code segment, the base address of the data segment is loaded into the ax register. The value of a is moved into the lower 8 bits of ax (register al). Next, the program adds the value of b to al, storing the result in al. If the addition produces a carry, the Carry Flag (CF) is set, and the jnc (Jump if No Carry) instruction checks this flag. If a carry occurs, the program increments the carry variable; otherwise, it skips this step. Finally, the result in al (72h in hex, 114 in decimal) is stored in sum. The program ends with sum holding the result and carry set to 1, indicating an overflow**

***● Code:***

***;IJ Roy***

***data segment***

***a db 0C1h***

***b db 0B1h***

***sum db ?***

***carry db 00h***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***MOV ax,data***

***mov ds,ax***

***mov al,a***

***add al,b***

***jnc skip***

***inc carry***

***skip:mov sum,al***

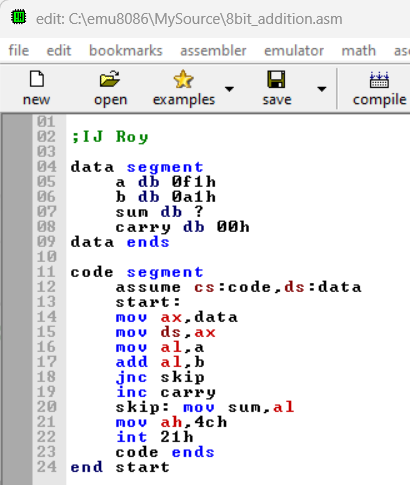
***mov ah,4ch***

***int 21h***

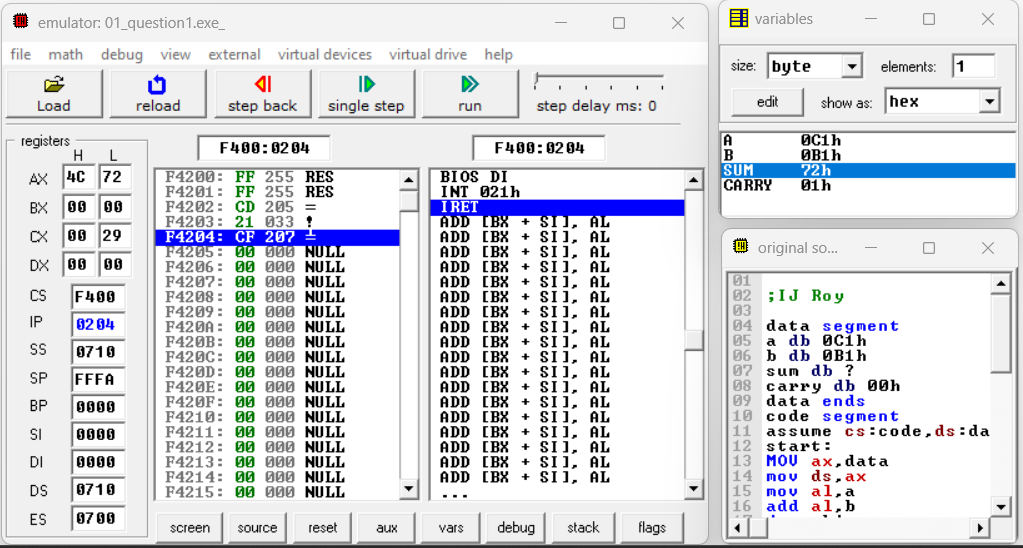
***code ends***

***end start***

***● Code Snapshot :***

******

***● Results:***



**Question 2**

**Write an ALP for 16-bit addition.**

**Theory : The code below demonstrates the addition of two 16-bit numbers in assembly language. It begins with the data segment, where input values and additional variables, such as sum (to store the result) and carry (to track overflow), are defined. Both are initialized, with sum left unassigned to hold the output and carry set to 0. The data inputs are a = 1234h and b = 4567h, representing hexadecimal values. Using the code segment, the base address of the data segment is loaded into the AX register, a 16-bit register. The value of a is then moved into AX, and b is added to it, with the result stored back in AX. The instruction JNC (Jump if No Carry) ensures that if no carry is generated, the process skips incrementing carry; otherwise, carry is updated. The result in AX is then moved to sum, yielding 579Bh as the sum and a carry value of 1. The program concludes with interrupts and termination instructions.**

***● Code:***

***;IJ Roy***

***data segment***

***a dw 1234h***

***b dw 4567h***

***sum dw ?***

***carry db 00h***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***mov ax,data***

***mov ds,ax***

***mov ax,a***

***add ax,b***

***jnc skip***

***inc carry***

***skip: mov sum,ax***

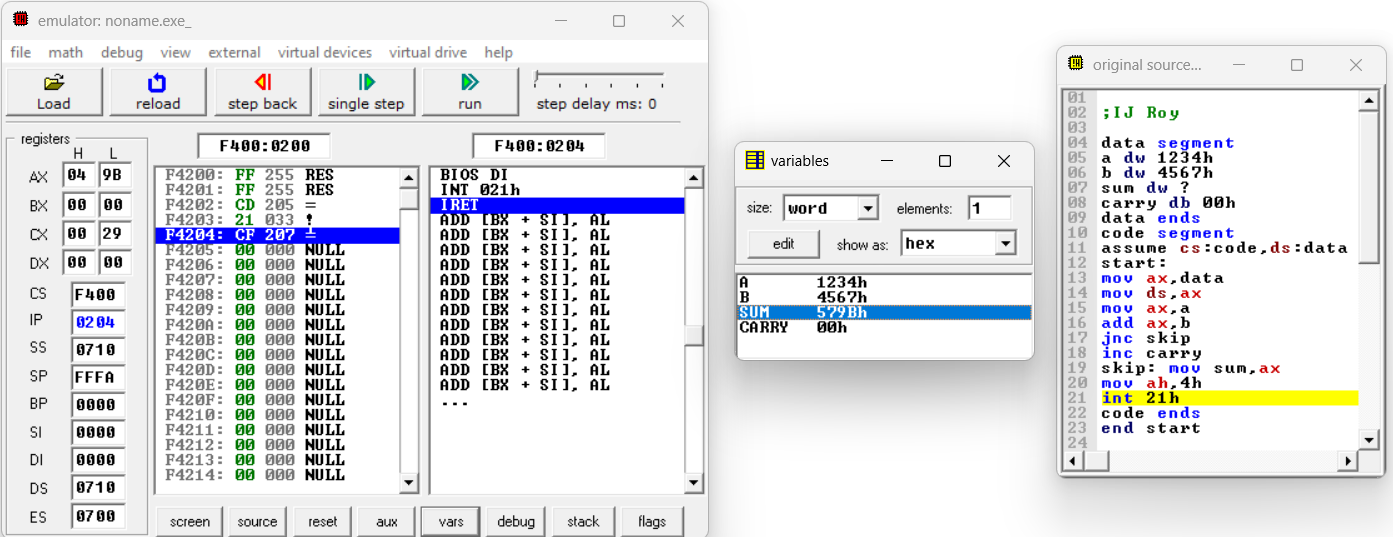
***mov ah,4h***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 3**

**Write an ALP for 8-bit unsigned subtraction.**

**Theory : The code below demonstrates the subtraction of two 8-bit numbers in assembly language. It begins with the data segment, where input values and additional variables, such as subtraction (to store the result) and borrow (to track overflow or underflow), are defined. Both are initialized, with subtraction left unassigned to hold the output and borrow set to 0. The data inputs are a = 55h and b = 41h, representing hexadecimal values. In the code segment, the base address of the data segment is loaded into the AX register, with AL (the lower 8 bits of AX) used for operations. The value of a is moved into AL, and then b is subtracted from it, with the result stored in AL. The JNC (Jump if No Carry) instruction checks for a borrow; if there is no borrow, the process skips incrementing borrow. Otherwise, borrow is updated. Finally, the result in AL is moved to subtraction, yielding 14h as the result with a borrow value of 0. The program concludes with interrupts and termination instructions.**

***● Code:***

***;IJ Roy***

***data segment***

***a db 55h***

***b db 41h***

***subtraction db ?***

***borrow db 00h***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***mov ax,data***

***mov ds,ax***

***mov al,a***

***sub al,b***

***jnc skip***

***inc borrow***

***skip: mov subtraction,al***

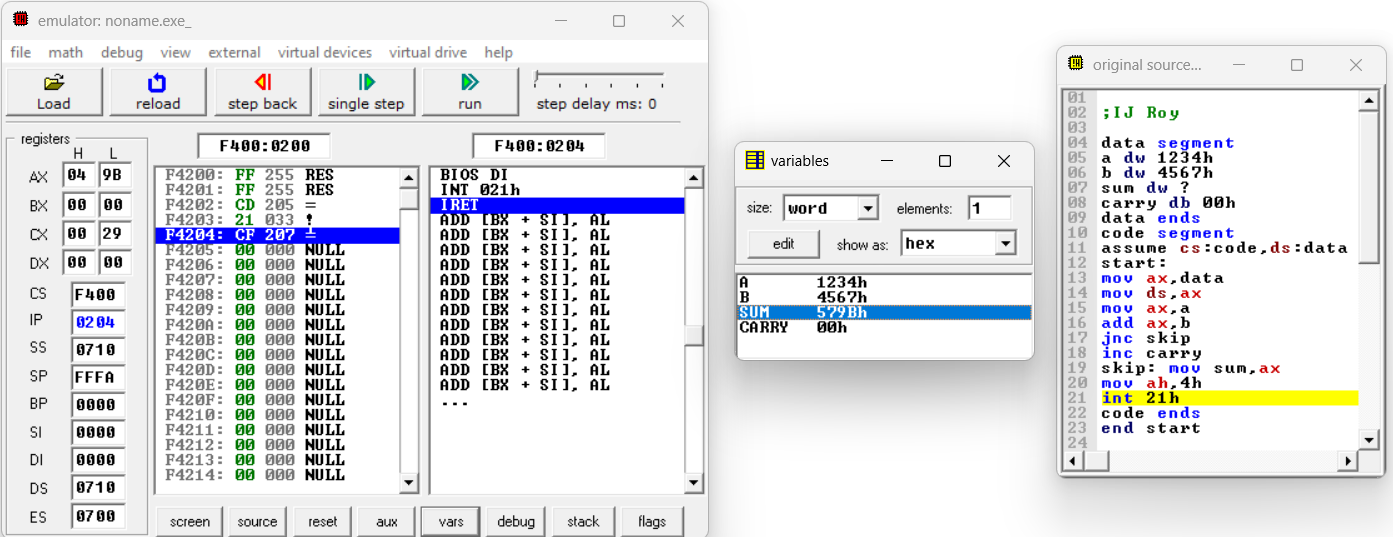
***mov ah,4h***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 4**

**Write an ALP for 16-bit unsigned subtraction.**

**Theory : In the code below we are subtracting two 16 bits number. First we write the data segment code in which take the data as input and whatever other data we might required. We take the data’s input and two extra data as subtraction and borrow as we might require it while subtraction, and subtraction as 16 bit and borrow same initialised to 0. We use dw as Define Word for 16 bit data. Then we write our code in the code segment. In the example below a is 1055h in which input is 1055 in hexadecimal and b is 0041h as other input in which 41 in hexadecimal. The subtraction is not assigned any value as we required it as the output or the result and borrow initially assigned to 0. First we assume the code segment as ds and data segment as ds then we start. We move the base address of the data segment to the ax which is a 16 bit register and ax is of 16 bit which contains a now. Now we move a (1055) to ax in which a total capacity is of 16 bits and then sub ax,b(0041) which subtracts the value and then stores result in ax. Now jnc is for jump if no carry to skip (which is a label) and then increment borrow else. The skip label moves the data of ax to subtraction and we get the result in subtraction(1014 in hexadecimal with borrow 0). The rest are interrupts and then code ends.**

***● Code:***

***;IJ Roy***

***data segment***

***a dw 1055h***

***b dw 0041h***

***subtraction dw ?***

***borrow db 00h***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***mov ax,data***

***mov ds,ax***

***mov ax,a***

***sub ax,b***

***jnc skip***

***inc borrow***

***skip: mov subtraction,ax***

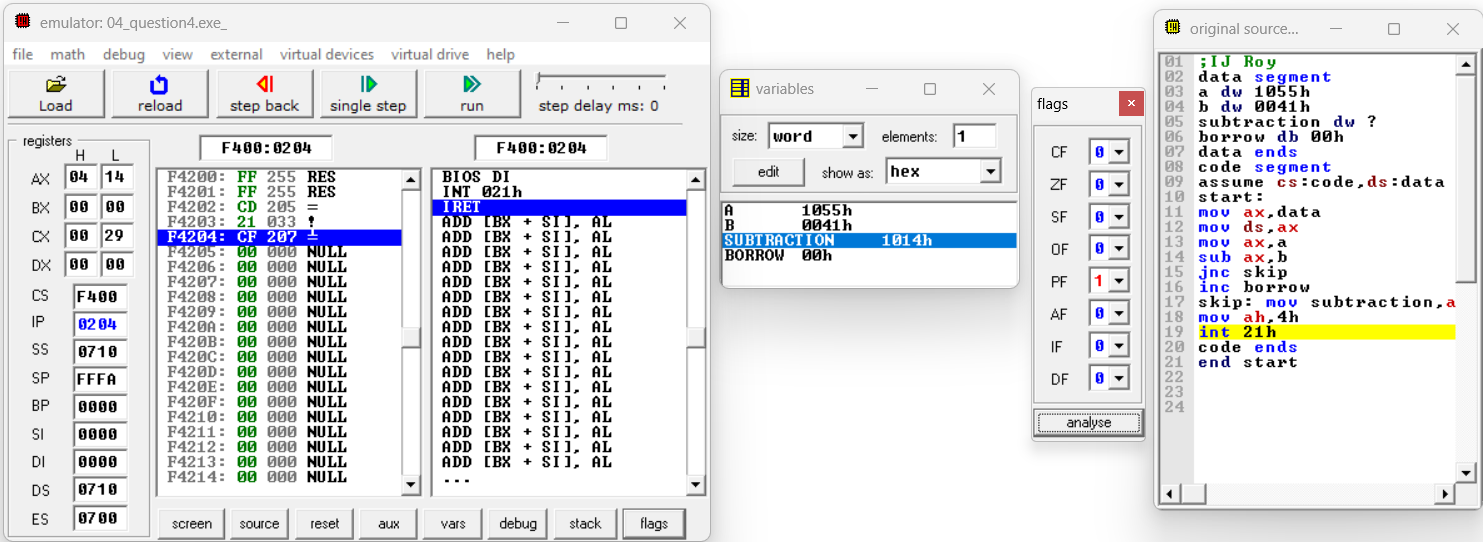
***mov ah,4h***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 5**

**Write an ALP to multiply two 8-bit numbers. Operands and results in Data Segment.**

**Theory :In the code below we are multiplying two 8 bits number. First we write the data segment code in which take the data as input and whatever other data we might required. We take the data’s input and one extra data as product as we might require it while multiplication and make it as dw Define word as multiplication is stored in 2n bits , and multiplication as 8 bit. We use db as Define Byte for 8 bit data. Then we write our code in the code segment. In the example below a is 08h in which input is 08 in hexadecimal and b is 09h as other input in which 09 in hexadecimal. The multiplication is not assigned any value as we required it as the output or the result . First we assume the code segment as ds and data segment as ds then we start. We move the base address of the data segment to the ax which is a 16 bit register and al is of 8 bit which contains a now. Now we move a (08)to lower bits of ax in which a total capacity is of 16 bits and move b (09) to lower bits of bx and stored in b1 and the result is now stored in ax a 16 bit register by mul bl operation.The result is (48h) which is correct.The rest are interrupts and then code ends.**

***● Code:***

***;IJ Roy***

***data segment***

***a db 08h***

***b db 09h***

***product dw ?***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***mov ax,data***

***mov ds,ax***

***mov al,a***

***mov bl,b***

***mul bl***

***mov product,ax***

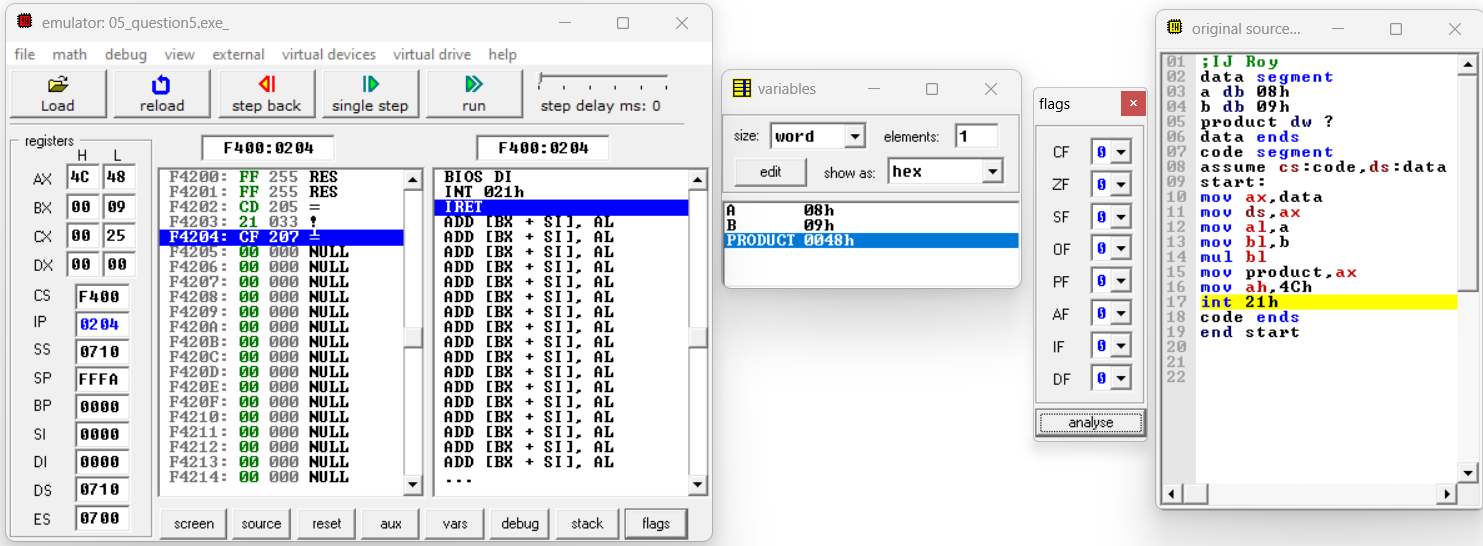
***mov ah,4Ch***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 6**

**Write an ALP to multiply two 16-bit numbers. Operands and result in Data Segment**

**Theory : In the code below we are multiplying two 16 bits number. First we write the data segment code in which take the data as input and whatever other data we might required. We take the data’s input and two extra data as producthigh and productlow as the result can be stored in max of 32 bits so and we might require it while multiplication and make it as dw Define word as multiplication is stored in 2n bits , and multiplication as 8 bit. We use dw as Define word for 8 bit data. Then we write our code in the code segment. In the example below a is 0001h in which input is 0001 in hexadecimal and b is 0002h as other input in which 0002 in hexadecimal. The multiplication is not assigned any value as we required it as the output or the result . First we assume the code segment as ds and data segment as ds then we start. We move the base address of the data segment to the ax which is a 16 bit register and ax is of 16 bit which contains a now. Now we move a (0001) to ax and b(0002) to bx in which a total capacity is of 16 bits and the result is now stored in ax-dx pairs the higher bits with dx and lower bits with ax.The result is (00000002h) which is correct.The rest are interrupts and then code ends**

***● Code:***

***;IJ Roy***

***data segment***

***a dw 0001h***

***b dw 0002h***

***productlow dw ?***

***producthigh dw ?***

***data ends***

***code segment***

***assume cs:code,ds:data***

***start:***

***mov ax,data***

***mov ds,ax***

***mov ax,a***

***mov bx,b***

***mul bx***

***mov productlow,ax***

***mov producthigh,dx***

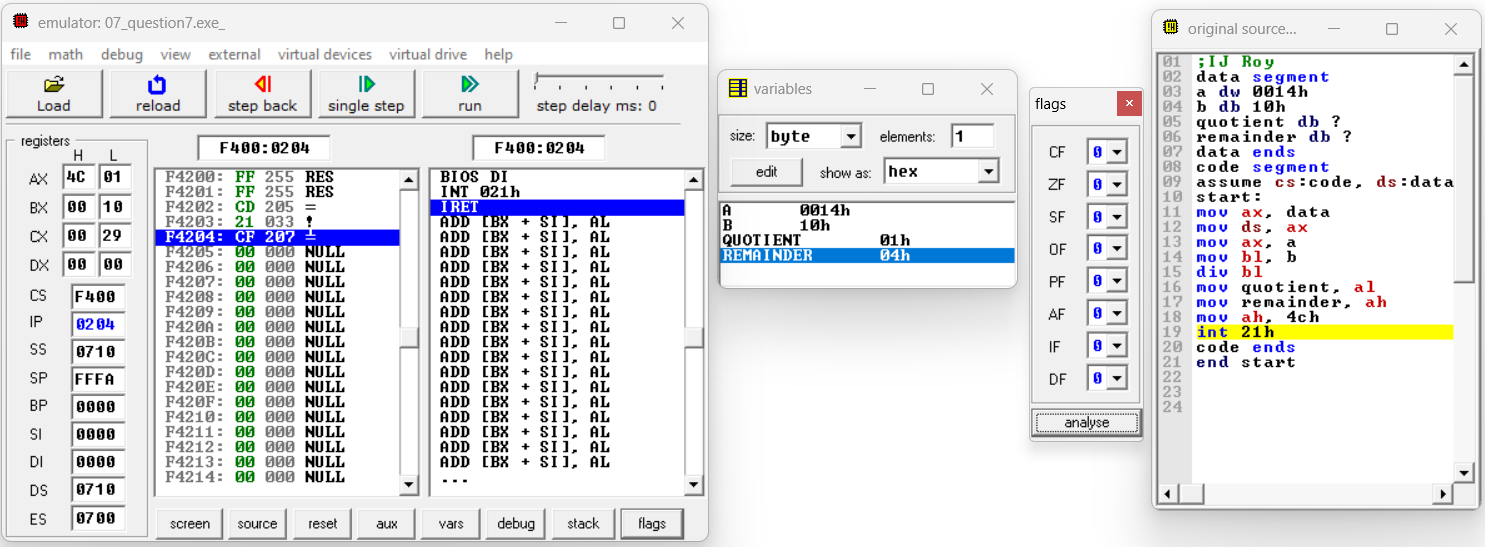
***mov ah,4Ch***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 7**

**Write** **an ALP to divide a 16-bit number by an 8-bit number**

**Theory :In the code below we are dividing 16 bit number by 8 bit number. First we write the data segment code in which take the data as input and whatever other data we might required. We take the data’s input in which 1 is db and the other is dw and two extra data as quotient and remainder as we might require it while dividing and make it as db both remainder and as well as quotient. We use db as Define Byte for 8 bit data and dw as Define Word. Then we write our code in the code segment. In the example below a is 0014h in which input is 0014 in hexadecimal and b is 10h as other input in which 10 in hexadecimal. The quotient and remainder are not assigned any value as we required it as the output or the result . First we assume the code segment as ds and data segment as ds then we start. We move the base address of the data segment to the ax which is a 16 bit register and ax is of 16 bit which contains a now. Now we move a (0014)to ax in which a total capacity is of 16 bits and move b (10) to lower bits of bx and stored in b1 and the result is now stored in al-ah pair 8 bit registers by div bl operation.The quotient is in al and remainder in ah.The result is quotient(01h) and remainder(04h) which is correct.The rest are interrupts and then code ends.**

***● Code:***

***;IJ Roy***

***data segment***

***a dw 0014h***

***b db 10h***

***quotient db ?***

***remainder db ?***

***data ends***

***code segment***

***assume cs:code, ds:data***

***start:***

***mov ax, data***

***mov ds, ax***

***mov ax, a***

***mov bl, b***

***div bl***

***mov quotient, al***

***mov remainder, ah***

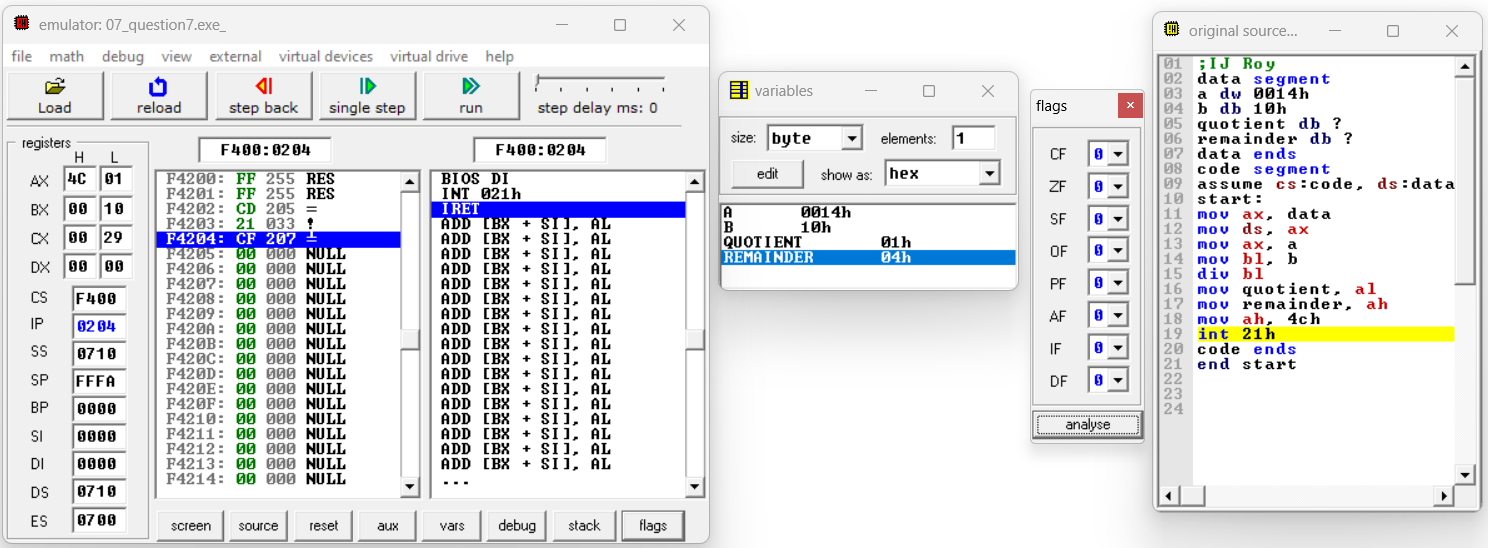
***mov ah, 4ch***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Question 8**

**Write an ALP to divide a 32-bit number by a 16 bit number**

**Theory :In the code below we are dividing 32 bit number by 16 bit number. First we write the data segment code in which take the data as input and whatever other data we might required. We take the data’s input in which 3 input are dw as 1 for ahigher and other for alower to store 32 bit number and two extra data as quotient and remainder as we might require it while dividing and make it as dw both remainder and as well as quotient. We use dw as Define Word. Then we write our code in the code segment. In the example below a\_h is 0000h and a\_l 1011h in which input is 00001011 in hexadecimal and b is 0001h as other input in which 01 in hexadecimal. The quotient and remainder are not assigned any value as we required it as the output or the result . First we assume the code segment as ds and data segment as ds then we start. We move the base address of the data segment to the ax which is a 16 bit register and ax is of 16 bit which contains a now.Now we mov a\_h to dx and a\_l to ax as both are 16 bit registers and b in bx to store and now we had done div bx which divides and gives the result in ax-dx pair and in ax quotient and dx remainder.The result is quotient(1011h) and remainder(0000h) which is correct.The rest are interrupts and then code ends.**

***● Code:***

***;IJ Roy***

***data segment***

***a\_h dw 0000h***

***a\_l dw 1011h***

***b dw 0001h***

***quotient dw ?***

***remainder dw ?***

***data ends***

***code segment***

***assume cs:code, ds:data***

***start:***

***mov ax, data***

***mov ds, ax***

***mov dx, a\_h***

***mov ax, a\_l***

***mov bx, b***

***div bx***

***mov quotient, ax***

***mov remainder, dx***

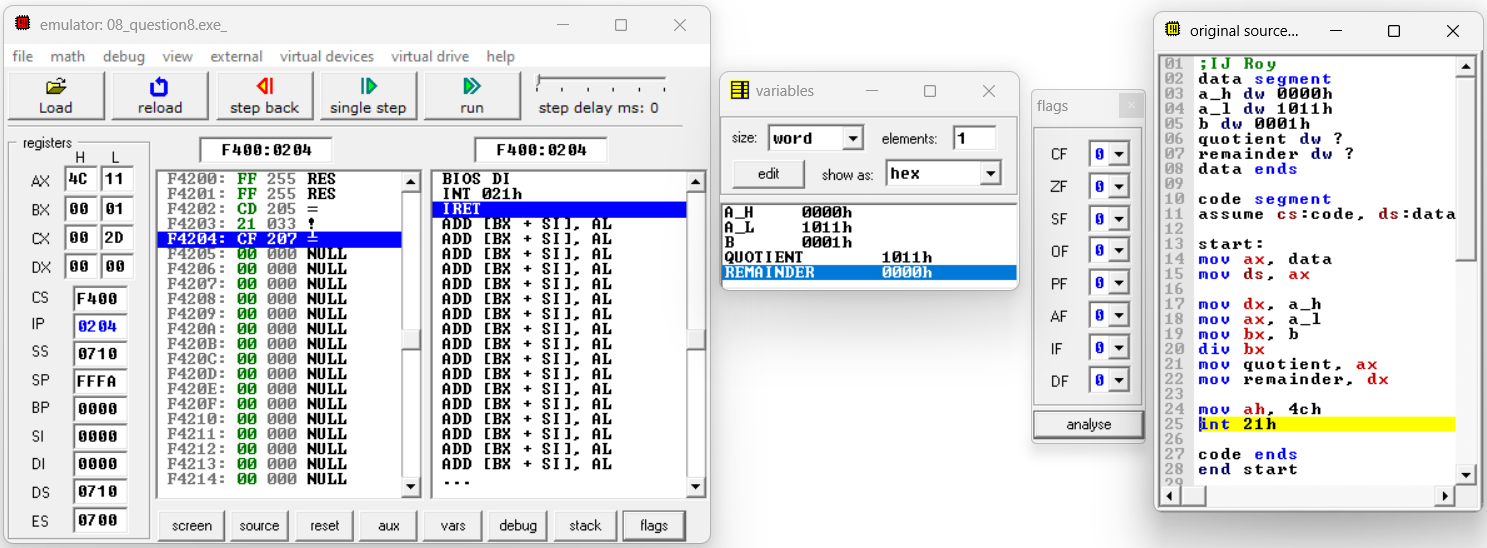
***mov ah, 4ch***

***int 21h***

***code ends***

***end start***

***● Results:***

******

**Learnings:-**

**1. In this after running the codes we can see changes in the registers and the flags that brings clarity that the hexadecimal operations performed by us is correct and the changes in the flag bits confirms the other parts.**

**2. In this efficient use of registers for optimization of space in the memory. Also the good thing is this was the optimization of the use of everything that could be good enough for further programmes.**

**3. Understanding how each arithmetic operaions works in assembly, including how to handle borrow and carry. Using jnc to handle the carry flag and manage branching in code.**

**4.Efficient use of registers and understanding how to move and manipulate data. Correctly setting up and using the data and code segment.**

**Thank You**