|  |
| --- |
| FAST National University |
| **Introduction to DOSBox**  **Lab 1** |
|  |

**Computer Organization and Assembly Language**

|  |  |
| --- | --- |
| **Student Name** | Moeez Ali |
| **Registration #** | 21L-1798 |
| **Instructor** | Hazoor Ahmad |
| **Class** | CS3 |
| **Section** | A1, A2, D1, D2 |
| **Semester** | Fall 2022 |

Fast School of Computing

FAST-NU, Lahore, Pakistan

# Activity 1

## **Assembly Language Code**

[[org 0x0100]

mov ax,1798 h

mov bx,0xAAAA

AND bx,ax

AND cx,ax

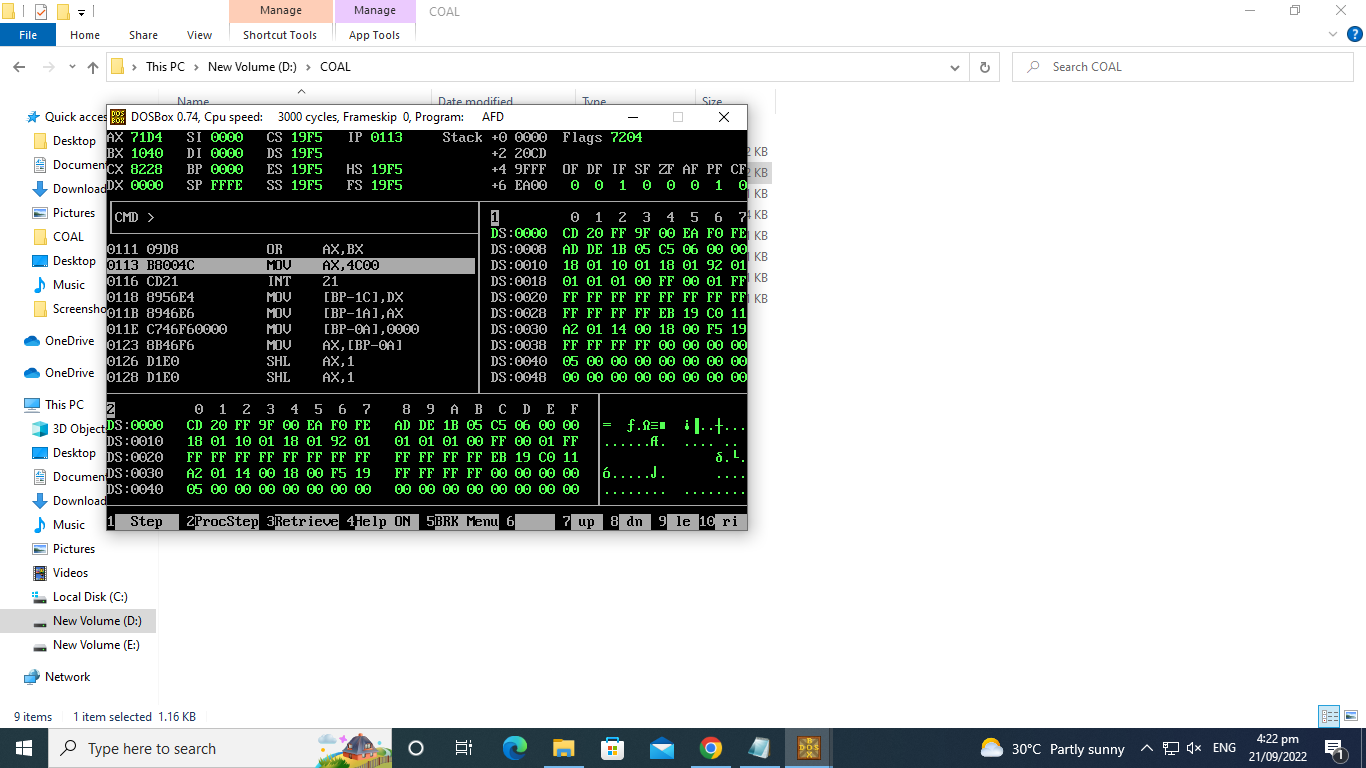
shr bx,1

shl cx,1

OR ax,bx

mov ax,0x4c00

## **Debugging Screenshots**



# Activity 2

## **Assembly Language Code**

[org 0x0100]

mov ax,1798 h

mov bx,3333h

AND bx,ax

mov cx,0xCCCC

AND cx,ax

shr bx,2

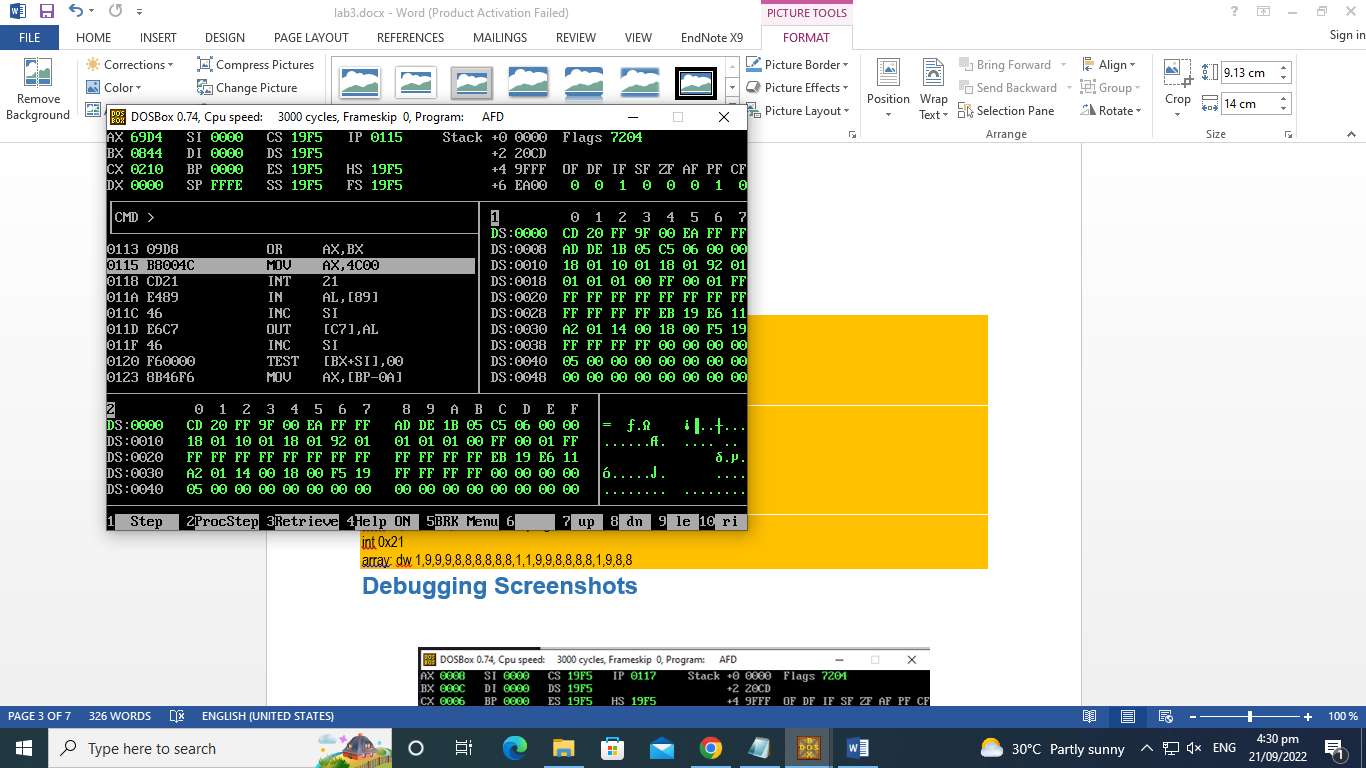
shl cx,2

OR ax,bx

mov ax,0x4c00

int 0x21

## **Debugging Screenshots**



# Activity 3

## **Assembly Language Code**

[org 0x0100]

mov ax,1798 h

mov bx,0xFOFO

AND bx,ax

mov cx,0xOFOF

AND cx,ax

shr bx,4

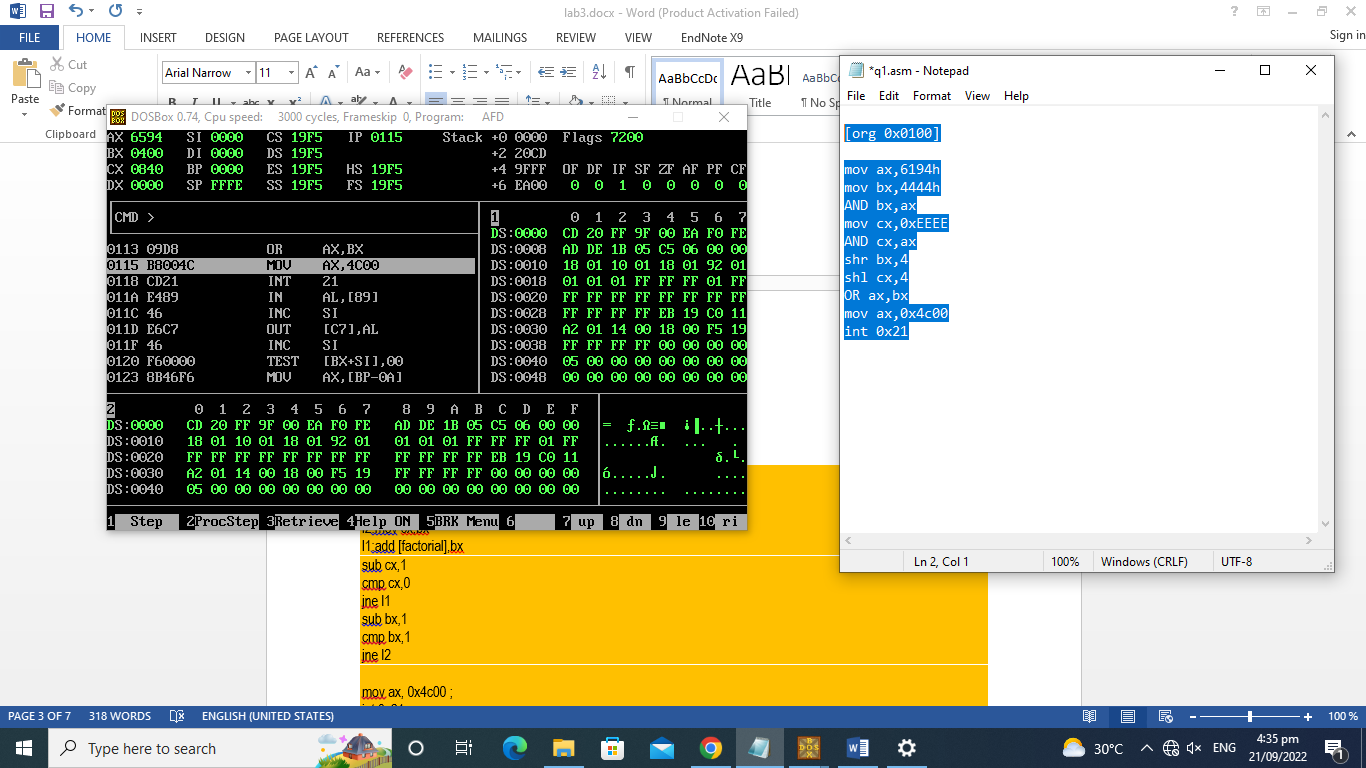
shl cx,4

OR ax,bx

mov ax,0x4c00

int 0x21

## **Debugging Screenshots**



# Activity 4

## **Assembly Language Code**

[[org 0x0100]

mov ax,1798

mov cx,1111111111111111b

mov bx, ax

xor bx, cx

or bx , ax

mov cx,ax

xor ax,1BCDh

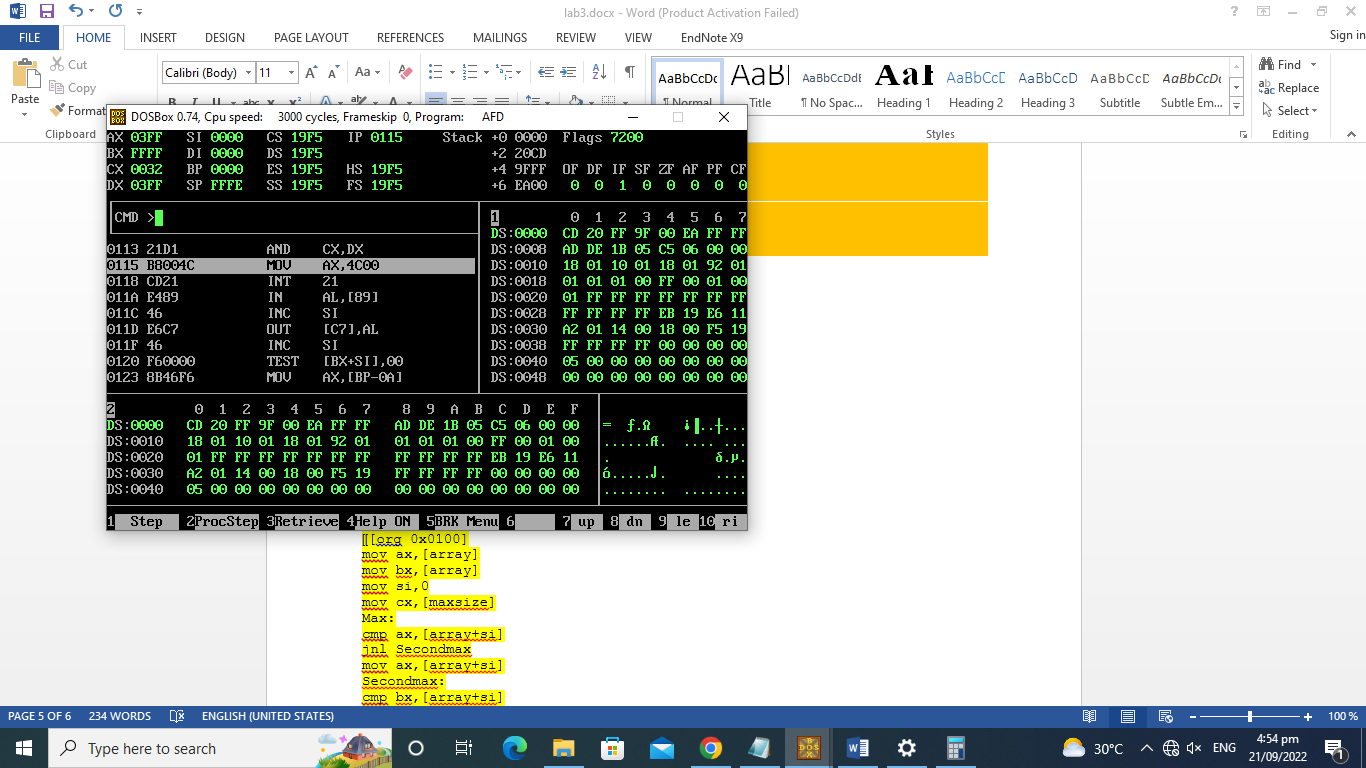
mov dx,ax

and cx,dx

mov ax,0x4c00

int 0x21

**Debugging Screenshots**



# Activity 5

## **Assembly Language Code**

[org 0x0100]

mov ax,1798

mov bx, ax

not bx

mov word [multiplicand], ax

mov word [multiplier], bx

mov cl, 16

mov bx, 1

checkbit:

test bx, [multiplier]

jz skip

mov ax, [multiplicand]

add [result], ax

mov ax, [multiplicand + 2]

adc [result + 2], ax

skip:

shl word [multiplicand], 1

rcl word [multiplicand + 2], 1

shl bx , 1

dec cl

jnz checkbit

mov ax, 5380

mov bx, ax

not bx

add bx, [result]

adc ax, [result+2]

mov [f], bx

mov [f+2], ax

mov ax, 0x4c00

int 21h

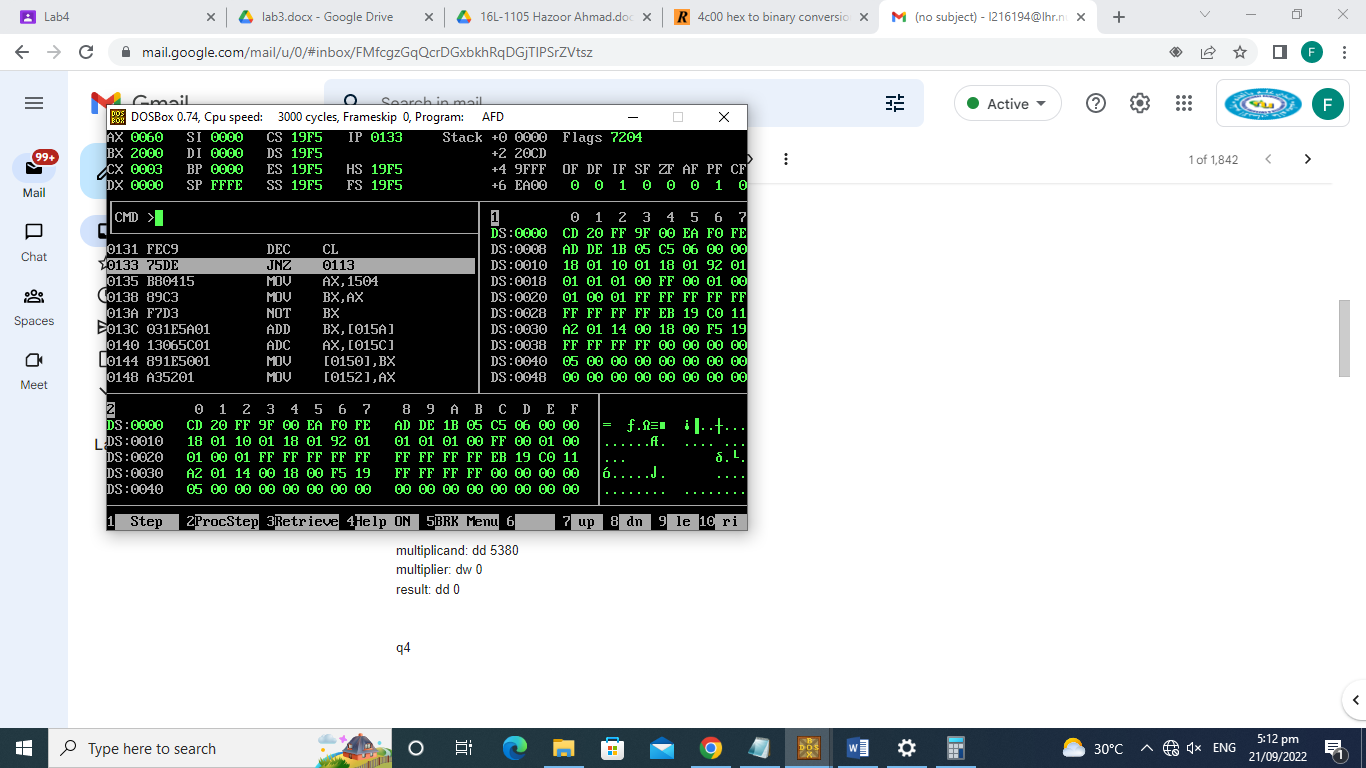
f: dd 0

multiplicand: dd 6194

multiplier: dw 0

result: dd 0

## **Debugging Screenshots**



# Activity 6

Near Jump:

When the relative address stored with the instruction is in 16 bits the jump is called a near jump. Using a near jump we can jump anywhere within a segment. If we add a large number it will wrap around to the lower part. A negative number actually is a large number and works this way using the wraparound behavior.

Example:

; a program to add ten numbers without a separate counter

[org 0x0100]

jmp start ; unconditionally jump over data

num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50

total: dw 0

start: mov bx, 0 ; initialize array index to zero

mov ax, 0 ; initialize sum to zero

l1: add ax, [num1+bx] ; add number to ax

add bx, 2 ; advance bx to next index

cmp bx, 20 ; are we beyond the last index

jne l1 ; if not add next number

mov [total], ax

mov ax, 0x4c00

Short Jump:

If the offset is stored in a single byte as in 75F2 with the opcode 75 and operand F2, the jump is called a short jump. F2 is added to IP as a signed byte. If the byte is negative the complement is negated from IP otherwise the byte is added. Conditional jumps can only be short. A short jump can go +127 bytes ahead in code and -128 bytes backwards. This is the limitation of a byte in signed representation.

example:

mov ax,4h

mov bx,5h

cmp ax,bx

jb change

jmp exit

change:

mov ax,bx

exit:

mov ax,0x4c00

int 21h

Far Jump:

Far jump is not position relative but is absolute. Both segment and offset must be given to a far jump. The previous two jumps were used to jump within a segment. Sometimes we may need to go from one code segment to another, and near and short jumps cannot take us there. The three instructions that have a far form are JMP, CALL, and RET.Far jump must be used and a two byte segment and a two byte offset are given to it.

example:

mov ax,5h

mov bx,4h

call func

jmp exit

func:

add ax,bx

ret ax

exit:

mov ax,0x4c00

int 21h