National University of Computer and Emerging Sciences

Lab Manual 5

COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE

Course Instructor	Dr Asma, Miss Aleena
Lab Instructor(s)	Maham Saleem Sidra
Section	3F,3E
Semester	Fall 2021

Department of Computer Science
FAST-NU, Lahore, Pakistan

DIV Instruction
The division used in the process is integer division and not floating-point division. Integer division gives an integer quotient and an integer remainder.

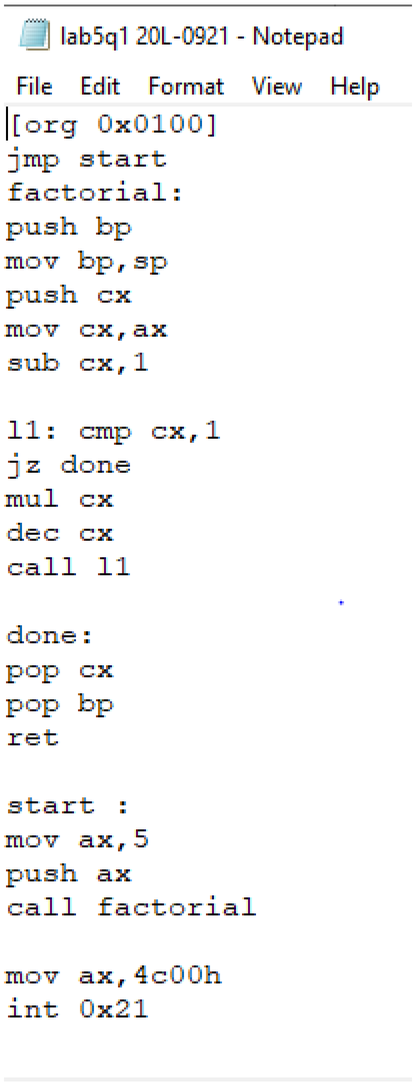
There are two forms of the DIV instruction. The first form divides a 32bit number in DX:AX by its 16bit operand and stores the 16bit quotient in AX and the 16bit remainder in DX.

The second form divides a 16bit number in AX by its 8bit operand and stores the 8bit quotient in AL and the 8bit remainder in AH. For example, “DIV BL” has an 8bit operand, so the implied dividend is 16bit and is stored in the AX register and “DIV BX” has a 16bit operand, so the implied dividend is 32bit and is therefore stored in the concatenation of the DX and AX registers. The higher word is stored in DX and the lower word in AX.

If a large number is divided by a very small number, it is possible that the quotient is larger than the space provided for it in the implied destination. In this case an interrupt is automatically generated and the program is usually terminated as a result. This is called a divide overflow error; just like the calculator shows an –E– when the result cannot be displayed. This interrupt will be discussed later in the discussion of interrupts. DIV (divide) performs an unsigned division of the accumulator (and its extension) by the source operand. If the source operand is a byte, it is divided into the two-byte dividend assumed to be in registers AL and AH. The byte quotient is returned in AL, and the byte remainder is returned in AH. If the source operand is a word, it is divided into the two-word dividend in registers AX and DX. The word quotient is returned in AX, and the word remainder is returned in DX. If the quotient exceeds the capacity of its destination register (FF for byte source, FFFF for word source), as when division by zero is attempted, a type 0 interrupt is generated, and the quotient and remainder are undefined.

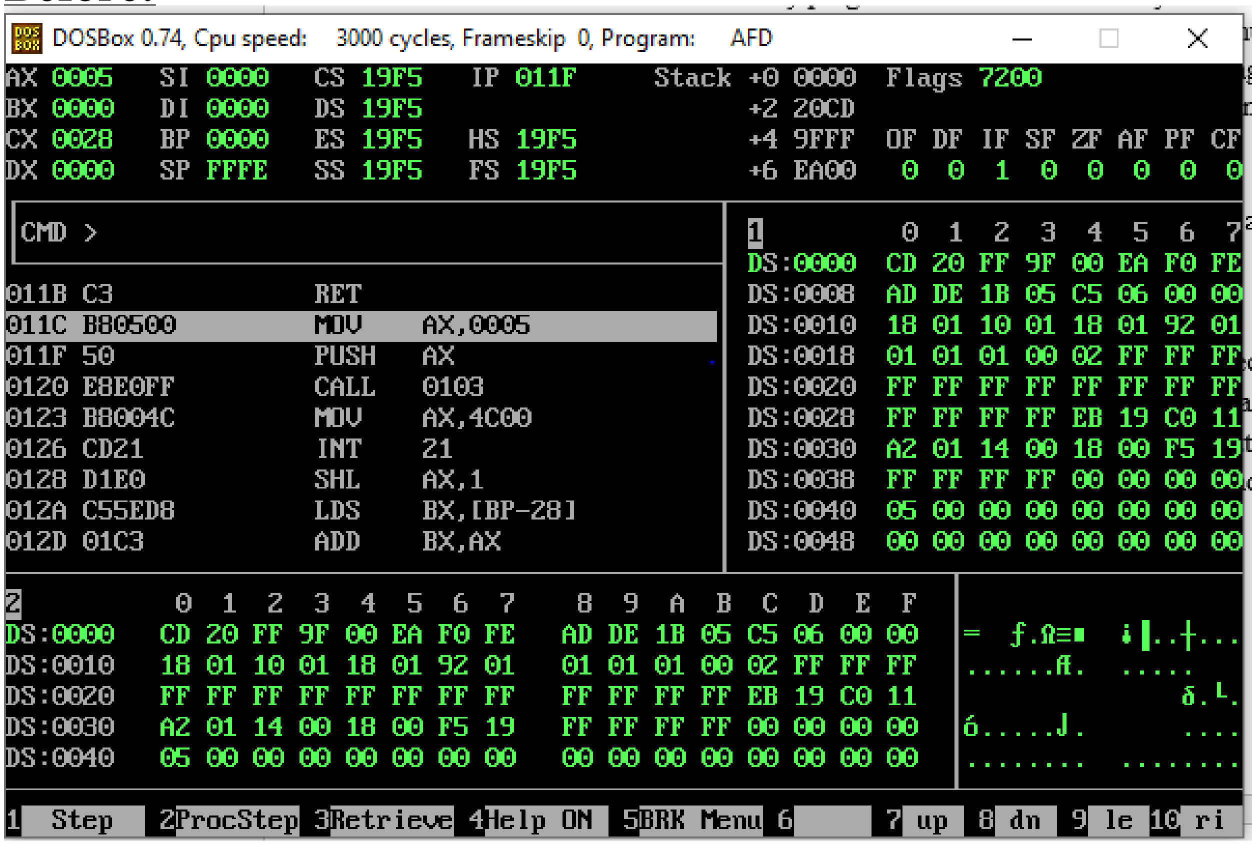
MUL Instruction
MUL (multiply) performs an unsigned multiplication of the source operand and the accumulator. If the source operand is a byte, then it is multiplied by register AL and the double-length result is returned in AH and AL. If the source operand is a word, then it is multiplied by register AX, and the double-length result is returned in registers DX and AX.

Problem 1: Write an assembly program to calculate factorial of a number. (Use MUL instruction and a loop)



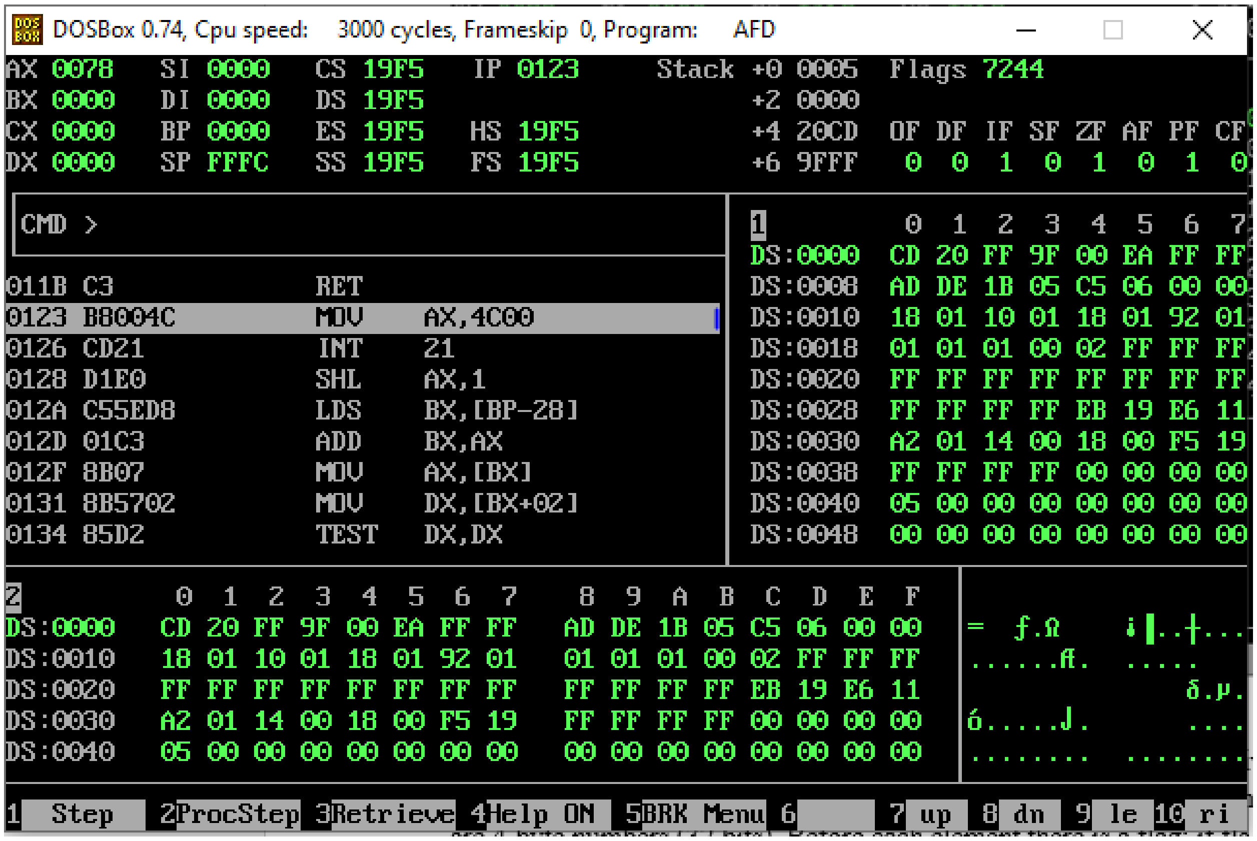
```
[org 0x0100]
jmp start
factorial:
push bp
mov bp,sp
push cx
mov cx,ax
sub cx,1

l1: cmp cx,1
jz done
mul cx
dec cx
call l1

done:
pop cx
pop bp
ret

start :
mov ax,5
push ax
call factorial

mov ax,4c00h
int 0x21
```

Before:



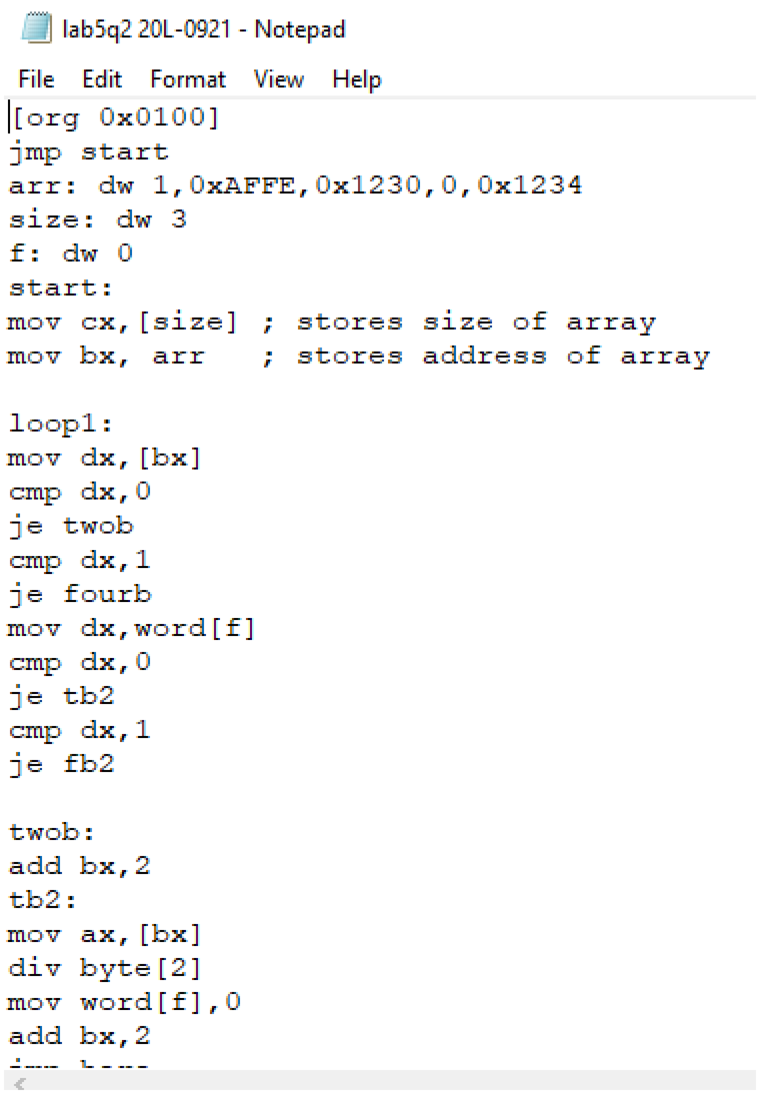
```
DOSBox 0.74, Cpu speed:    3000 cycles, Frameskip  0, Program:    AFD
AX 0005   SI 0000   CS 19F5   IP 011F      Stack +0 0000  Flags 7200
BX 0000   DI 0000   DS 19F5                     +2 20CD
CX 0028   BP 0000   ES 19F5   HS 19F5           +4 9FFF  OF DF IF SF ZF AF PF CF
DX 0000   SP FFFE   SS 19F5   FS 19F5           +6 EA00   0  0  1  0  0  0  0  0

 CMD >                                           1          0  1  2  3  4  5  6  7
                                                 DS:0000   CD 20 FF 9F 00 EA F0 FE
011B C3                RET                       DS:0008   AD DE 1B 05 C5 06 00 00
011C B80500            MOV     AX,0005           DS:0010   18 01 10 01 18 01 92 01
011F 50                PUSH    AX                DS:0018   01 01 01 00 02 FF FF FF
0120 E8E0FF            CALL    0103              DS:0020   FF FF FF FF FF FF FF FF
0123 B8004C            MOV     AX,4C00           DS:0028   FF FF FF FF EB 19 C0 11
0126 CD21              INT     21                DS:0030   A2 01 14 00 18 00 F5 19
0128 D1E0              SHL     AX,1              DS:0038   FF FF FF FF 00 00 00 00
012A C55ED8            LDS     BX,[BP-28]        DS:0040   05 00 00 00 00 00 00 00
012D 01C3              ADD     BX,AX             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   = ƒ.Ω≡■  ↓▌..┼...
DS:0010  18 01 10 01 18 01 92 01   01 01 01 00 02 FF FF FF   ......Æ.  .....
DS:0020  FF FF FF FF FF FF FF FF   FF FF FF FF EB 19 C0 11             δ.└.
DS:0030  A2 01 14 00 18 00 F5 19   FF FF FF FF 00 00 00 00   ó.....⌡.      ....
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
```

After:

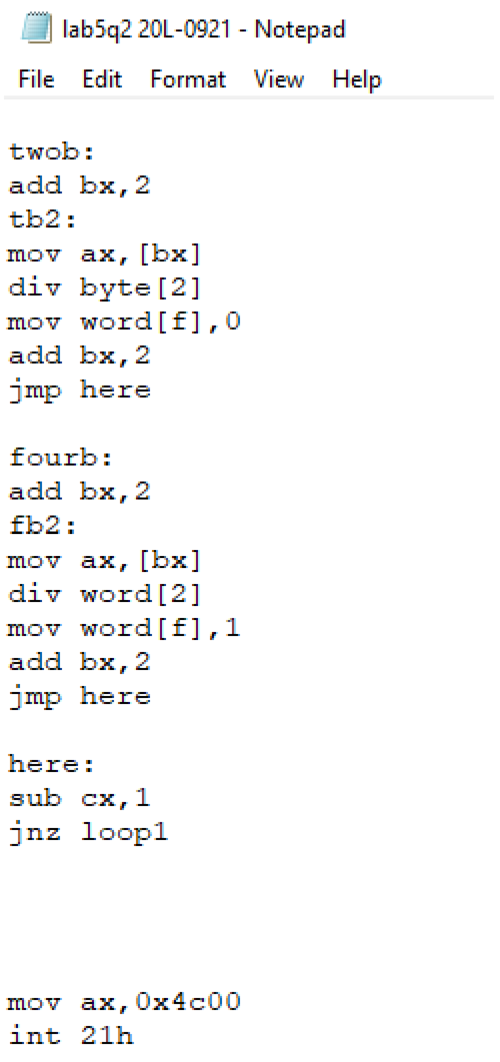


```
DOSBox 0.74, Cpu speed:   3000 cycles, Frameskip  0, Program:   AFD
AX 0078   SI 0000   CS 19F5   IP 0123      Stack +0 0005  Flags 7244
BX 0000   DI 0000   DS 19F5                      +2 0000
CX 0000   BP 0000   ES 19F5   HS 19F5            +4 20CD  OF DF IF SF ZF AF PF CF
DX 0000   SP FFFC   SS 19F5   FS 19F5            +6 9FFF   0  0  1  0  1  0  1  0

CMD >                                          1          0  1  2  3  4  5  6  7
                                               DS:0000   CD 20 FF 9F 00 EA FF FF
011B C3            RET                         DS:0008   AD DE 1B 05 C5 06 00 00
0123 B8004C        MOV    AX,4C00              DS:0010   18 01 10 01 18 01 92 01
0126 CD21          INT    21                   DS:0018   01 01 01 00 02 FF FF FF
0128 D1E0          SHL    AX,1                 DS:0020   FF FF FF FF FF FF FF FF
012A C55ED8        LDS    BX,[BP-28]           DS:0028   FF FF FF FF EB 19 E6 11
012D 01C3          ADD    BX,AX                DS:0030   A2 01 14 00 18 00 F5 19
012F 8B07          MOV    AX,[BX]              DS:0038   FF FF FF FF 00 00 00 00
0131 8B5702        MOV    DX,[BX+02]           DS:0040   05 00 00 00 00 00 00 00
0134 85D2          TEST   DX,DX                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 FF FF  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 E6 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
```

Problem 2: Write an assembly program that traverses the array and divide each element by some number. The array does not have same size element. Some of them are 2-byte numbers (16 bits) and some of them are 4-byte numbers (32 bits). Before each element there is a flag; if flag is 0 it means that the number is a 2-byte number. If the flag is 1, it means that the number is a 4-byte number.
For example:
Numbers: dw 1, 0xAFFE,0x1230, 0, 0x1234
In the above example, there are two numbers which are 0x1230AFFE and 0x1234. 0x1230AFFE has been stored in 4 bytes whereas as 0x1230 has been stored in 2 bytes.



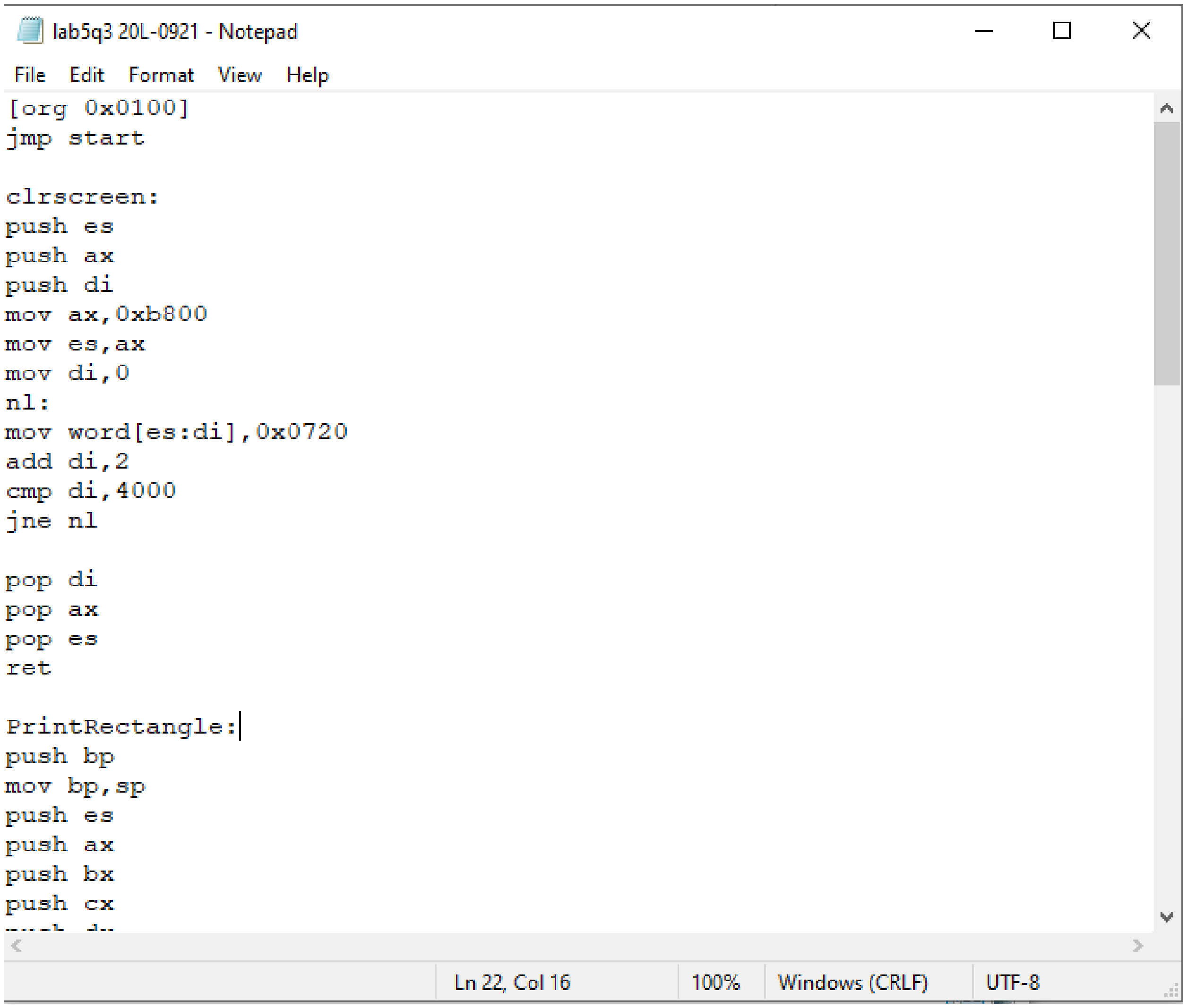
```
[org 0x0100]
jmp start
arr: dw 1,0xAFFE,0x1230,0,0x1234
size: dw 3
f: dw 0
start:
mov cx,[size] ; stores size of array
mov bx, arr   ; stores address of array

loop1:
mov dx,[bx]
cmp dx,0
je twob
cmp dx,1
je fourb
mov dx,word[f]
cmp dx,0
je tb2
cmp dx,1
je fb2

twob:
add bx,2
tb2:
mov ax,[bx]
div byte[2]
mov word[f],0
add bx,2
```


lab5q2 20L-0921 - Notepad

File Edit Format View Help

```
twob:
add bx,2
tb2:
mov ax,[bx]
div byte[2]
mov word[f],0
add bx,2
jmp here

fourb:
add bx,2
fb2:
mov ax,[bx]
div word[2]
mov word[f],1
add bx,2
jmp here

here:
sub cx,1
jnz loop1




mov ax,0x4c00
int 21h
```

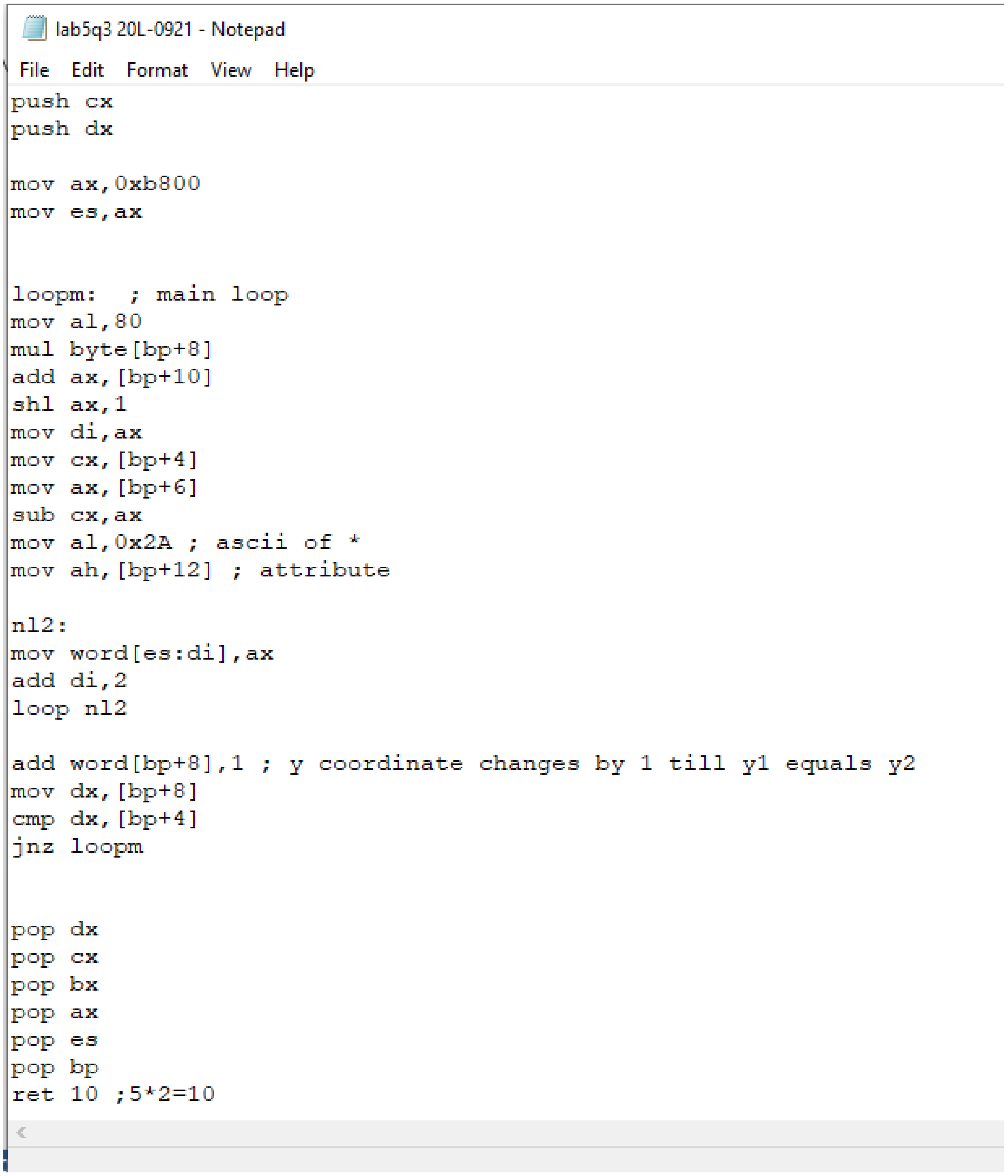
Problem 3: Write a function **PrintRectanlge** that prints a rectangle having its TopLeft and BottomRight corners at (top, left) and (bottom, right) coordinates respectively where top, left, bottom and right are parameters passed by caller. Also pass attribute by caller to print colored rectangle. Following is a red rectangle with TopLeft = (2, 10) and BottomRight = (20, 60).


lab5q3 20L-0921 - Notepad

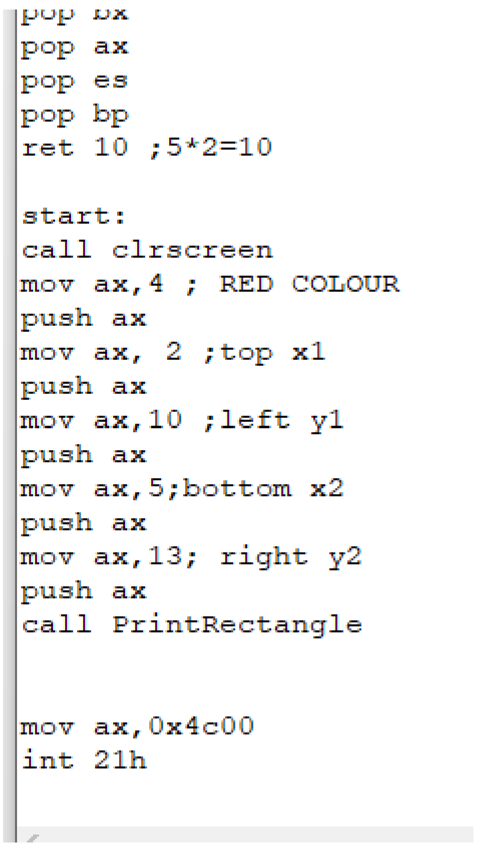
File Edit Format View Help

```
[org 0x0100]
jmp start

clrscreen:
push es
push ax
push di
mov ax,0xb800
mov es,ax
mov di,0
nl:
mov word[es:di],0x0720
add di,2
cmp di,4000
jne nl

pop di
pop ax
pop es
ret

PrintRectangle:
push bp
mov bp,sp
push es
push ax
push bx
push cx
```

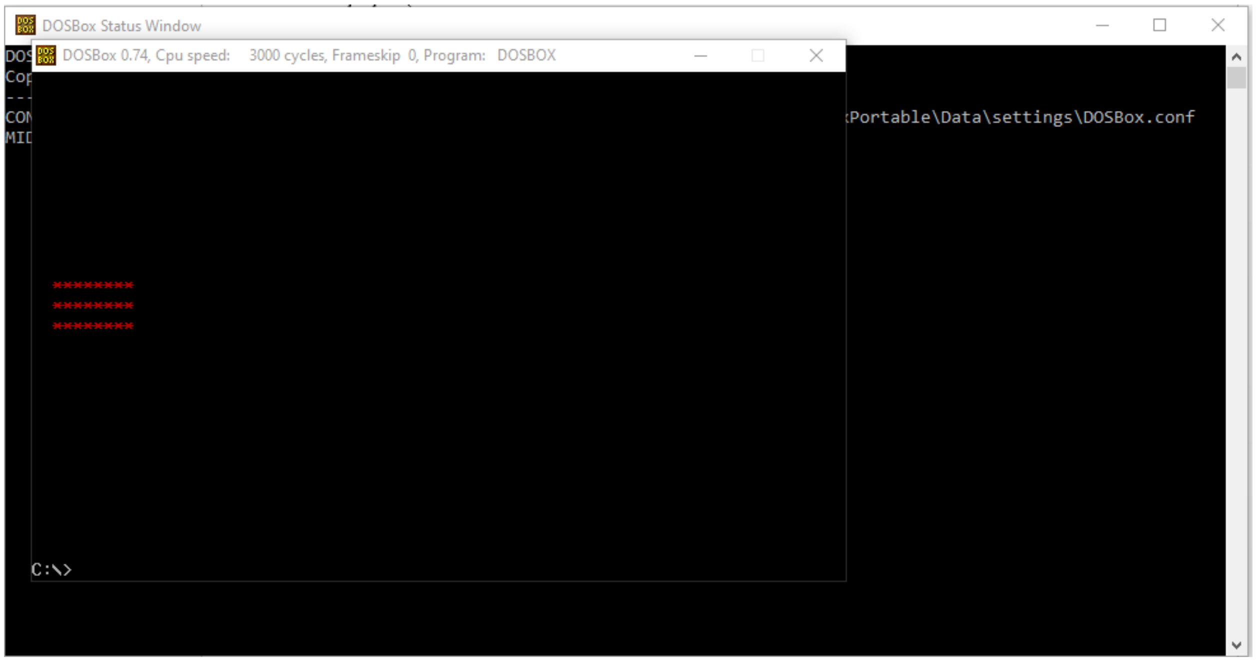
Ln 22, Col 16 100% Windows (CRLF) UTF-8



```
push cx
push dx

mov ax,0xb800
mov es,ax


loopm:  ; main loop
mov al,80
mul byte[bp+8]
add ax,[bp+10]
shl ax,1
mov di,ax
mov cx,[bp+4]
mov ax,[bp+6]
sub cx,ax
mov al,0x2A ; ascii of *
mov ah,[bp+12] ; attribute

nl2:
mov word[es:di],ax
add di,2
loop nl2

add word[bp+8],1 ; y coordinate changes by 1 till y1 equals y2
mov dx,[bp+8]
cmp dx,[bp+4]
jnz loopm


pop dx
pop cx
pop bx
pop ax
pop es
pop bp
ret 10 ;5*2=10
```

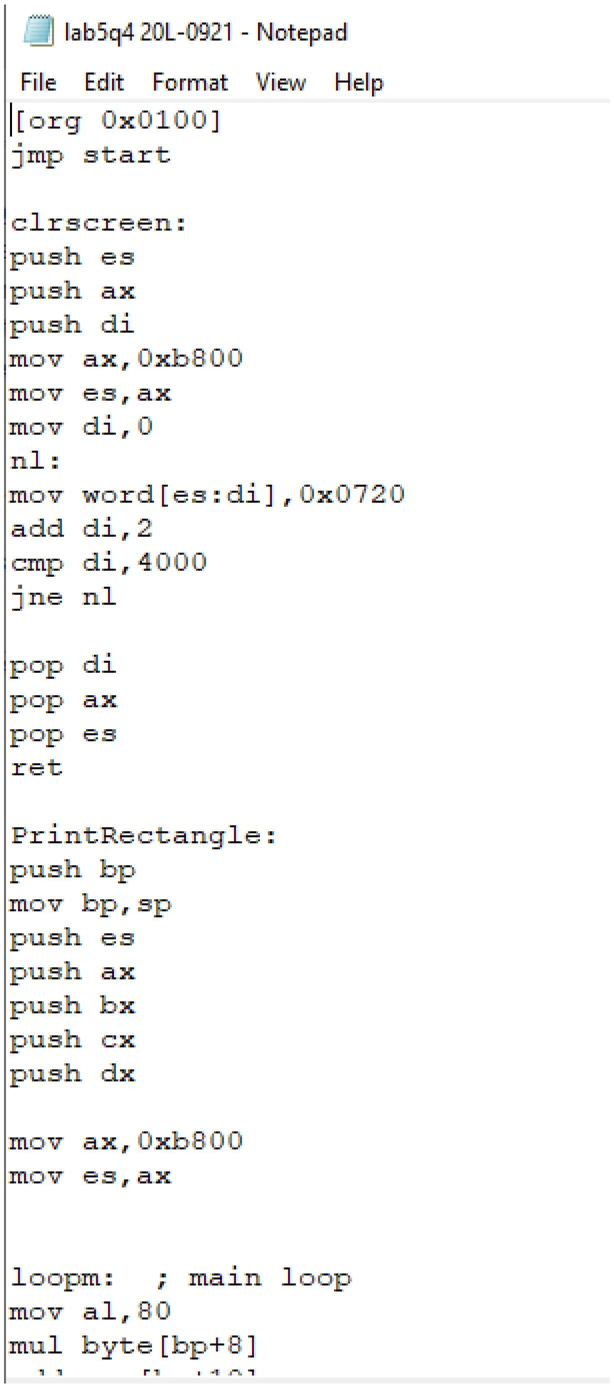


```
pop bx
pop ax
pop es
pop bp
ret 10 ;5*2=10

start:
call clrscreen
mov ax,4 ; RED COLOUR
push ax
mov ax, 2 ;top x1
push ax
mov ax,10 ;left y1
push ax
mov ax,5;bottom x2
push ax
mov ax,13; right y2
push ax
call PrintRectangle


mov ax,0x4c00
int 21h
```

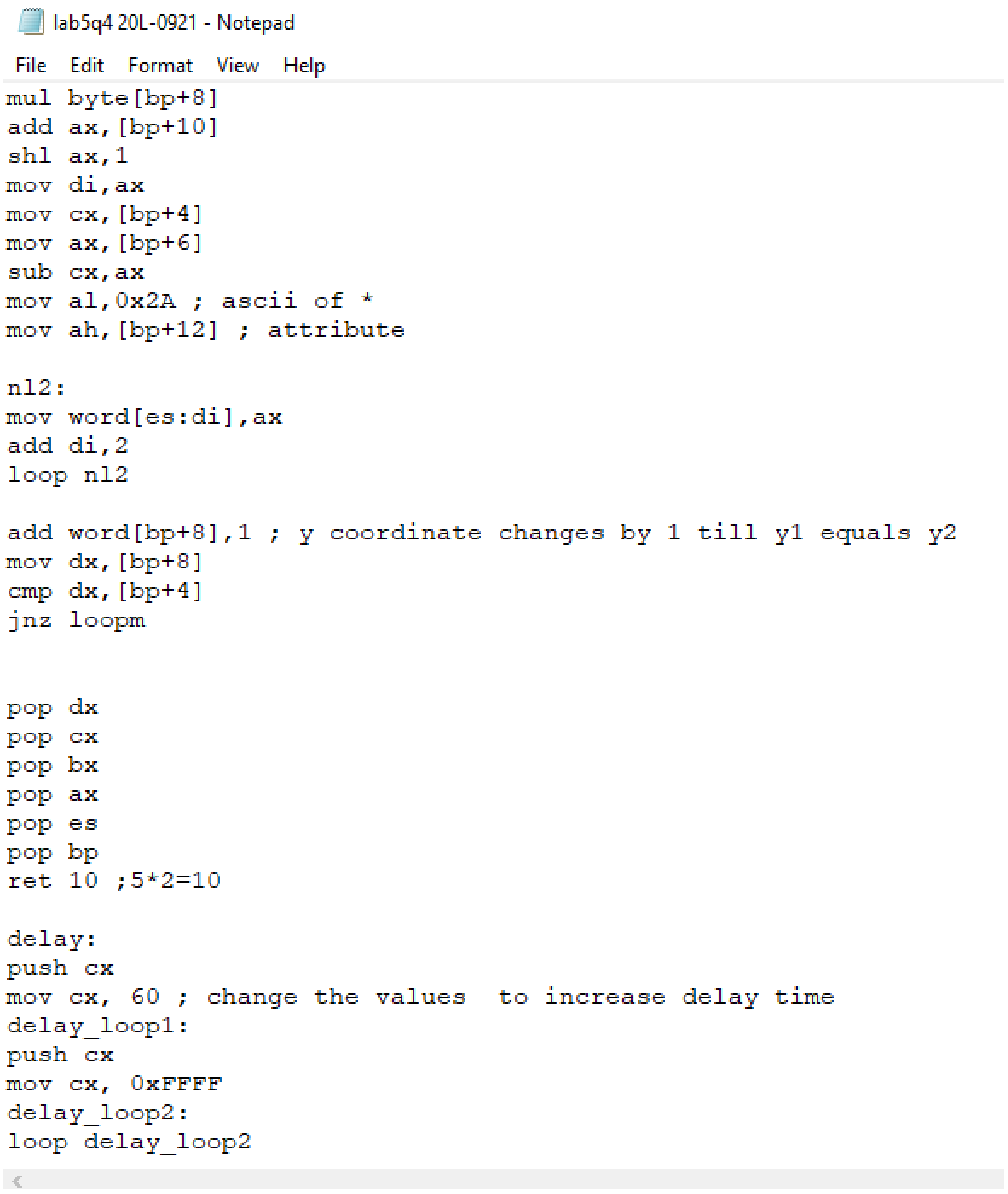

DOSBox Status Window
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```
DOS
Cop
---
CON                                          :Portable\Data\settings\DOSBox.conf
MID

*********
*********
*********

C:\>
```

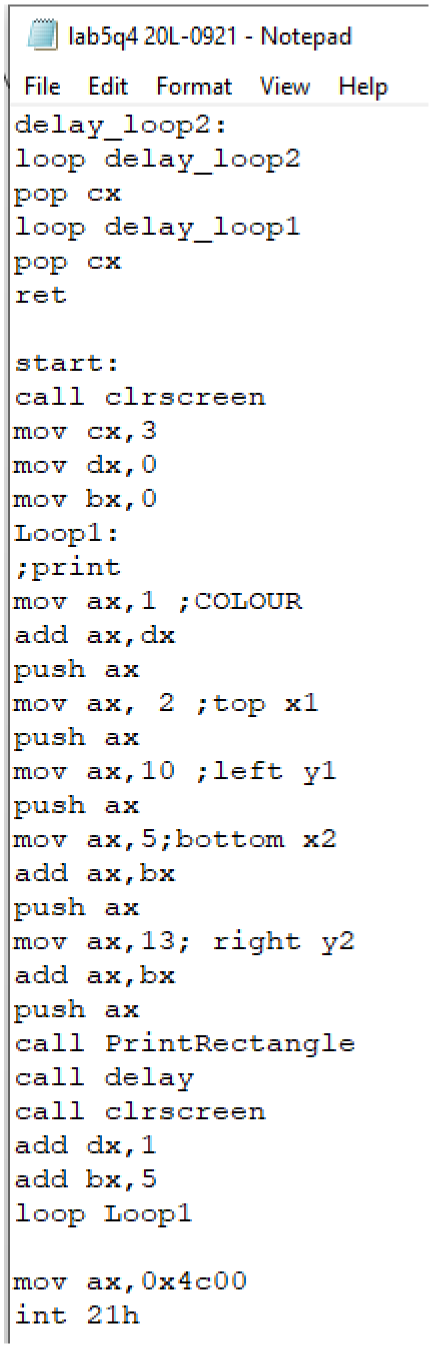
Problem 4: Call above function PrintRectangle and print 3 rectangles of different sizes and colors.



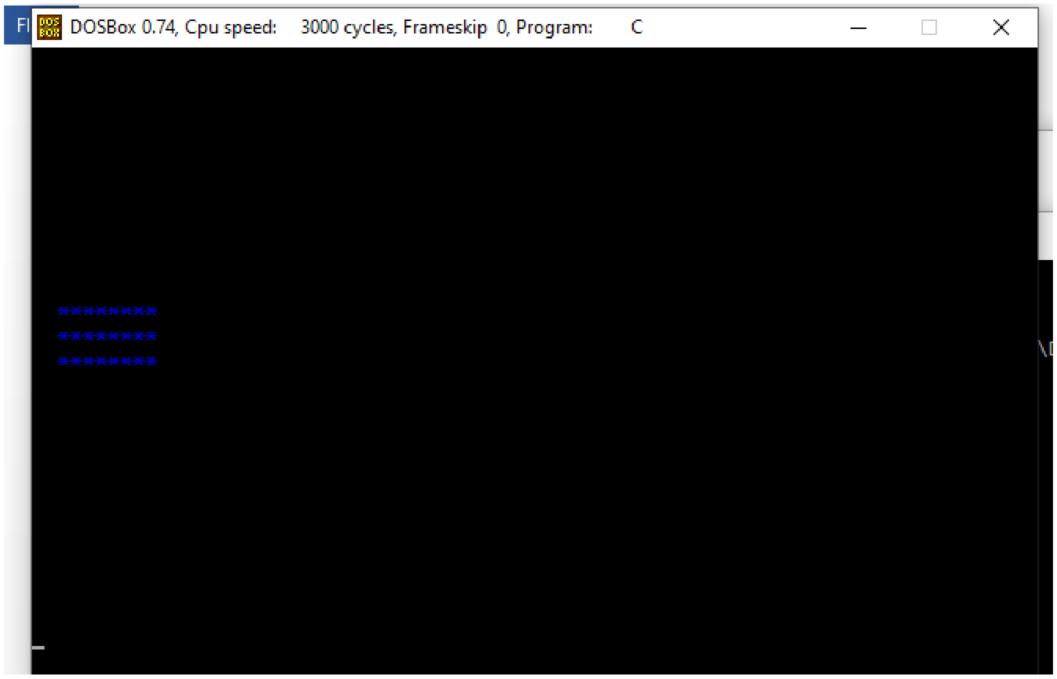
```
[org 0x0100]
jmp start

clrscreen:
push es
push ax
push di
mov ax,0xb800
mov es,ax
mov di,0
nl:
mov word[es:di],0x0720
add di,2
cmp di,4000
jne nl

pop di
pop ax
pop es
ret

PrintRectangle:
push bp
mov bp,sp
push es
push ax
push bx
push cx
push dx

mov ax,0xb800
mov es,ax


loopm:  ; main loop
mov al,80
mul byte[bp+8]
```



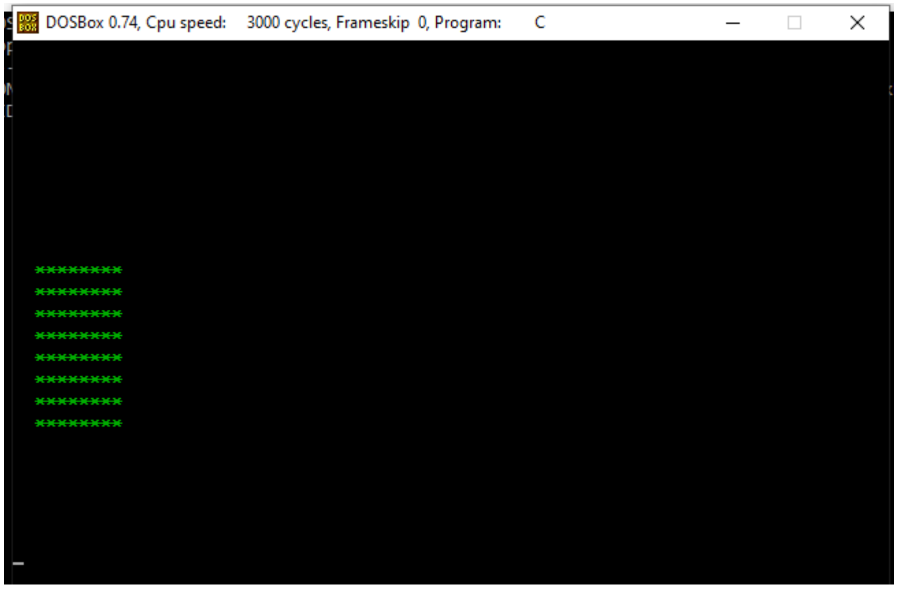
```
mul byte[bp+8]
add ax,[bp+10]
shl ax,1
mov di,ax
mov cx,[bp+4]
mov ax,[bp+6]
sub cx,ax
mov al,0x2A ; ascii of *
mov ah,[bp+12] ; attribute

nl2:
mov word[es:di],ax
add di,2
loop nl2

add word[bp+8],1 ; y coordinate changes by 1 till y1 equals y2
mov dx,[bp+8]
cmp dx,[bp+4]
jnz loopm


pop dx
pop cx
pop bx
pop ax
pop es
pop bp
ret 10 ;5*2=10

delay:
push cx
mov cx, 60 ; change the values  to increase delay time
delay_loop1:
push cx
mov cx, 0xFFFF
delay_loop2:
loop delay_loop2
```



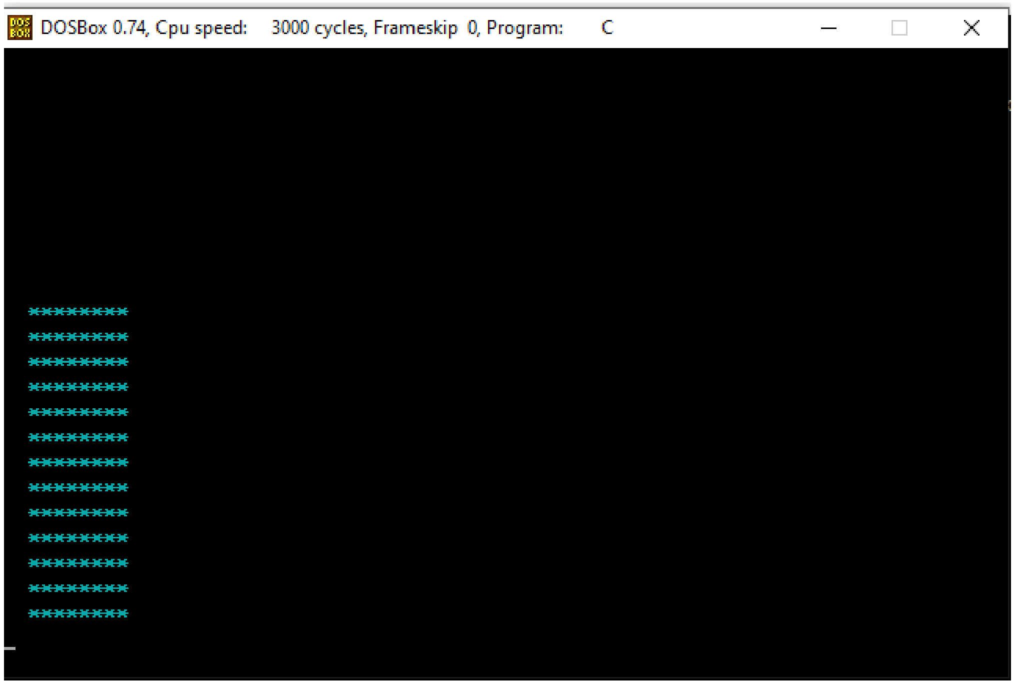
```
delay_loop2:
loop delay_loop2
pop cx
loop delay_loop1
pop cx
ret

start:
call clrscreen
mov cx,3
mov dx,0
mov bx,0
Loop1:
;print
mov ax,1 ;COLOUR
add ax,dx
push ax
mov ax, 2 ;top x1
push ax
mov ax,10 ;left y1
push ax
mov ax,5;bottom x2
add ax,bx
push ax
mov ax,13; right y2
add ax,bx
push ax
call PrintRectangle
call delay
call clrscreen
add dx,1
add bx,5
loop Loop1

mov ax,0x4c00
int 21h
```


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: C

```
********
********
********
```


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: C

```
********
********
********
********
********
********
********
********
```


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: C

```
********
********
********
********
********
********
********
********
********
********
********
********
********
```

HINT: the Loop should have 3 steps,

Loop1:

print * at new location

delay

clear screen

jmp loop1

code for delay is

```
delay:
push cx
mov cx, 3 ; change the values  to increase delay time
delay_loop1:
push cx
mov cx, 0xFFFF
delay_loop2:
loop delay_loop2
pop cx
loop delay_loop1
```

```
pop cx
ret
```