Computer Architecture Lab – CS322

Name: M. Maheeth Reddy **Roll No.**: 1801CS31 **Date**: 16 September 2020

Lab 3 – Assembly Language Programming

P1_0:

Debug Report

Debuggers are very important while learning assembly language programming. A debugger displays the contents of memory and lets us view the registers and variables as they change. One can easily trace logical errors in a program with the help of a debugger because it allows us to step through a program one line at a time. Debugger plays a significant role in getting acquainted with the Intel 8086.

Important Commands in Debugger

?	Help
A	Assemble
C	Compare
D	Dump
E	Enter
F	Fill
G	Go
H	Hexarithmetic
I	Input
L	Load
M	Move
N	Name
O	Output
P	Proceed
Q	Quit
R	Register
S	Search
T	Trace
U	Unassemble
W	Write

A – Assemble
Assemble a program into machine language.
Command formats: A
A *address*
address is the offset from CS

C – Compare
Compares bytes within a specified range with the same number of bytes at a target address.
Command formats: C *range address*

D – Dump
Displays memory on the screen as single bytes in both hexadecimal and ASCII. If no address or range is given, the location begins from where the last D command left off.
Command formats: D
D *address*
D *range*
address is a segment-offset address.
range consists of beginning and ending addresses to dump.

E – Enter
Used to enter data or instructions directly into memory locations. Starting Memory Location for storing the values must be supplied. If only an offset is entered, it is assumed to be from DS.
Command formats: E *address*
E *address list*

F – Fill
Fills a range of memory with a single value or list of values. Range must be specified as two offset addresses or segment-offset addresses.
Command formats: F *range list*

G – Go
Execute the program in memory. Breakpoint (16-/32-bit address) can be specified to stop at a given address. A starting address is an optional parameter. If no breakpoints are specified, program runs until it stops by itself. Maximum of 10 breakpoints can be specified.
Command formats: G
G *breakpoint*
G = *startAddr breakpoint*
G = *startAddr breakpoint1 breakpoint2 …*

H – Hexarithmetic
Performs addition and subtraction on two hexadecimal numbers.
Command formats: H *value1 value2*

I – Input
Inputs a byte from a specified I/O port and displays the value in hexadecimal.
Command formats: I *port*

L - Load
Loads a file or logical disk sectors into memory at given address (default is CS:100). Debug sets BX and CX to number of bytes read. Command formats:
L
L *address*
L *address drive firstsector number*
Note: To read a file, its name must be initialized with the **Name** command.

M – Move
Copies a block of data from one memory location to another.
Command format: M *range address*

N – Name
Initializes a filename in memory before using Load or Write commands.
Command format: N [*d:*] [*filename*] [*.ext*]

P – Proceed
Executes one or more instructions or subroutines. Address of the instruction can be mentioned. For a series of instructions, number of instructions can also be mentioned.
Command formats: P
P *=address*
P *=address number*

Q – Quit
Quits Debug and returns to DOS

R – Register
Display contents of register(s), flags and the next instruction about to be executed. In case of one register and flags, Debug prompts user for a new value.
Command formats: R
R *register*

S – Search
Searches a range of addresses for a sequence of one or more bytes.
Command format: S *range list*

T – Trace
Executes one or more instructions starting at either the current CS:IP location or at an optional address. Contents of the registers are shown after execution of each instruction.
Command formats: T
T *count*
T *=address count*
count is the number of instructions to trace

U – Unassemble
Disassembles or translates memory into assembly language mnemonics. If starting address is not specified, disassembling takes place from the location where the last U command left off. Ending address can also be specified.
Command formats: U
U *startaddr*
U *startaddr endaddr*

W – Write
Writes a block of memory to a file or individual disk sectors. For a file, its name must be initialized with the N command.
Command formats: W
W *address*
W *address drive firstsector number*

P1_1:

Program to find sum of 10 random numbers

Specification: Find sum of given 10 random numbers

Source Code:

```
; Program to find sum of 10 random numbers
.model small
.stack 64
.data
   ; numbers are 1,4,2,7,5,9,6,3,8,0
   num db 01h,04h,02h,07h,05h,09h,06h,03h,08h,00h
   sum db 0        ; variable to store sum
.code
main proc far
   mov ax,@data
   ; load data segment address
   mov ds,ax

   ; bl stores count of remaining numbers
   mov bl,0ah
   ; load num
   lea si,num
   mov ax,0000h

repeat: add al,[si]
        inc si
        dec bl
        jnz repeat

   mov sum,al

   ; return to dos
   mov ah,4ch
   int 21h
main endp
    end main
```

Input

num:
1,4,2,7,5,9,6,3,8,0

Output

sum:
45

P1_2:

Program to find average of given set of 16-bit numbers

Specification: Find average of given set of 16-bit numbers

Source Code:

```
; Program to find average of given 16-bit numbers
.model small
.stack 64
.data
   num dw 0005h, 0002h, 0004h, 0001h, 0003h
   avg dw 0000h
.code
main proc far
   ; initialize data segment register
   mov ax, @data
   mov ds, ax

   ; load address of num into si
   lea si, num
   ; cx stores count of remaining numbers
   mov cx, 05h
   mov dl, 05h

   ; ax stores sum of given numbers
   mov ax, 0000h

   ; calculate sum of given numbers
   repeat: add ax, [si]
         add si,2
         dec cx
         ;if cx is not zero, jump to repeat
         jnz repeat

   div dl         ;Divide Acc with DL
   mov avg, ax      ;Store value

   mov ah,4ch
   int 21h
main endp
    end main
```

Input

num:
5,2,4,1,3

Output

avg:
3

P1_3:

Program to find largest and smallest among an array of numbers and print the difference between them

```
.model small
.stack 256
.data
   ; numbers list
   num   db 07,04,02,07,15,09,06,03,08,04
   large db 00     ; largest number
   small db 00     ; smallest number
   diff  db 00     ; difference
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; find smalles number
   mov ax,0000h
   lea si,num
   mov cl,0ah
   mov al,[si]

   dec cl
   inc si
   repeat: cmp al,[si]
         jc cmd
         mov al,[si]

         cmd: inc si
         loop repeat

   ; store smallest number
   mov small,al

   ; find largest number
   lea si,num
   mov cl,0ah
   mov al,[si]

   dec cl
   inc si
   repeat2: cmp al,[si]
         jnc cmd2
            mov al,[si]

         cmd2: inc si
   loop repeat2

   ; find difference
   sub al,small
   mov diff,al

   ; store difference
   xor ax,ax
   mov al,diff
   call print

   ; return to dos
   mov ah,4ch
   int 21h
main endp
   print proc
      ; cx stores count of digits in number
      mov cx,0
      mov dx,0
      cmd:
         cmp ax,0
         je printN
         mov bx,10     ; initialize bx to 10
         div bx        ; extract the last digit
         push dx       ; push it to stack
         inc cx        ; increment the count
         mov dx,0
         jmp cmd
      printN:
         ; check if count is greater than zero
         cmp cx,0
         je exit
         ; pop the top of stack
         pop dx
         ; convert to ASCII
         add dx,'0'
         ; print character interrupt
         mov ah,02h
         int 21h
         ; decrease the count
         dec cx
         jmp printN
      exit: ret
   print endp
   end main
```

Input	Output
num: 07,04,02,07,15,09,06,03,08,04	diff: 13

P1_4a:

Program to find gray code of a number

Source Code:

```
.model small
.stack 64
.data
   num db 38h   ; given number
   gray db ?    ; gray code
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; calculate gray code
   mov al,num
   mov bl,al
   shr al,01
   xor bl,al

   ; store gray code
   mov gray,bl

   ; return to dos
   mov ah,4ch
   int 21h
main endp
    end main
```

Input

num: 38h

Output

gray: 24h

P1 4b:

Program to find BCD representation of hexadecimal number

Source Code:

```
.model small
.stack 64
.data
   hex db 0fh  ; given number
   bcd dw ?    ; bcd representation
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; ax stores bcd form
   mov ax,0
   ; cl stores num
   mov cl,hex

; calculate bcd form
loop2:  add al,01h
        daa
        jnc loop1
        inc ah
loop1:  dec cx
        jnz loop2

   ; store bcd representation
   mov bcd,ax

   ; return to dos
   mov ah,4ch
   int 21h
main endp
   end main
```

Input

hex: 0fh

Output

bcd: 15

P1 4c:

Program to generate Arithmetic Progression of n numbers using given First Term and Common Difference

Source Code:

```
; Program to generate arithmetic progression of n numbers
.model small
.stack 256
.data
   n db 05h    ; number of terms
   a db 03h    ; first term
   d db 09h    ; common difference
   s db ?      ; sequence
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; cl stores count remaining numbers
   xor cx,cx
   mov cl,n

   ; al holds next term
   xor ax,ax
   mov al,a

   ; bl holds common difference
   mov bl,d

   ; load address of s into di register
   lea di,s

   ; store the sequence
   ; by calculating and storing
   ; each term
   mov [di],al
   inc di
   dec cl
   repeat: add al,bl
         mov [di],al

         inc di
         loop repeat

   ; return to dos
   mov ah,4ch
   int 21h
main endp
    end main
```

Input

n: 0fh
a: 03h
d: 09h

Output

s: 03h, 0ch, 15h, 1eh, 27h

P1 4d:

Program to find Factorial of a number

Source Code:

```
; Program to find factorial of a number
.model small
.stack 64
.data
   n dw 8  ; number
   f  dd 0  ; factorial
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; load n value to cx
   xor cx,cx
   mov cx, n

   ; load address of f in di
   lea di, f

   ; ax stores less significant byte
   mov ax, 0001
   ; dx stores more significant byte
   mov dx, 0000

   ; calculate factorial
   cmd: mul cx
   loop cmd

   mov [di], ax
   inc di
   mov [di], dx

   ; return to dos
   mov ah,4ch
   int 21h

main endp
    end main
```

Input

n: 8

Output

f: 9d80h

P1 4e:

Program to Read a Character from Keyboard and Display on Screen

Source Code:

```
.model small
.stack 64
.data
   prompt db 'Enter a character: $'
   newline db ' ',13,10,'$'
   finish db 'You entered $'
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; show prompt message
   ; ask user to enter a character
   mov ah,09
   lea dx,prompt
   int 21h

   ; keyboard input interrupt
   mov ah, 1h
   ; read character into al
   int 21h
   mov cl, al  ; copy character to cl

   ; print newline
   mov   ah, 09
   lea   dx, newline
   int   21h

   ; end of prompt
   mov ah,09
   lea dx,finish
   int 21h

   ; character output interrupt
   mov dl,cl
   mov ah, 2h
   int 21h

   ; return to dos
   mov ah,4ch
   int 21h
main endp
    end main
```

Input:

Enter a character: p

Output:

You entered p

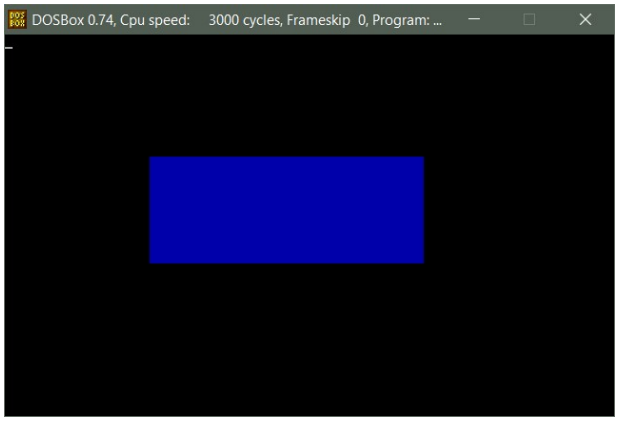
P1_4f:

Program to Change colour of a region of the display when a key is pressed on keyboard

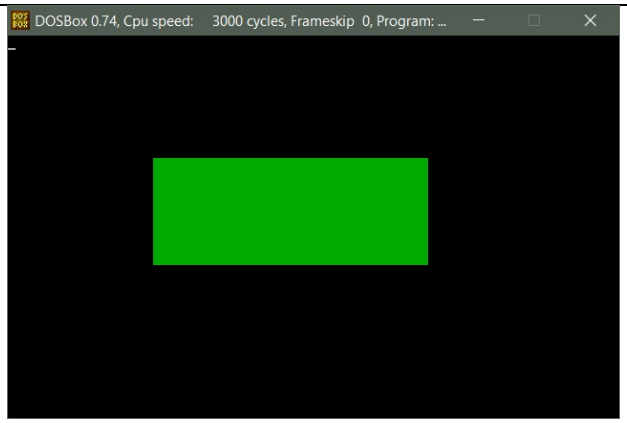
Specification: Change the colour of a region of the display when **0 is pressed** on the keyboard. For any other key press exit program.

Source Code:

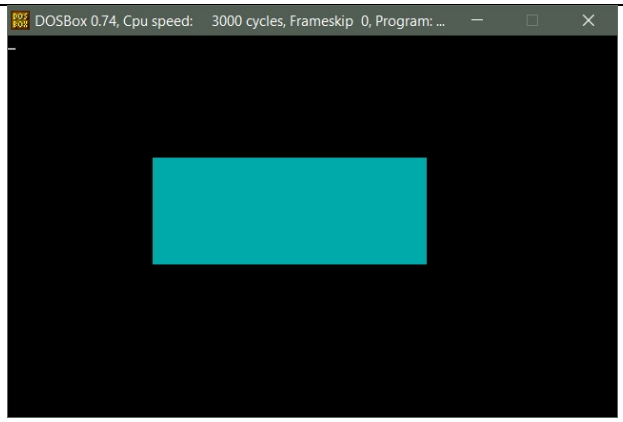
```
.model small
.stack 64
.data
   color db 10h
.code
main proc near
   ; initialize data segment register
   mov ax,@data
   mov ds,ax

   ; clear screen
   mov ax,3h
   int 10h

   repeat:
      mov ax,0600h
      mov bh,color
      mov ch,08h          ; row of window's upper left corner
      mov cl,13h          ; column of window's upper left corner
      mov dh,0eh          ; row of window's lower right corner
      mov dl,36h          ; column of window's lower right corner
      int 10h
      add color,10h       ;change background color

   ; keyboard input interrupt
   mov ah,8h
   int 21h
   ; if 0 is pressed then change display color
   cmp al,'0'
   ; otherwise exit
   je repeat

   ; clear screen
   mov ah,06
   mov bh,07h
   mov cx,0000h
   mov dx,184fh
   int 10h

   ; return to dos
   mov ah,4ch
   int 21h

main endp
    end main
```



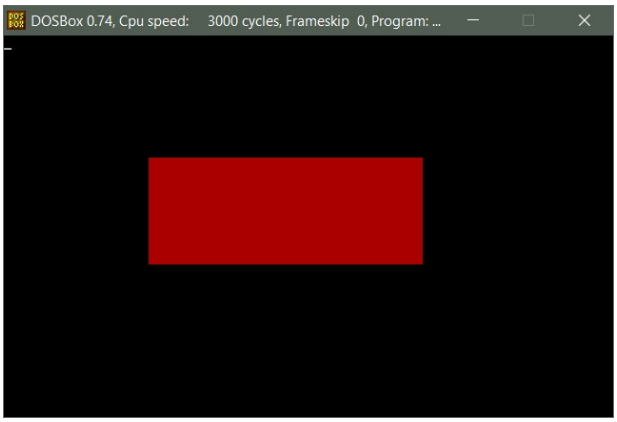
Beginning of Execution
Colour of a region of display becomes **BLUE** colour



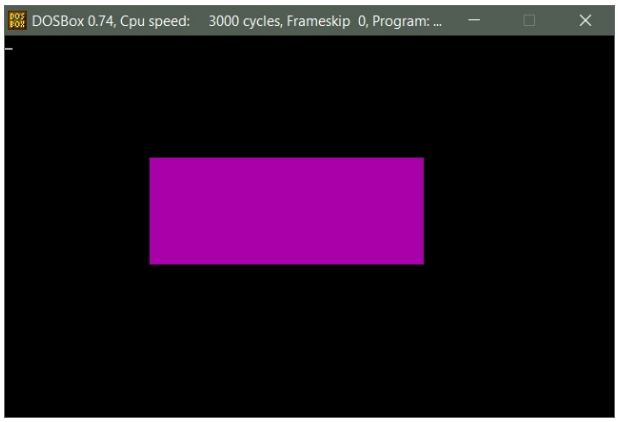
When **0** is pressed, region colour changes to **GREEN**



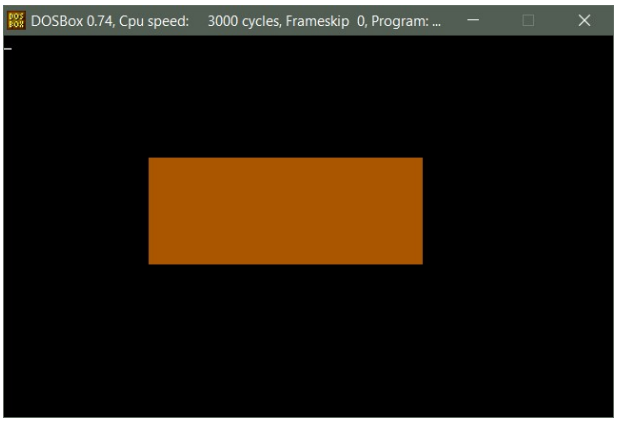
Followed by **TURQUOISE**



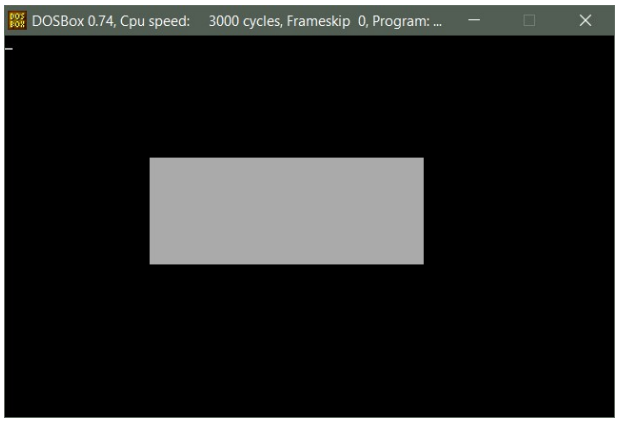
RED



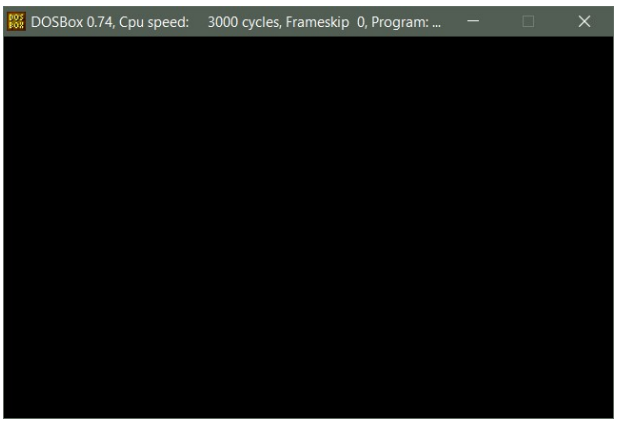
PINK



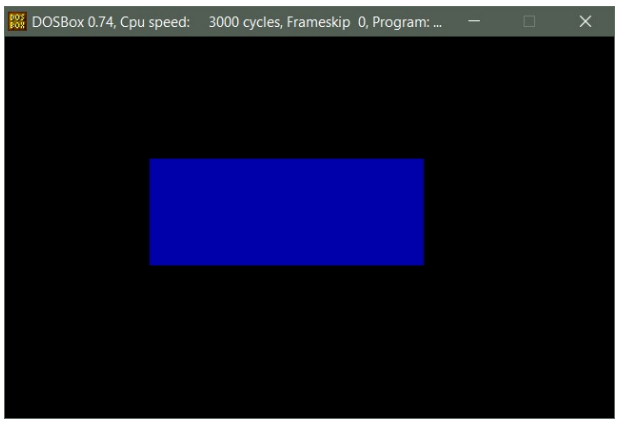
ORANGE



WHITE



BLACK



BLUE, the cycle repeats as long as **0** is pressed