COLEGIO DE MUNTINLUPA

DEPARTMENT OF COMPUTER ENGINEERING

COEN 3211 - Microprocessors Lab

Advanced Screen Processing with BIOS Interrupts

Laboratory Experiment No. 3

Grade

STUDENT NAME	: <**DAVID, Raven A.**>
STUDENT NUMBER	: <**20202011637**>
DATE PERFORMED	: <**08 February 2024**>
DATE SUBMITTED	: <**09 February 2024**>

Engr. Ricrey E. Marquez, PCpE

(Lab Instructor)

PRINCIPLES

Video System

Figure 3.1, illustrates the video system in a PC can be used to display text, or display graphics, by switching the device into the respective mode.

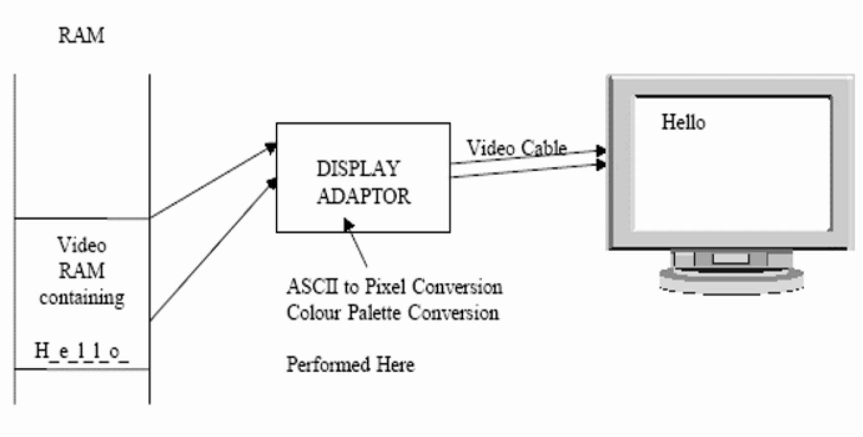


Figure 3.1 - Components of a video system

Video Buffer or Video RAM

Video Buffer or Video RAM is a block of memory locations for storing the data and information on how they are to be displayed. Likewise, there are also alternative memory locations that a video buffer may be mapped to.

The different video systems have their video buffers at different memory locations and in different sizes as shown in Figure 3.2. The other purpose of the video buffer is to make the CPU (and the program it is running) and the video adaptor independent of each other as far as transferring of information from the former to the latter is concerned.

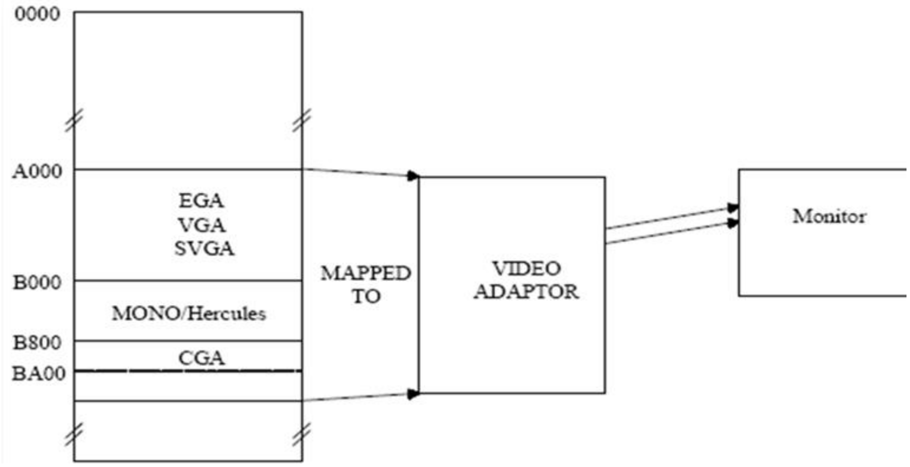


Figure 3.2 - Locations of the video RAMs for different adaptors

In graphics mode, each pixel is represented by:

- 2 bits for 320 x 200 (4 colors), or
- 1 bit for 640 x 200 (2 colors).

The video buffer is divided into two halves starting at:

- B800:0000 - store data for even-numbered scan lines.
- B800:2000 - store data for odd-numbered scan lines.

Syntax some useful INT 10H BIOS service for text or graphics modes:

`00H` - Setting the Video Mode

Syntax:

```
MOV AH, 00h        ;BIOS Interrupt service 00h
MOV AL, video_mode;screen or video mode
INT 10h            ;execute service AH 00h
```

Text mode video screen supported by emu8086:

- **`00h`** : 40 x 25 B & W Text CGA 8
- **`03h`** : 80 x 25 16 Color Text (MDPA)/CGA 4 or 8

Graphics mode video screen supported by emu8086:

- **`13h`** : 40 x 25 / 320 x 200 256 colors VGA 1

08H - Reading Attribute or Character at Cursor Position

Syntax:

```
MOV AH, 08h        ;request INT 10h service AH 08h
MOV BH, 00h        ;set default page number
INT 10h            ;execute service AH 08h
```

09H - Display Attribute or Character at Cursor Position

Syntax:

```
MOV AH, 09h        ;request INT 10h service 09h
MOV AL, character ;character to display
MOV BH, page_num   ;set page number (0)
MOV BL, attri_val ;attribute value (text or graphics)
MOV CX, num_times ;number of repeated characters
INT 10h            ;execute service AH 09h
```

- **AL** contains a single character that is to display any number times
- **BH** contains the page number
- **BL** contains the attribute
- **CX** contains the value that determines the number of times to display character contained by **AL**.

0AH - Display Character at Cursor Position

Syntax:

```
MOV AH, 0Ah        ;request INT 10h service 0Ah
MOV AL, char_val   ;character to display
MOV BH, page_num   ;set page number
MOV CX, num_times ;number of repeated character
INT 10h            ;execute service AH 0Ah
```

0CH - Write Pixel Dot

Syntax:

```
MOV AH, 0Ch        ;request INT 10h service 0Ch
MOV AL, color_val ;color of pixel dot
MOV BH, page_num   ;set page number (default 0)
MOV CX, pix_col    ;pixel column
MOV DX, pix_row    ;pixel row
INT 10h            ;execute service AH 0Ch
```

- **AL** = color of the pixel
- **BH** = page number
- **CX** = column
- **DX** = row

13H - Display Character String

Syntax:

```
MOV AH, 13h        ;request INT 10h service 13h
MOV AL, func_val   ;0, 1, 2, or 3
MOV BH, page_num   ;set page number (0)
MOV BL, attri_val  ;screen attribute
LEA BP, str_add    ;address of string (ES:BP)
MOV CX, str_len    ;length of the string
MOV DH, row_coor   ;relative screen row coordinates
MOV DL, col_coor   ;relative screen column coordinates
INT 10h            ;execute service AH 13h
```

Four functions in the AL in service AH 13h

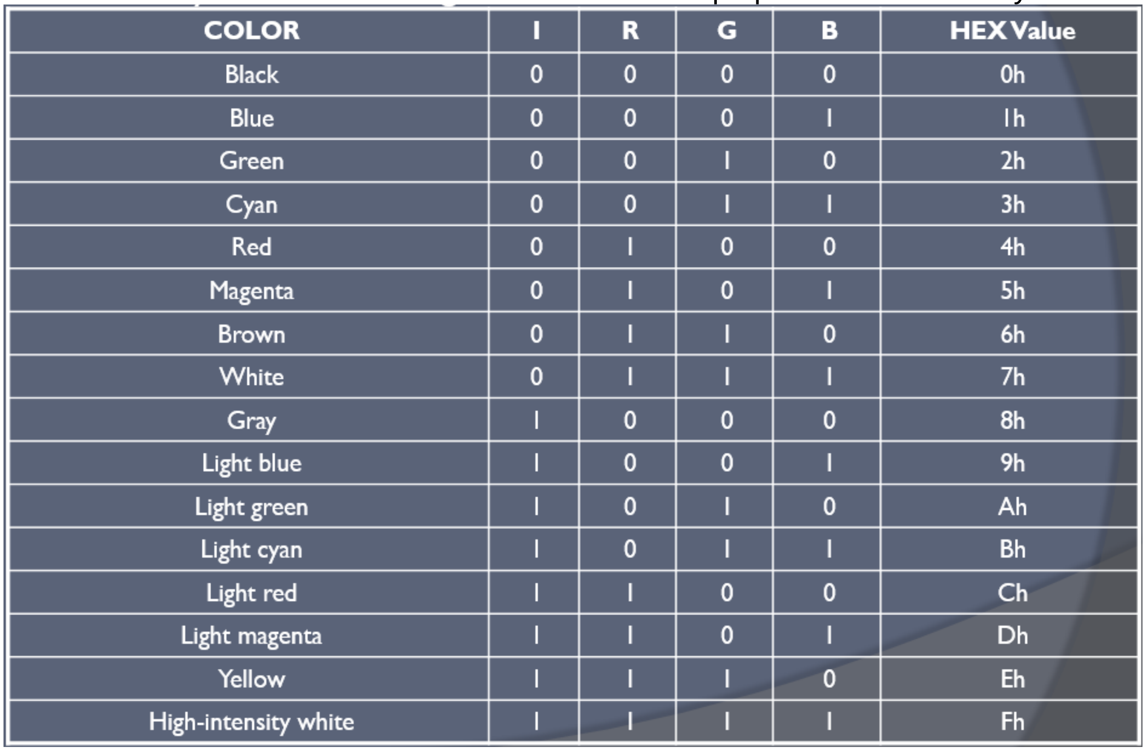
- 0 – Display attribute and string an does not advance cursor.
- 1 – Display attribute and string and advance cursor.
- 2 – Display character then attribute and do not advance cursor.
- 3 – Display character then attributes and advance cursor

Attributes Byte. This determines characteristics of each character has the following format according to bit position:

Functions of Attribute Byte Bit Numbers:

- **Bits 0 - 2 (Foreground)**
  - Control the screen foreground of the character being displayed.
- **Bits 3 (Intensity)**
  - Sets color high intensity
- **Bits 4 – 6 (Background)**
  - Control the background of the character being displayed
- **Bit 7 (Blinking)**
  - Sets blinking properties of the character being displayed

Table 3.1 – Colors that can be used to set the properties of attribute byte



COLOR	I	R	G	B	HEX Value
Black	0	0	0	0	0h
Blue	0	0	0	1	1h
Green	0	0	1	0	2h
Cyan	0	0	1	1	3h
Red	0	1	0	0	4h
Magenta	0	1	0	1	5h
Brown	0	1	1	0	6h
White	0	1	1	1	7h
Gray	1	0	0	0	8h
Light blue	1	0	0	1	9h
Light green	1	0	1	0	Ah
Light cyan	1	0	1	1	Bh
Light red	1	1	0	0	Ch
Light magenta	1	1	0	1	Dh
Yellow	1	1	1	0	Eh
High-intensity white	1	1	1	1	Fh

Text Color Attrbute Byte:

Example	**BL (Blinking)**	**Text Background**			**I (Intensity)**	**Text Foreground**		
		R	**B**	**G**		**R**	**B**	**G**
1	0	0	0	0	0	0	1	0
2	1	0	0	1	1	0	0	0

Example:

- 1 –> (Green text in black background) = 0000 0010 = 02h
- 2 –> (Blinking gray text in blue background) = 1001 1000 = 98h

OBJECTIVES AND MATERIALS

Objectives:

After this lab experiment, student should be able to:

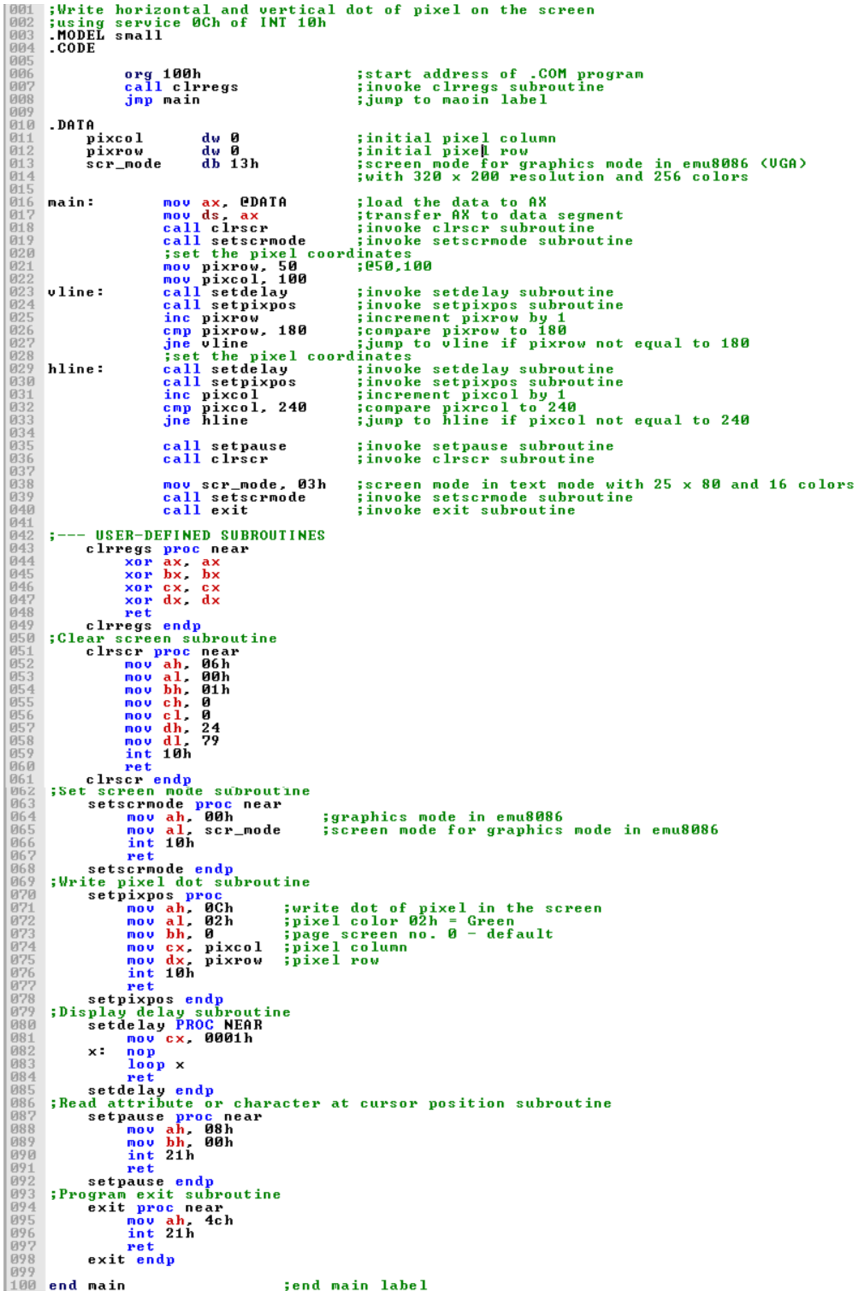
1. familiar with the operation of some useful BIOS interrupt service routines,
2. create assembly programs using some INT 10h BIOS service routines,
3. test and simulate the functionality of the assembly program with emu8086 assembler software,

Materials:

QUANTITY	PART NUMBER	DESCRIPTION
1	-	PC/Laptop with emu8086 software installed

DRILL EXERCISES

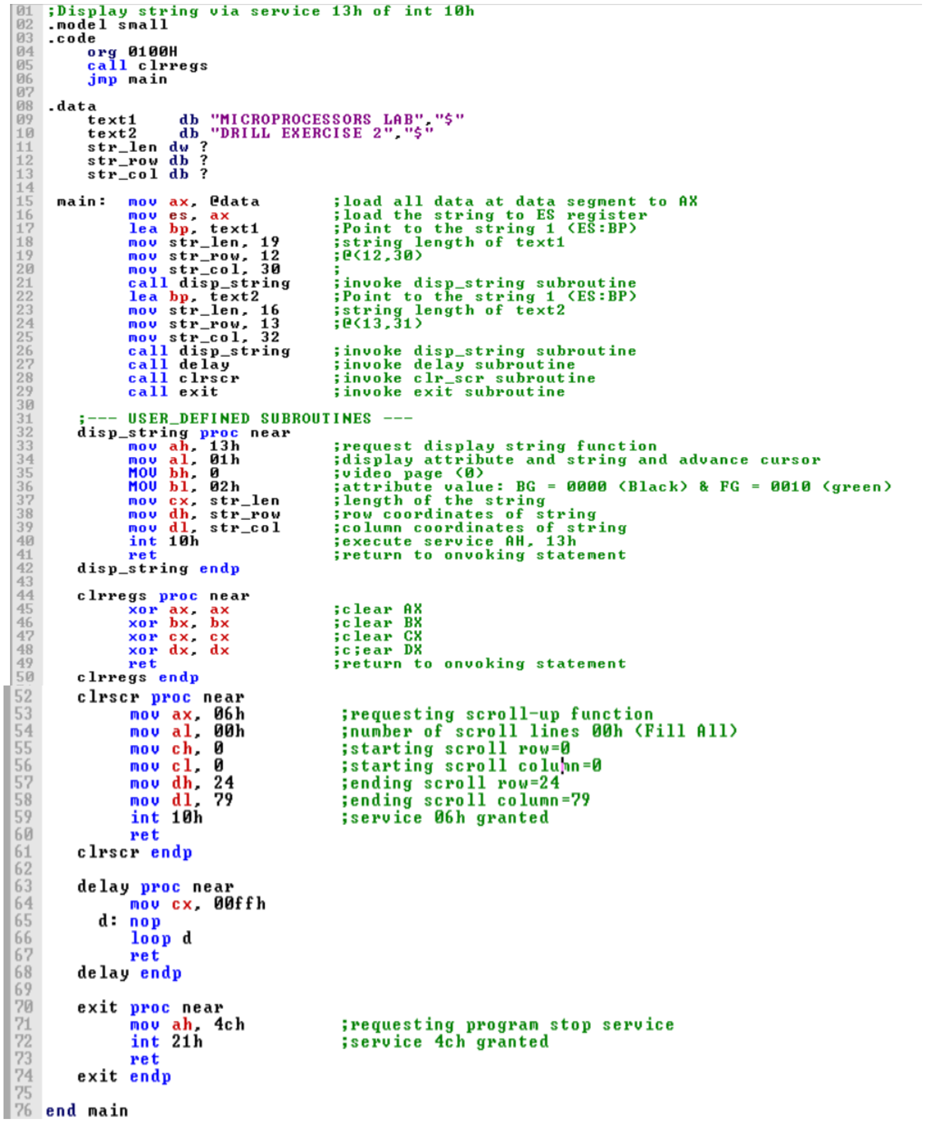
Drill Exercise 3-1 – Given the assembly language source code illustrated in Figure 3-1, re-type and test the program. Save as **`COEN3211-x_Drill3_1.asm`**.



```
;Write horizontal and vertical dot of pixel on the screen
;using service 0Ch of INT 10h
.MODEL small
.CODE

        org 100h                ;start address of .COM program
        call clrregs            ;invoke clrregs subroutine
        jmp main                ;jump to maoin label

.DATA
    pixcol      dw 0            ;initial pixel column
    pixrow      dw 0            ;initial pixel row
    scr_mode    db 13h          ;screen mode for graphics mode in emu8086 (VGA)
                                ;with 320 x 200 resolution and 256 colors

main:       mov ax, @DATA       ;load the data to AX
            mov ds, ax          ;transfer AX to data segment
            call clrscr         ;invoke clrscr subroutine
            call setscrmode     ;invoke setscrmode subroutine
            ;set the pixel coordinates
            mov pixrow, 50      ;@50,100
            mov pixcol, 100
vline:      call setdelay       ;invoke setdelay subroutine
            call setpixpos      ;invoke setpixpos subroutine
            inc pixrow          ;increment pixrow by 1
            cmp pixrow, 180     ;compare pixrow to 180
            jne vline           ;jump to vline if pixrow not equal to 180
            ;set the pixel coordinates
hline:      call setdelay       ;invoke setdelay subroutine
            call setpixpos      ;invoke setpixpos subroutine
            inc pixcol          ;increment pixcol by 1
            cmp pixcol, 240     ;compare pixrcol to 240
            jne hline           ;jump to hline if pixcol not equal to 240

            call setpause       ;invoke setpause subroutine
            call clrscr         ;invoke clrscr subroutine

            mov scr_mode, 03h   ;screen mode in text mode with 25 x 80 and 16 colors
            call setscrmode     ;invoke setscrmode subroutine
            call exit           ;invoke exit subroutine

;--- USER-DEFINED SUBROUTINES
    clrregs proc near
        xor ax, ax
        xor bx, bx
        xor cx, cx
        xor dx, dx
        ret
    clrregs endp
;Clear screen subroutine
    clrscr proc near
        mov ah, 06h
        mov al, 00h
        mov bh, 01h
        mov ch, 0
        mov cl, 0
        mov dh, 24
        mov dl, 79
        int 10h
        ret
    clrscr endp
;Set screen mode subroutine
    setscrmode proc near
        mov ah, 00h         ;graphics mode in emu8086
        mov al, scr_mode    ;screen mode for graphics mode in emu8086
        int 10h
        ret
    setscrmode endp
;Write pixel dot subroutine
    setpixpos proc
        mov ah, 0Ch     ;write dot of pixel in the screen
        mov al, 02h     ;pixel color 02h = Green
        mov bh, 0       ;page screen no. 0 - default
        mov cx, pixcol  ;pixel column
        mov dx, pixrow  ;pixel row
        int 10h
        ret
    setpixpos endp
;Display delay subroutine
    setdelay PROC NEAR
        mov cx, 0001h
    x:  nop
        loop x
        ret
    setdelay endp
;Read attribute or character at cursor position subroutine
    setpause proc near
        mov ah, 08h
        mov bh, 00h
        int 21h
        ret
    setpause endp
;Program exit subroutine
    exit proc near
        mov ah, 4ch
        int 21h
        ret
    exit endp

end main                ;end main label
```

Figure 3-1. Code listing of Drill Exercise 3-1

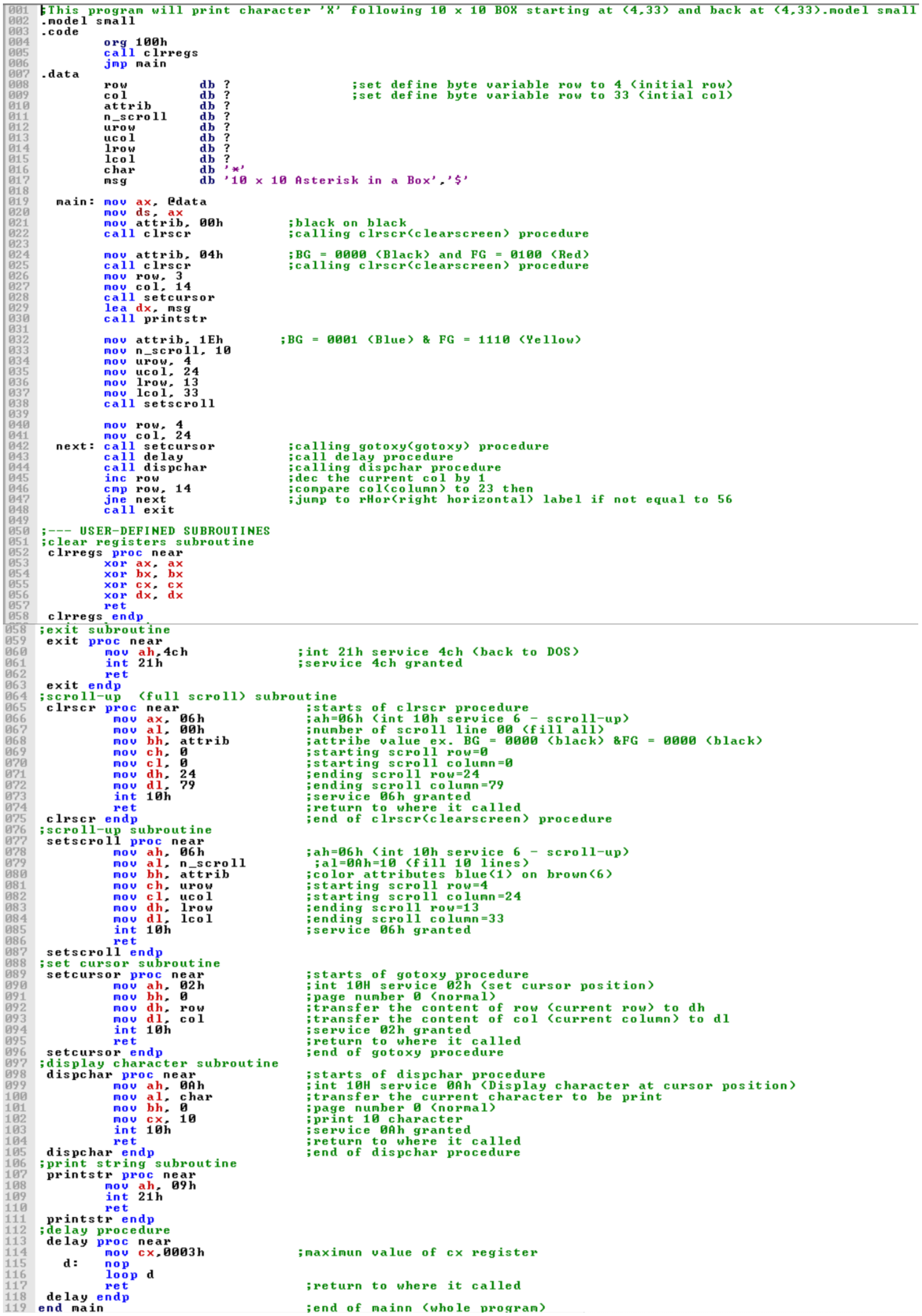
Drill Exercise 3-2 – Re-type and test the given assembly language in Figure 3-2. Save as **`COEN3211-x_Drill3_2.asm`**.



```
;Display string via service 13h of int 10h
.model small
.code
     org 0100H
     call clrregs
     jmp main

.data
     text1     db "MICROPROCESSORS LAB","$"
     text2     db "DRILL EXERCISE 2","$"
     str_len dw ?
     str_row db ?
     str_col db ?

 main:   mov ax, @data          ;load all data at data segment to AX
         mov es, ax             ;load the string to ES register
         lea bp, text1          ;Point to the string 1 (ES:BP)
         mov str_len, 19        ;string length of text1
         mov str_row, 12        ;@(12,30)
         mov str_col, 30        ;
         call disp_string       ;invoke disp_string subroutine
         lea bp, text2          ;Point to the string 1 (ES:BP)
         mov str_len, 16        ;string length of text2
         mov str_row, 13        ;@(13,31)
         mov str_col, 32
         call disp_string       ;invoke disp_string subroutine
         call delay             ;invoke delay subroutine
         call clrscr            ;invoke clr_scr subroutine
         call exit              ;invoke exit subroutine

   ;--- USER_DEFINED SUBROUTINES ---
   disp_string proc near
         mov ah, 13h            ;request display string function
         mov al, 01h            ;display attribute and string and advance cursor
         MOV bh, 0              ;video page (0)
         MOV bl, 02h            ;attribute value: BG = 0000 (Black) & FG = 0010 (green)
         mov cx, str_len        ;length of the string
         mov dh, str_row        ;row coordinates of string
         mov dl, str_col        ;column coordinates of string
         int 10h                ;execute service AH, 13h
         ret                    ;return to onvoking statement
   disp_string endp

   clrregs proc near
         xor ax, ax             ;clear AX
         xor bx, bx             ;clear BX
         xor cx, cx             ;clear CX
         xor dx, dx             ;c;ear DX
         ret                    ;return to onvoking statement
   clrregs endp
   clrscr proc near
         mov ax, 06h            ;requesting scroll-up function
         mov al, 00h            ;number of scroll lines 00h (Fill All)
         mov ch, 0              ;starting scroll row=0
         mov cl, 0              ;starting scroll column=0
         mov dh, 24             ;ending scroll row=24
         mov dl, 79             ;ending scroll column=79
         int 10h                ;service 06h granted
         ret
   clrscr endp

   delay proc near
         mov cx, 00ffh
      d: nop
         loop d
         ret
   delay endp

   exit proc near
         mov ah, 4ch            ;requesting program stop service
         int 21h                ;service 4ch granted
         ret
   exit endp

end main
```

Figure 3-2. Code listing of Drill Exercise 3-2

Drill Exercise 3-3 – Encode, compile, and test the given assembly language in Figure 3-3. Save as **`COEN3211-x_Drill3_3.asm`**.



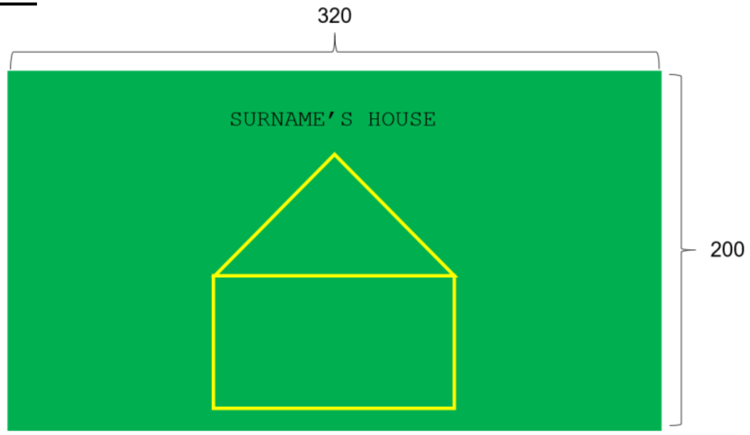
```
;This program will print character 'X' following 10 x 10 BOX starting at (4,33) and back at (4,33).model small
.model small
.code
        org 100h
        call clrregs
        jmp main
.data
        row          db ?                 ;set define byte variable row to 4 (initial row)
        col          db ?                 ;set define byte variable row to 33 (intial col)
        attrib       db ?
        n_scroll     db ?
        urow         db ?
        ucol         db ?
        lrow         db ?
        lcol         db ?
        char         db '*'
        msg          db '10 x 10 Asterisk in a Box','$'

  main: mov ax, @data
        mov ds, ax
        mov attrib, 00h        ;black on black
        call clrscr            ;calling clrscr(clearscreen) procedure

        mov attrib, 04h        ;BG = 0000 (Black) and FG = 0100 (Red)
        call clrscr            ;calling clrscr(clearscreen) procedure
        mov row, 3
        mov col, 14
        call setcursor
        lea dx, msg
        call printstr

        mov attrib, 1Eh       ;BG = 0001 (Blue) & FG = 1110 (Yellow)
        mov n_scroll, 10
        mov urow, 4
        mov ucol, 24
        mov lrow, 13
        mov lcol, 33
        call setscroll

        mov row, 4
        mov col, 24
  next: call setcursor         ;calling gotoxy(gotoxy) procedure
        call delay             ;call delay procedure
        call dispchar          ;calling dispchar procedure
        inc row                ;dec the current col by 1
        cmp row, 14            ;compare col(column) to 23 then
        jne next               ;jump to rHor(right horizontal) label if not equal to 56
        call exit

;--- USER-DEFINED SUBROUTINES
;clear registers subroutine
 clrregs proc near
        xor ax, ax
        xor bx, bx
        xor cx, cx
        xor dx, dx
        ret
 clrregs endp
;exit subroutine
 exit proc near
        mov ah,4ch              ;int 21h service 4ch (back to DOS)
        int 21h                 ;service 4ch granted
        ret
 exit endp
;scroll-up  (full scroll) subroutine
 clrscr proc near                ;starts of clrscr procedure
         mov ax, 06h             ;ah=06h (int 10h service 6 - scroll-up)
         mov al, 00h             ;number of scroll line 00 (fill all)
         mov bh, attrib          ;attribe value ex. BG = 0000 (black) &FG = 0000 (black)
         mov ch, 0               ;starting scroll row=0
         mov cl, 0               ;starting scroll column=0
         mov dh, 24              ;ending scroll row=24
         mov dl, 79              ;ending scroll column=79
         int 10h                 ;service 06h granted
         ret                     ;return to where it called
 clrscr endp                     ;end of clrscr(clearscreen) procedure
;scroll-up subroutine
 setscroll proc near
         mov ah, 06h             ;ah=06h (int 10h service 6 - scroll-up)
         mov al, n_scroll         ;al=0Ah=10 (fill 10 lines)
         mov bh, attrib          ;color attributes blue(1) on brown(6)
         mov ch, urow            ;starting scroll row=4
         mov cl, ucol            ;starting scroll column=24
         mov dh, lrow            ;ending scroll row=13
         mov dl, lcol            ;ending scroll column=33
         int 10h                 ;service 06h granted
         ret
 setscroll endp
;set cursor subroutine
 setcursor proc near             ;starts of gotoxy procedure
         mov ah, 02h             ;int 10H service 02h (set cursor position)
         mov bh, 0               ;page number 0 (normal)
         mov dh, row             ;transfer the content of row (current row) to dh
         mov dl, col             ;transfer the content of col (current column) to dl
         int 10h                 ;service 02h granted
         ret                     ;return to where it called
 setcursor endp                  ;end of gotoxy procedure
;display character subroutine
 dispchar proc near              ;starts of dispchar procedure
         mov ah, 0Ah             ;int 10H service 0Ah (Display character at cursor position)
         mov al, char            ;transfer the current character to be print
         mov bh, 0               ;page number 0 (normal)
         mov cx, 10              ;print 10 character
         int 10h                 ;service 0Ah granted
         ret                     ;return to where it called
 dispchar endp                   ;end of dispchar procedure
;print string subroutine
 printstr proc near
        mov ah, 09h
        int 21h
        ret
 printstr endp
;delay procedure
 delay proc near
        mov cx,0003h            ;maximun value of cx register
   d:   nop
        loop d
        ret                      ;return to where it called
 delay endp
end main                         ;end of mainn (whole program)
```

Figure 3-3. Code listing of Drill Exercise 3-3

PROGRAM EXERCISES

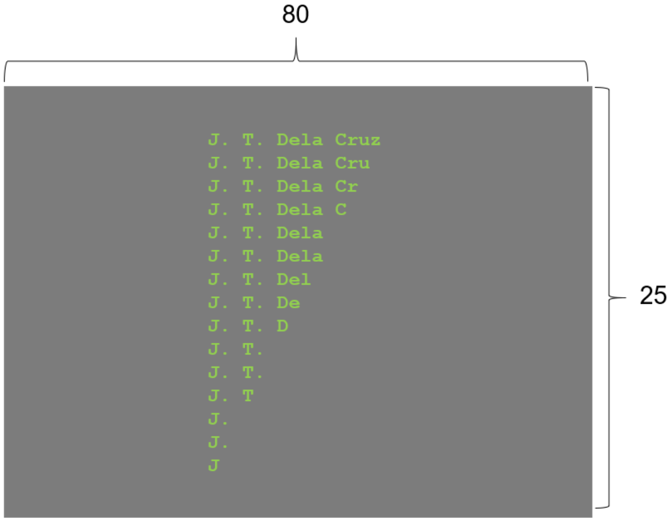
Program Exercise 3-1. Create an assembly program applying advanced BIOS screen/video services that will draw dot pixel similar to the Figure below. Save as **`COEN3211-x_ProgExer3_1.asm`**.

Sample Output:



Program Exercise 3-2. Create an assembly program applying advanced BIOS screen/video services that print your full name similar to shown below using AH, 13h. Save as **`COEN3211-x_ProgExer3_2.asm`**.

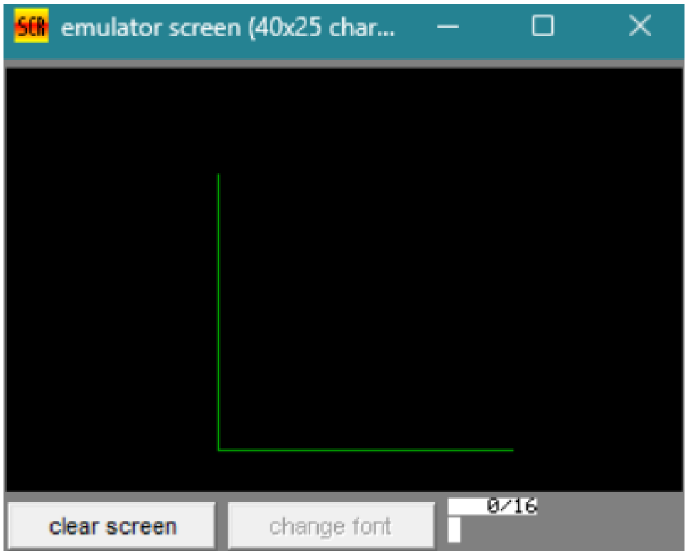
Sample Output:



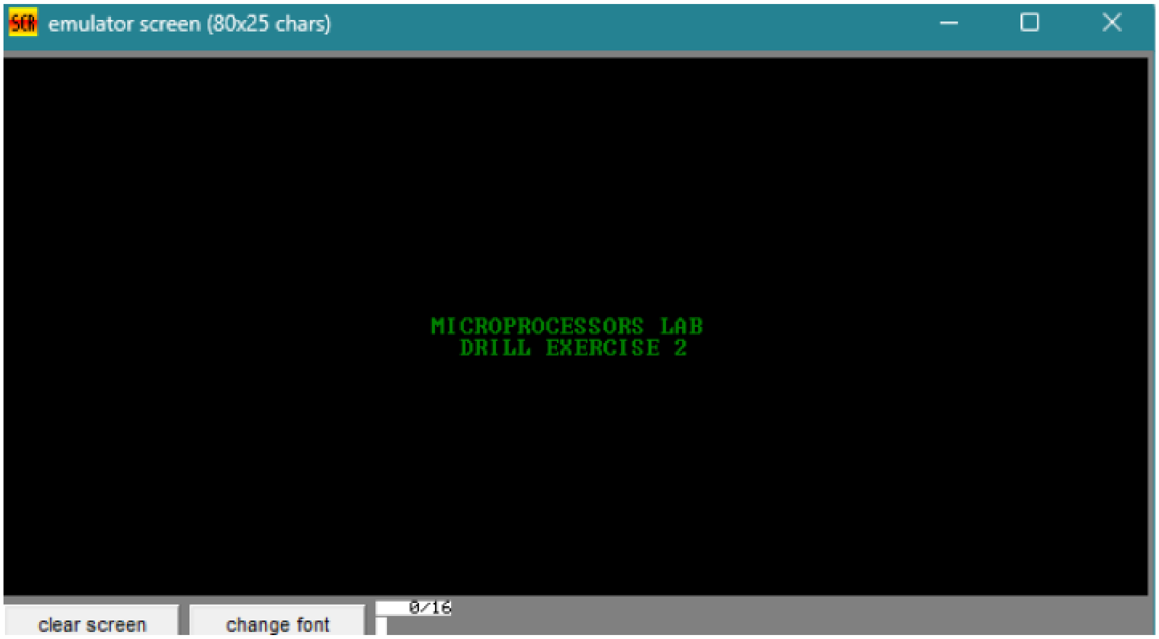
```
J. T. Dela Cruz
J. T. Dela Cru
J. T. Dela Cr
J. T. Dela C
J. T. Dela
J. T. Dela
J. T. Del
J. T. De
J. T. D
J. T.
J. T.
J. T
J.
J.
J
```

DATA RESULTS

Drill Exercise 3-1. Test output

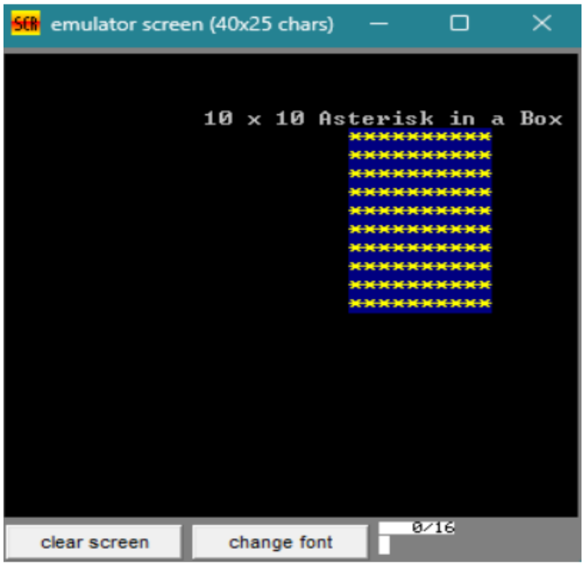


Drill Exercise 3-2. Test output



```
MICROPROCESSORS LAB
 DRILL EXERCISE 2
```

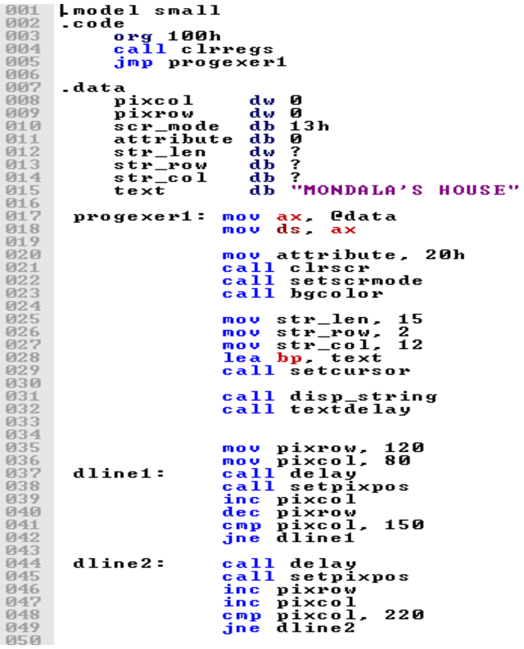
Drill Exercise 3-3. Test output



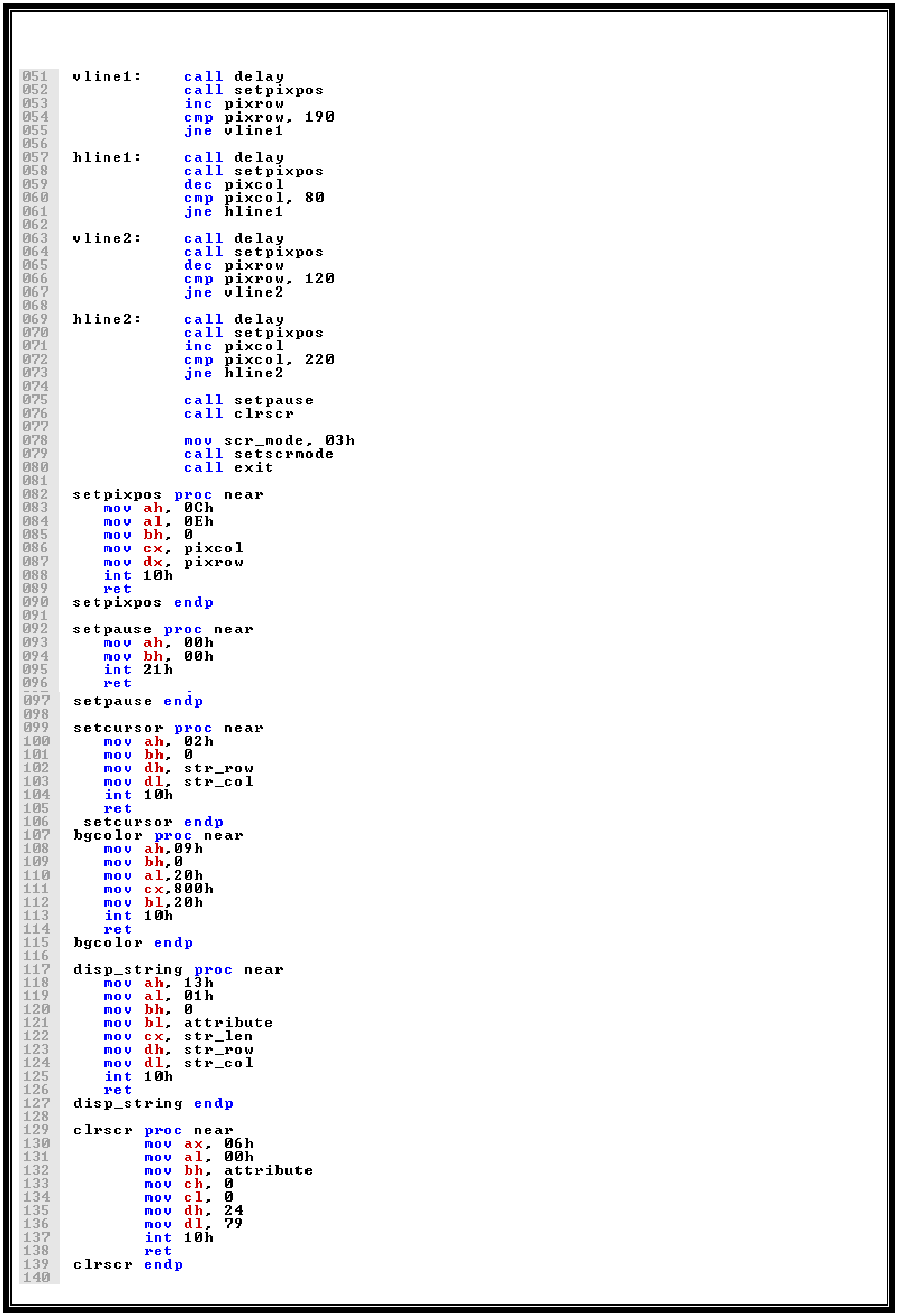
```
10 x 10 Asterisk in a Box
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

Program Exercise 3-1. Program Listing and Outputs

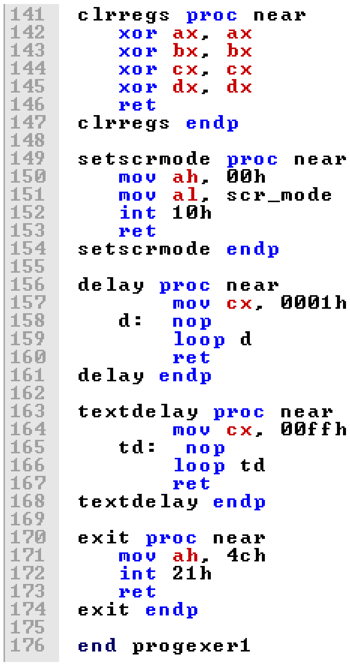
Program Listing:



```
.model small
.code
        org 100h
        call clrregs
        jmp progexer1

.data
        pixcol      dw 0
        pixrow      dw 0
        scr_mode    db 13h
        attribute   db 0
        str_len     dw ?
        str_row     db ?
        str_col     db ?
        text        db "MONDALA'S HOUSE"

 progexer1: mov ax, @data
            mov ds, ax

            mov attribute, 20h
            call clrscr
            call setscrmode
            call bgcolor

            mov str_len, 15
            mov str_row, 2
            mov str_col, 12
            lea bp, text
            call setcursor

            call disp_string
            call textdelay


            mov pixrow, 120
            mov pixcol, 80
 dline1:    call delay
            call setpixpos
            inc pixcol
            dec pixrow
            cmp pixcol, 150
            jne dline1

 dline2:    call delay
            call setpixpos
            inc pixrow
            inc pixcol
            cmp pixcol, 220
            jne dline2

```

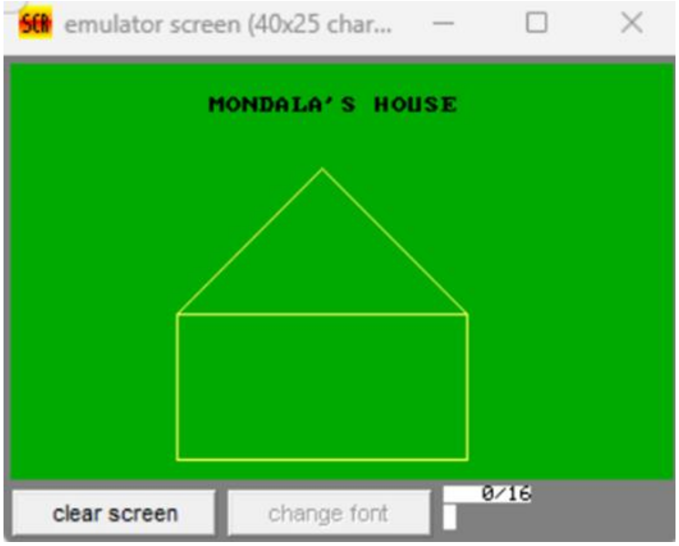


```
vline1:    call delay
           call setpixpos
           inc pixrow
           cmp pixrow, 190
           jne vline1

hline1:    call delay
           call setpixpos
           dec pixcol
           cmp pixcol, 80
           jne hline1

vline2:    call delay
           call setpixpos
           dec pixrow
           cmp pixrow, 120
           jne vline2

hline2:    call delay
           call setpixpos
           inc pixcol
           cmp pixcol, 220
           jne hline2

           call setpause
           call clrscr

           mov scr_mode, 03h
           call setscrmode
           call exit

setpixpos proc near
   mov ah, 0Ch
   mov al, 0Eh
   mov bh, 0
   mov cx, pixcol
   mov dx, pixrow
   int 10h
   ret
setpixpos endp

setpause proc near
   mov ah, 00h
   mov bh, 00h
   int 21h
   ret
setpause endp

setcursor proc near
   mov ah, 02h
   mov bh, 0
   mov dh, str_row
   mov dl, str_col
   int 10h
   ret
 setcursor endp
bgcolor proc near
   mov ah,09h
   mov bh,0
   mov al,20h
   mov cx,800h
   mov bl,20h
   int 10h
   ret
bgcolor endp

disp_string proc near
   mov ah, 13h
   mov al, 01h
   mov bh, 0
   mov bl, attribute
   mov cx, str_len
   mov dh, str_row
   mov dl, str_col
   int 10h
   ret
disp_string endp

clrscr proc near
       mov ax, 06h
       mov al, 00h
       mov bh, attribute
       mov ch, 0
       mov cl, 0
       mov dh, 24
       mov dl, 79
       int 10h
       ret
clrscr endp
```



```
clrregs proc near
    xor ax, ax
    xor bx, bx
    xor cx, cx
    xor dx, dx
    ret
clrregs endp

setscrmode proc near
    mov ah, 00h
    mov al, scr_mode
    int 10h
    ret
setscrmode endp

delay proc near
        mov cx, 0001h
    d:  nop
        loop d
        ret
delay endp

textdelay proc near
        mov cx, 00ffh
    td:  nop
        loop td
        ret
textdelay endp

exit proc near
    mov ah, 4ch
    int 21h
    ret
exit endp

end progexer1
```

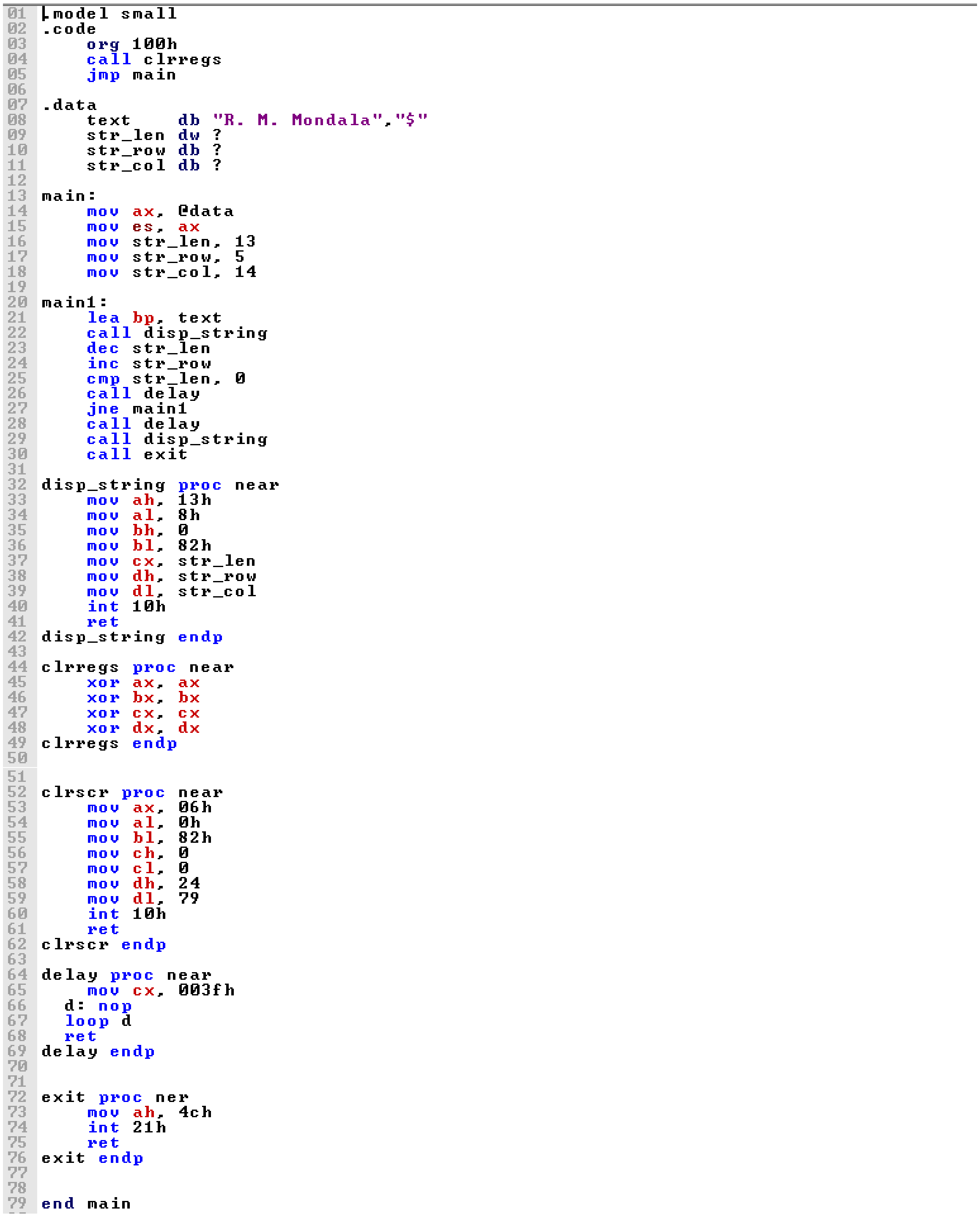
Output:



```
MONDALA'S HOUSE
```

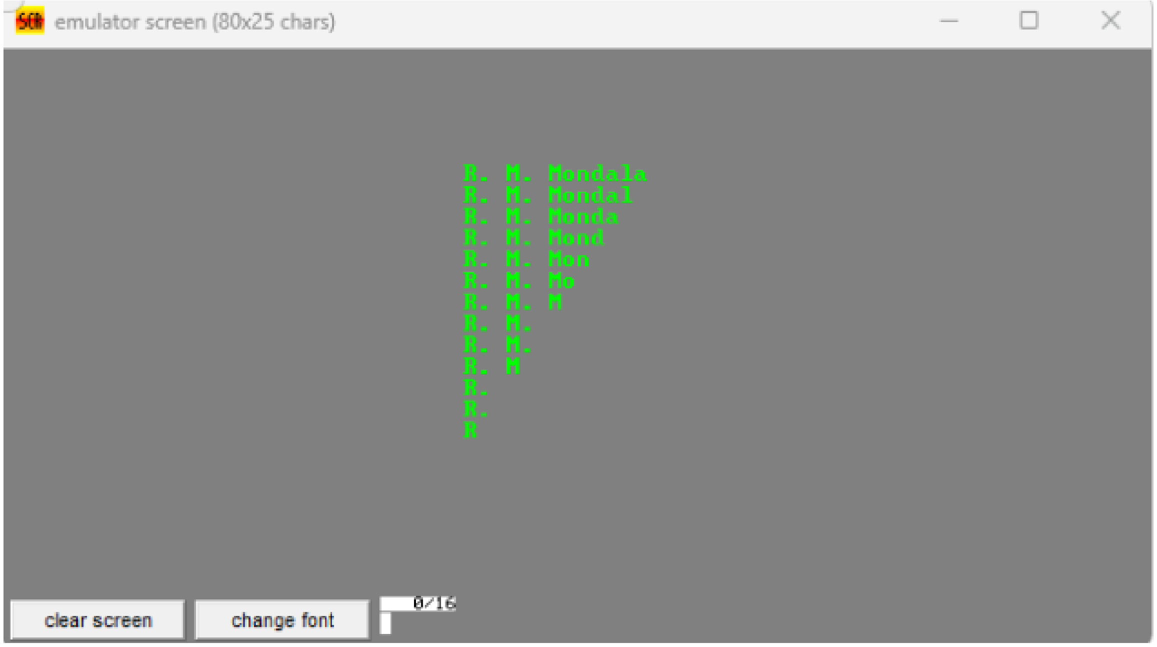
Program Exercise 3-2. Program Listing and Outputs

Program Listing:



```
.model small
.code
    org 100h
    call clrregs
    jmp main

.data
    text    db "R. M. Mondala","$"
    str_len dw ?
    str_row db ?
    str_col db ?

main:
    mov ax, @data
    mov es, ax
    mov str_len, 13
    mov str_row, 5
    mov str_col, 14

main1:
    lea bp, text
    call disp_string
    dec str_len
    inc str_row
    cmp str_len, 0
    call delay
    jne main1
    call delay
    call disp_string
    call exit

disp_string proc near
    mov ah, 13h
    mov al, 8h
    mov bh, 0
    mov bl, 82h
    mov cx, str_len
    mov dh, str_row
    mov dl, str_col
    int 10h
    ret
disp_string endp

clrregs proc near
    xor ax, ax
    xor bx, bx
    xor cx, cx
    xor dx, dx
clrregs endp


clrscr proc near
    mov ax, 06h
    mov al, 0h
    mov bl, 82h
    mov ch, 0
    mov cl, 0
    mov dh, 24
    mov dl, 79
    int 10h
    ret
clrscr endp

delay proc near
    mov cx, 003fh
  d: nop
  loop d
  ret
delay endp


exit proc ner
    mov ah, 4ch
    int 21h
    ret
exit endp


end main
```

Output:



```
R. M. Mondala
R. M. Mondal
R. M. Monda
R. M. Mond
R. M. Mon
R. M. Mo
R. M. M
R. M.
R. M.
R. M
R.
R.
R
```

DATA ANALYSIS

In the initial phase, which is the first drill, we focused on creating a pixel dot line using BIOS interrupts and services. By leveraging the appropriate interrupt calls and service routines provided by the BIOS, we were able to manipulate the display buffer effectively to generate a straight line of pixels on the screen. This exercise allowed us to grasp the fundamentals of interacting with the display hardware at a low level.

Moving forward to the next drill exercise, we advanced to printing strings on the screen using BIOS services that require string length, row, and column parameters. Through careful utilization of BIOS interrupt calls and service routines tailored for string printing, we achieved the desired output by specifying the string length along with the row and column coordinates where the string should be displayed. This exercise enhanced our understanding of managing memory and coordinates within the display buffer.

In another aspect of our exploration, which is the last drill, we experimented with printing individual characters on the screen using the 0Ah BIOS service interrupt. Additionally, we utilized BIOS interrupts for clearing the screen or scrolling when necessary. By combining these functionalities, we gained insights into character-based display manipulation and screen management, laying the groundwork for more sophisticated graphical rendering tasks.

Subsequently, for the program exercise 3_1, we ventured into creating graphical elements, such as a house, utilizing the BIOS's pixel dot writing service (09h) and leveraging graphics mode. This involved setting the background color, positioning individual pixels, and coordinating graphical elements to compose the desired image. Through this exercise, we honed our skills in graphical representation and manipulation within the constraints of BIOS services.

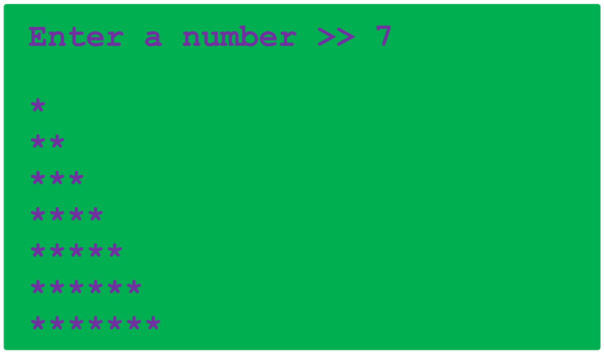
In the final segment of our lab activity, the program exercise 3-2, we explored printing strings in graphics mode, implementing a loop mechanism to dynamically decrease the string length. This involved transitioning into graphics mode and manipulating the display buffer directly to render the string with decreasing length iteratively. By combining string manipulation techniques with graphics mode capabilities, we expanded our repertoire of display rendering methods and gained proficiency in dynamic content generation within graphical environments.

QUESTIONS AND ANSWERS

Questions:

1. When to use INT 21H service 09H and INT 10H service 0AH? Provide sample example for each.
2. Create and test assembly program that will display information similar to the output below using INT 10H and INT 00H, 06H, and 0AH services. Save as **`SURNAME_COEN3211_QA2.asm`**.

Sample output:

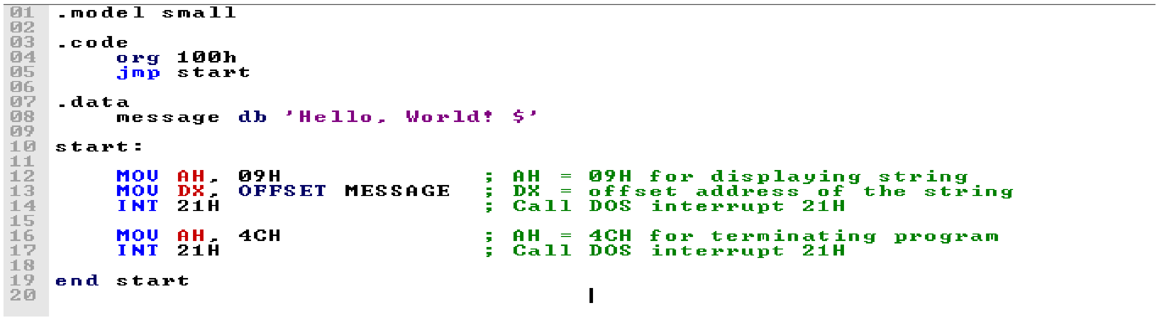


```
Enter a number >> 7

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

Answers:

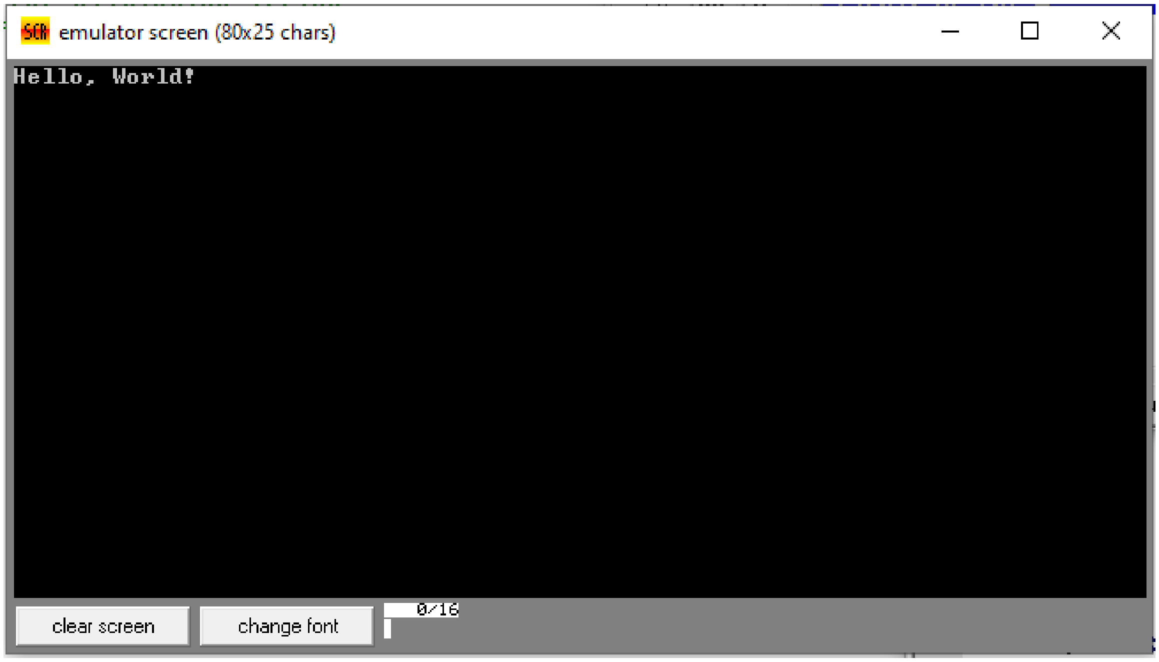
1. **INT 21H Service 09H:** This interrupt service is used to display a string of characters terminated by the '$' character in the DOS environment.

   EXAMPLE:



```
.model small

.code
     org 100h
     jmp start

.data
     message db 'Hello, World! $'

start:

     MOV  AH, 09H                 ; AH = 09H for displaying string
     MOV  DX, OFFSET MESSAGE      ; DX = offset address of the string
     INT  21H                     ; Call DOS interrupt 21H

     MOV  AH, 4CH                 ; AH = 4CH for terminating program
     INT  21H                     ; Call DOS interrupt 21H

end start
```

OUTPUT:



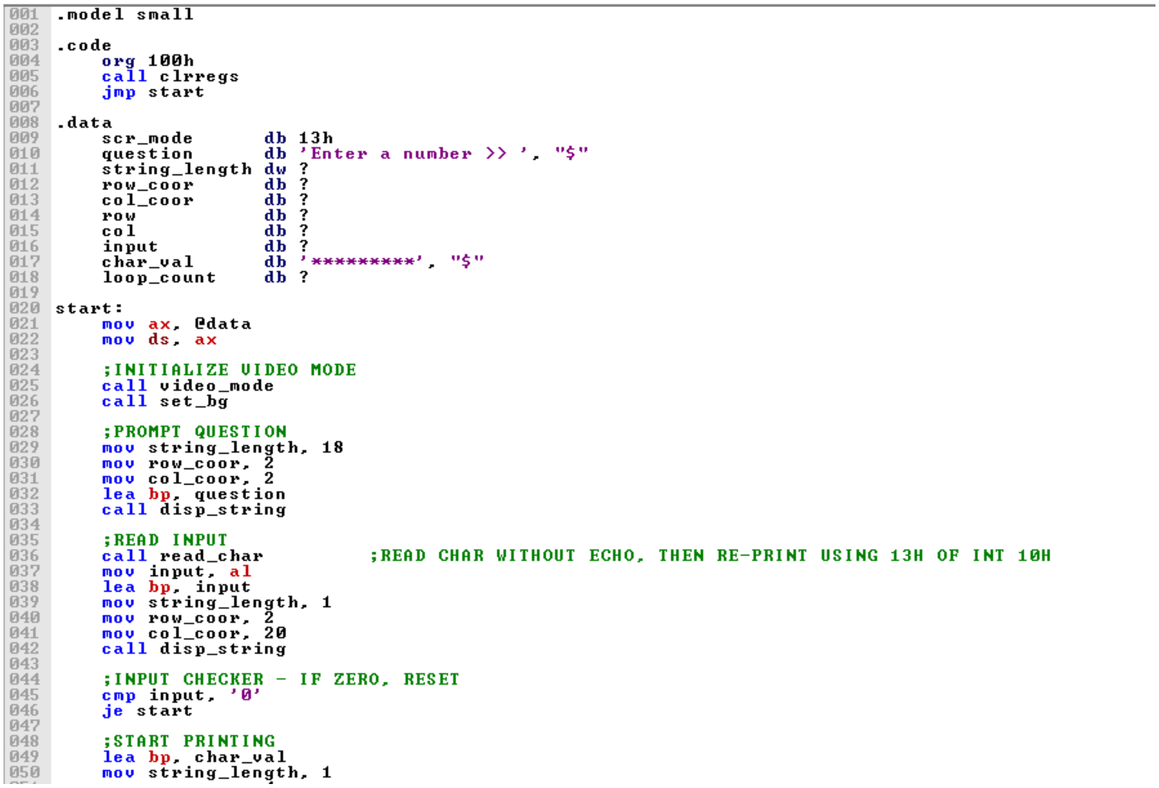
```
Hello, World!
```

INT 10H Service 0AH: This interrupt service is used for writing a character only at the cursor position on the screen. It doesn’t handle attributes or special characters.

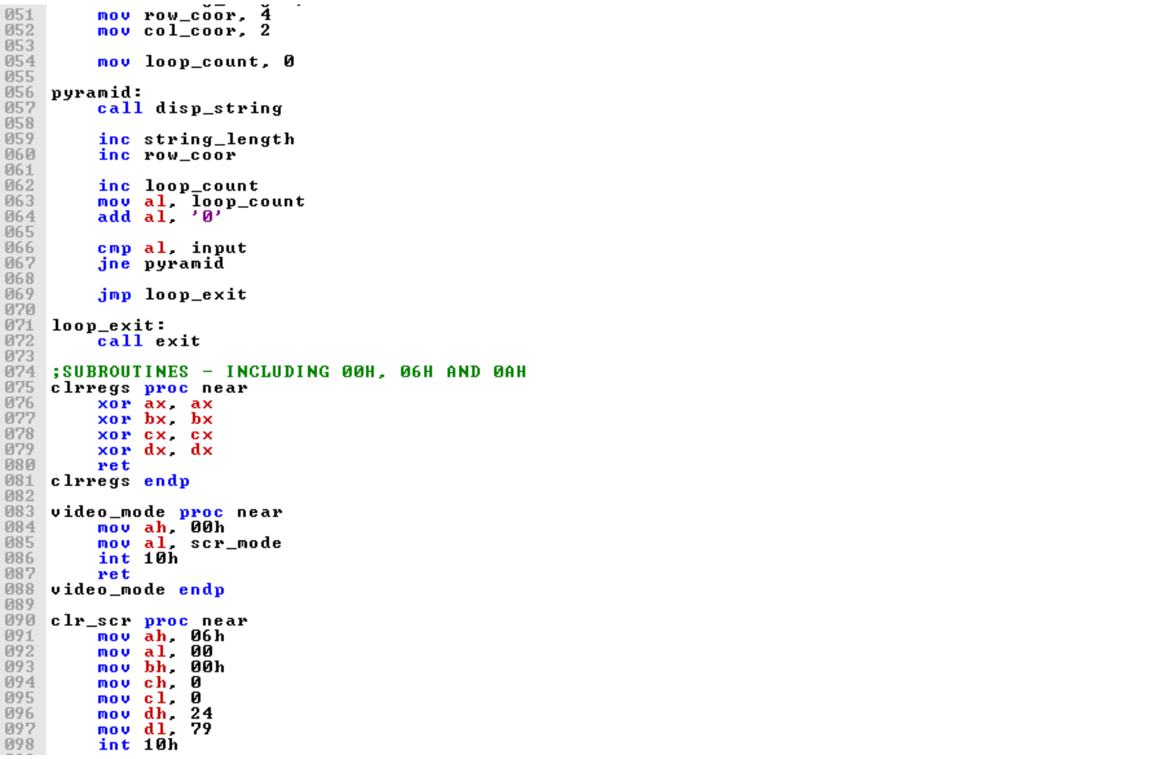
EXAMPLE: For example, to display the letter ‘A’ at the cursor position:

```
mov ah, 0Ah
mov al, 'A'
int 10h
```

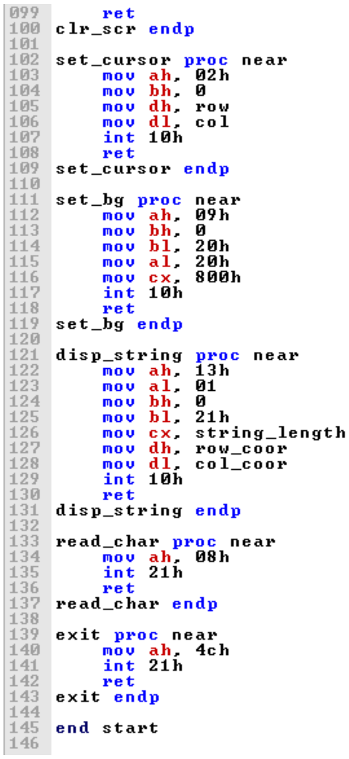
2. PROGRAM AND OUTPUT:



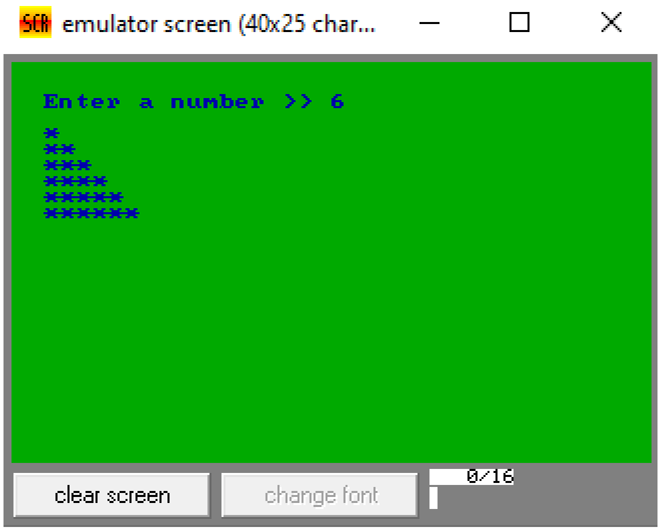
```
.model small

.code
    org 100h
    call clrregs
    jmp start

.data
    scr_mode       db 13h
    question       db 'Enter a number >> ', "$"
    string_length  dw ?
    row_coor       db ?
    col_coor       db ?
    row            db ?
    col            db ?
    input          db ?
    char_val       db '*********', "$"
    loop_count     db ?

start:
    mov ax, @data
    mov ds, ax

    ;INITIALIZE VIDEO MODE
    call video_mode
    call set_bg

    ;PROMPT QUESTION
    mov string_length, 18
    mov row_coor, 2
    mov col_coor, 2
    lea bp, question
    call disp_string

    ;READ INPUT
    call read_char          ;READ CHAR WITHOUT ECHO, THEN RE-PRINT USING 13H OF INT 10H
    mov input, al
    lea bp, input
    mov string_length, 1
    mov row_coor, 2
    mov col_coor, 20
    call disp_string

    ;INPUT CHECKER - IF ZERO, RESET
    cmp input, '0'
    je start

    ;START PRINTING
    lea bp, char_val
    mov string_length, 1
```



```
    mov row_coor, 4
    mov col_coor, 2

    mov loop_count, 0

pyramid:
    call disp_string

    inc string_length
    inc row_coor

    inc loop_count
    mov al, loop_count
    add al, '0'

    cmp al, input
    jne pyramid

    jmp loop_exit

loop_exit:
    call exit

;SUBROUTINES - INCLUDING 00H, 06H AND 0AH
clrregs proc near
    xor ax, ax
    xor bx, bx
    xor cx, cx
    xor dx, dx
    ret
clrregs endp

video_mode proc near
    mov ah, 00h
    mov al, scr_mode
    int 10h
    ret
video_mode endp

clr_scr proc near
    mov ah, 06h
    mov al, 00
    mov bh, 00h
    mov ch, 0
    mov cl, 0
    mov dh, 24
    mov dl, 79
    int 10h
```



```
    ret
clr_scr endp

set_cursor proc near
    mov ah, 02h
    mov bh, 0
    mov dh, row
    mov dl, col
    int 10h
    ret
set_cursor endp

set_bg proc near
    mov ah, 09h
    mov bh, 0
    mov bl, 20h
    mov al, 20h
    mov cx, 800h
    int 10h
    ret
set_bg endp

disp_string proc near
    mov ah, 13h
    mov al, 01
    mov bh, 0
    mov bl, 21h
    mov cx, string_length
    mov dh, row_coor
    mov dl, col_coor
    int 10h
    ret
disp_string endp

read_char proc near
    mov ah, 08h
    int 21h
    ret
read_char endp

exit proc near
    mov ah, 4ch
    int 21h
    ret
exit endp

end start
```



```
Enter a number >> 6
*
**
***
****
*****
******
```

CONCLUSION

Firstly, we have successfully familiarized ourselves with the operation of several essential BIOS interrupt service routines. By studying and implementing these routines, such as those responsible for pixel manipulation and screen clearing, we have gained invaluable insights into the inner workings of low-level system operations. This familiarity lays a solid foundation for future endeavors in system programming and hardware interaction, as understanding BIOS interrupts is fundamental to effectively harnessing system resources.

Secondly, we have demonstrated proficiency in creating assembly programs utilizing INT 10h BIOS service routines. By leveraging the capabilities offered by INT 10h, we were able to accomplish tasks ranging from basic pixel dot line creation to more complex graphical rendering. The ability to interface with BIOS services not only facilitates efficient programming but also empowers us to harness the full potential of underlying hardware resources. Through hands-on experimentation and coding exercises, we have honed our skills in crafting efficient and functional assembly programs tailored to specific system requirements.

Lastly, we have thoroughly tested and simulated the functionality of our assembly programs using the emu8086 assembler software. Through rigorous testing and debugging, we have ensured the reliability and robustness of our code, validating its behavior under various scenarios and inputs. The emulation environment provided by emu8086 has proven invaluable in facilitating rapid prototyping and iterative refinement of our programs. By meticulously examining program output and behavior within a controlled environment, we have gained confidence in the correctness and effectiveness of our assembly implementations.

REFERENCES

Wikipedia Article:

Wikipedia. (2015, June). INT 10H*: In Wikipedia. Retrieved February 9, 2024, from https://en.wikipedia.org/wiki/INT_10H.*