COLEGIO DE MUNTINLUPA

DEPARTMENT OF COMPUTER ENGINEERING

COEN 3211 - Microprocessors Lab

Video Screen and Keyboard Processing with BIOS Interrupts

Laboratory Experiment No. 2

Grade

STUDENT NAME	: <**DAVID, Raven A.**>
STUDENT NUMBER	: <**20202011637**>
DATE PERFORMED	: <**28 January 2024**>
DATE SUBMITTED	: <**29 January 2024**>

Engr. Ricrey E. Marquez, PCpE

(Lab Instructor)

PRINCIPLES

ROM / Flash ROM contains a suite of permanent programs that control:

- Data transfers between the CPU and the peripheral devices; and
- Initialization of the system at boot-up time.

BIOS I/O routines provide services, in the form of common routines, that are easily available to assembly language programmers. These routines enable simple transfers of data between the CPU and:

- Display screen memory,
- Keyboard interface,
- Printer interface,
- Serial I/O interface and
- Disk drives.

The tasks associated with the routines are:

- Primary memory test
- Set up the interrupt vector table
- Initialize the main circuit-board support chips
- Check which equipment is attached to the system.
- Fetch the current time-of-day and date from the M6818 real-time clock chip
- Allow extension-board equipment to initialize itself
- Load the operating system or invoke BASIC ROM

An assembly language programmer, not only has the access to low-level BIOS routine, but they also have access to higher-level routines in the form of DOS functions. Both types of access make use of the interrupt vector table (IVT) Therefore, knowledge of CPU interrupts is essential.

Syntax some useful INT 10H BIOS service:

02H - Setting the Cursor in the Screen

Syntax:

```
mov ah, 02h        ;BIOS Service AH 02 (set cursor)
mov bh, page_no    ;page (screen) no. default 0
mov dh, row_value  ;row value (dec. or hex value)
mov dl, col_value  ;column value (dec. or hex value)
int 10h            ;execute service AH 02
```

06H - Clearing the Screen

Syntax:

```
mov ah, 06h         ;BIOS Function AH 06 (scroll-up)
mov bh, attrib_no.  ;Attributes (bgnd & fgnd)
mov al, no._lines   ;if AL=00 (full screen)
mov ch, upper_row   ;upper row value
mov cl, upper_col   ;upper column value
mov dh, lower_row   ;lower row value
mov dl, lower_col   ;lower column value
int 10h             ;execute service AH 06h
```

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 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 Attribute 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 2-1 – Given the assembly language source code list below, re-type and test the program. Save as **`COEN3211_3x-x_Drill2_1.asm`**.

```
.model small
;code starts here
.code
       org 100h          ;indicates the instruction starts at offset 100h
       jmp entry         ;jump to label entry

;declaration of data (variable)
.data
        pass        db "Guest The Three Letter Password : ",9,'$'
        inva        db "..Invalid Password Try Again..",'$'
        hello       db "A C C E S S  G R A N T E D !",'$'
        msg1        db "Welcome",'$'
        msg2        db "to",'$'
        msg3        db "Assembly Language",'$'
        msg4        db "Programmed by: Engr. Ricrey E. Marquez, CpE",'$'
        row         db 0
        col         db 0
        attri       db 0

entry:  mov attri,02h       ;attributes (BH):bgd-0=black & fgd-2=green
        call clrscr         ;call procedure clrscr (clear screen)

        mov row, 12         ;set row equal to 12
        mov col, 10         ;set col equal to 10
        call gotoxy         ;call gotoxy procedure

        lea dx, pass         ;transfer the content of pass variable @ .data
        mov ah, 09h         ;int 21h service AH 09h (display string)
        int 21h             ;execute service AH 09h
        jmp read            ;jump to read label

start:  mov attri, 74h      ;attributes (BH):bgd-7=white & fgd-4=red
        call clrscr

        mov row, 13
        mov col, 15
        call gotoxy
        call invalid
        jmp entry

read:   mov ah,07h          ;INT 21h service AH 07 (read key with wait)
        int 21h             ;DOS service request for AH=07h
        cmp al, 'R'         ;compare if AL equal to 'R' then
        jne start           ;jump to label start if AL not equal 'R'(Read Again)

        mov ah,07h
        int 21h             ;if AL equal 'R' then get next character
        cmp al, 'E'         ;compare if AL equal to 'E' then
        jne start           ;jump to label start if AL not equal 'E' (Read Again)

        mov ah,07h
        int 21h             ;if AL equal 'E' then get next character
```

```
        cmp al, 'M'        ;compare if AL equal to 'M' then
        jne start          ;jump to label start if AL not equal 'M'

        jmp print         ;if AL equal 'M' jump to label print (to print the
content of .data)

print:  mov attri,0A8H
        call clrscr
        mov row, 13
        mov col, 20
        call gotoxy

        mov dx,offset hello  ;transfer the content of .data to data register (dx)
        mov ah,09h         ;INT 21h service to display string (DX) from the screen
        int 21h               ;DOS service request for AH=09h
        call delay

        mov attri,5Eh
        call clrscr
        mov row, 10
        mov col, 35
        call gotoxy

        mov dx, offset msg1
        mov ah, 09h
        int 21h
        call delay

        mov row, 12
        mov col, 37
        call gotoxy

        mov dx, offset msg2
        mov ah, 09h
        int 21h
        call delay

        mov row, 14
        mov col, 29
        call gotoxy

        mov dx, offset msg3
        mov ah, 09h
        int 21h
        call delay

        mov attri,04h
        call clrscr

        mov row, 12
        mov col, 15
        call gotoxy

        mov dx, offset msg4
        mov ah, 09h
        int 21h
        call delay
        mov attri, 07h
        call clrscr
        call exit

;--- USER-DEFINED PROCEDURES ---
;procedure exit starts here
```

```
exit proc near
        mov ah, 4ch       ;DOS Function service AH 4Ch (return to DOS)
        int 21h           ;execute service 4Ch
exit endp
;procedure delay start here
delay proc near
        mov cx, 000fh
  del1: nop
        loop del1
        ret
delay endp
;procedure set cursor starts here
gotoxy proc near
       mov ah, 02h
       mov bh, 0
       mov dh, row
       mov dl, col
       int 10h
       ret
gotoxy endp
;procedure clear screen starts here
clrscr proc near
        mov ah, 06h     ;BIOS Service AH 06h (Screen Scrolling)
        mov al, 00      ;AL=00 (Full Screen)
        mov bh, Attri   ;attributes (background & foreground)
        mov ch, 0
        mov cl, 0
        mov dh, 24
        mov dl, 79
        int 10h
        ret
clrscr endp

invalid proc near
        lea dx, inva
        mov ah, 09h
        int 21h
        call delay
        ret
invalid endp

end entry               ;end of the main program
```

Figure 2-1. Code listing of Drill Exercise 2-1

PROGRAM EXERCISES

Program Exercise 2-1. Create an assembly program applying DOS and BIOS function services that prompt the user to select an operation from the menu, then program will execute the said operation. And also, program will be asked you to select another operation if user enter 'y' or 'Y' otherwise the program will terminate. Save as **COEN3211_3x-x_Drill2_1.asm**

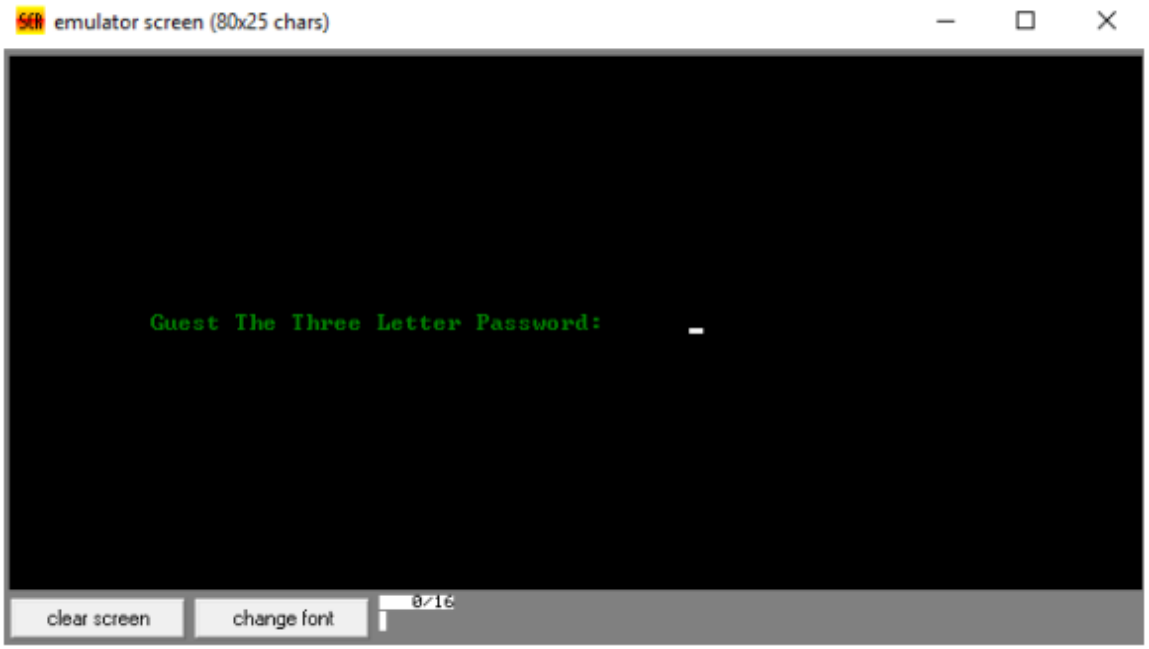
Sample Input and Output:

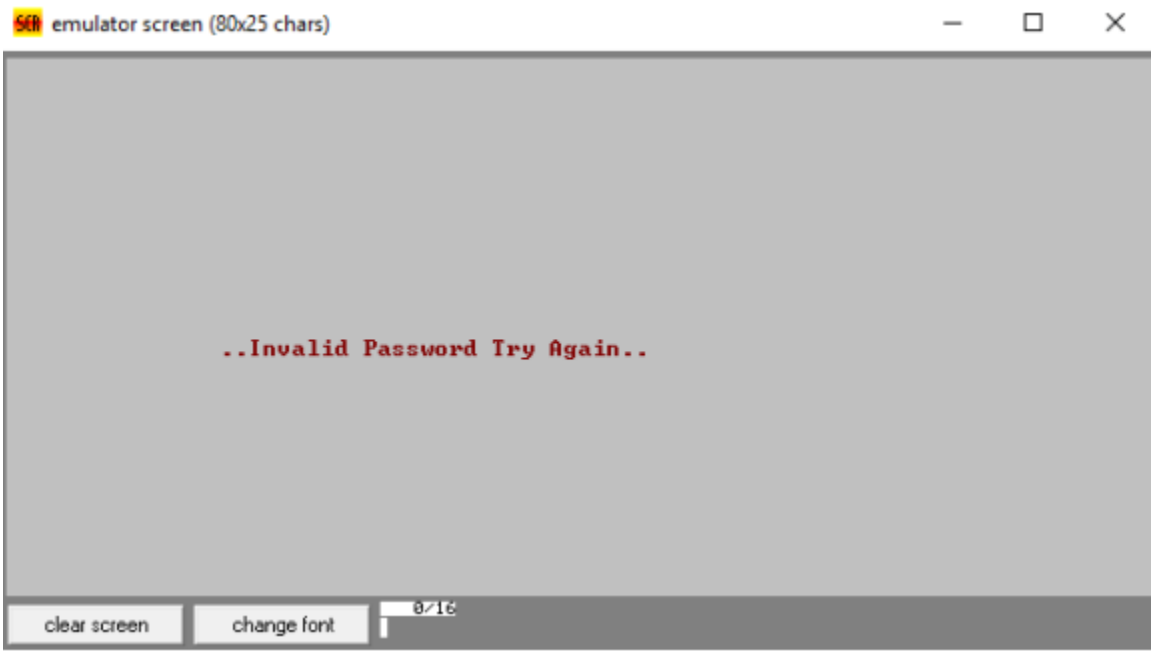
```
                    This assembly program will prompt
                    the user to select an operation then
                    execute the desired sequence of character.

                    programmed by: JTamad
                    <clear the screen after the delay>

--------------------------------------------------------------------------------
                              CHARACTER OPERATIONS
--------------------------------------------------------------------------------
                    1 - Print lowercase letter from a to z
                    2 - Print all odd character from Z to A
                    3 - Print character in alternate lowercase & uppercase
                    4 - Print all digit in descending order
                    5 - Print all digit in ascending order
                    0 - Exit
--------------------------------------------------------------------------------
                    Select an operation from the menu: 1
                       <clear the screen after the delay>

                             Character operation 1:
                          abcdefghijklmnopqrstuvwxyz
                       <clear the screen after the delay>

                 Do you want to try another operation? [y/n]: y
```

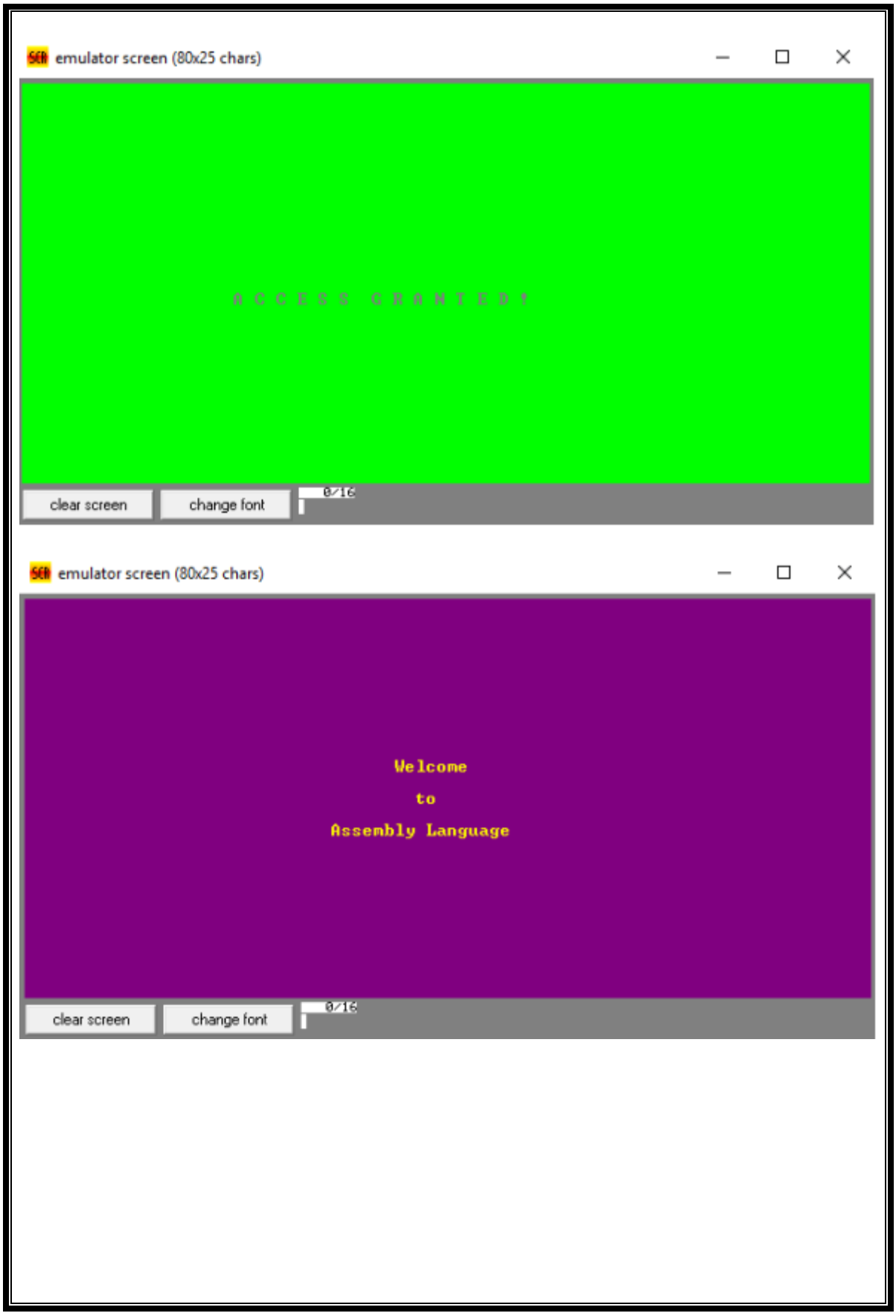
```
                    This assembly program will prompt
                    the user to select an operation then
                    execute the desired sequence of character.

                    programmed by: JTamad
                    <clear the screen after the delay>

--------------------------------------------------------------------------------
                               CHARACTER OPERATIONS
--------------------------------------------------------------------------------
                    1 - Print lowercase letter from a to z
                    2 - Print all odd character from Z to A
                    3 - Print character in alternate lowercase & uppercase
                    4 - Print all digit in descending order with space
                    5 - Print all digit in ascending order with space
                    0 - Exit
--------------------------------------------------------------------------------
                    Select an operation from the menu: 5

                           <clear the screen after the delay>

                              Character operation 5:
                                0 1 2 3 4 5 6 7 8 9
                          <clear the screen after the delay>

                 Do you want to try another operation? [y/n]: n
                      <clear the screen then program exit>
```

DATA RESULTS

Drill Exercise 2-1. Test output






emulator screen (80x25 chars)
A C C E S S G R A N T E D !
clear screen
change font
0/16
emulator screen (80x25 chars)
Welcome
to
Assembly Language
clear screen
change font
0/16

Program Exercise 2-1. Program Listing and Outputs

Program Listing:

```
.model small

.code
    org 100h
    call clr_regs
    jmp start

.data
    introOne db "This assembly program will prompt",10,13, "$"
    introTwo db "the user to select an operation then", 10,13, "$"
    introThree db "execute the desired sequence of character.",10,13,"$"
    newline db 10, "$"
    programmer db "programmed by: Group 5 ", 10,13, "$"

    menuLine db "--------------------------------------------------------------------------------",10,13, "$"
    menuOps db "CHARACTER OPERATIONS", 10,13, "$"
    optionOne db "1 - Print lowercase letter from a to z", 10,13,"$"
    optionTwo db "2 - Print all odd character from Z to A",10,13,"$"
    optionThree db "3 - Print character in alternate lowercase & uppercase",10,13,"$"
    optionFour db "4 - Print all digit in descending order",10,13,"$"
    optionFive db "5 - Print all digit in ascending order",10,13,"$"
    optionSix db "0 - Exit",10,13,"$"
    questionPrompt db "Select an operation from the menu: $"
    userInput db ?

    firstOperation db "Character operation 1: ",10,13, "$"
    secondOperation db "Character operation 2: ",10,13, "$"
    thirdOperation db "Character operation 3: ",10,13, "$"
    fourthOperation db "Character operation 4: ",10,13, "$"
    fifthOperation db "Character operation 5: ",10,13, "$"
    sixthOperation db "Character operation 0: ",10,13, "$"

    firstLetter db "a"
    reverseLetter db "Z"

    whiteSpace db " "

    tryAgain db "Do you want to try another operation? [y/n]: $"
    userResponse db ?

    attri db 0
    col db 0
    row db 0

start:
    mov attri, 02h
    call clr_scr

    mov row, 9
    mov col, 18


    call set_cursor

    lea dx, introOne
    call print_string
    call disp_del

    mov row, 10
    mov col, 18
    call set_cursor

    lea dx, introTwo
    call print_string
    call disp_del

    mov row, 11
    mov col, 18
    call set_cursor

    lea dx, introThree
    call print_string
    call disp_del

    mov row, 12
    mov col, 18
    call set_cursor

    lea dx, newline
    call print_string
    call disp_del

    mov row, 14
    mov col, 18
    call set_cursor

    lea dx, programmer
    call print_string
    call disp_del

    jmp menu_start

menu_start:
    mov attri, 02h
    call clr_scr

    mov row, 6
    mov col, 0
    call set_cursor
```

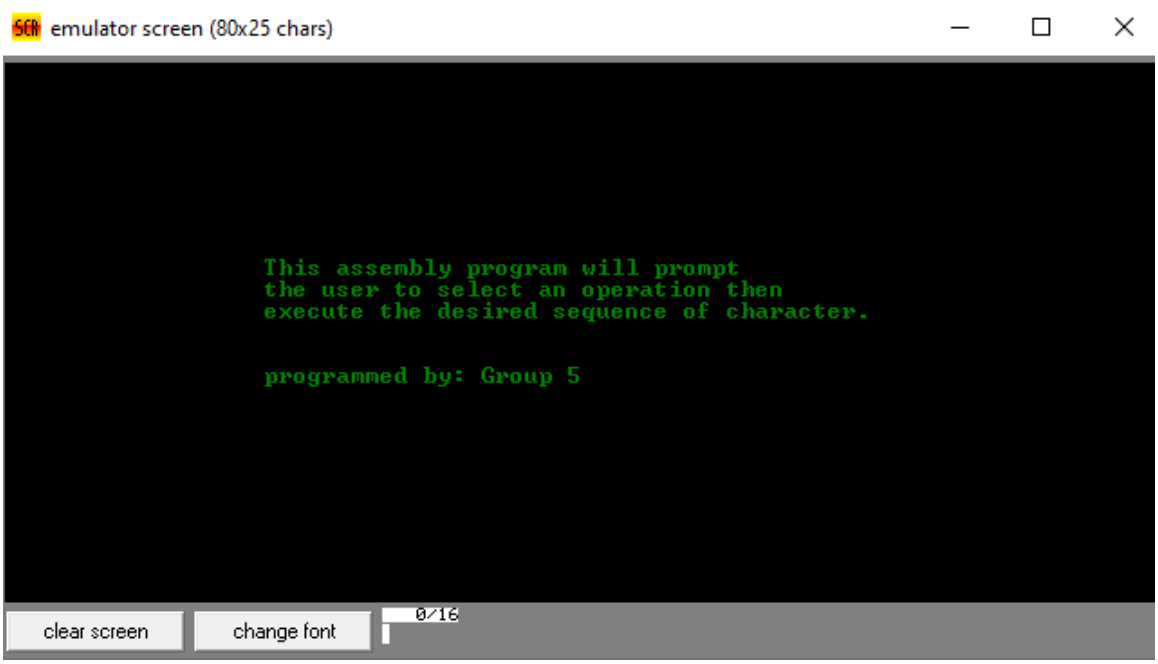
```
099     lea dx, menuLine
100     call print_string
101
102     mov row, 7
103     mov col, 29
104     call set_cursor
105
106     lea dx, menuOps
107     call print_string
108
109     mov row, 8
110     mov col, 0
111     call set_cursor
112
113     lea dx, menuLine
114     call print_string
115
116     mov row, 9
117     mov col, 13
118     call set_cursor
119
120     lea dx, optionOne
121     call print_string
122
123     mov row, 10
124     mov col, 13
125     call set_cursor
126
127     lea dx, optionTwo
128     call print_string
129
130     mov row, 11
131     mov col, 13
132     call set_cursor
133
134     lea dx, optionThree
135     call print_string
136
137     mov row, 12
138     mov col, 13
139     call set_cursor
140
141     lea dx, optionFour
142     call print_string
143
144     mov row, 13
145     mov col, 13
146     call set_cursor


147
148     lea dx, optionFive
149     call print_string
150
151     mov row, 14
152     mov col, 13
153     call set_cursor
154
155     lea dx, optionSix
156     call print_string
157
158     mov row, 15
159     mov col, 0
160     call set_cursor
161
162     lea dx, menuLine
163     call print_string
164
165     mov row, 16
166     mov col, 21
167     call set_cursor
168
169     lea dx, questionPrompt
170     call print_string
171     call read_char
172     mov userInput, al
173
174
175     cmp userInput, '5'
176     jg menu_start
177
178     jmp operationOne
179
180 try_again:
181     mov attri, 02h
182     call clr_scr
183
184     mov row, 11
185     mov col, 17
186     call set_cursor
187
188     lea dx, tryAgain
189     call print_string
190
191     call read_char
192     mov userResponse, al
193
194     call clr_regs
195
```

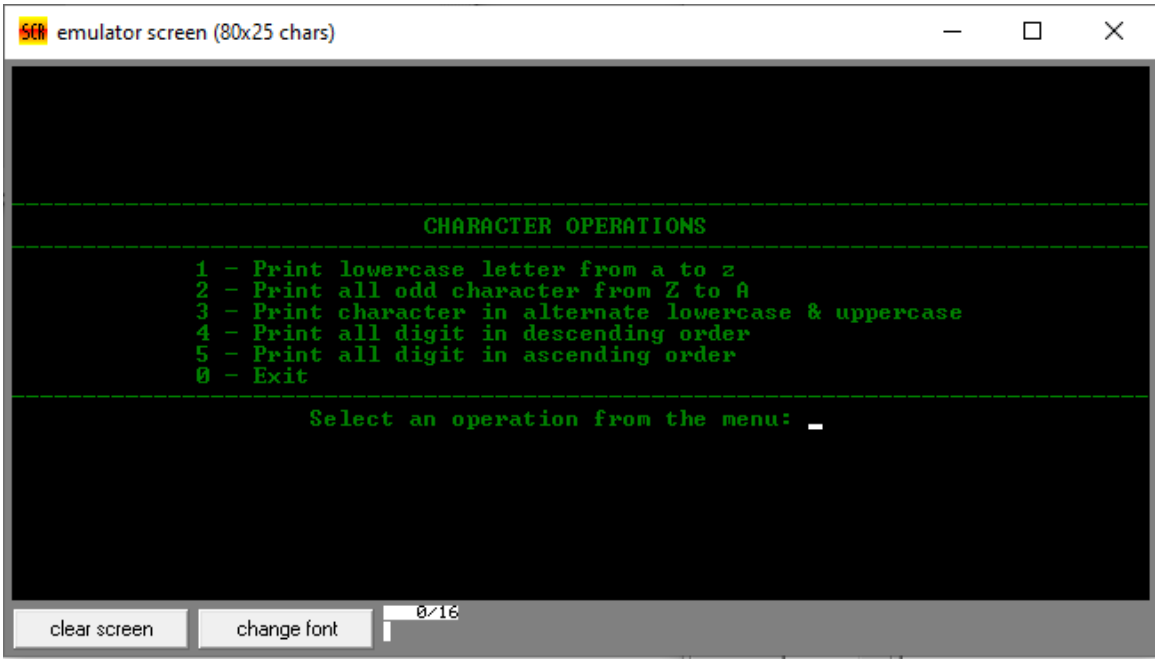
```
    cmp userResponse, 'y'
    je menu_start

    cmp userResponse, 'Y'
    je menu_start

    cmp userResponse, 'n'
    je loop_exit

    cmp userResponse, 'N'
    je loop_exit

    jmp try_again

operationOne:
    cmp userInput, '1'
    jne operationTwo

    jmp print_a_to_z

operationTwo:
    cmp userInput, '2'
    jne operationThree

    jmp print_all_odds

operationThree:
    cmp userInput, '3'
    jne operationFour

    jmp print_alternate_characters

operationFour:
    cmp userInput, '4'
    jne operationFive

    jmp print_in_descending

operationFive:
    cmp userInput, '5'
    jne operationSix

    jmp print_in_ascending


operationSix:
    cmp userInput, '0'
    jne operationOne

    jmp loop_exit

;OPTION 1
print_a_to_z:
    mov attri, 02h
    call clr_scr

    mov row, 10
    mov col, 28
    call set_cursor

    lea dx, firstOperation
    call print_string

    mov row, 11
    mov col, 26
    call set_cursor

    mov al, firstLetter

 output:
    mov dl, al
    call print_char
    inc al
    call disp_del

    cmp al, 'z'+ 1
    je try_again

    loop output

;OPTION 2
print_all_odds:
    mov attri, 02h
    call clr_scr

    mov row, 10
    mov col, 28
    call set_cursor

    lea dx, secondOperation
    call print_string


    mov row, 11
    mov col, 26
    call set_cursor

    mov bl, reverseLetter

 outputTwo:
    mov al, bl
    dec al
    mov dl, al
    call print_char
    dec al
    mov bl, al
    call disp_del

    cmp bl, 'A' - 1
    je try_again

    mov ah, 02h
    mov dl, whiteSpace
    int 21h
    call disp_del

    loop outputTwo
```

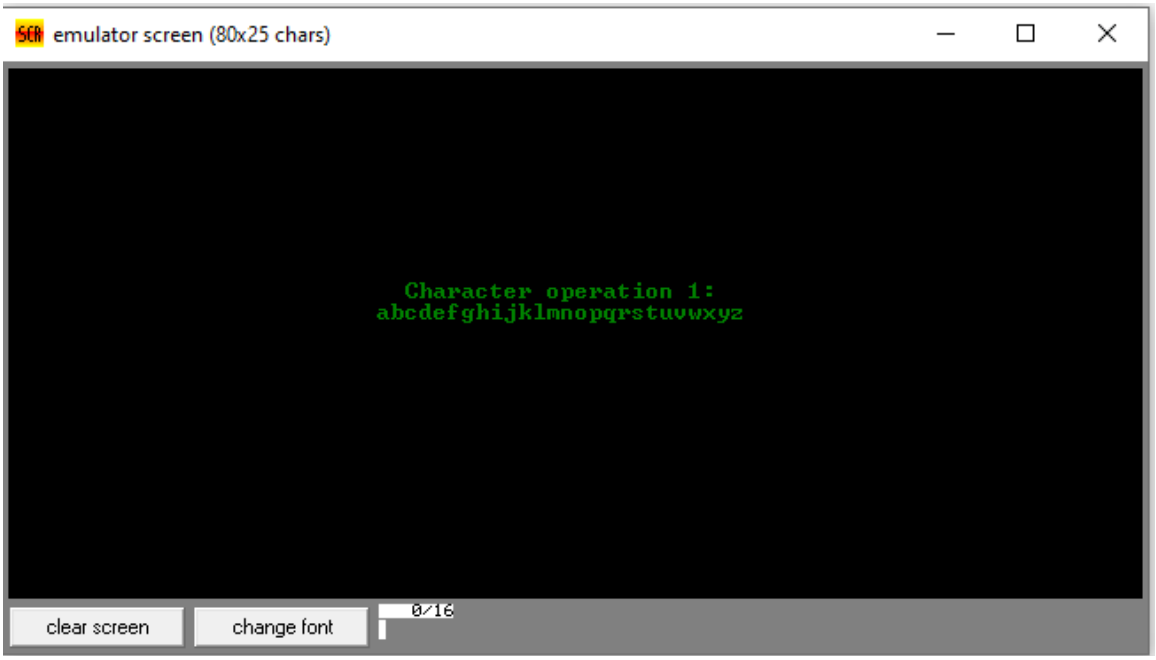
```
;OPTION 3
print_alternate_characters:
    mov attri, 02h
    call clr_scr

    mov row, 10
    mov col, 28
    call set_cursor

    lea dx, thirdOperation
    call print_string

    mov row, 11
    mov col, 13
    call set_cursor

    mov bl, firstLetter

 lower_case:
    mov al, bl
    mov dl, al
    call print_char
    inc bl
    call disp_del

    mov ah, 02h
    mov dl, whiteSpace
    int 21h
    call disp_del

    jmp upper_case


 upper_case:
    mov al, bl
    sub al, 32
    mov dl, al
    call print_char
    inc bl
    call disp_del

    cmp bl, 'z' + 1
    je try_again

    add al, 32

    mov ah, 02h
    mov dl, whiteSpace
    int 21h


    call disp_del

    jmp lower_case
;OPTION 4
print_in_descending:
    mov attri, 02h
    call clr_scr

    mov row, 10
    mov col, 28
    call set_cursor

    lea dx, fourthOperation
    call print_string

    mov row, 11
    mov col, 29
    call set_cursor

    mov bl, 9

 descending:
    mov al, bl
    add al, '0'
    mov dl, al
    call print_char
    call disp_del

    cmp bl, 0
    je try_again

    cmp al, '0'
    jne next_order

 separate:
    mov dl, whiteSpace
    call print_char
    call disp_del

    jmp descending

 next_order:
    sub al, '0'
    sub al, 1
    mov bl, al

    jmp separate


;OPTION 5
print_in_ascending:
    mov attri, 02h
    call clr_scr

    mov row, 10
    mov col, 28
    call set_cursor

    lea dx, fifthOperation
    call print_string

    mov row, 11
    mov col, 29
    call set_cursor

    mov bl, 0
```

```
 ascending:
    mov al, bl
    add al, '0'
    mov dl, al
    call print_char
    call disp_del

    cmp bl, 9
    je try_again

    cmp al, '9'
    jne next_seq

 separateTwo:
    mov dl, whiteSpace
    call print_char
    call disp_del

    jmp ascending

 next_seq:
    sub al, '0'
    add al, 1
    mov bl, al

    jmp separateTwo


loop_exit:
    mov attri, 00h
    call clr_scr
    call disp_del

    call exit


;DOS/BIOS FUNCSTIONS :: SUB-ROUTINES
set_cursor proc near
       mov ah, 02h
       mov bh, 0
       mov dh, row
       mov dl, col
       int 10h
       ret
set_cursor endp


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

clr_regs proc near
    xor ax, ax
    xor bx, bx
    xor cx, cx
    xor dx, dx
    ret
clr_regs endp

print_string proc near
    mov ah, 09h
    int 21h
    ret
print_string endp

print_char proc near
    mov ah, 02h
    int 21h
    ret
print_char endp

read_char proc near
    mov ah, 01h
    int 21h
    ret
read_char endp

prompt_start proc near
    jmp start
    ret
prompt_start endp

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


disp_del proc near
       mov cx, 000Fh
 again: nop
       loop again
       ret
disp_del endp

end start
```

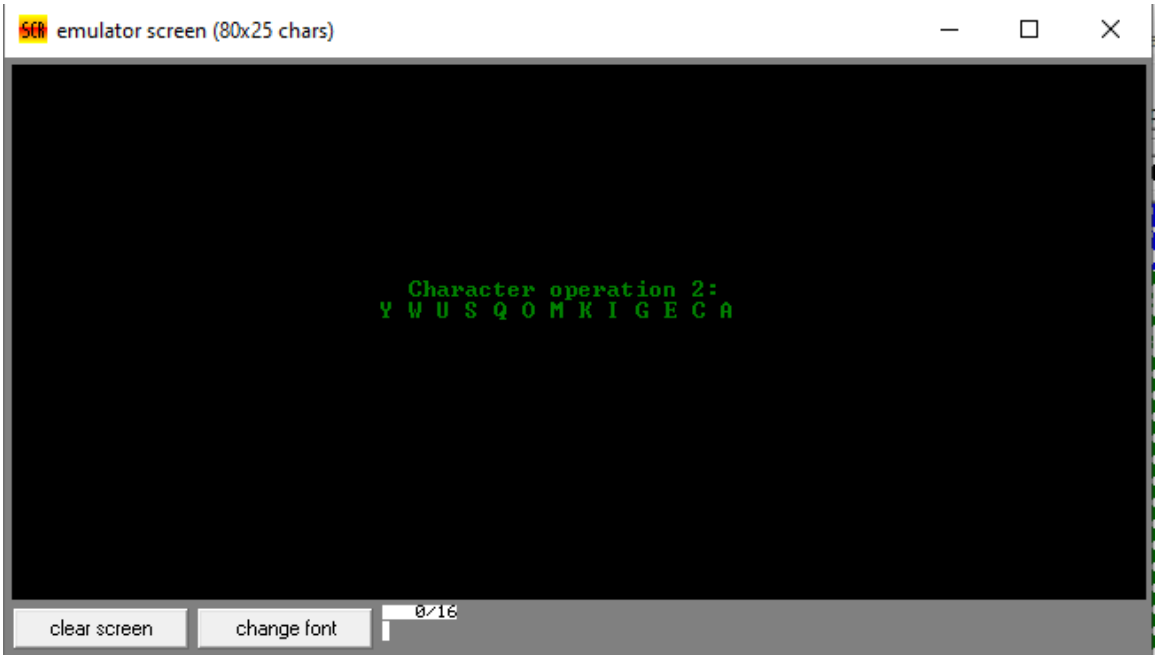
Output (at least 3 scenarios):



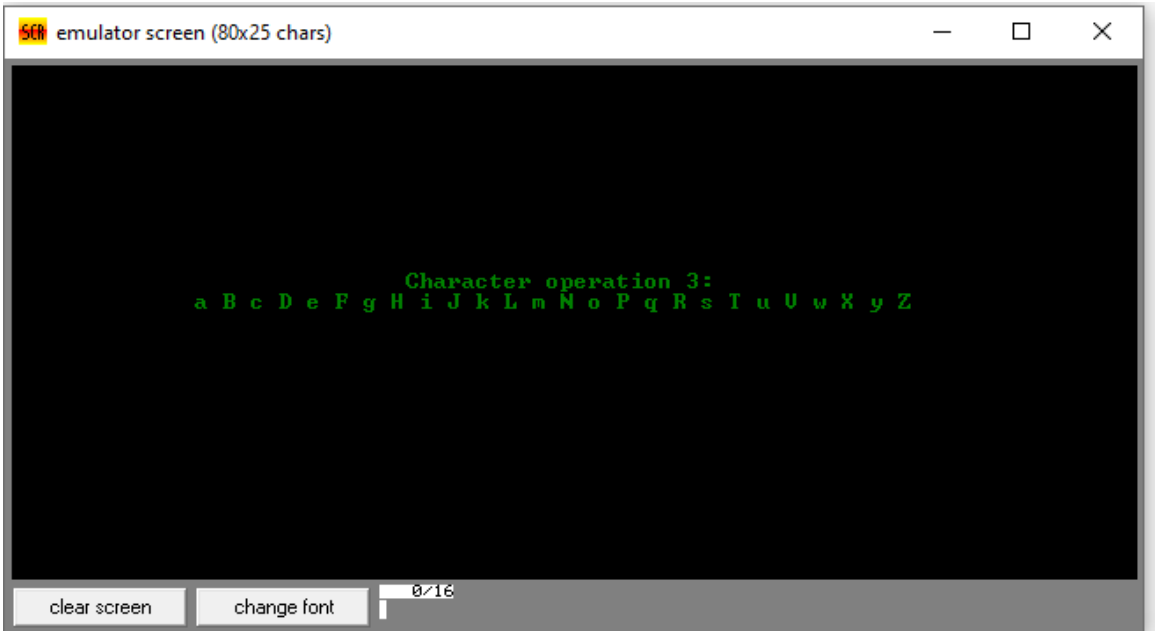


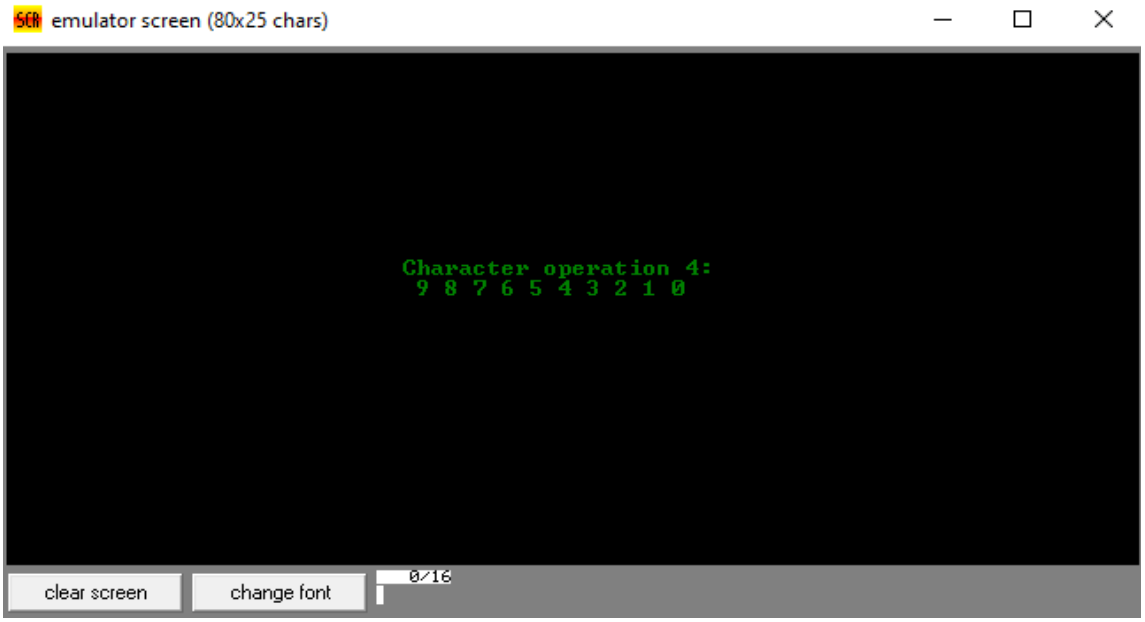


CHARACTERS ARE PRINTED WITH SPACES BETWEEN THEM:



CHARACTERS ARE PRINTED WITH SPACES BETWEEN THEM:






emulator screen (80x25 chars)
Character operation 5:
0 1 2 3 4 5 6 7 8 9
clear screen
change font
0/16
emulator screen (80x25 chars)
Do you want to try another operation? [y/n]:
clear screen
change font
0/16

DATA ANALYSIS

The provided assembly language code for our drill activity implements a basic password-guessing program with interactive features. The code structure includes a series of sections, such as initialization, data declaration, and user-defined procedures. The program incorporates fundamental assembly language concepts like control flow through conditional jumps (jmp), interrupt services (int 21h), and procedures for tasks like screen clearing, cursor positioning (int 10h), and introducing delays. The core functionality involves prompting the user to guess a three-letter password ('REM'), displaying appropriate messages, and handling input validation. The code also contains a set of welcome messages and terminates by returning control to the operating system. To validate its correctness and effectiveness, testing the program, especially input scenarios and edge cases, is essential during the lab activity. Additionally, ensuring that the code adheres to assembly language conventions and best practices would contribute to a comprehensive analysis.

For our program exercise, it implements a menu-driven system, prompting the user to select a character operation from a menu. The available operations include printing lowercase letters, odd characters, alternating lowercase and uppercase characters, descending digits with spaces, ascending digits with spaces, and exiting the program. The user is asked if they want to try another operation after each execution.

The code features a structured layout with comments for clarity. It utilizes DOS and BIOS function services to perform operations such as displaying characters, clearing the screen, and obtaining user input. The program handles user input effectively, and the character operations are implemented as specified in the prompt. The inclusion of delays enhances the user experience by separating distinct sections of the output. The program also terminates gracefully upon the user's decision to exit.

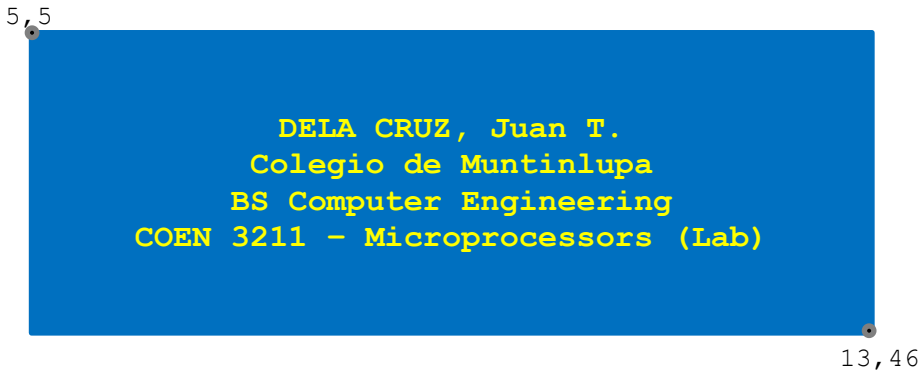
To validate its functionality, thorough testing of various inputs, edge cases, and user interactions with the menu options is recommended during the program exercise. Additionally, ensuring that the code adheres to assembly language conventions and best practices would contribute to a comprehensive analysis.

QUESTIONS AND ANSWERS

Questions:

1. What is the between INT 21H service 07H, and INT 10H service 08H? Provide sample example for each.
2. Create and test assembly program that will display information similar to the output below using INT 21H and INT 10H services.

Sample output:



Answers:

1. The key difference between INT 21H service 07H and INT 10H service 08H lies in their purposes and functionalities. These are two different interrupt services that serve distinct roles in a computer system.

   INT 21H Service 07H: Keyboard Input with Wait (DOS Function)
   Purpose: This interrupt service is part of the DOS (Disk Operating System) functions and is used for keyboard input. It waits for a key to be pressed and then returns the ASCII code of the pressed key.

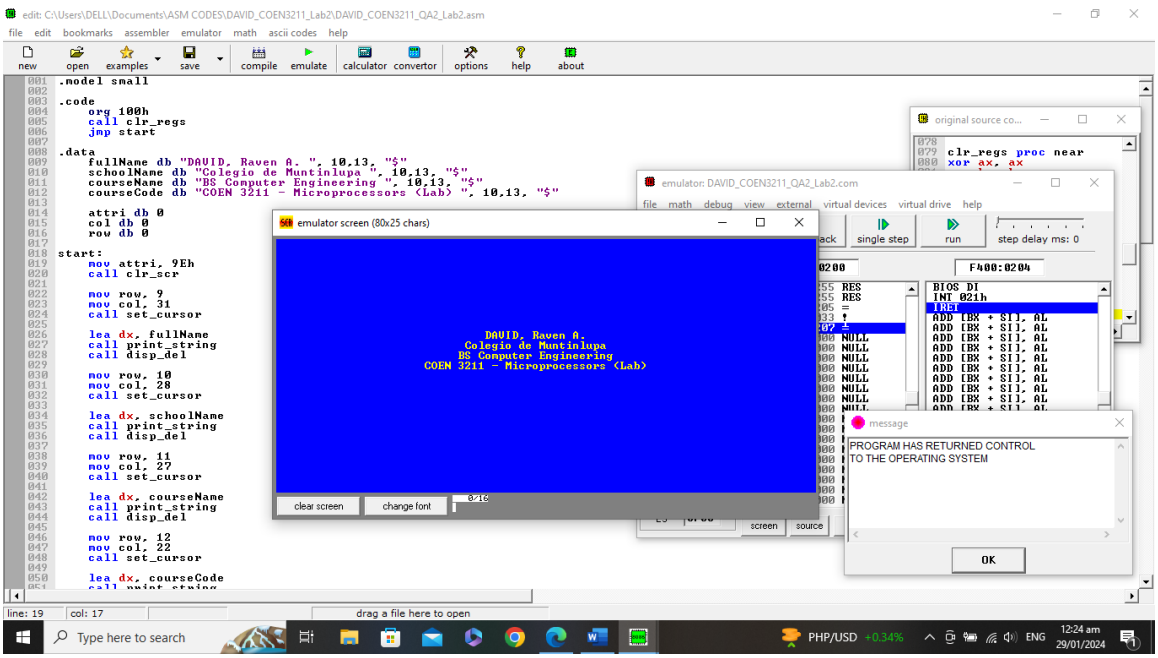
```
01 ;INT 21H Service 07H - EXAMPLE
02
03 mov ah, 07h        ; Function 07h - Keyboard Input with Wait
04 int 21h            ; Call DOS interrupt service
05
06 ; AL now contains the ASCII code of the pressed key
```

INT 10H Service 08H: Read Character and Attribute at Cursor Position (BIOS Function)

Purpose: This interrupt service is part of the BIOS (Basic Input/Output System) functions and is used for reading the character and attribute (color) at the current cursor position on the text screen.

```
;INT 10H Service 08H - EXAMPLE

mov ah, 08h        ; Function 08h - Read Character and Attribute at Cursor Position
int 10h            ; Call BIOS interrupt service
; AH contains the character at the cursor position
; AL contains the attribute (color) at the cursor position
```

2. Program and Output:



CONCLUSION

After completing this lab experiment, students have gained valuable insights into the intricacies of BIOS interrupt service routines, setting the stage for a deeper understanding of low-level system operations. The first objective aimed at fostering familiarity with the operation of several essential BIOS interrupt service routines (ISR). Through hands-on exploration, students delved into the intricacies of INT 10h, a BIOS ISR primarily responsible for video services. This provided a foundational understanding of how the BIOS interacts with hardware components, particularly in managing text and video displays.

Moving on to the second objective, students successfully honed their skills in creating assembly programs utilizing the INT 10h BIOS service routines. The lab facilitated a practical application of theoretical knowledge as students crafted assembly programs to manipulate text and background colors on the display. This objective aimed not only at understanding the intricacies of BIOS routines but also at applying this knowledge to solve real-world problems programmatically. The assembly language, being a low-level language, demanded careful consideration of memory locations and interrupt vectors, offering students a hands-on experience in managing system resources effectively.

To fulfill the third objective, the lab provided an environment for testing and simulating the functionality of the assembly program using emu8086 assembler software. This step was crucial in ensuring that the assembly code produced the expected results and operated seamlessly within the designated environment. The emulation aspect not only allowed for a safe exploration of code behavior but also provided a means to observe the impact of the assembly program on the simulated system, fostering a comprehensive understanding of the program's functionality.

REFERENCES

GitHub Forum Post:

Nitish Kumar. (2021). 8086 BIOS and DOS Interrupts. GitHub. https://github.com/er-nitish/8086assembly/blob/main/8086-bios-and-dos-interrupts.html