COLEGIO DE MUNTINLUPA

DEPARTMENT OF COMPUTER ENGINEERING

COEN 3211 - Microprocessors Lab

Assembly Language Programming with DOS Interrupts

Laboratory Experiment No. 1

GRADE

STUDENT NAME	: <**DAVID, Raven A.**>
STUDENT NUMBER	: <**20202011637**>
DATE PERFORMED	: <**24 January 2024**>
DATE SUBMITTED	: <**24 January 2024**>

Engr. Ricrey E. Marquez, PCpE

(Lab Instructor)

PRINCIPLES

The Intel CPU recognizes two types of interrupts namely *hardware interrupt* when a peripheral device needs attention from the CPU, and *software interrupt* that is call to a subroutine located in the operating system.

The common software interrupts used here are INT 10H for video services and INT 21H for DOS services.

The INT 21H is called the DOS function call for keyboard and display operations follow the function or service number.

Syntax some useful INT 21H DOS service:

`01H` - Read a character with echo

Syntax:

```
MOV AH, 01H     ;request keyboard input
INT 21H         ;service function granted/executed
```

Returns character in AL.
If AL= non-zero value, operation echoes on the screen.
If AL = zero means that user has pressed an extended function key such as F1 or home.

`02H` **- Display single character**

Syntax:

```
MOV AH, 02H     ;request display character
MOV DL, char    ;load character to display to DL
INT 21H         ;service function granted/executed
```

Display character in DL at current cursor position. The tab, carriage return and line feed characters act normally and the operation automatically advances the cursor.

`09H` **– Display string**

Syntax:

```
MOV AH, 09H     ;request display string
MOV DX, strloc  ;load string to DX
```

```
INT 21H          ;service function granted/executed
```

Displays string in the data area, immediately followed by a dollar sign ($ or 24H), which uses to end the display. Note that 10h and 13h

```
text db “Print string!”, 10, 13, “$”
```

Note that 10 (0Ah) is the ASCII control code for line feed while 13 (0Fh) is the code for carriage return.

OBJECTIVES AND MATERIALS

Objectives:

After this lab experiment, student should be able to:

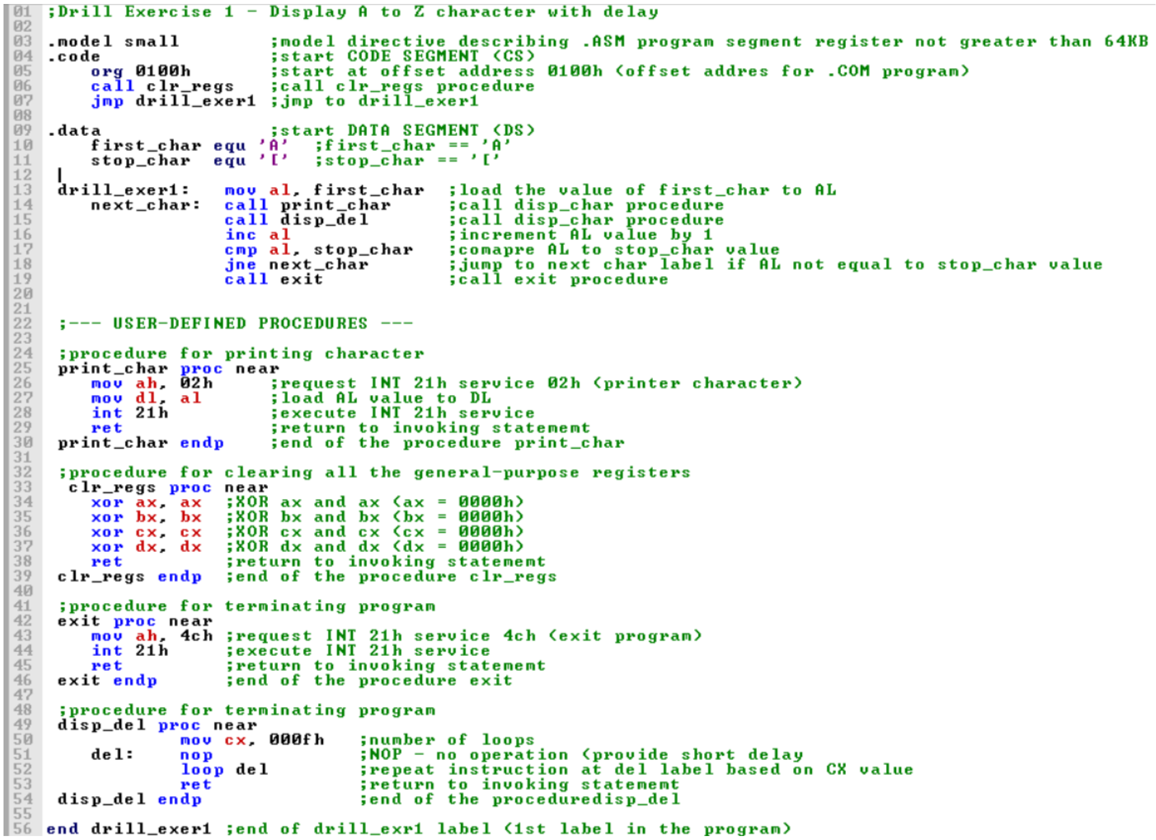
1. familiar with the operation of some useful DOS interrupt service routines,
2. create assembly programs using INT 21h DOS service routines,
3. test and simulate the functionality of the assembly program with emu8086 assembler software,

Materials:

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

DRILL EXERCISES

Drill Exercise 1 – Given the assembly language source code list below, re-type and test the program. Save as **`COEN3211_3x-x_Drill1.asm`**.

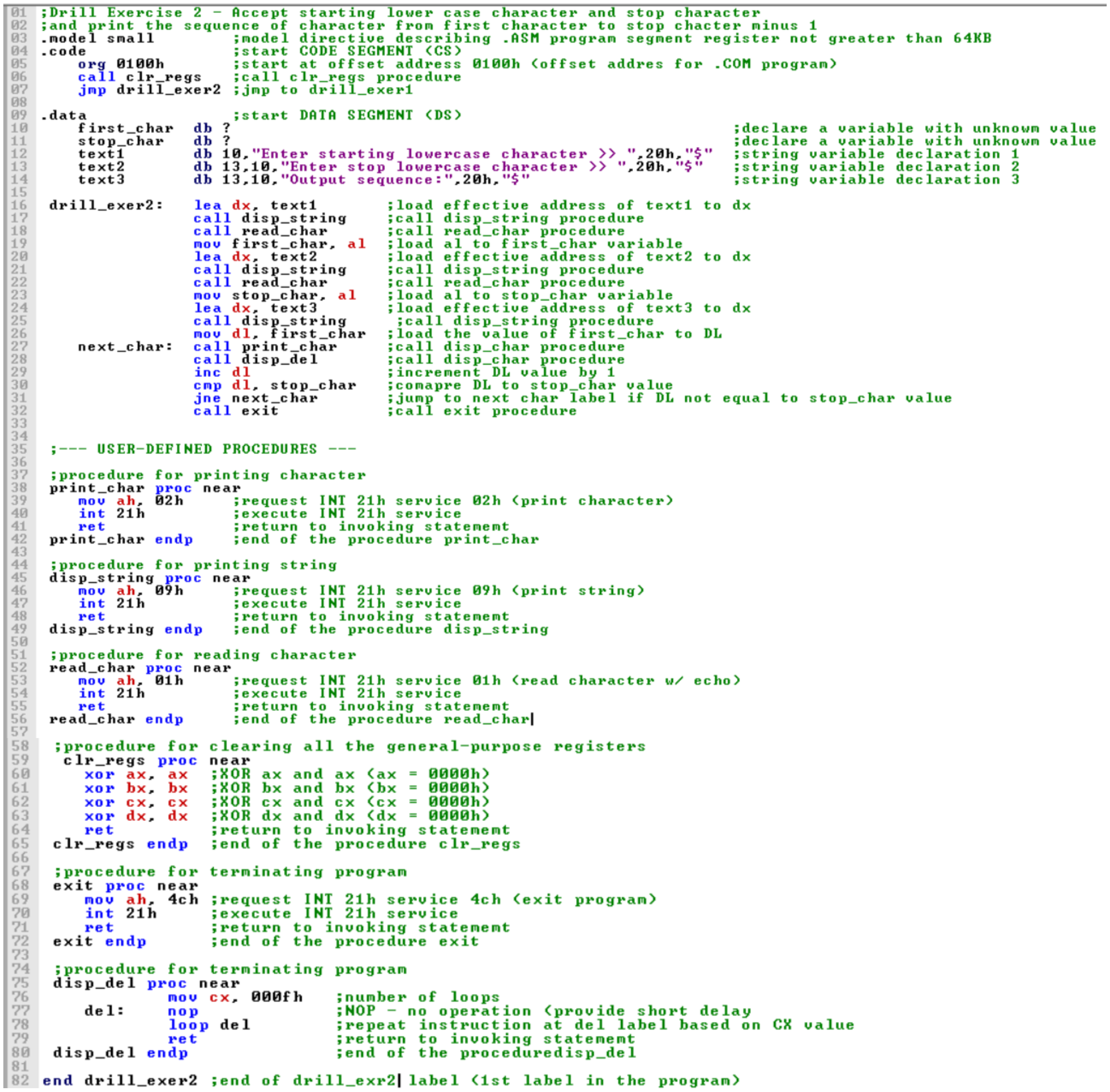


```
;Drill Exercise 1 - Display A to Z character with delay

.model small        ;model directive describing .ASM program segment register not greater than 64KB
.code               ;start CODE SEGMENT (CS)
    org 0100h       ;start at offset address 0100h (offset addres for .COM program)
    call clr_regs   ;call clr_regs procedure
    jmp drill_exer1 ;jmp to drill_exer1

.data               ;start DATA SEGMENT (DS)
    first_char equ 'A'  ;first_char == 'A'
    stop_char  equ '['  ;stop_char == '['

 drill_exer1:   mov al, first_char  ;load the value of first_char to AL
    next_char:  call print_char     ;call disp_char procedure
                call disp_del       ;call disp_char procedure
                inc al              ;increment AL value by 1
                cmp al, stop_char   ;comapre AL to stop_char value
                jne next_char       ;jump to next char label if AL not equal to stop_char value
                call exit           ;call exit procedure


 ;--- USER-DEFINED PROCEDURES ---

 ;procedure for printing character
 print_char proc near
    mov ah, 02h     ;request INT 21h service 02h (printer character)
    mov dl, al      ;load AL value to DL
    int 21h         ;execute INT 21h service
    ret             ;return to invoking statememt
 print_char endp    ;end of the procedure print_char

 ;procedure for clearing all the general-purpose registers
  clr_regs proc near
    xor ax, ax  ;XOR ax and ax (ax = 0000h)
    xor bx, bx  ;XOR bx and bx (bx = 0000h)
    xor cx, cx  ;XOR cx and cx (cx = 0000h)
    xor dx, dx  ;XOR dx and dx (dx = 0000h)
    ret         ;return to invoking statement
 clr_regs endp  ;end of the procedure clr_regs

 ;procedure for terminating program
 exit proc near
    mov ah, 4ch ;request INT 21h service 4ch (exit program)
    int 21h     ;execute INT 21h service
    ret         ;return to invoking statememt
 exit endp      ;end of the procedure exit

 ;procedure for terminating program
 disp_del proc near
            mov cx, 000fh   ;number of loops
    del:    nop             ;NOP - no operation (provide short delay
            loop del        ;repeat instruction at del label based on CX value
            ret             ;return to invoking statememt
 disp_del endp              ;end of the proceduredisp_del

end drill_exer1 ;end of drill_exr1 label (1st label in the program)
```

Figure 1-1. Code listing of Drill Exercise 1

Drill Exercise 2 – Given the assembly language source code list below, re-type and test the program. Save as **`COEN3211_3x-x_Drill2.asm`**.

.



```
;Drill Exercise 2 - Accept starting lower case character and stop character
;and print the sequence of character from first character to stop chacter minus 1
.model small         ;model directive describing .ASM program segment register not greater than 64KB
.code                ;start CODE SEGMENT (CS)
    org 0100h        ;start at offset address 0100h (offset addres for .COM program)
    call clr_regs    ;call clr_regs procedure
    jmp drill_exer2 ;jmp to drill_exer1

.data                ;start DATA SEGMENT (DS)
    first_char  db ?                                                  ;declare a variable with unknown value
    stop_char   db ?                                                  ;declare a variable with unknown value
    text1       db 10,"Enter starting lowercase character >> ",20h,"$"  ;string variable declaration 1
    text2       db 13,10,"Enter stop lowercase character >> ",20h,"$"   ;string variable declaration 2
    text3       db 13,10,"Output sequence:",20h,"$"                   ;string variable declaration 3

 drill_exer2:   lea dx, text1       ;load effective address of text1 to dx
                call disp_string    ;call disp_string procedure
                call read_char      ;call read_char procedure
                mov first_char, al  ;load al to first_char variable
                lea dx, text2       ;load effective address of text2 to dx
                call disp_string    ;call disp_string procedure
                call read_char      ;call read_char procedure
                mov stop_char, al   ;load al to stop_char variable
                lea dx, text3       ;load effective address of text3 to dx
                call disp_string     ;call disp_string procedure
                mov dl, first_char  ;load the value of first_char to DL
    next_char:  call print_char     ;call disp_char procedure
                call disp_del       ;call disp_char procedure
                inc dl              ;increment DL value by 1
                cmp dl, stop_char   ;comapre DL to stop_char value
                jne next_char       ;jump to next char label if DL not equal to stop_char value
                call exit           ;call exit procedure


 ;--- USER-DEFINED PROCEDURES ---

 ;procedure for printing character
 print_char proc near
    mov ah, 02h      ;request INT 21h service 02h (print character)
    int 21h          ;execute INT 21h service
    ret              ;return to invoking statememt
 print_char endp     ;end of the procedure print_char

 ;procedure for printing string
 disp_string proc near
    mov ah, 09h      ;request INT 21h service 09h (print string)
    int 21h          ;execute INT 21h service
    ret              ;return to invoking statememt
 disp_string endp    ;end of the procedure disp_string

 ;procedure for reading character
 read_char proc near
    mov ah, 01h      ;request INT 21h service 01h (read character w/ echo)
    int 21h          ;execute INT 21h service
    ret              ;return to invoking statememt
 read_char endp      ;end of the procedure read_char

 ;procedure for clearing all the general-purpose registers
  clr_regs proc near
    xor ax, ax  ;XOR ax and ax (ax = 0000h)
    xor bx, bx  ;XOR bx and bx (bx = 0000h)
    xor cx, cx  ;XOR cx and cx (cx = 0000h)
    xor dx, dx  ;XOR dx and dx (dx = 0000h)
    ret         ;return to invoking statememt
 clr_regs endp  ;end of the procedure clr_regs

 ;procedure for terminating program
 exit proc near
    mov ah, 4ch ;request INT 21h service 4ch (exit program)
    int 21h     ;execute INT 21h service
    ret         ;return to invoking statememt
 exit endp      ;end of the procedure exit

 ;procedure for terminating program
 disp_del proc near
            mov cx, 000fh    ;number of loops
    del:    nop              ;NOP - no operation (provide short delay
            loop del         ;repeat instruction at del label based on CX value
            ret              ;return to invoking statememt
 disp_del endp               ;end of the proceduredisp_del

end drill_exer2 ;end of drill_exr2 label (1st label in the program)
```

Figure 1-2. Code listing of Drill Exercise 2

PROGRAM EXERCISES

Program Exercise 1. Create and test an optimized and procedural-based assembly program that will display character sequence as shown below. **Note:** Execute each character output with delay. Save as **`COEN3211_3x-x_ProgExer1.asm.`**

```
aA bB cC … yY zZ
```

Program Exercise 2. Create and test an optimized and procedural-based assembly program in that will accepts start and stop character then the program will display the following. **Note:** Execute each character output with delay. Save as **`COEN3211_3x-x_ProgExer1.asm`**.

Sample input and output:

```
Enter start letter (lowercase letter only): b
Enter last letter (lowercase letter only): g
Sequence of uppercase characters b to g in descending order is
G F E D C B
```

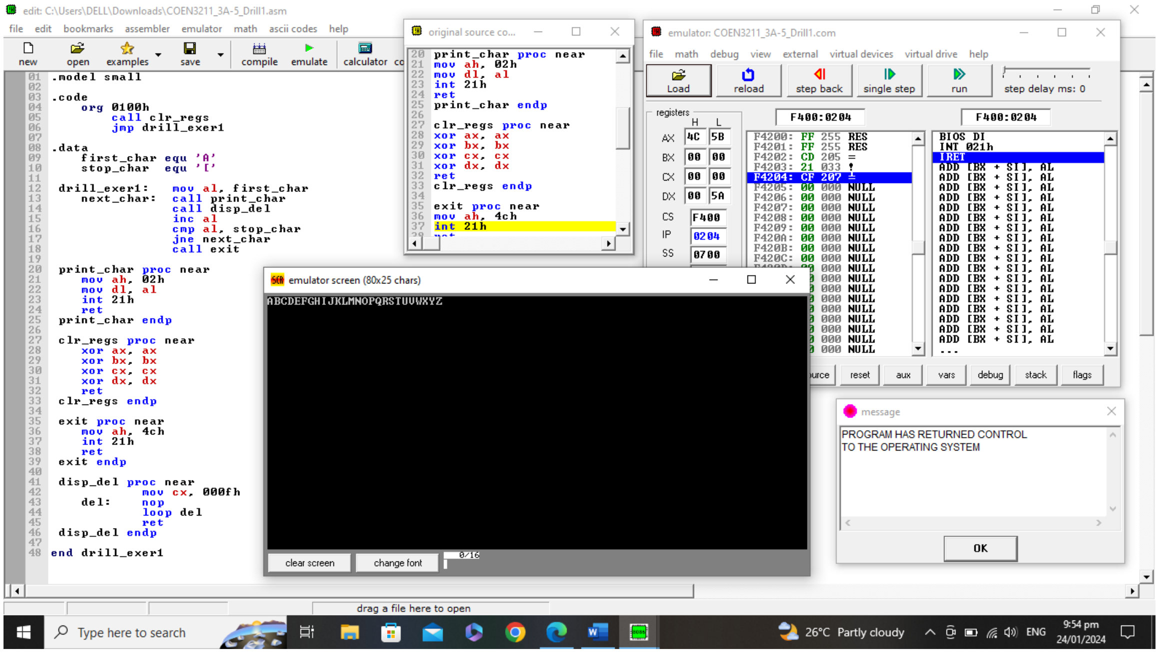
Program Exercise 1-3. Create and test optimized and procedural-based assembly program that will prompt the user to enter single digit number from (0 to 9) then program will display the sum of the numbers. **Note:** digit ‘0’ = 30h to ‘9’ = 39h. If the sum is greater than 9 then print digit ‘1’ and subtract sum by digit ‘9’ to get the least significant digit (LSB) and print, else display the sum. Save as **`COEN3211_3x-x_ProgExer1.asm`**.

Sample input and output:

```
Enter the first single digit number: 5
Enter the second single digit number: 7
The sum of 5 and 7 is 12
```

DATA RESULTS

Drill Exercise 1-1. Test output



edit: C:\Users\DELL\Downloads\COEN3211_3A-5_Drill1.asm

```
.model small

.code
    org 0100h
        call clr_regs
        jmp drill_exer1

.data
    first_char equ 'A'
    stop_char  equ '['

 drill_exer1:   mov al, first_char
    next_char:  call print_char
                call disp_del
                inc al
                cmp al, stop_char
                jne next_char
                call exit

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

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

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

 disp_del proc near
                mov cx, 000fh
    del:        nop
                loop del
                ret
 disp_del endp

end drill_exer1
```

emulator: COEN3211_3A-5_Drill1.com

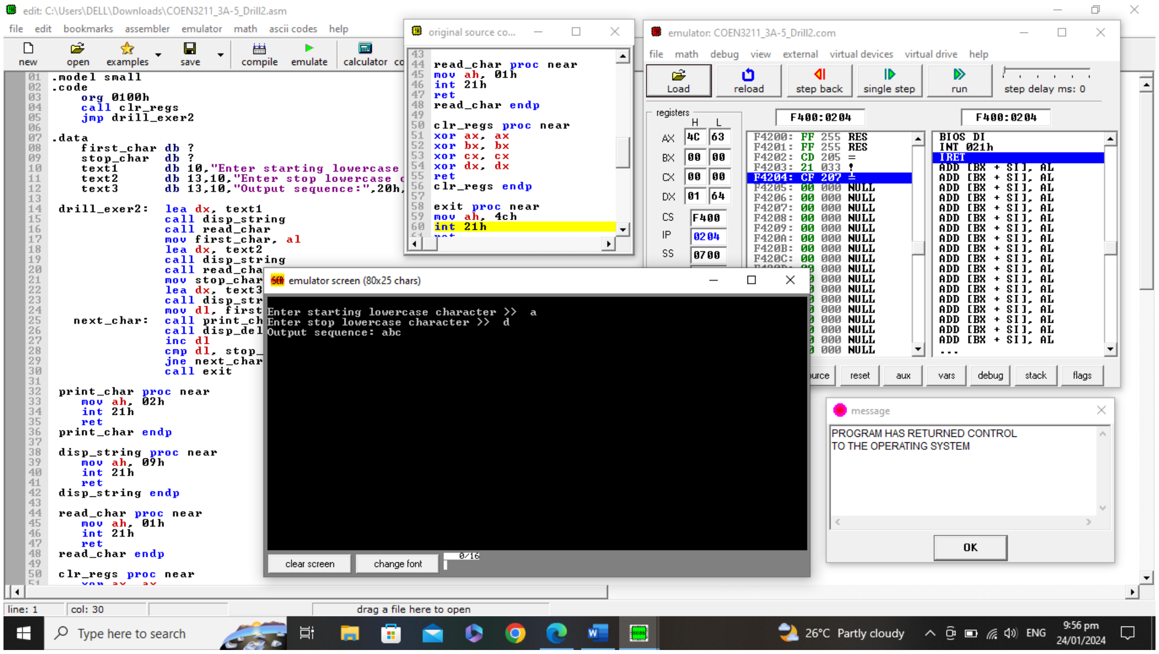
Registers: AX 4C 5B, BX 00 00, CX 00 00, DX 00 5A, CS F400, IP 0204, SS 0700

emulator screen (80x25 chars)

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

message: PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM

Drill Exercise 1-2. Test output



edit: C:\Users\DELL\Downloads\COEN3211_3A-5_Drill2.asm

```
.model small
.code
    org 0100h
    call clr_regs
    jmp drill_exer2

.data
    first_char db ?
    stop_char  db ?
    text1      db 10,"Enter starting lowercase
    text2      db 13,10,"Enter stop lowercase c
    text3      db 13,10,"Output sequence:",20h,

 drill_exer2:  lea dx, text1
               call disp_string
               call read_char
               mov first_char, al
               lea dx, text2
               call disp_string
               call read_cha
               mov stop_char
               lea dx, text3
               call disp_str
               mov dl, first
   next_char:  call print_ch
               call disp_del
               inc dl
               cmp dl, stop_
               jne next_char
               call exit

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

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

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

 clr_regs proc near
```

emulator: COEN3211_3A-5_Drill2.com

Registers: AX 4C 63, BX 00 00, CX 00 00, DX 01 64, CS F400, IP 0204, SS 0700

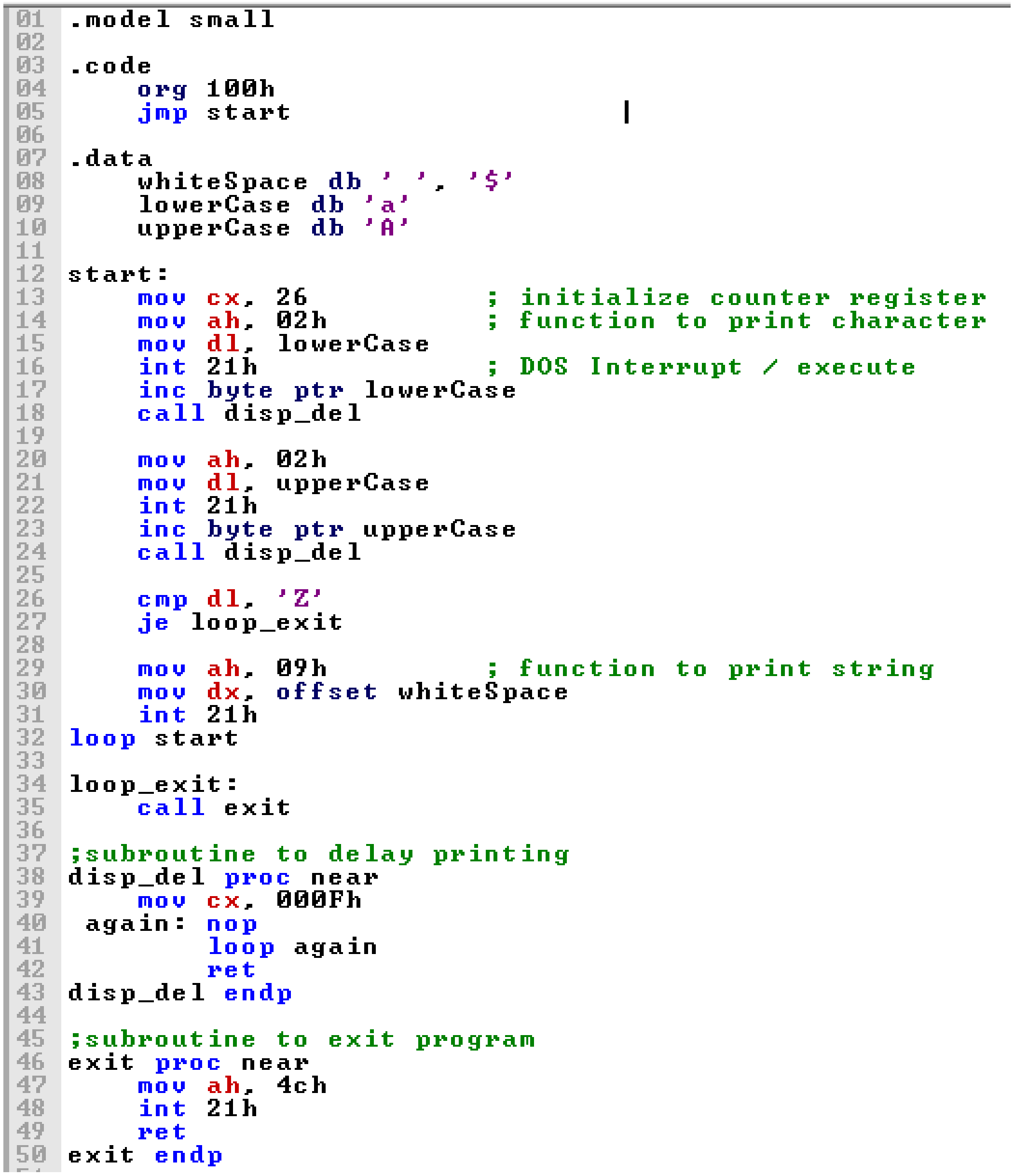
emulator screen (80x25 chars)

```
Enter starting lowercase character >>  a
Enter stop lowercase character >>  d
Output sequence: abc
```

message: PROGRAM HAS RETURNED CONTROL TO THE OPERATING SYSTEM

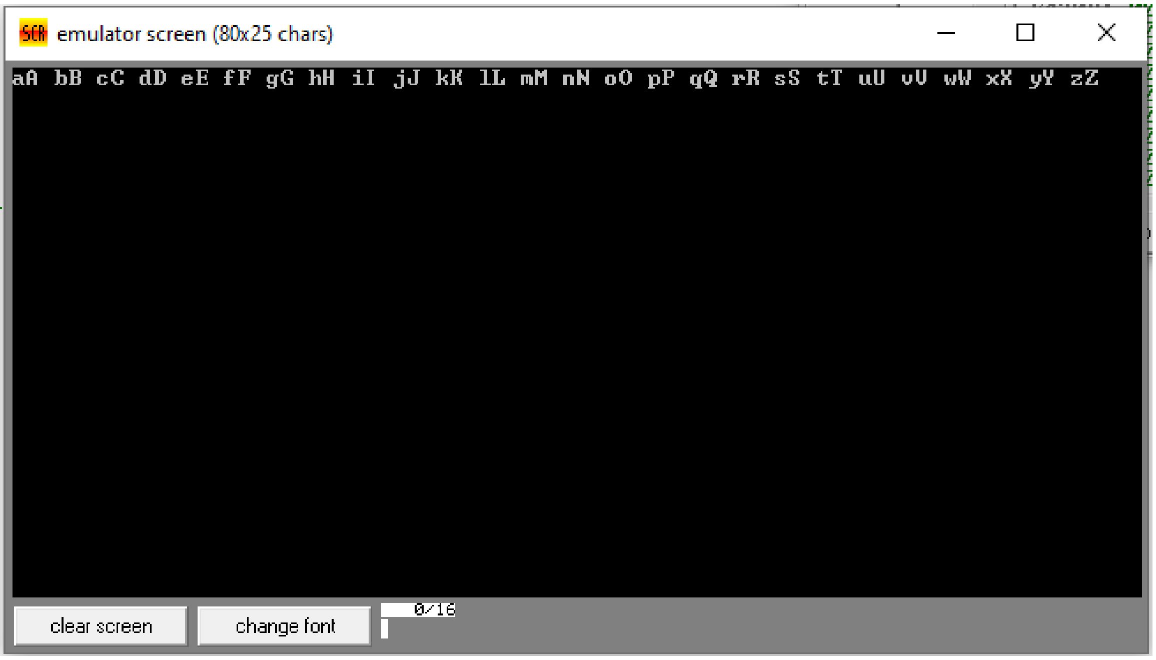
Program Exercise 1-1. Program Listing and Outputs

Program Listing:

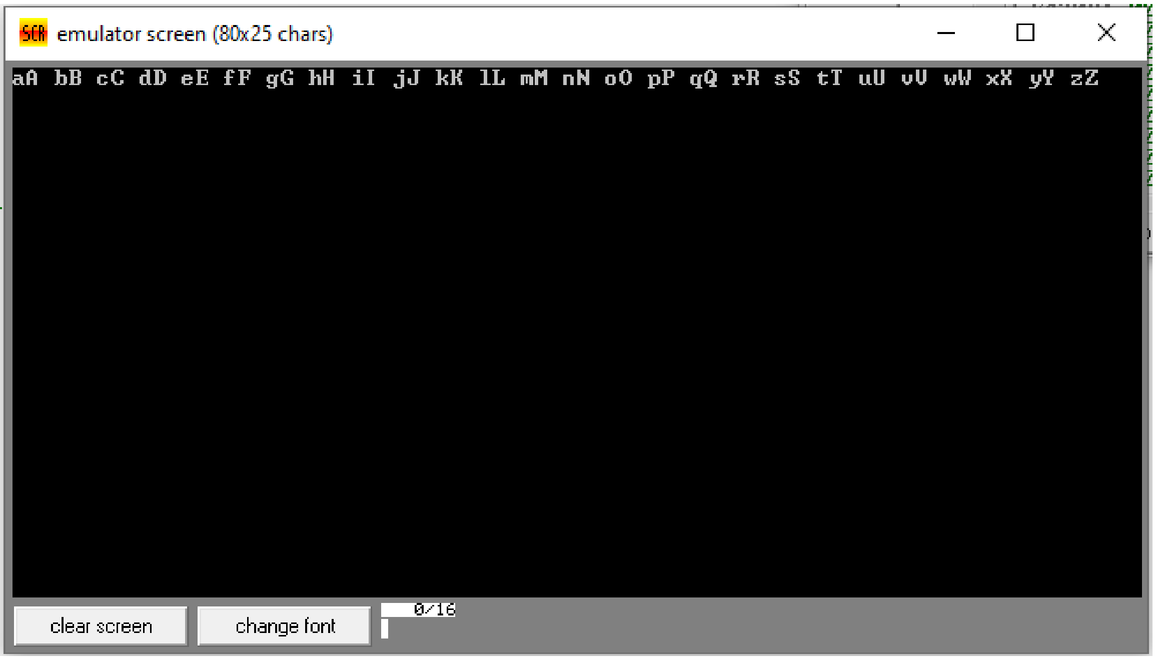


```
.model small

.code
    org 100h
    jmp start

.data
    whiteSpace db ' ', '$'
    lowerCase db 'a'
    upperCase db 'A'

start:
    mov cx, 26              ; initialize counter register
    mov ah, 02h             ; function to print character
    mov dl, lowerCase
    int 21h                 ; DOS Interrupt / execute
    inc byte ptr lowerCase
    call disp_del

    mov ah, 02h
    mov dl, upperCase
    int 21h
    inc byte ptr upperCase
    call disp_del

    cmp dl, 'Z'
    je loop_exit

    mov ah, 09h             ; function to print string
    mov dx, offset whiteSpace
    int 21h
loop start

loop_exit:
    call exit

;subroutine to delay printing
disp_del proc near
    mov cx, 000Fh
 again: nop
        loop again
        ret
disp_del endp

;subroutine to exit program
exit proc near
    mov ah, 4ch
    int 21h
    ret
exit endp
```

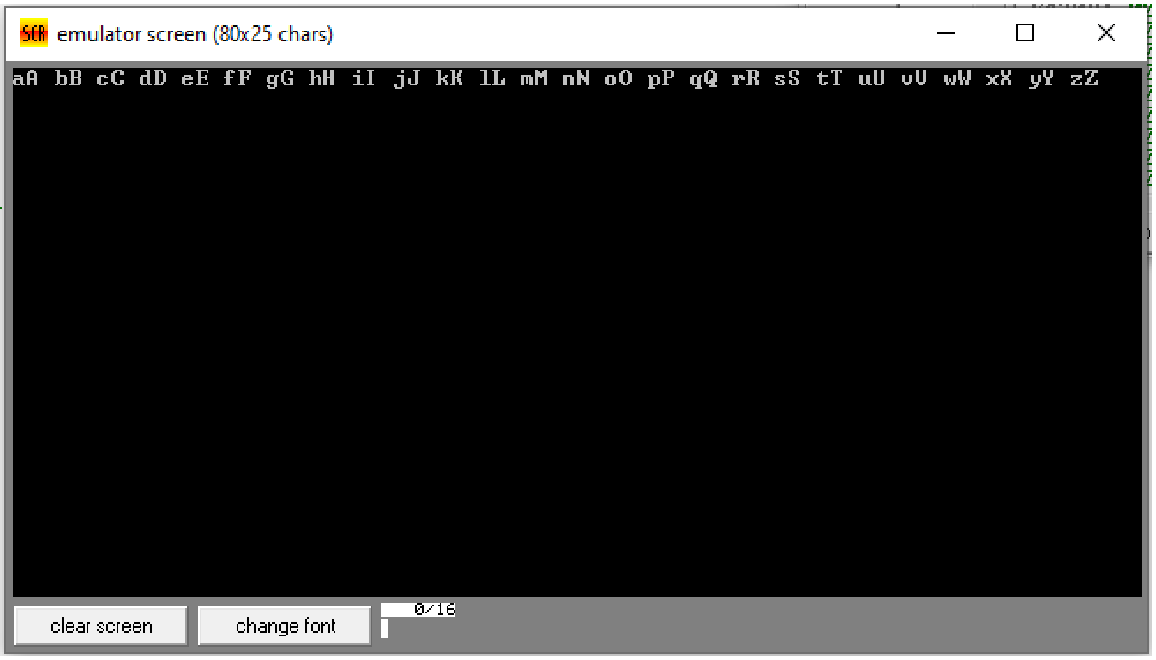
Output (at least 3 scenarios):



```
aA bB cC dD eE fF gG hH iI jJ kK lL mM nN oO pP qQ rR sS tT uU vV wW xX yY zZ
```



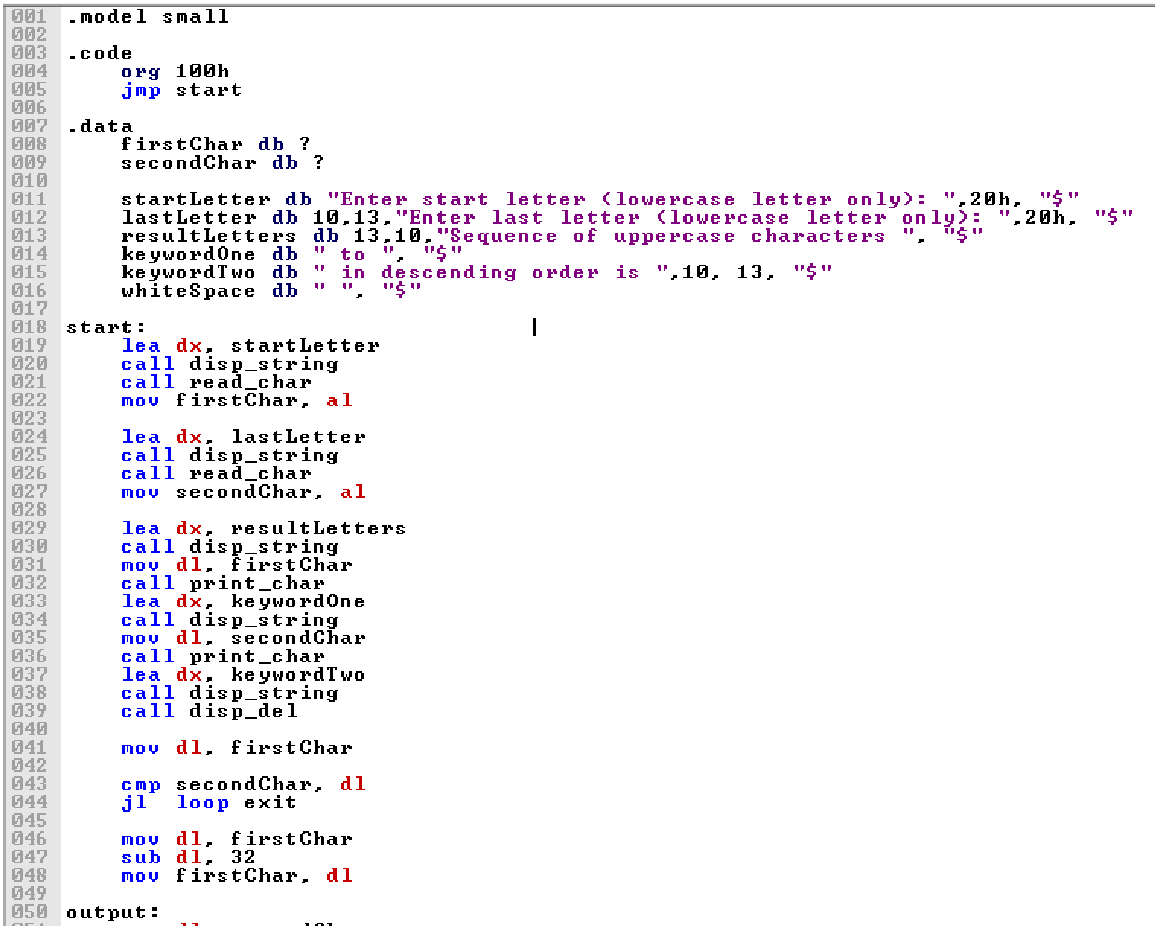
```
aA bB cC dD eE fF gG hH iI jJ kK lL mM nN oO pP qQ rR sS tT uU vV wW xX yY zZ
```



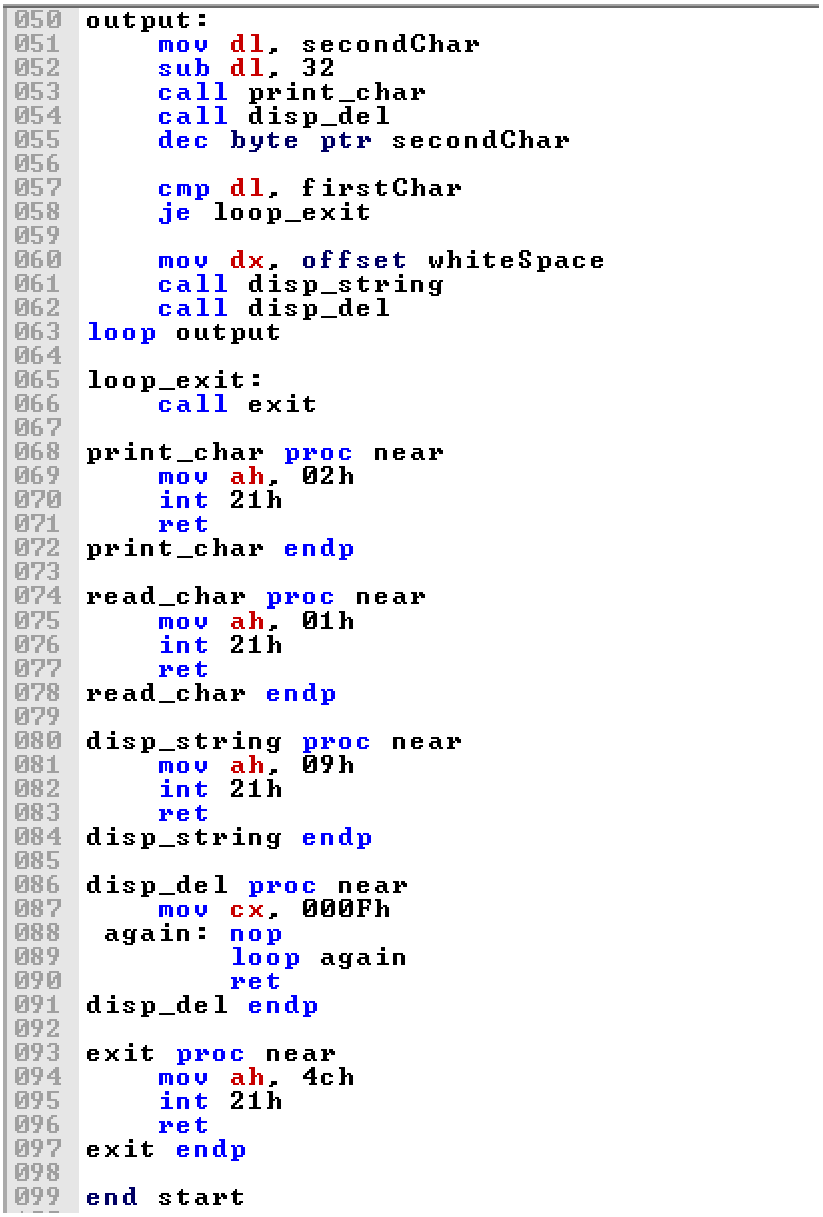
```
aA bB cC dD eE fF gG hH iI jJ kK lL mM nN oO pP qQ rR sS tT uU vV wW xX yY zZ
```

Program Exercise 1-2. Program Listing and Outputs

Program Listing:

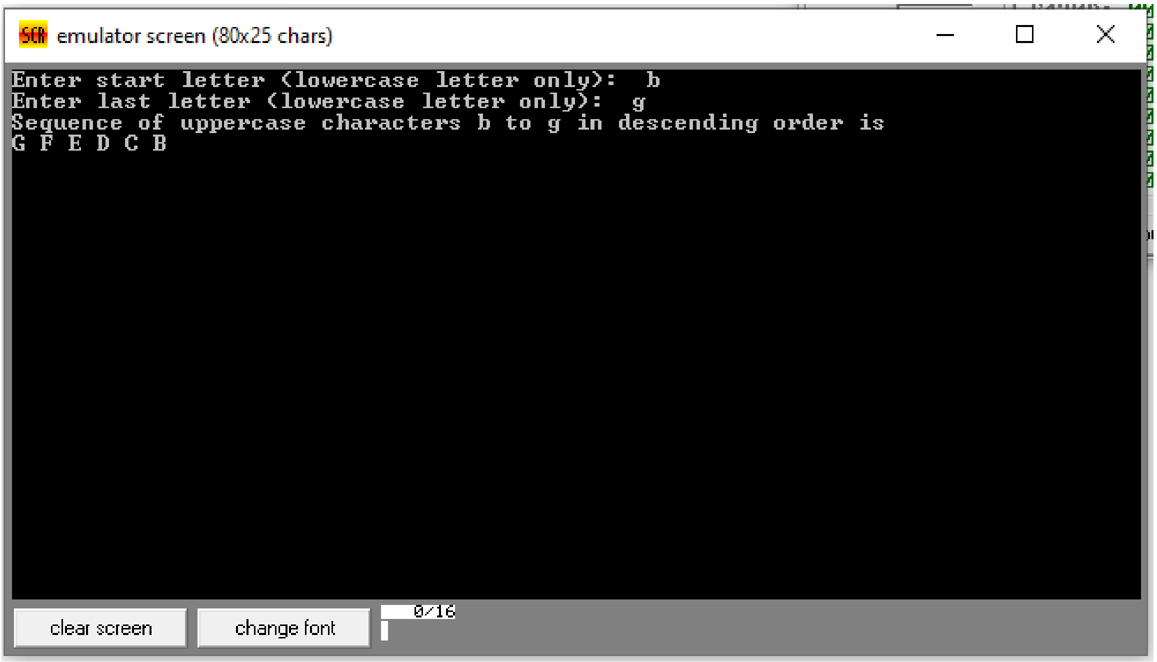


```
.model small

.code
    org 100h
    jmp start

.data
    firstChar db ?
    secondChar db ?

    startLetter db "Enter start letter (lowercase letter only): ",20h, "$"
    lastLetter db 10,13,"Enter last letter (lowercase letter only): ",20h, "$"
    resultLetters db 13,10,"Sequence of uppercase characters ", "$"
    keywordOne db " to ", "$"
    keywordTwo db " in descending order is ",10, 13, "$"
    whiteSpace db " ", "$"

start:
    lea dx, startLetter
    call disp_string
    call read_char
    mov firstChar, al

    lea dx, lastLetter
    call disp_string
    call read_char
    mov secondChar, al

    lea dx, resultLetters
    call disp_string
    mov dl, firstChar
    call print_char
    lea dx, keywordOne
    call disp_string
    mov dl, secondChar
    call print_char
    lea dx, keywordTwo
    call disp_string
    call disp_del

    mov dl, firstChar

    cmp secondChar, dl
    jl  loop exit

    mov dl, firstChar
    sub dl, 32
    mov firstChar, dl

output:
```

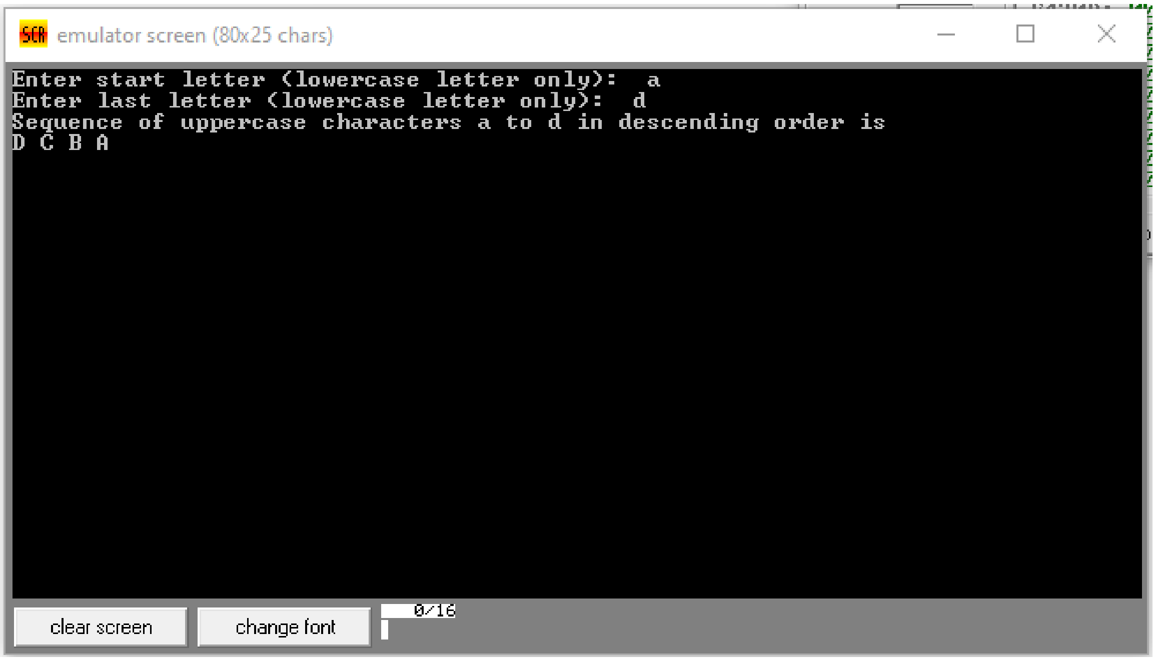


```
output:
    mov dl, secondChar
    sub dl, 32
    call print_char
    call disp_del
    dec byte ptr secondChar

    cmp dl, firstChar
    je loop_exit

    mov dx, offset whiteSpace
    call disp_string
    call disp_del
loop output

loop_exit:
    call exit

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

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

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

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

end start
```

Output (at least 3 scenarios):

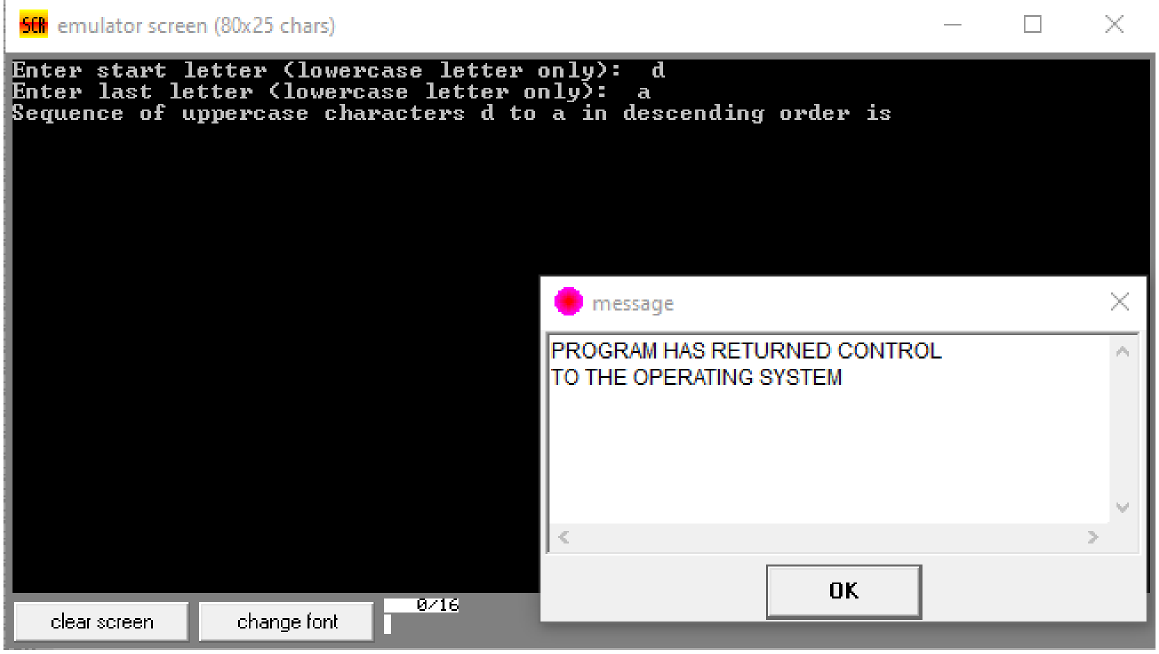


```
Enter start letter (lowercase letter only):  b
Enter last letter (lowercase letter only):  g
Sequence of uppercase characters b to g in descending order is
G F E D C B
```



```
Enter start letter (lowercase letter only):  a
Enter last letter (lowercase letter only):  d
Sequence of uppercase characters a to d in descending order is
D C B A
```

PROGRAM TERMINATES IF SECOND LETTER IS LESS THAN THE FIRST LETTER: (TO AVOID INFINITE LOOP)

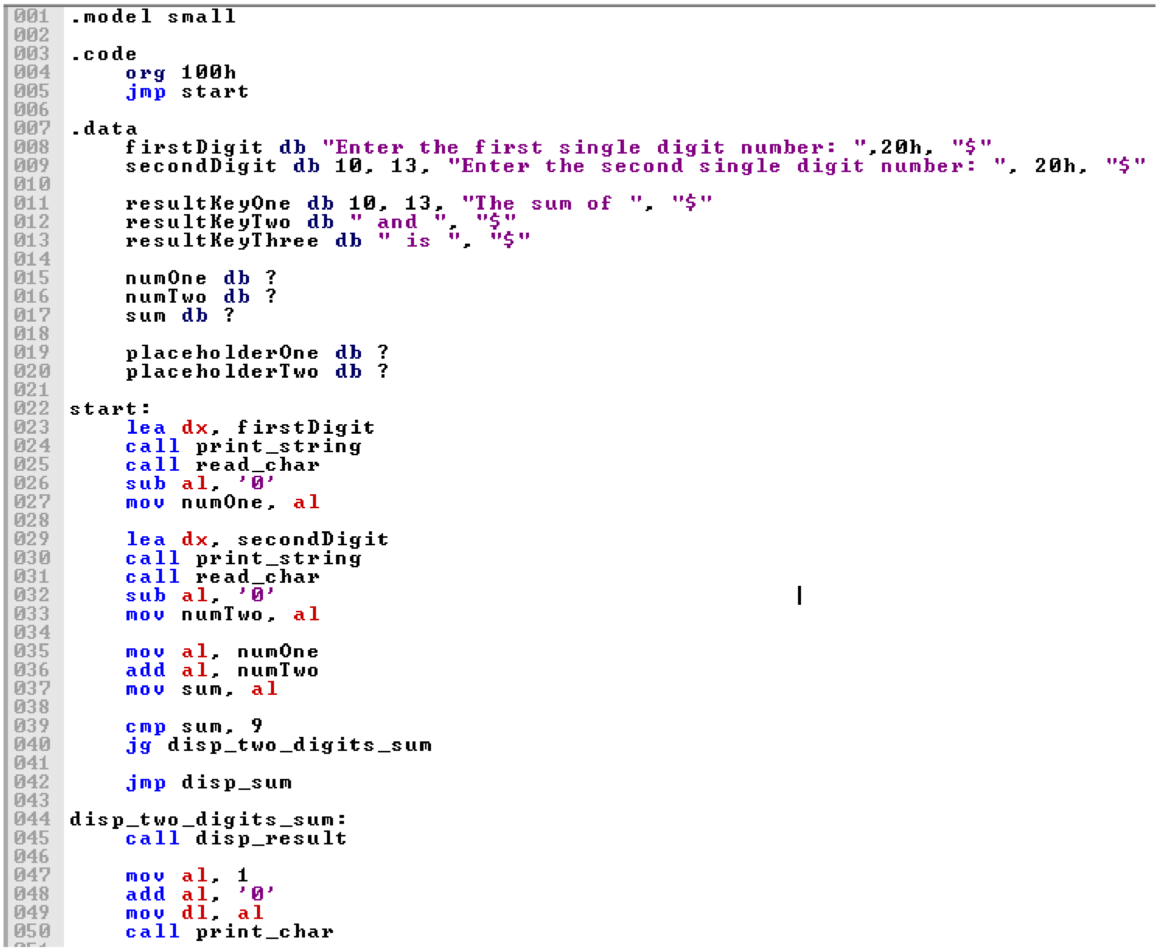


```
Enter start letter (lowercase letter only):  d
Enter last letter (lowercase letter only):  a
Sequence of uppercase characters d to a in descending order is
```

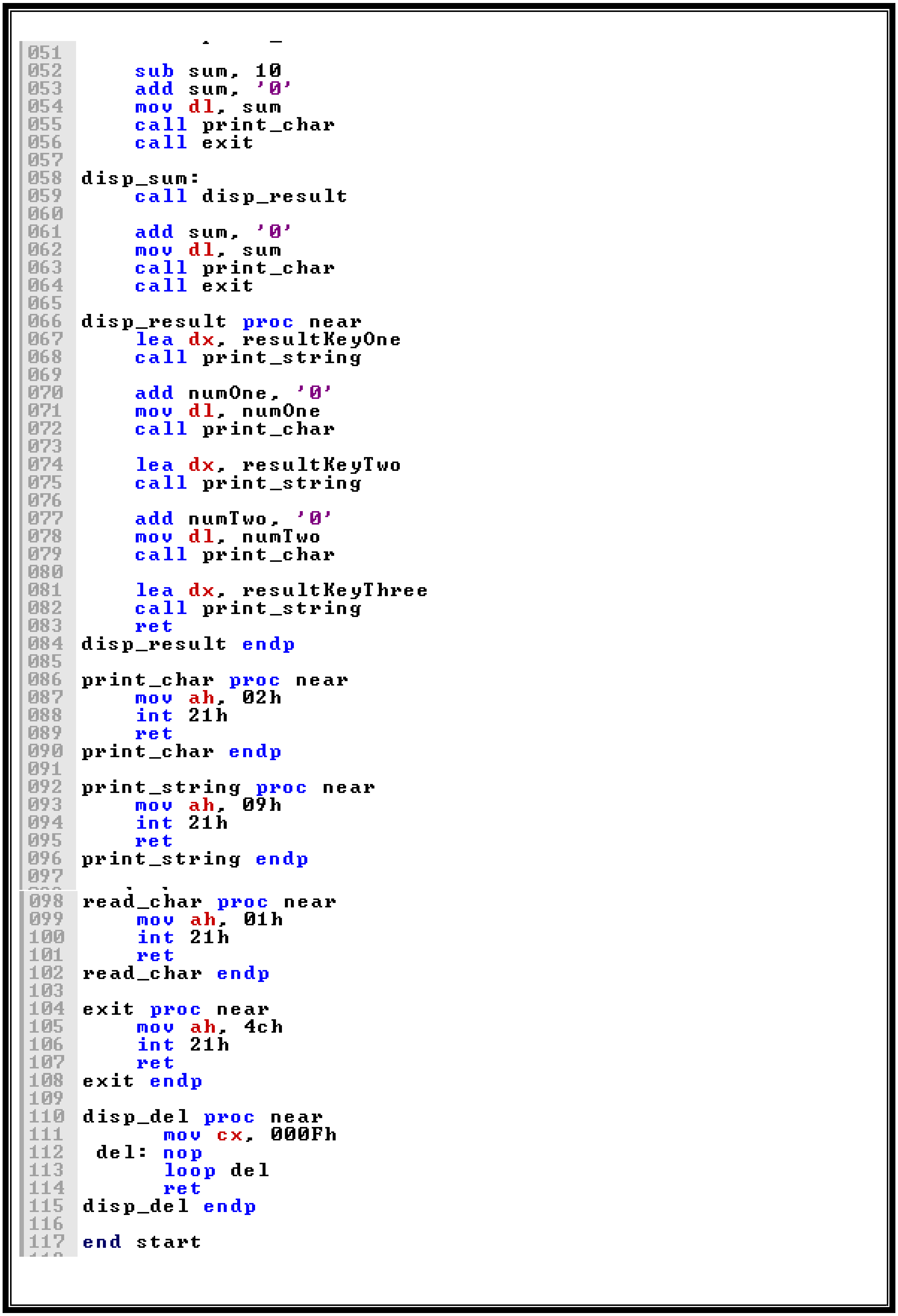
PROGRAM HAS RETURNED CONTROL
TO THE OPERATING SYSTEM

Program Exercise 1-3. Program Listing and Outputs

Program Listing:

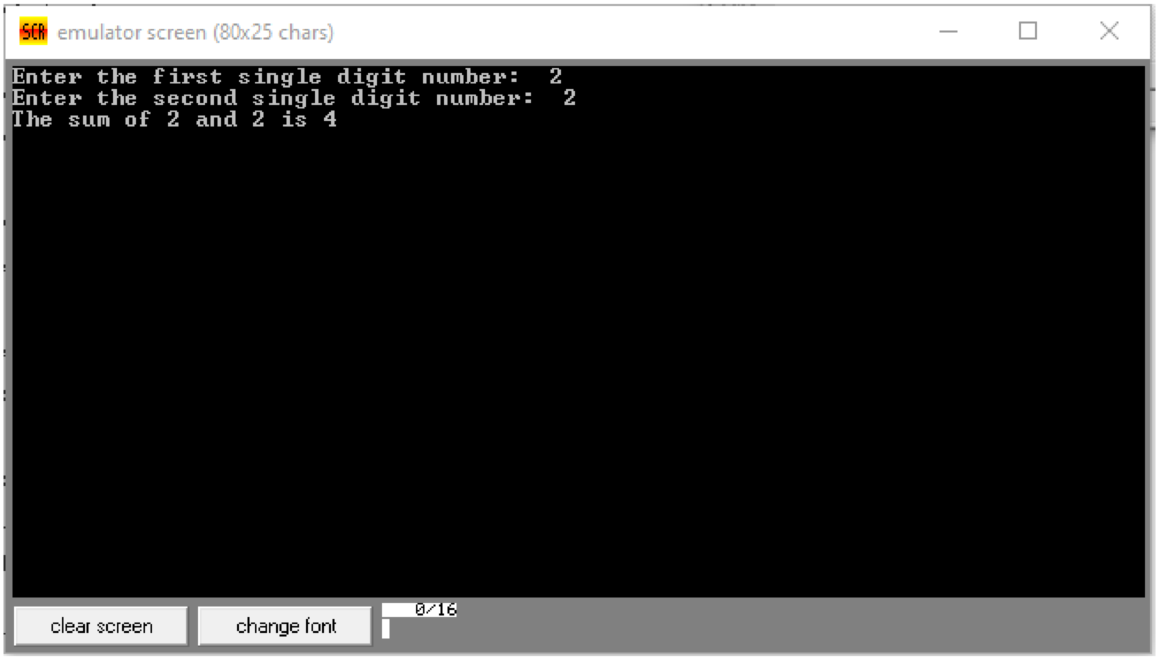


```
.model small

.code
    org 100h
    jmp start

.data
    firstDigit db "Enter the first single digit number: ",20h, "$"
    secondDigit db 10, 13, "Enter the second single digit number: ", 20h, "$"

    resultKeyOne db 10, 13, "The sum of ", "$"
    resultKeyTwo db " and ", "$"
    resultKeyThree db " is ", "$"

    numOne db ?
    numTwo db ?
    sum db ?

    placeholderOne db ?
    placeholderTwo db ?

start:
    lea dx, firstDigit
    call print_string
    call read_char
    sub al, '0'
    mov numOne, al

    lea dx, secondDigit
    call print_string
    call read_char
    sub al, '0'
    mov numTwo, al

    mov al, numOne
    add al, numTwo
    mov sum, al

    cmp sum, 9
    jg disp_two_digits_sum

    jmp disp_sum

disp_two_digits_sum:
    call disp_result

    mov al, 1
    add al, '0'
    mov dl, al
    call print_char
```

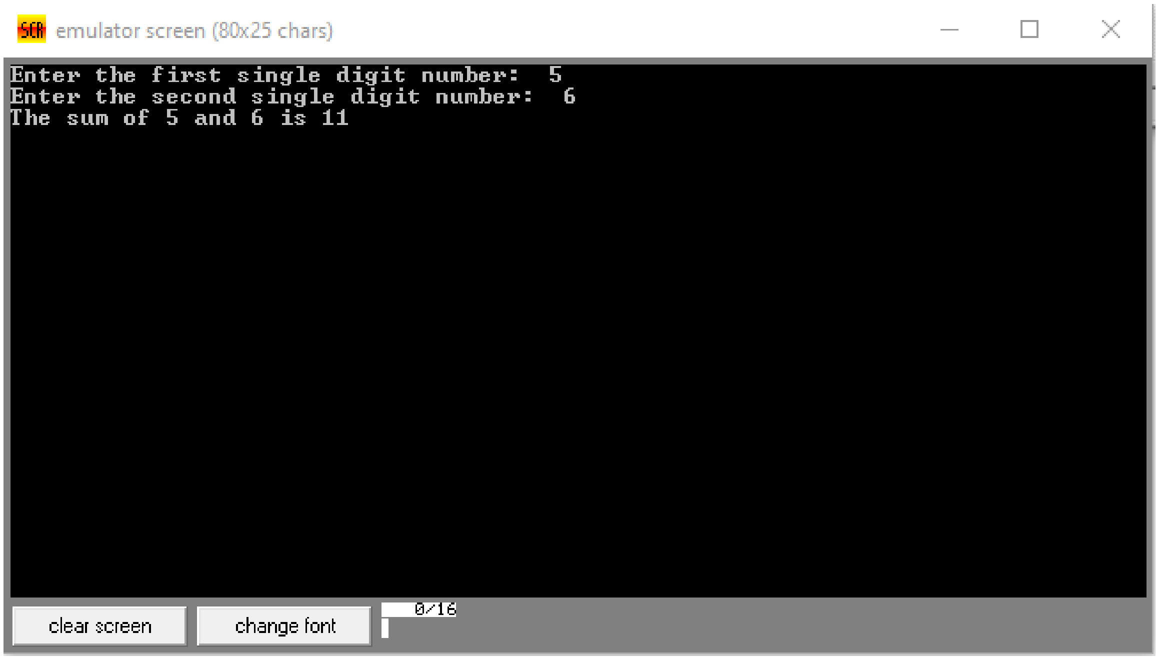


```
    sub sum, 10
    add sum, '0'
    mov dl, sum
    call print_char
    call exit

disp_sum:
    call disp_result

    add sum, '0'
    mov dl, sum
    call print_char
    call exit

disp_result proc near
    lea dx, resultKeyOne
    call print_string

    add numOne, '0'
    mov dl, numOne
    call print_char

    lea dx, resultKeyTwo
    call print_string

    add numTwo, '0'
    mov dl, numTwo
    call print_char

    lea dx, resultKeyThree
    call print_string
    ret
disp_result endp

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

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

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

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

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

end start
```

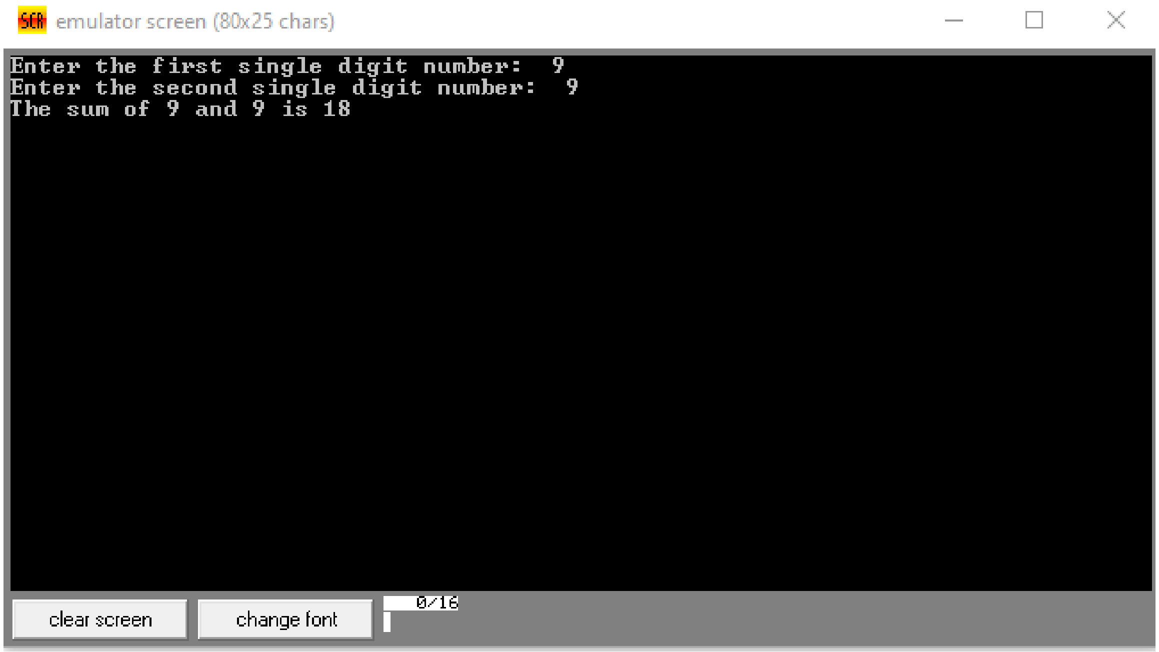
Output (at least 3 scenarios):



```
Enter the first single digit number:  2
Enter the second single digit number:  2
The sum of 2 and 2 is 4
```



```
Enter the first single digit number:  5
Enter the second single digit number:  6
The sum of 5 and 6 is 11
```



```
Enter the first single digit number:  9
Enter the second single digit number:  9
The sum of 9 and 9 is 18
```

DATA ANALYSIS

In addressing a drill exercise like Program Exercise 1, our focus is on developing an optimized and procedural-based assembly program. This specific drill requires us to display a character sequence that progresses from uppercase 'A' to 'Z'. Crucially, we introduce delays between character outputs to enhance the user experience.

Turning our attention to drill exercise 2 inspired by Program Exercise 2, our objective is to create and test an optimized assembly program. This drill prompts users to input start and stop characters (in lowercase) and then displays the corresponding uppercase characters in ascending order. Like Program Exercise 2, controlled delays between character outputs are implemented for clarity.

Revisiting Program Exercise 1, our focus is on crafting an optimized and procedural-based assembly program. This exercise requires the display of a character sequence like the one in our first drill exercise. However, in this program exercise, we emphasize a more holistic approach to assembly programming by incorporating additional features such as structured delays.

Like our drill exercise inspired by Program Exercise 2, our attention in Program Exercise 2 itself is centered on creating an optimized assembly program. This program is designed to dynamically handle user input for start and stop characters and present the corresponding uppercase characters in descending order. Our approach prioritizes procedural programming and efficiency, with deliberate delays between character outputs.

The final program exercise, Program Exercise 3, introduces a unique challenge. Our objective is to create and test an optimized and procedural-based assembly program that prompts users to input two single-digit numbers (ranging from '0' to '9'). The program then displays the sum of these numbers, addressing specific cases where the sum may exceed 9. This exercise demonstrates our adaptability in assembly programming, showcasing the ability to handle complex computation tasks efficiently.

QUESTIONS AND ANSWERS

Questions:

1. What is the between INT 21H service 01H, 07H, and 08H?
2. Create and test assembly program that will display information similar to the output below using INT 21H service 09H. Save as SURNAME_**COEN3211_QA2.asm.**

   Sample output:

   DELA CRUZ, Juan T.

   *<space>***Colegio de Muntinlupa**

   *<space><space>***BS Computer Engineering Student**

   *<space><space><space>***COEN3211 – Microprocessors Lab**

Answers:

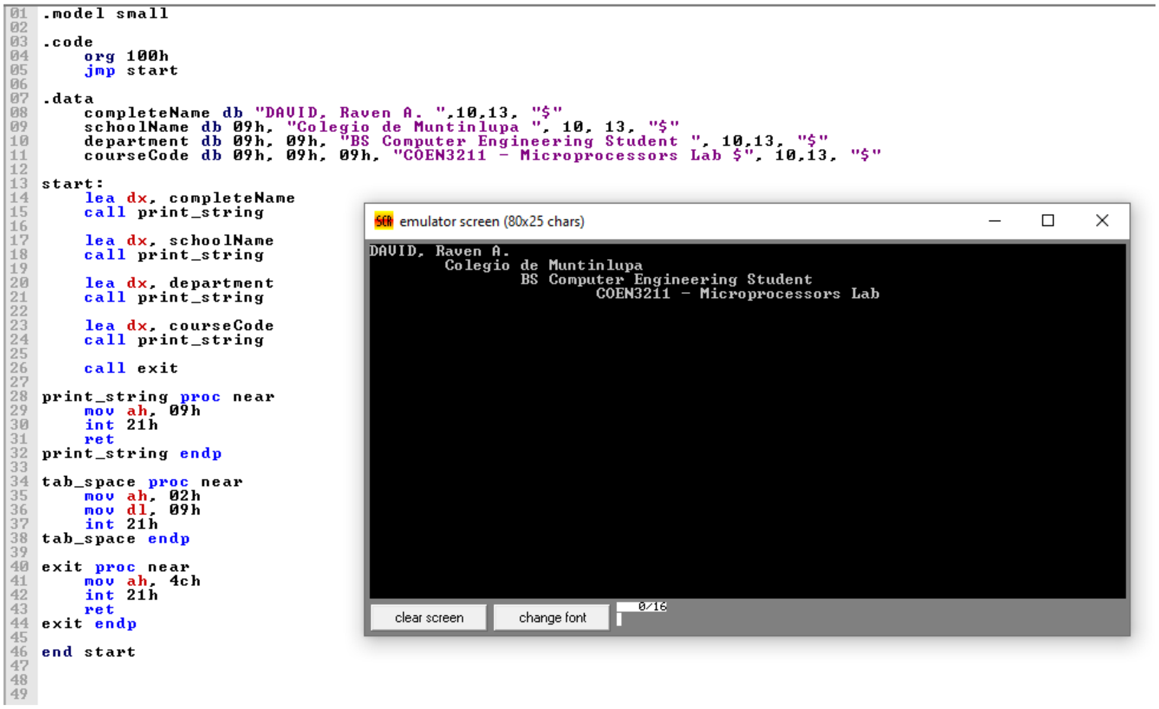
1. **INT 21H** Service 01H is used for reading a single character, Service 07H is used for reading a string with echo, and Service 08H is used for reading a string without echo from the standard input.

   INT 21H Service 01H (Read Character Input):
   Function: This service reads a character from the standard input (keyboard).
   Registers Used:
   AH = 01H (function code for read character)
   AL = (returned) ASCII code of the read character

   INT 21H Service 07H (Read String Input with Echo):
   Function: This service reads a string from the standard input with echo (characters are displayed as they are typed).
   Registers Used:
   AH = 07H (function code for read string with echo)
   AL = (returned) ASCII code of the last character in the string
   DX = Address of the buffer to store the input string

INT 21H Service 08H (Read String Input without Echo):
Function: This service reads a string from the standard input without echo (characters are not displayed as they are typed).
Registers Used:
AH = 08H (function code for read string without echo)
AL = (returned) ASCII code of the last character in the string
DX = Address of the buffer to store the input string

2. Program and Output:



```
.model small

.code
    org 100h
    jmp start

.data
    completeName db "DAVID, Raven A. ",10,13, "$"
    schoolName db 09h, "Colegio de Muntinlupa ", 10, 13, "$"
    department db 09h, 09h, "BS Computer Engineering Student ", 10,13, "$"
    courseCode db 09h, 09h, 09h, "COEN3211 - Microprocessors Lab $", 10,13, "$"

start:
    lea dx, completeName
    call print_string

    lea dx, schoolName
    call print_string

    lea dx, department
    call print_string

    lea dx, courseCode
    call print_string

    call exit

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

tab_space proc near
    mov ah, 02h
    mov dl, 09h
    int 21h
tab_space endp

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

end start
```

emulator screen (80x25 chars)

```
DAVID, Raven A.
        Colegio de Muntinlupa
                BS Computer Engineering Student
                        COEN3211 - Microprocessors Lab
```

CONCLUSION

In our first Microprocessors lab, we're diving into the basics of assembly language programming. We're focusing on Intel x86 architecture and DOS interrupt services. This lab helps students gain practical skills, especially in working with DOS interrupt service routines through hands-on activities.

Our goal is to make students comfortable with how DOS interrupt service routines work. We'll explore both hardware and software interruptions, with a special focus on INT 21H DOS service routines. These routines handle tasks like reading characters, displaying single characters, and presenting strings, giving students a practical understanding of dealing with the operating system.

We're also asking students to create assembly programs using INT 21H DOS service routines through some drill exercises. These exercises are not just challenges; they're opportunities for students to practice and solidify their understanding of syntax and using interrupt service routines effectively.

Finally, we're putting an emphasis on testing and simulating assembly programs using emu8086 assembler software. This step is crucial for students to not only write code but also thoroughly evaluate how it runs. Through these drills, we're ensuring a strong understanding of program behavior.

In essence, this lab is designed to offer a complete and hands-on learning experience in assembly language programming. We're combining theoretical concepts with practical exercises and using modern tools to prepare students for working with Intel x86 assembly language and DOS interrupt services. The ultimate goal is to empower students with the skills needed to create, test, and optimize assembly programs, laying a solid foundation for further exploration in low-level programming.

REFERENCES

YouTube Video:

Hack. (2008, November 13). *Bermagui forest disputed turf [Video]. The Hack Half Hour. Retrieved from https://youtu.be/N_lZDj9L0YM?si=Q8C8nVx9BQhqzsrF*

Stack Overflow Forum Post:

Anonymous. (2014, January 1). 8086 Assembly Language Program to Print from A to Z. Stack Overflow. Retrieved from https://stackoverflow.com/questions/20868746/8086-assembly-language-program-to-print-from-a-to-z