

COAL LAB MANUALS

Submitted to:

Dr. Tauqir

Teacher assistant : Sir Shahzad

Submitted By:

Komal Shehzadi

Registration no : 2016-CS-178

Section:

Section B

Department:

Department of Computer Science

Institute:

University of Engineering and Technology Lahore (Main Campus)



Date of Submission: March 29, 2018

Contents

LAB 1	5
TASK: Take a character input from keyboard and display it on screen	5
Commands :.....	5
OUTPUT:	6
Description:	7
LAB 2	7
Task : ADDING TWO NUMBERS FROM 0 TO 9 AND SHOWING THERE SUM TILL ONE DIGIT.....	7
Commands:.....	10
OUTPUT:	11
Description:	11
LAB 3	12
Task : Adding two digits and displaying result up to two digits	12
Command:	16
OUTPUT:	17
Description:	18
LAB 4	18

Task 1:.....	18
Take a digit input from user and check whether it is even or odd.....	18
Commands:.....	21
OUTPUT:	24
Description:	24
TASK 2 : Home Task	25
Take a character input from user and convert it to its opposite case means lower to upper and upper to lower case.....	25
Commands:.....	30
OUTPUT:	33
Description:	34
LAB 5	35
Task 1 : Print 0 to 9 digits on screen without using loop and then by using loop	35
Commands:.....	37
Output :	40
Description:	41

Task 2 : Take 3 single digit inputs from keyboard and sort them in ascending order	42
Commands:.....	50
OUTPUT :	52
Description:	53
LAB 6	55
Task: Take a string from data segment and display vowels in it and count number of vowel in it and display count on screen	55
Commands:.....	59
OUTPUT :	62
Description:	62

.....

.....

.....**THE_END**.....

.....

.....

LAB 1

TASK: Take a character input from keyboard and display it on screen

The description inside the “// ‘ ‘ // “ are the comments to make the code more clear. These are not the part of program.

```
// PROGRAM //
```

```
// Taking input from Key Board //
```

```
Org 100
```

```
// by moving 1 in AH interrupt will produce a character input prompt  
on screen //
```

```
MOV AH, 1
```

```
INT 21H
```

```
// Displaying it on screen //
```

```
MOV DL, AL
```

```
MOV AH, 2
```

```
INT 21H
```

```
ret
```

Commands :

Org 100 : Set offset of the segment originated at 100hex

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

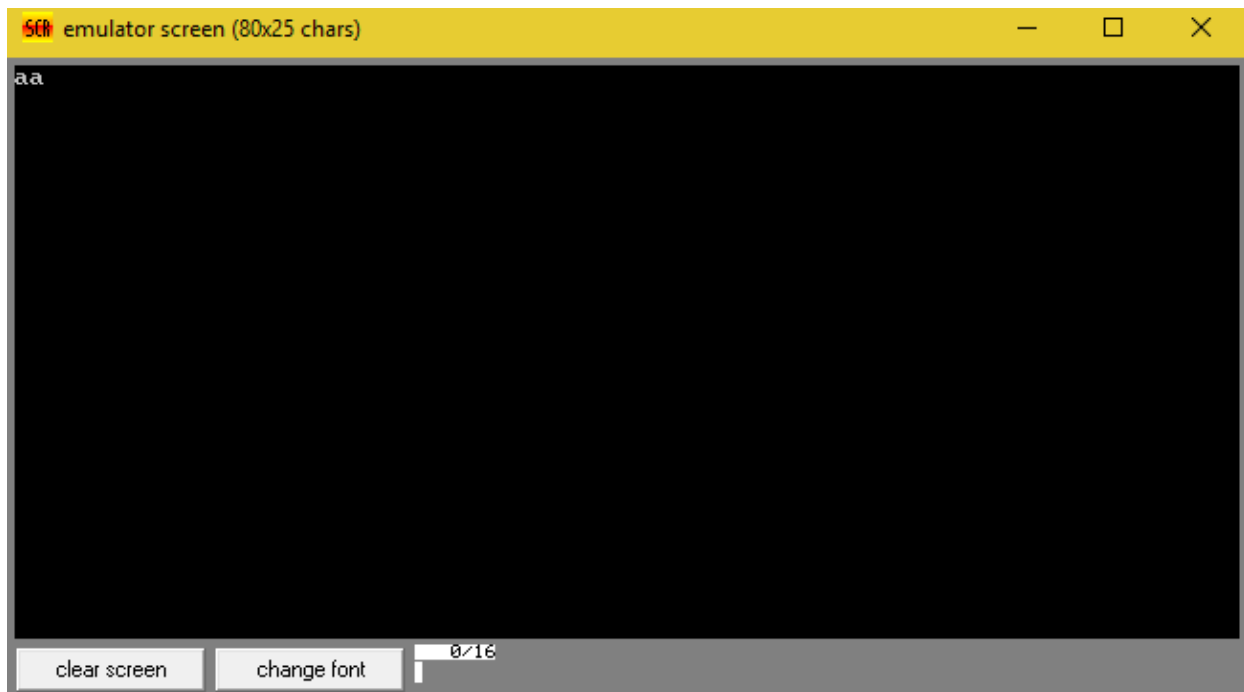
Mov Destination, Source

INT 21H: Call to DOS interrupt Handler

Ret: return statement to end the program

OUTPUT:

When the above program is run following output screen is displayed first 'a' is input character and 2nd 'a' is the output character.



Description:

In this lab task we are taking a character input from keyboard and displaying it on screen. For this purpose we have used interrupts after moving certain values in AH, AL and DL register

By moving 1 in AH register interrupt prompt screen to take character input from screen and that input will be saved in AL register automatically. So further we have moved 2 in AH register to display the character stored in DL register so move the AL contents in DL register and then produce interrupt so the input character would be displayed on screen.

LAB 2

Task : ADDING TWO NUMBERS FROM 0 TO 9 AND SHOWING THERE SUM TILL ONE DIGIT

The description inside the “// ‘ ‘ // “ are the comments to make the code more clear. These are not the part of program.

```
// PROGRAM //
```

```
org 100h
```

// taking first number from keyboard //

mov AH, 1

INT 21h

//saving it in BH register in its real value by subtracting it from 30 //

mov BH, AL

sub BH, 30h

// displaying new line and carriage return to place the cursor on the start of next line //

mov ah, 2

mov dl, 0Ah

INT 21h

mov dl, 0Dh

INT 21h

// taking 2nd input number from keyboard//


```
mov ah, 1
```

```
INT 21h
```

```
// saving it in BL in its original state//
```

```
mov BL, AL
```

```
sub BL, 30h
```

```
// Displaying a new line on screen//
```

```
mov Ah, 2
```

```
mov DL, 0Ah
```

```
INT 21h
```

```
// adding 2nd input placed in BL to BH//
```

```
ADD BH, BL
```

```
// adding 30 in result to display the decimal value //
```

```
add BH, 30h
```

```
// Displaying the result of sum of two digits //
```

```
mov AH, 2  
mov DL, BH  
int 21h  
ret  
  
// END //
```

Commands:

Org 100: Set offset of the segment originated at 100hex

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler

Sub: It is used to subtract destination from source and store the result of computation in destination.

Sub Destination, Source

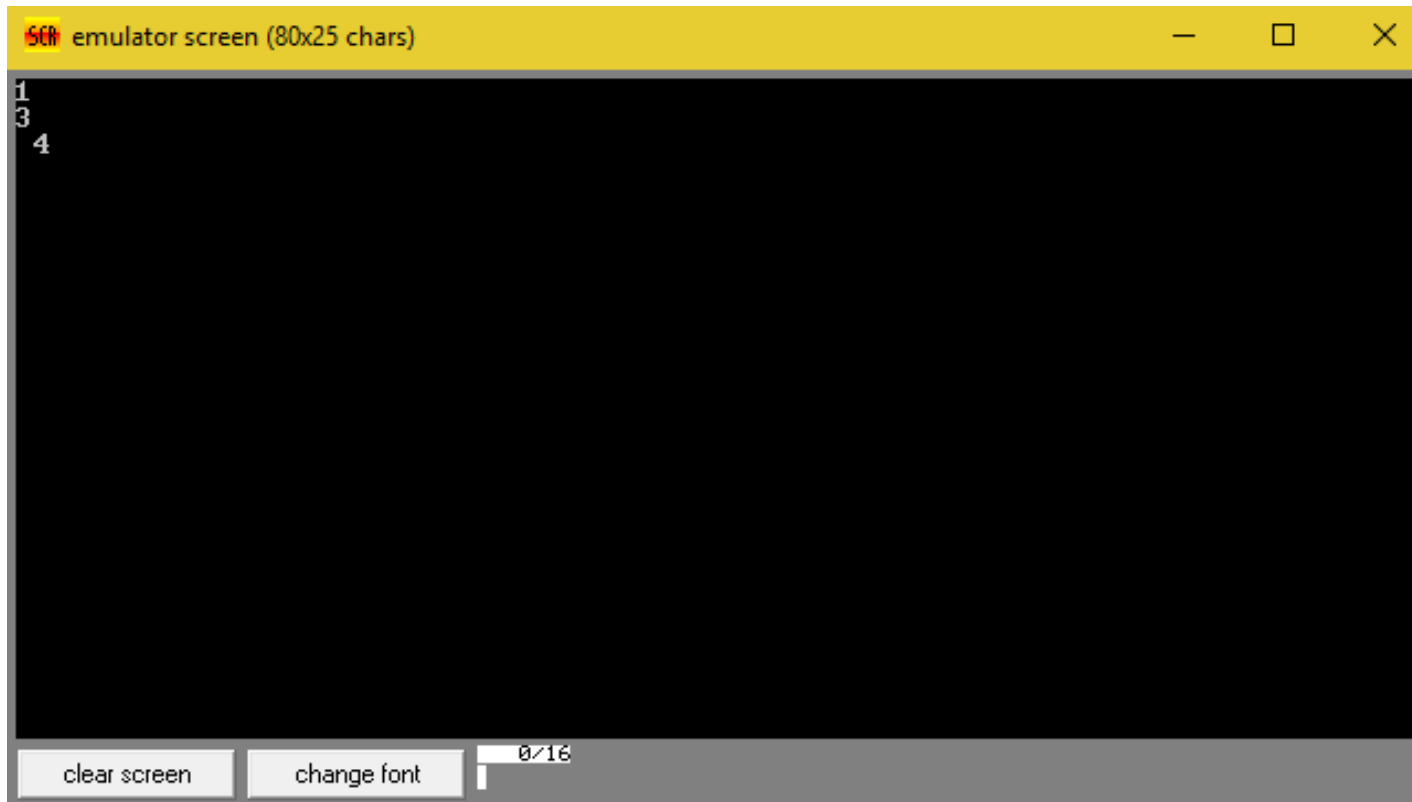
Add: It is used to add source in destination and store the result of computation in destination.

Add Destination, Source

Ret: Return statement to end the program

OUTPUT:

The following output screen is displayed when the above program is run 3rd digit is the sum of first and 2nd digit.



Description:

in this lab we are adding two numbers entered from keyboard both numbers are from 0 to 9 and by adding them we can display there sum till one digit means maximum digit 9 can be displayed correctly as the sum of two digits because till now we can handle hexadecimal single digit through register so we are displaying a sum of maximum one digit value. For this purpose we have used simple interrupts to take input and display output on screen and to store the hexadecimal value in

decimal state we have subtracted 30 from the value and vice versa to display the value we have added 30 in it.

LAB 3

Task : Adding two digits and displaying result up to two digits

The description inside the “// ‘ ‘ // ” are the comments to make the code more clear. These are not the part of program.

// PROGRAM //

```
org 100h
```

```
// taking first input from user//
```

```
mov AH, 1
```

```
INT 21h
```

```
// storing it in BH register//
```

```
mov BH, AL
```

```
sub BH, 30h
```

```
// displaying a newline on screen//
```

```
mov ah, 2
```

```
mov dl, 0Ah
```

```
INT 21h
```

```
// moving cursor to the start of line//
```

```
mov dl, 0Dh
```

```
INT 21h
```

// taking 2nd input from user//

mov ah, 1

INT 21h

// storing it in BL register in its decimal form by subtracting it from 30h//

mov BL, AL

sub BL, 30h

// displaying a new line//

mov Ah, 2

mov DL, 0Ah

INT 21h

// adding 2nd input to first input//

ADD BH, BL

// making things ready to display on screen//

```
mov AH, 2
```

```
// by default div instruction divides something from BL  
register's value so we have to divide our value by 10 so  
moving 10 in bl//
```

```
mov BL, 10
```

```
// clear data in AL and AH so nullify AX//
```

```
mov AL, 00h
```

```
mov AH, 00h
```

```
mov AL, BH
```

```
// divide AL from BL and split it in AL and AH //
```

```
div BL
```

```
//moving results in suitable registers//
```

```
mov BH, AH
```

```
mov BL, AL
```

```
// converting them in decimal form to display on screen//
```

```
add BH, 30h
```

```
add BL, 30H
```

//displaying BL and BH on screen//

mov AH, 2

mov DL, BL

int 21h

mov DL, BH

int 21h

ret

Command:

Org 100: Set offset of the segment originated at 100hex

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler

Sub: It is used to subtract destination from source and store the result of computation in destination.

Sub Destination, Source

Add: It is used to add source in destination and store the result of computation in destination.

Add Destination, Source

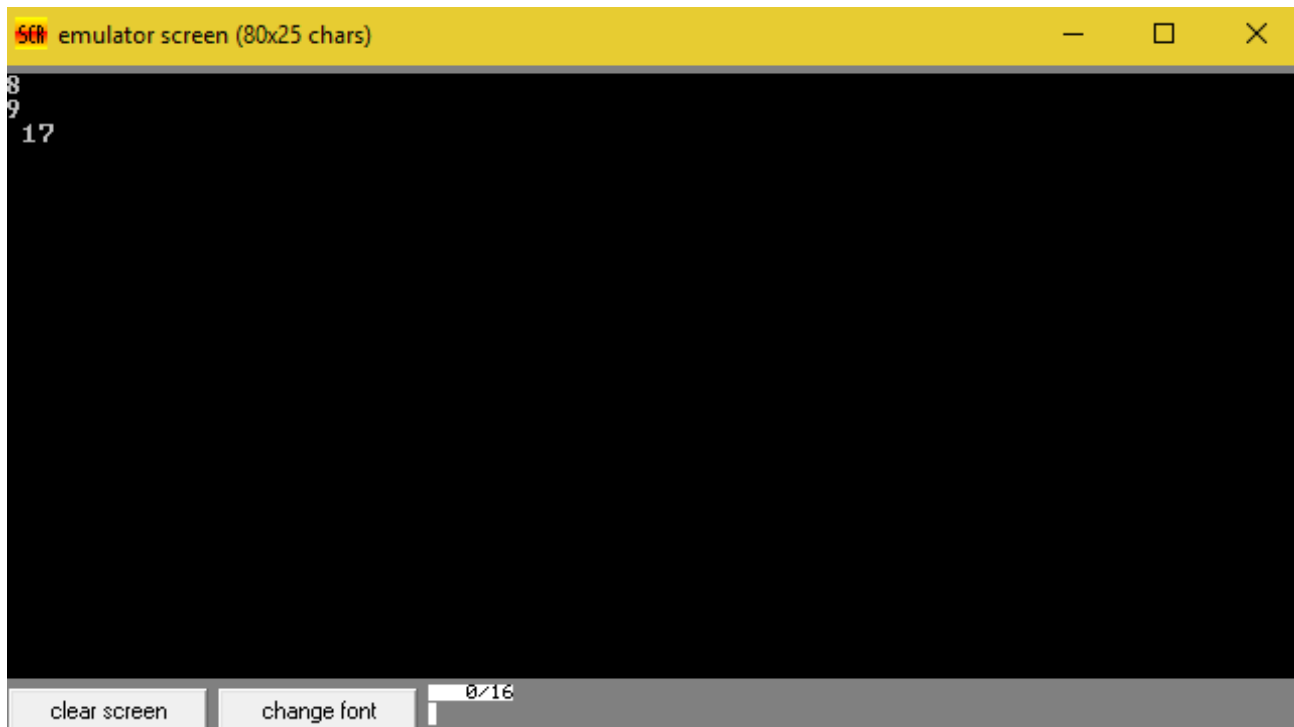
Div: It is used to Divide source contents by BL register's contents and store the quotient and remainder of computation in AL and AH register respectively

Div Source

Ret: Return statement to end the program

OUTPUT:

The following output screen will display when we sum 8 and 9 in the above code 3rd line contains the sum of first and 2nd digit.

A screenshot of a software emulator window titled "emulator screen (80x25 chars)". The window has a yellow title bar with standard minimize, maximize, and close buttons. The main area is black with white text. On the left side, the numbers "8", "9", and "17" are displayed vertically, one above the other. At the bottom of the window, there is a grey control bar containing two buttons labeled "clear screen" and "change font", followed by a small status indicator showing "0/16".

Description:

In this lab we have extended the work of 2nd lab we are taking two inputs from user and adding them in 2nd lab in previous lab we can only display a single digit on screen as a sum of two inputs but in this lab we can display sum up to two digits maximum 18 on screen when a user enters 9 and 9 to be sum .For this purpose we have used a new command of div to split our digit in two parts and by default the remainder and quotient are stored in AL and AH register respectively div command automatically divides the operand by the contents in BL register so move 10 in BL so that we can get both parts of a digit . And then we can display a two digit number created in a result of arithmetic calculation on screen easily.

LAB 4

Task 1:

Take a digit input from user and check whether it is even or odd.

The description inside the “// ‘ ‘ // “ are the comments to make the code more clear. These are not the part of program.

```
// PROGRAM //
```

```
.model
```

```
// Data Segment//
```

// All variables defined here //

.Data

msg DB 10, 13, 'Enter A NUMBER \$'

msg1 DB 10, 13, 'number is even!\$'

msg2 DB 10, 13, 'number is odd! \$'

msg3 DB 10, 13, 'do you want to continue(y/n) for yes or no?', 10, 13, '\$'

// Code Segment //

// all the functions to be performed written here //

.Code

// Basic commands to use the variables in data segment //

// Compulsory!! //

mov AX, @data

mov DS, AX

// Jump on label p1 //

JMP p1

// Label p1 //

// in label p1 we are taking single digit input from keyboard

And checking its remainder from 2 to check the even or odd digit if remainder is 0, it is even digit, control jumps on label p2 otherwise, it is odd digit, it will jump on p3 //

p1:

LEA DX, msg

mov ah, 9

int 21h

mov ah, 1

int 21h

mov bl, 2

```
div bl
CMP ah, 0
JE p2
JNE p3
```

// Label p2 //

// it will display that the number is even and then the control jumps on label p4 //

```
p2:
LEA DX, msg1
mov ah, 9
int 21h
jmp p4
```

// Label p3 //

// it will display that the number is odd and then control jumps on label p4 //

p3:

```
LEA DX, msg2
mov ah, 9
int 21h
jmp p4
```

// Label p4 //

// this will prompt the user for another input or just exit the program by entering, 'y' to continue for another input at this control will again jump on p1 label and if user enters 'n' control will jump on exit label //

```
p4:  
LEA DX, msg3  
mov Ah, 9  
int 21h  
mov ah, 1  
int 21h  
CMP al, 'y'  
JE p1  
JNE exit
```

```
// Label exit //  
// Exits the program //
```

```
exit:  
.exit
```

Commands:

.model: it is used to tell the assembler that program is in protected mode.

.code: it contains all the codes and commands.

.data: it contains all the definitions and declarations of variables.

Labels: these are the customized names of specific code segment to call that code segment anywhere in program by its name

LABEL_NAME:

Write the code here.

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler.

INT 21H

LEA: Load effective address of a memory block in a register it moves the starting address of a memory block in a register.

LEA Source, Destination

JMP: JMP command is used for unconditional jump on a label.

JMP LABEL_NAME

CMP: CMP is used to compare the contents of two register or variables and give a value in result if source is less than , greater than or equal to destination then -1, 1 or 0 is returned by CMP respectively.

CMP Source, Destination

JNE: in the result of CMP command this label will be executed if the contents of source and destination are not equal and by this we can jump on a label

JNE LABEL_NAME

JLE: In the result of CMP command this label will be executed if the contents of source is less or equal to destination by this we can jump on a label

JLE LABEL_NAME

JE: In the result of CMP command this label will be executed if the contents of source and destination are equal and by this we can jump on a label

JE LABEL_NAME

JL: In the result of CMP command this label will be executed if the contents of source is less than destination by this we can jump on a label

JL LABEL_NAME

JG: : In the result of CMP command this label will be executed if the contents of source is greater than destination by this we can jump on a label

JG LABEL_NAME

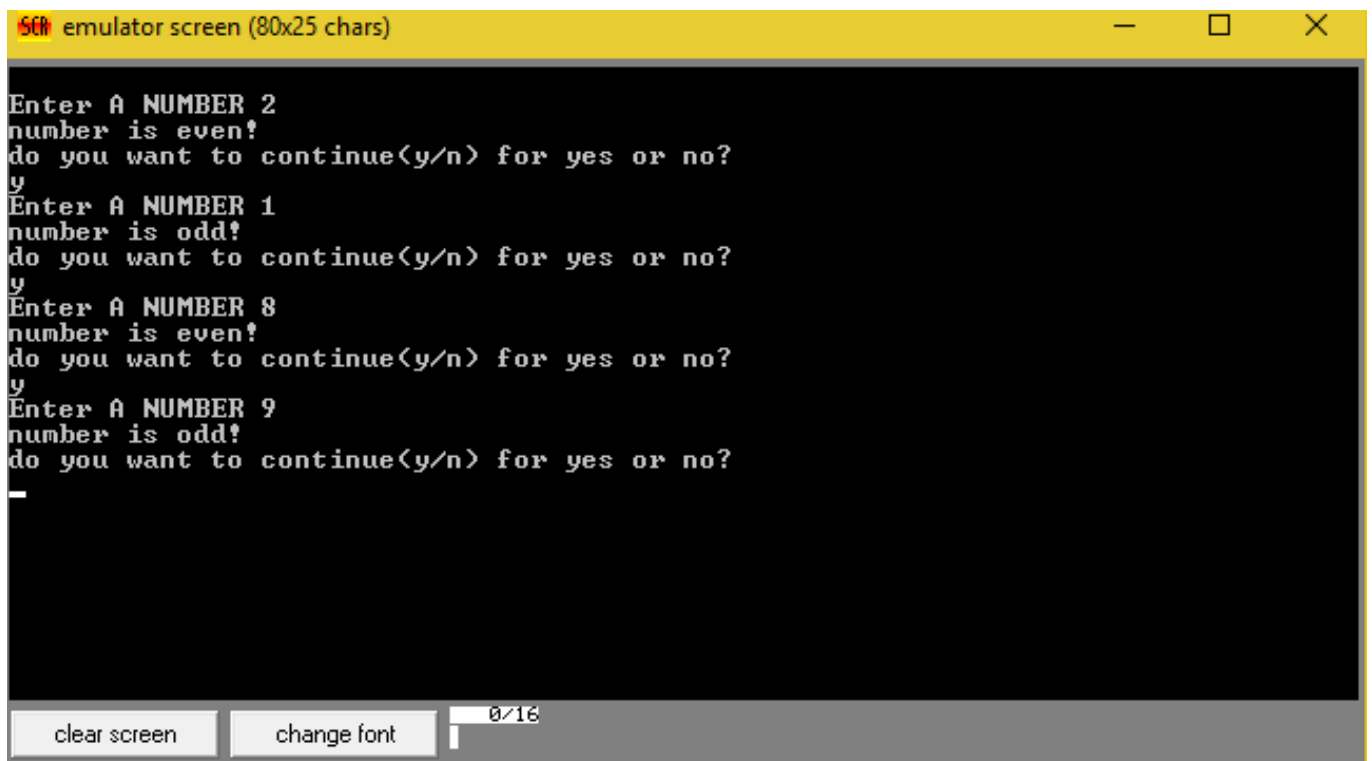
JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

JGE LABEL_NAME

.exit: Command to exit the program.

OUTPUT:

The following output screen will appear when the above code is run enter the number and it will tell either the number is even or odd and then prompt for continue or exit.



```
emulator screen (80x25 chars)
Enter A NUMBER 2
number is even!
do you want to continue(y/n) for yes or no?
y
Enter A NUMBER 1
number is odd!
do you want to continue(y/n) for yes or no?
y
Enter A NUMBER 8
number is even!
do you want to continue(y/n) for yes or no?
y
Enter A NUMBER 9
number is odd!
do you want to continue(y/n) for yes or no?
_
```

The screenshot shows a terminal window titled "emulator screen (80x25 chars)". It displays the execution of a program that repeatedly asks for a number, checks if it's even or odd, and prompts the user to continue (y/n) or exit. The user has entered 'y' for all four prompts. The interface includes a yellow title bar, a black terminal area, and a grey footer bar with buttons for "clear screen" and "change font", along with a cursor position indicator "0/16".

Description:

In this lab we take a digit input from user and check whether it is even or odd and display the result on screen. For this purpose we have used number of labels to take input and display the desired even or odd output and then again prompt for next

input or exit the program. In data segment the string variables are stored. Where we have to display the string variable move 9 in AH register. And load the string variable to be displayed in DX register by LEA command. And all the other tasks are done as explained in code.

TASK 2 : Home Task

Take a character input from user and convert it to its opposite case means lower to upper and upper to lower case.

The description inside the “// ‘ ‘ // “ are the comments to make the code more clear. These are not the part of program.

```
// PROGRAM //
```

```
.model
```

```
// DATA SEGMENT //
```

```
// All the required variables and string messages to be displayed on  
screen are declared here //
```

```
.data
```

```
msg DB 10, 13, 'enter a alphabet character from keyboard $'
```

msg1 DB 0ah, 0dh,'the upper case of letter is \$'

msg2 DB 0ah, 0dh,'the lower case of letter is \$'

msg3 DB 0ah, 0dh,'enter 1 to continue and 0 to exit: \$'

// CODE SEGMENT //

// All the codes are written here to perform the action //

.code

mov AX, @data

mov DS, AX

jmp p0

// LABEL p0 //

// Takes input from user //

p0:

LEA DX, msg

mov ah, 9

int 21h

jmp p1

// LABEL p1 //

// Compares the input character with suitable hexadecimal ascii values to check whether it is upper case or lower case then jump on suitable label to convert it in its opposite case //

p1:

mov ah, 1

int 21h

mov bl, al

Cmp bl, 91

JL p3

CMP bl, 96

JG p2

// LABEL p3 //

// it converts lower case letter in upper case and shows the screen message “the upper case of letter is “ and then display the upper case letter. And then jumps on label p4 //

p2:

LEA DX, msg1

mov ah, 9

int 21h

```
sub bl, 32
mov ah, 2
mov dl, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, bl
int 21h
jmp p4
```

```
// LABEL p3 //
```

```
// it converts upper case letter in lower case and shows the screen  
message "the lower case of letter is " and then display the lower case  
letter. And then jumps on label p4 //
```

```
p3:
LEA DX, msg2
mov ah, 9
int 21h
add bl, 32
mov ah, 2
```

mov dl, 0ah

int 21h

mov dl, 0dh

int 21h

mov dl, bl

int 21h

jmp p4

// LABEL p4 //

**// It prompts screen for user to either continue for next character or
exit the program //**

p4:

Lea dx, msg3

mov ah, 9

int 21h

mov ah, 1

int 21h

cmp al, 49

JE p0

JNE exit

```
// exit Label //
```

```
// To exit the program //
```

exit:

.exit

Commands:

.model: it is used to tell the assembler that program is in protected mode.

.code: it contains all the codes and commands.

.data: it contains all the definitions and declarations of variables.

Labels: these are the customized names of specific code segment to call that code segment anywhere in program by its name

LABEL_NAME:

Write the code here.

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler.

INT 21H

LEA: Load effective address of a memory block in a register it moves the starting address of a memory block in a register.

LEA Source, Destination

JMP: JMP command is used for unconditional jump on a label.

JMP LABEL_NAME

CMP: CMP is used to compare the contents of two register or variables and give a value in result if source is less than , greater than or equal to destination then -1, 1 or 0 is returned by CMP respectively.

CMP Source, Destination

JNE: in the result of CMP command this label will be executed if the contents of source and destination are not equal and by this we can jump on a label

JNE LABEL_NAME

JLE: In the result of CMP command this label will be executed if the contents of source is less or equal to destination by this we can jump on a label

JLE LABEL_NAME

JE: In the result of CMP command this label will be executed if the contents of source and destination are equal and by this we can jump on a label

JE LABEL_NAME

JL: In the result of CMP command this label will be executed if the contents of source is less than destination by this we can jump on a label

JL LABEL_NAME

JG: : In the result of CMP command this label will be executed if the contents of source is greater than destination by this we can jump on a label

JG LABEL_NAME

JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

JGE LABEL_NAME

Sub: It is used to subtract destination from source and store the result of computation in destination.

Sub Destination, Source

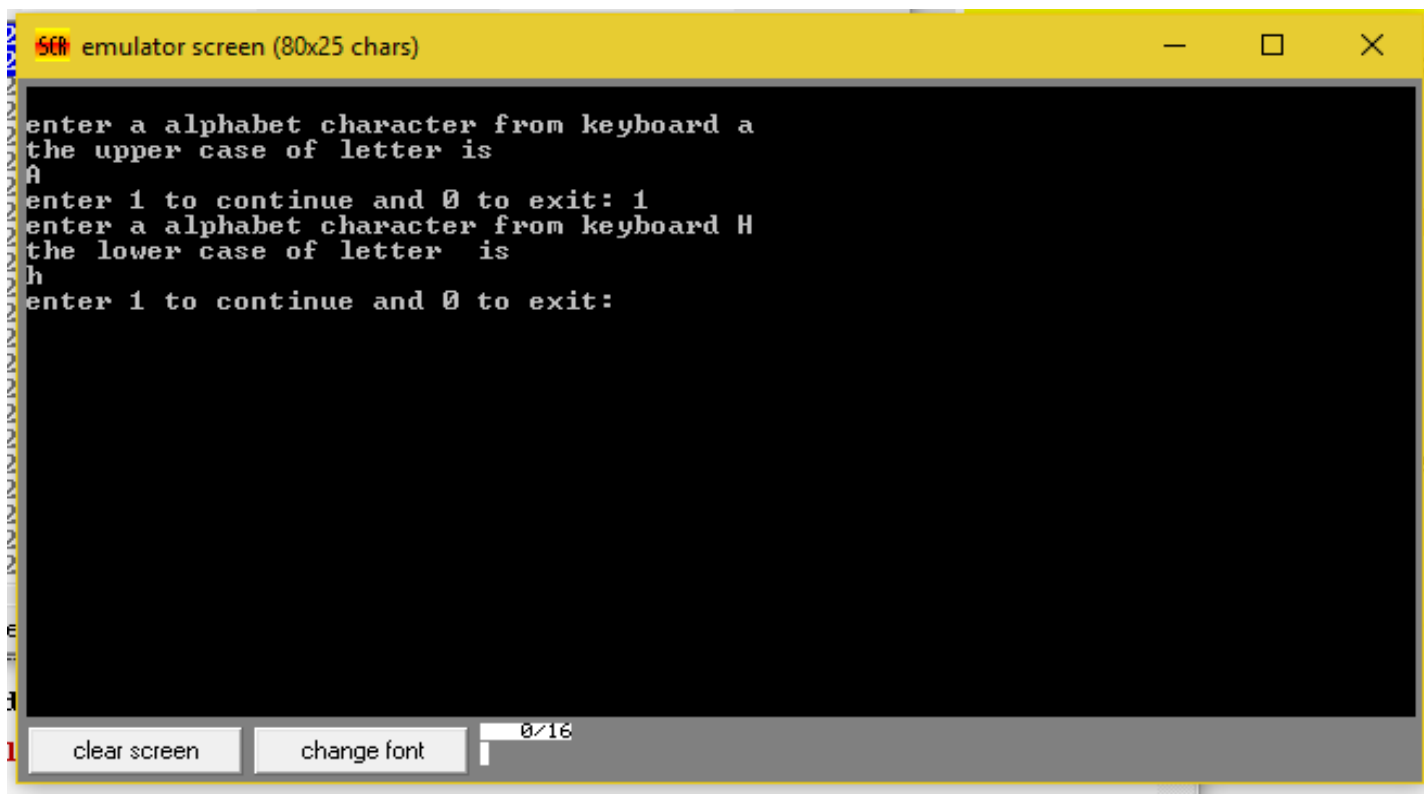
Add: It is used to add source in destination and store the result of computation in destination.

Add Destination, Source

.exit: command to exit the program.

OUTPUT:

Output screen looks like this when input is taken and required output is displayed so the above code works like this.



The screenshot shows a window titled "emulator screen (80x25 chars)" with a black background and white text. The text displayed is as follows:

```
enter a alphabet character from keyboard a
the upper case of letter is
A
enter 1 to continue and 0 to exit: 1
enter a alphabet character from keyboard H
the lower case of letter is
h
enter 1 to continue and 0 to exit:
```

At the bottom of the window, there are two buttons: "clear screen" and "change font". To the right of these buttons is a small text field showing "0/16".

Description:

In this home task we have to take input character from user and convert it in its opposite case .Upper to lower and lower to upper case. For this purpose I have used multiple labels and compared the hexadecimal ascii values of upper case and lower case with the input character and then generated the output character upper and lower case for lower and upper case respectively and then again prompt the screen to continue or exit as shown in output screen above.

LAB 5

Task 1 : Print 0 to 9 digits on screen without using loop and then by using loop

The description inside the “// ‘ ‘ //” are the comments to make the code more clear. These are not the part of program.

// PROGRAM //

org 100h

// First Print without using loop I am using DL register as a counter stored 0 in it and incrementing it in label l1 and repeating label l1 unless until DL is less than 10 and printing all the digits on screen //

MOV ah, 2

mov dl, 30h

// LABEL l1 //

l1:

```
int 21h  
add dl, 01h  
CMP dl, 39h  
JLE l1
```

```
// LABEL l2 //
```

```
// Now printing 0 to 9 digits using loop statement available in  
assembly language I have stored 10 in CX , counter register , and  
stored 0 in DL then looped the label l3 every time in l3 we print DL  
and increment DL which runs CXs time so the digit from 0 to 9 are  
printed on screen and then control moves to exit label //
```

```
l2:
```

```
mov CX, 10  
mov ah, 2  
mov dl, 30h  
jmp l3
```

```
// LABEL l3 //
```

```
// label to print 0 to 9 digits by a loop which repeats 10 times a label  
l3 for this purpose //
```

```
l3:
```

```
int 21h;
add dl, 01h
Loop l3
JE exit
// Label exit //
// To exit the program //
exit:
.exit
```

Commands:

Org 100 : set offset of the segment originated at 100hex.

Labels: these are the customized names of specific code segment to call that code segment anywhere in program by its name

LABEL_NAME:

Write the code here.

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler.

INT 21H

LEA: Load effective address of a memory block in a register it moves the starting address of a memory block in a register.

LEA Source, Destination

JMP: JMP command is used for unconditional jump on a label.

JMP LABEL_NAME

CMP: CMP is used to compare the contents of two register or variables and give a value in result if source is less than , greater than or equal to destination then -1, 1 or 0 is returned by CMP respectively.

CMP Source, Destination

JNE: in the result of CMP command this label will be executed if the contents of source and destination are not equal and by this we can jump on a label

JNE LABEL_NAME

JLE: In the result of CMP command this label will be executed if the contents of source is less or equal to destination by this we can jump on a label

JLE LABEL_NAME

JE: In the result of CMP command this label will be executed if the contents of source and destination are equal and by this we can jump on a label

JE LABEL_NAME

JL: In the result of CMP command this label will be executed if the contents of source is less than destination by this we can jump on a label

JL LABEL_NAME

JG: : In the result of CMP command this label will be executed if the contents of source is greater than destination by this we can jump on a label

JG LABEL_NAME

JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

JGE LABEL_NAME

Loop: it is used to repeat a label the content in CX register times

LABEL_NAME:

Loop LABEL_NAME

Sub: It is used to subtract destination from source and store the result of computation in destination.

Sub Destination, Source

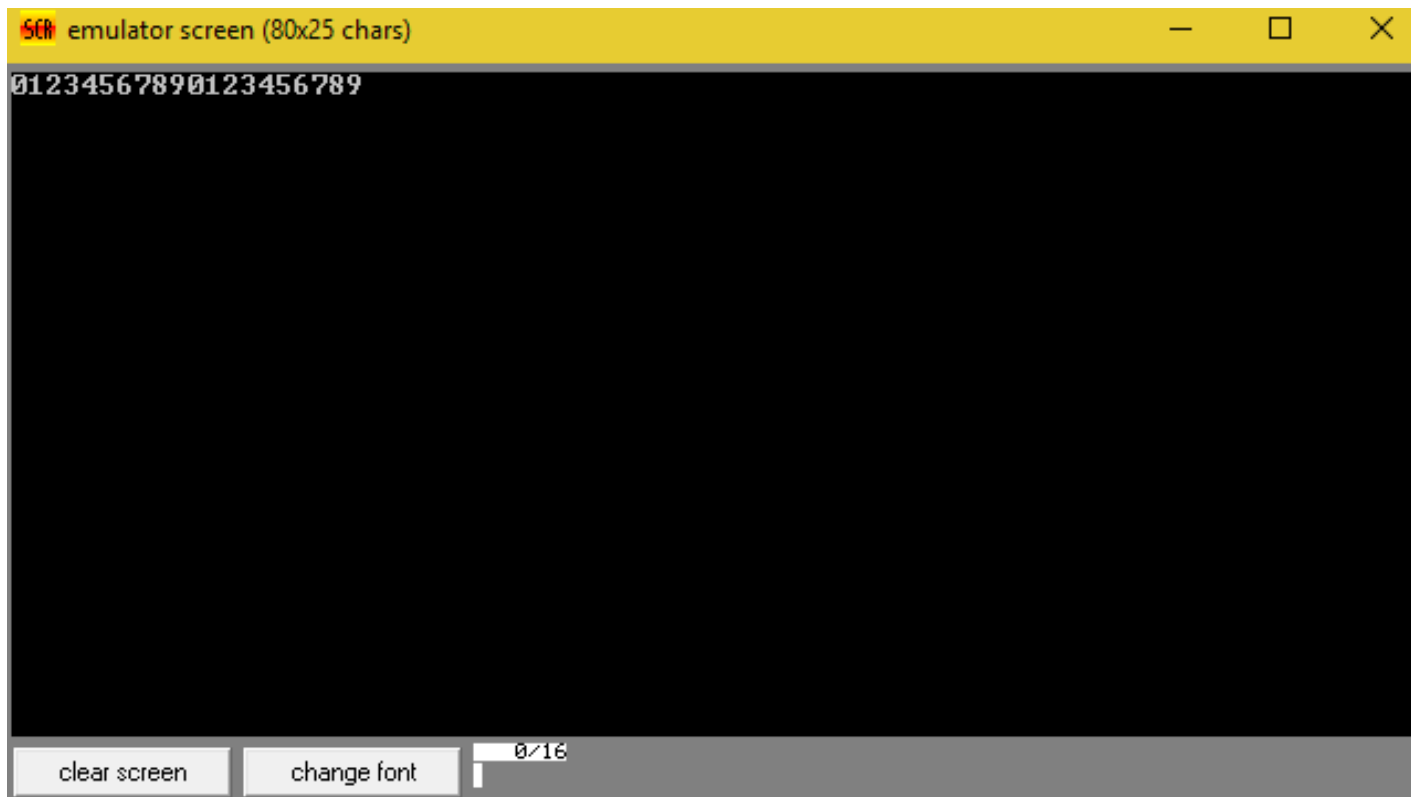
Add: It is used to add source in destination and store the result of computation in destination.

Add Destination, Source

.ret : Return statement to end the program.

Output :

The following output screen will appear after running the above code first 0 to 9 digits are printed without loop and after that 0 to 9 digits are printed with loop.



Description:

In this lab we are printing 0 to 9 digits without loop and then with loop. For this purpose I have used multiple labels first to print 0 to 9 digits without label I have used cmp command and a DL register as a counter and then for loop I have stored 10 in CX register which is default counter register and loop instruction runs CX times every time when loop runs there would be a decrement in CX and loop runs until CX becomes 0. So we have used this logic in our code we have run the loop 10 times and printed 0 to 9 digits on screen.

Task 2 : Take 3 single digit inputs from keyboard and sort them in ascending order

The description inside the “// ‘ ‘ // ” are the comments to make the code more clear. These are not the part of program.

// PROGRAM //

.model

// DATA SEGMENT //

// All the variable declaration and definition here //

.data

msg1 DB 10, 13, 'enter first number\$'

msg2 DB 10, 13, 'Enter 2nd number\$'

msg3 DB 10, 13, 'Enter 3rd number\$'

msg4 DB 10, 13, 'Before sorting\$'

msg5 DB 10, 13, 'After sorting\$'

msg6 DB 10, 13, 'enter 1 to continue 0 to exit\$'

// CODE SEGMENT//

// All the command here //

.code

```
mov AX, @data
mov DS, AX
// LABEL I1 //
// to take input of three numbers from user //
I1:
LEA DX, msg1
mov ah, 9
int 21h
mov ah, 1
int 21h
mov bl, al
LEA DX, msg2
mov ah, 9
int 21h
mov ah, 1
int 21h
mov bh, al
LEA DX, msg3
mov ah, 9
int 21h
```

```
mov ah, 1
int 21h
mov cl, al
LEA DX, msg4
mov ah, 9
int 21h
mov ah, 2
MOV DL, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, bl
int 21h
MOV DL, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, bh
int 21h
MOV DL, 0ah
```

int 21h

mov dl, 0dh

int 21h

mov dl, cl

int 21h

jmp l2

// LABEL l2 //

// to compare 1st and 2nd digit and jump on next to either swap or to move on next comparison //

l2:

CMP bl, bh

JG l3

JLE l4

// LABEL l3 //

// To swap the contents of 1st and 2nd number if 1st number is greater than the 2nd one and then jump on next label for next comparison //

l3:

mov al, bl

mov bl, bh

mov bh, al

Jmp I4

// LABEL I4 //

// to compare 1st and 3rd digit and jump on next to either swap or to move on next comparison //

I4:

Cmp bl, cl

JG I5

JLE I6

// LABEL I5 //

// To swap the contents of 1st and 3rd number if 1st number is greater than the 3rd one and then jump on next label for next comparison //

I5:

mov al, bl

mov bl, cl

mov cl, al

Jmp I6

// LABEL I6 //

// to compare 2nd and 3rd digit and jump on next to either swap or to move on next label //

I6:

CMP bh, cl

JG l7

JLE l8

// LABEL l7 //

// To swap the contents of 2nd and 3rd number if 2nd number is greater than the 3rd one and then jump on next label for next comparison //

l7:

mov al, bh

mov bh, cl

mov cl, al

jmp l8

// LABEL l8 //

// To print the sorted numbers on screen one by one at the start of every line and then jump on next label to prompt for next value or exit the program //

l8:

LEA DX, msg5

mov ah, 9

int 21h

```
mov ah, 2
MOV DL, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, bl
int 21h
MOV DL, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, bh
int 21h
MOV DL, 0ah
int 21h
mov dl, 0dh
int 21h
mov dl, cl
int 21h
JMP I9
```


// LABEL I9 //

**// To prompt the screen for user to continue for next values or just
exit the program by just jump on 1st label or exit label respectively //**

I9:

LEA DX, msg6

mov ah, 9

int 21h

mov ah, 1

int 21h

CMP al, 30h

JNE I1

JE exit

// LABEL exit//

// exit the program//

exit:

.Exit

Commands:

.model: it is used to tell the assembler that program is in protected mode.

.code: it contains all the codes and commands.

.data: it contains all the definitions and declarations of variables.

Labels: these are the customized names of specific code segment to call that code segment anywhere in program by its name

LABEL_NAME:

Write the code here.

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler.

INT 21H

LEA: Load effective address of a memory block in a register it moves the starting address of a memory block in a register.

LEA Source, Destination

JMP: JMP command is used for unconditional jump on a label.

JMP LABEL_NAME

CMP: CMP is used to compare the contents of two register or variables and give a value in result if source is less than , greater than or equal to destination then -1, 1 or 0 is returned by CMP respectively.

CMP Source, Destination

JNE: in the result of CMP command this label will be executed if the contents of source and destination are not equal and by this we can jump on a label

JNE LABEL_NAME

JLE: In the result of CMP command this label will be executed if the contents of source is less or equal to destination by this we can jump on a label

JLE LABEL_NAME

JE: In the result of CMP command this label will be executed if the contents of source and destination are equal and by this we can jump on a label

JE LABEL_NAME

JL: In the result of CMP command this label will be executed if the contents of source is less than destination by this we can jump on a label

JL LABEL_NAME

JG: : In the result of CMP command this label will be executed if the contents of source is greater than destination by this we can jump on a label

JG LABEL_NAME

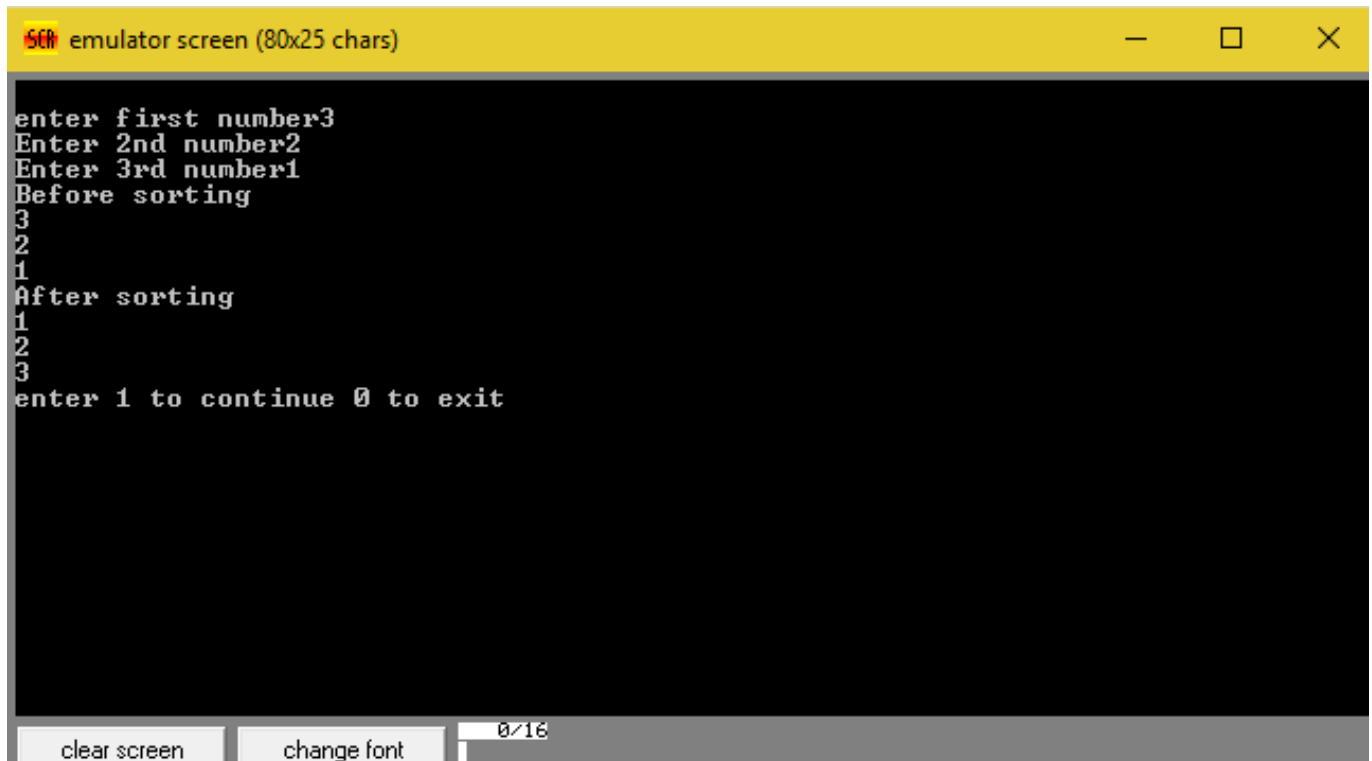
JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

JGE LABEL_NAME

.exit: Command to exit the program.

OUTPUT :

The following output screen will be shown when the above code is run on 3, 2 and 1 as input digits.



```
emulator screen (80x25 chars)
enter first number3
Enter 2nd number2
Enter 3rd number1
Before sorting
3
2
1
After sorting
1
2
3
enter 1 to continue 0 to exit
```

Description:

In this lab task we have to take 3 digits as input and sort them in ascending order for this purpose I have used multiple labels and the rule of compare and swap. In this rule we compare two digits and if the first digit is greater than the 2nd digit we swap them this rule is in case of ascending order sorting. For the sorting of two digits apply the rule on 1st digit and 2nd digit and then 1st and 3rd and then move to 2nd digit and apply the same rule on 2nd and 3rd digit after that ultimately all the digits are sorted in ascending order. We have stored multiple string variables to display on screen in data segment and displayed

them according to the situation in labels where required. And finally the sorted digits on screen will be displayed.

LAB 6

Task: Take a string from data segment and display vowels in it and count number of vowel in it and display count on screen

The description inside the “// ‘ ‘ // ” are the comments to make the code more clear. These are not the part of program.

```
// PROGRAM //
```

```
// Libraries to include //
```

```
.model
```

```
.stack
```

```
// DATA SEGMENT //
```

```
.data
```

```
str DB 'abcdeiighiklohoyekoe$'
```

```
counter DB 0
```

```
// CODE SEGMENT //
```

```
.CODE
```

```
// loads the starting address of string in BX register//
```

```
mov ax, @data
```

```
mov ds, ax
```

```
lea bx, str
```

```
mov dh, 0h
```

```
mov ah, 2h
```

```
mov CX, 40h
```

```
// LABEL I1 //
```

```
// Reads the string and jump on specific label //
```

```
I1:
```

```
mov al, [bx]
```

```
INC bx
```

```
CMP al, 'a'
```

```
JE I3
```

```
CMP al, 'e'
```

```
je I3
```

```
CMP al, 'i'
```

```
je I3
```

```
CMP al, 'o'
```

```
je I3
```

```
CMP al, 'u'
```

```
je I3
```

```
CMP al, '$'
```


JE l4

JNE l1

// LABEL l3 //

// It prints a new line and the character in AL register on screen//

l3:

inc DH

mov CL, AL

mov DL, 0ah

int 21h

mov DL, 0dh

int 21h

mov DL, CL

int 21h

inc counter

jmp l1

// LABEL l4 //

**// prints number of vowels, count, on screen and then jumps on exit
label //**

l4:

```
mov dl, 0ah
int 21h
mov dl, 0dh
int 21h
mov al, 00h
mov ah, 00h
mov al, counter
mov bl, 10
div bl
mov bl, al
mov bh, ah
mov ah, 2
mov dl, bl
add dl, 30h
int 21h
mov dl, bh
add dl, 30h
int 21h
JMP exit
// LABEL exit //
```

exit:

.exit

// END//

Commands:

.model: it is used to tell the assembler that program is in protected mode.

.code: it contains all the codes and commands.

.data: it contains all the definitions and declarations of variables.

Labels: these are the customized names of specific code segment to call that code segment anywhere in program by its name

LABEL_NAME:

Write the code here.

MOV: Moves the contents of source instruction in destination and the syntax is as follows:

Mov Destination, Source

INT 21H: Call to DOS interrupt Handler.

INT 21H

INC : unary operator increment, used to add 1 in the contents of source.

INC Source

LEA: Load effective address of a memory block in a register it moves the starting address of a memory block in a register.

LEA Source, Destination

JMP: JMP command is used for unconditional jump on a label.

JMP LABEL_NAME

CMP: CMP is used to compare the contents of two register or variables and give a value in result if source is less than , greater than or equal to destination then -1, 1 or 0 is returned by CMP respectively.

CMP Source, Destination

JNE: in the result of CMP command this label will be executed if the contents of source and destination are not equal and by this we can jump on a label

JNE LABEL_NAME

JLE: In the result of CMP command this label will be executed if the contents of source is less or equal to destination by this we can jump on a label

JLE LABEL_NAME

JE: In the result of CMP command this label will be executed if the contents of source and destination are equal and by this we can jump on a label

JE LABEL_NAME

JL: In the result of CMP command this label will be executed if the contents of source is less than destination by this we can jump on a label

JL LABEL_NAME

JG: : In the result of CMP command this label will be executed if the contents of source is greater than destination by this we can jump on a label

JG LABEL_NAME

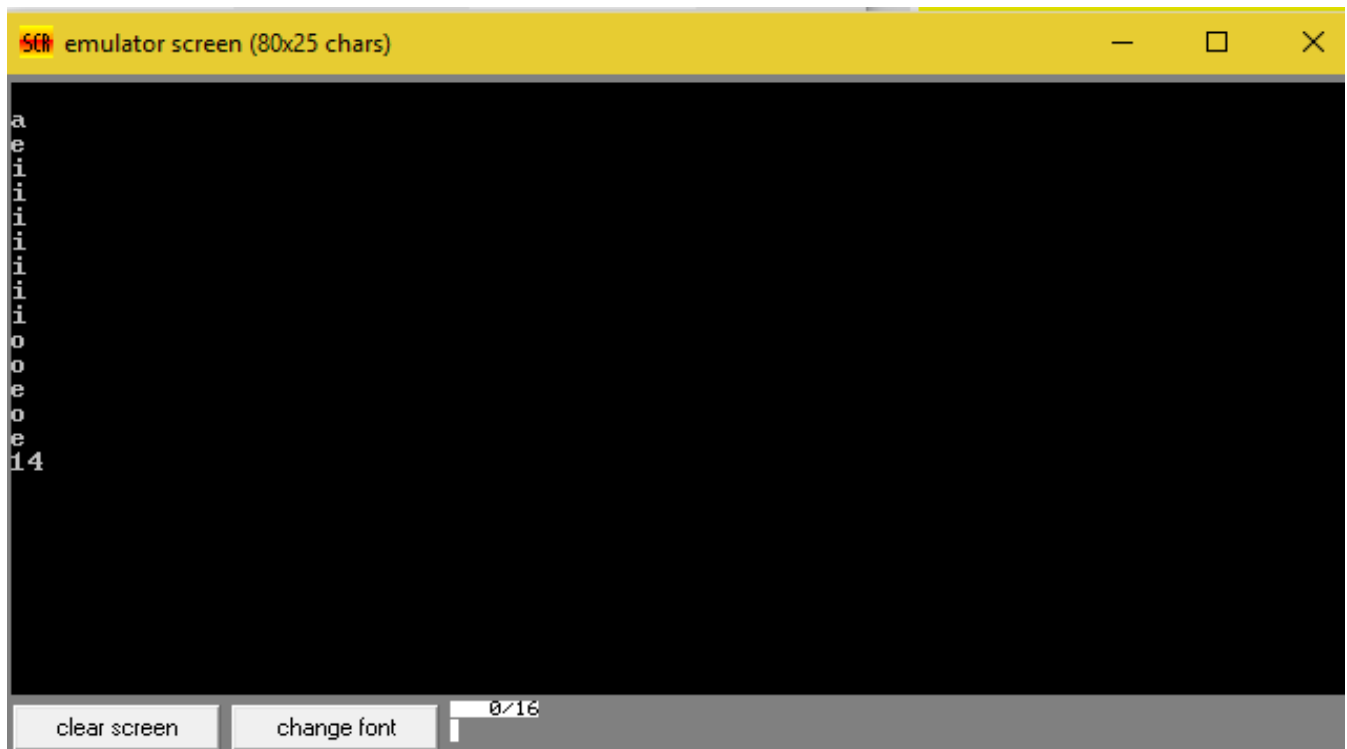
JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

JGE LABEL_NAME

.exit: Command to exit the program.

OUTPUT :

The following output screen will be displayed for 'abcdeiighiklohoeyekoe\$' string all the vowels in it are printed one by one and the number of vowels are also printed.



```
a
e
i
i
i
i
i
i
i
o
o
e
e
e
14
```

Description:

In this lab we have to take a string from data segment suppose str and read it character by character and print the vowels in it and count the number of vowels and print the count on screen. For this purpose I have used multiple labels and CMP statements to compare vowels and jump on different labels to display the vowel and increment the count and when the string is finished the count will be display to display the count we

have used div statement to get its quotient and remainder if it is a two digit number as shown in example string we divide it by 10 and then display the quotient and remainder on screen using div command we separate its parts and then print them one by one.