COAL LAB MANUAL

Submitted to:

Dr. Tauqir

Teacher assistant : Sir Shahzad

Submitted By:

Komal Shehzadi

Registeration 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

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

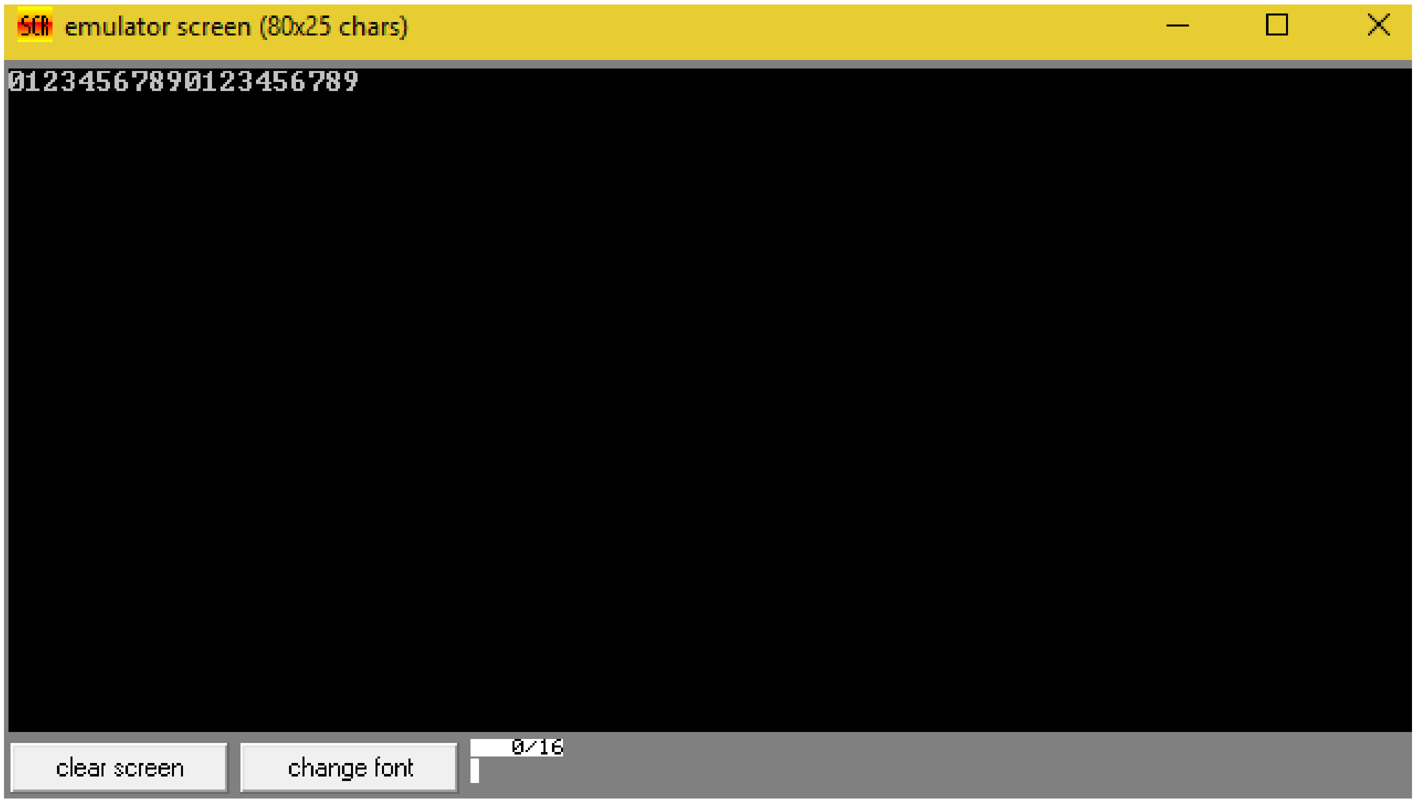
Add: It is used to add source in destination and store the result of computation in destination.

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.



```
01234567890123456789
```

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 l1 //

// to take input of three numbers from user //

l1:

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 l4

// LABEL l4 //

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

l4:

Cmp bl, cl

JG l5

JLE l6

// LABEL l5 //

// 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 //

l5:

mov al, bl

mov bl, cl

mov cl, al

Jmp l6

// LABEL l6 //

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

l6:

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 l9
```

// LABEL l9 //

// 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 //

```
l9:

LEA DX, msg6

mov ah, 9

int 21h

mov ah, 1

int 21h

CMP al, 30h
```

```
JNE l1

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
```

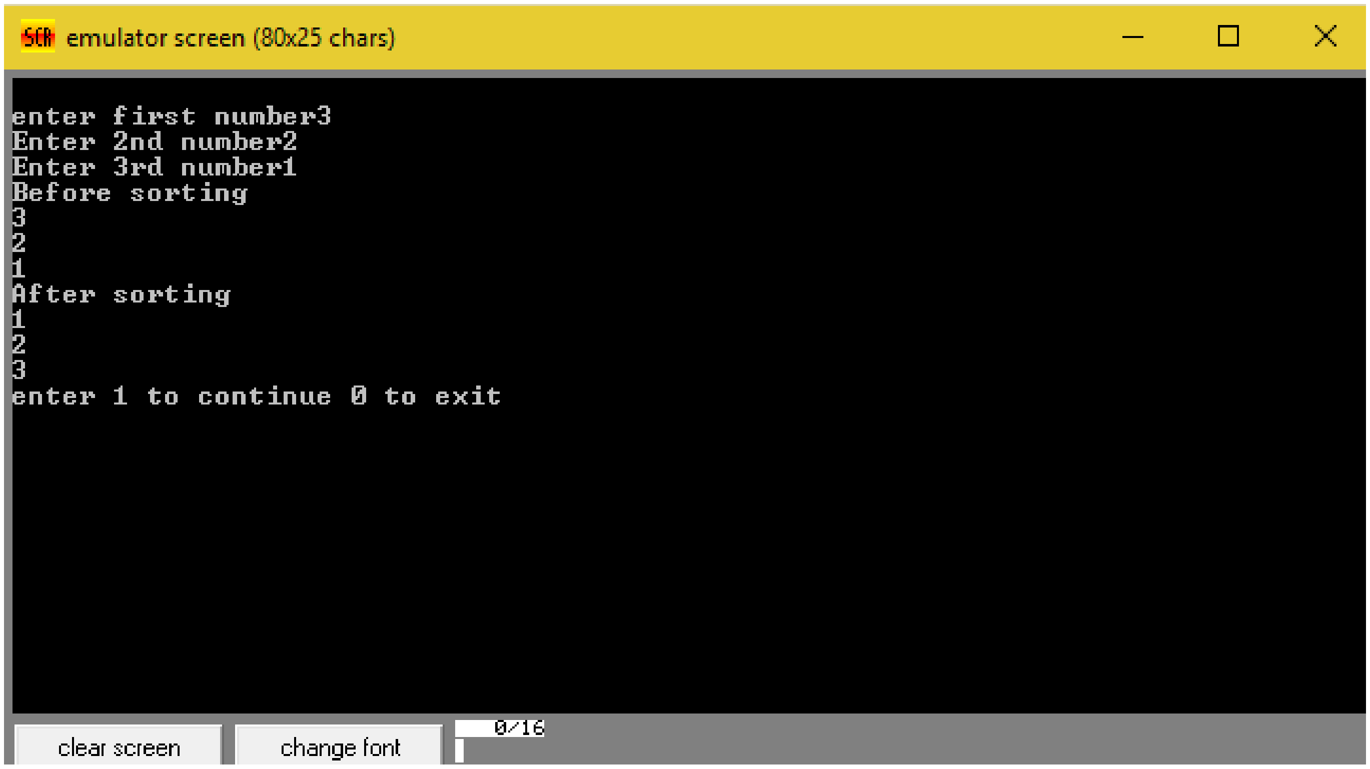
JGE: In the result of CMP command this label will be executed if the contents of source is greater or equal to destination by this we can jump on a label

```
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.



```
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.