Name:	Shagun Upadhyay
Roll No:	60
Class/Sem:	SE/IV
Experiment No.:	10
Title:	Program for printing the string using procedure and macro.
Date of Performance:	12/04/2024
Date of Submission:	12/04/2024
Marks:	
Sign of Faculty:	

Aim: Program for printing the string using procedure and macro.

Theory:

Procedures:-

• Procedures are used for large group of instructions to be repeated.

• Object code generated only once. Length of the object file is less the memory

• CALL and RET instructions are used to call procedure and return from procedure.

• More time required for its execution.

• Procedure Can be defined as:

```
Procedure_name PROC
……
……
Procedure_name ENDP
```

Example:

```
Addition PROC near
……
……
Addition ENDP
```

Macro:-

• Macro is used for small group of instructions to be repeated.

• Object code is generated every time the macro is called.

• Object file becomes very lengthy.

• Macro can be called just by writing.

• Directives MACRO and ENDM are used for defining macro.

• Less time required for its execution.

• Macro can be defined as:

```
Macro_name MACRO [Argument, .... , Argument N]
......
......
ENDM
```

Example:-

```
Display MACRO msg
.....
.....
ENDM
```

Code:

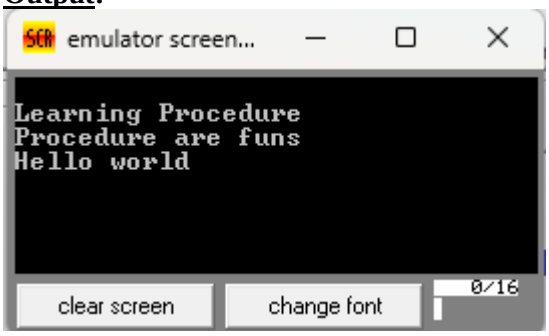
```
org 100h

   print macro p1
      lea dx,p1
      mov ah,09h
      int 21h
   endm
   .data
   m1 db 10,13,"Leaning Macro$"
   m2 db 10,13,"Macros are fun$"
   m3 db 10,13,"Hello World$"
   .code
   print m1
   print m2
   print m3

ret
```

Output:

Code:

```
org 100h

.data
msg1 db 10,13,'Learning Procedure$'
msg2 db 10,13,'Procedure are funs$'
msg3 db 10,13,'Hello world$'

.code

lea dx, msg1
call print

lea dx, msg2
call print

lea dx, msg3
call print
mov ah, 4CH
int 21h


print PROC
mov ah,09h
int 21h
ret
print ENDP

ret
```

Output:



Conclusion:

In this experiment, we learn that employing both procedures and macros to print strings in assembly language yields distinct advantages. Procedures offer structured, reusable blocks of code, enhancing clarity and facilitating debugging. On the other hand, macros provide compile-time expansion and customization options, though they may pose challenges in debugging due to inline expansion. The choice between procedures and macros depends on the specific needs of the program, with procedures favored for modularity and ease of debugging, while macros excel in flexibility and customization.

1. Differentiate between procedure and macros.

	Procedure	**Macro**
	Defined using a set of instructions or operations	Defined using a set of instructions or operations that can be expanded inline

	Called and executed at runtime	Expanded at compile-time
	Can take parameters/arguments	Can take parameters/arguments
	Has its own scope, can access global variables	Operates within the scope of the code in which it's used
	Called like a function	Expanded inline wherever it's invoked

2. Explain CALL and RET instructions.

The CALL and RET instructions are fundamental to subroutine (or function) invocation and return in assembly language programming. Here's an explanation of each:

CALL Instruction:

- The CALL instruction is used to transfer control to a subroutine or function. It allows the program to jump to a specific memory address where the subroutine begins.
- When the CALL instruction is executed, the address of the instruction following the CALL (the return address) is pushed onto the stack. This allows the program to return to the correct location after the subroutine completes its execution.
- Syntax: CALL destination
- Example: CALL subroutine_addressCALL subroutine_addr
- After executing the CALL instruction, the control flow of the program jumps to the specified subroutine.

RET Instruction:

- The RET instruction is used to return control from a subroutine back to the main program or to the calling code.
- When the RET instruction is executed, it pops the return address from the top of the stack and jumps to that address, resuming execution from the instruction following the original CALL.
- Syntax: RET
- Example: RET
- The RET instruction typically does not take any operands. It simply retrieves the return address from the stack and jumps to that address.