Name:	Ketan N Mahadik
Roll No:	24
Class/Sem:	SE/IV
Experiment No.:	10
Title:	Program for printing the string using procedure and macro.
Date of Performance:	01/04/24
Date of Submission:	08/04/24
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:

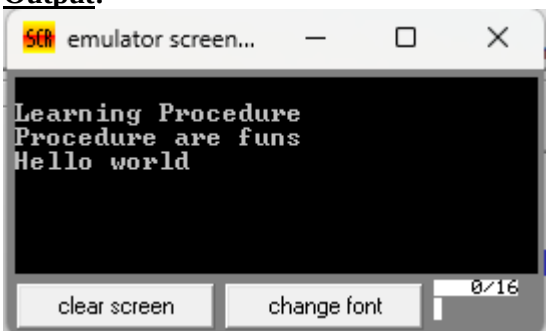
```
org 100h

   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.

Ans: Procedures and macros serve similar purposes in programming, but they have distinct characteristics:

1. Procedures:
- Procedures are reusable blocks of code that perform specific tasks and can be called from multiple places within a program.
- They are typically defined using a specific programming construct, such as a function or subroutine, and follow a specific calling convention for parameter passing and return values.
- Procedures provide encapsulation, allowing the programmer to organize code into manageable and modular units, enhancing readability, maintainability, and code reuse.

2. Macros:
- Macros are preprocessor directives that define a sequence of instructions or expressions which are substituted into the code wherever the macro is invoked.
- They are expanded inline during the preprocessing phase of compilation, effectively replacing the macro invocation with the defined code before the actual compilation process begins.
- Macros are often used for code generation, repetitive tasks, or defining shorthand notations, providing a convenient way to customize and extend the language syntax without modifying the compiler or interpreter.

In summary, procedures are reusable code blocks that provide encapsulation and follow a specific calling convention, while macros are preprocessor directives that are expanded inline during compilation and are primarily used for code generation and customization of the language syntax.

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.