Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL404**	**Course Name:**	**Microprocessor Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**10**
Title of the Experiment:	**Program for printing the string using procedure and macros**
Date of Performance:	**27/03/2025**
Date of Submission:	**03/04/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Ms. Sweety Patil

Signature :

Date:

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

Theory:

Procedure

A procedure is a reusable sequence of instructions that can be called from a program to perform a specific task. Procedures are defined using the PROC and ENDP directives, and can be called using the CALL instruction. Procedures are important for input-output in assembly language.

- 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

A macro is a named block of assembly language statements that can be invoked multiple times in a program. Macros are defined using the MACRO and ENDM directives. When a macro is invoked, a copy of its code is inserted into the program at the point of invocation. This process is known as inline expansion. Macros are faster to execute than procedures because they don't use the stack.

- Macro is used for a 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
```

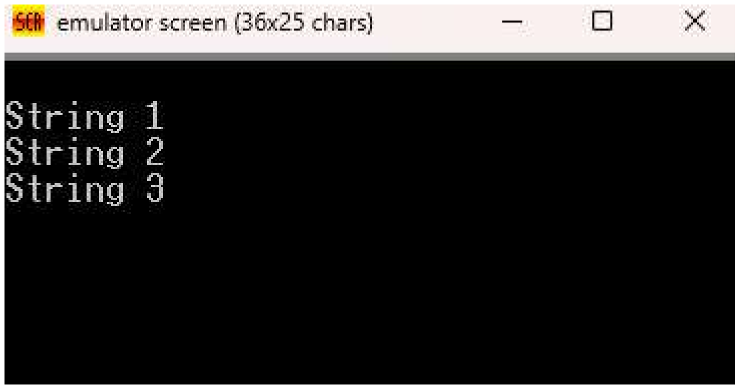
Code:

```
org 100h

.DATA
 M1 DB 10, 13,"String 1$"
 M2 DB 10, 13,"String 2$"
 M3 DB 10, 13,"String 3$"

 .CODE
 LEA DX, M1
 CALL PRINT

 LEA DX, M2
 CALL PRINT

 LEA DX, M3
 CALL PRINT
 ret


 PRINT PROC
    MOV AH, 09H
    INT 21H
    RET
    PRINT ENDP
```

MACRO

```
org 100h
PRINT MACRO
    MOV AH, 09H
    INT 21H
    ENDM

.DATA
M1 DB 10, 13,"String 1$"
M2 DB 10, 13,"String 2$"
M3 DB 10, 13,"String 3$"

.CODE

LEA DX, M1
PRINT

LEA DX, M2
PRINT

LEA DX, M3
PRINT

RET
```

Output:



```
String 1
String 2
String 3
```

Conclusion:
In this program, both procedures and macros are used to print strings. Procedures allow for reusable blocks of code that are called using the `CALL` instruction, saving memory and reducing code redundancy. However, they incur more execution time due to the need for stack operations. In contrast, macros provide inline code expansion, making them faster to execute but generating larger object files. While procedures are more efficient for complex tasks requiring multiple instructions, macros are better suited for small, repeated operations due to their faster execution and simplicity.

1. Differentiate between procedure and macros.

Aspect	Procedure	Macro
Definition	A reusable block of code defined with `PROC` and `ENDP`.	A block of code expanded inline using `MACRO` and `ENDM`.
Execution	Executes at runtime via `CALL` and `RET`.	Expands at assembly time.
Code Generation	Object code generated once.	Object code generated every time the macro is invoked.
Speed	Slower due to stack operations.	Faster due to inline expansion.
Memory Usage	Uses less memory.	Uses more memory due to repeated code.
Use Case	For complex, large instructions.	For small, frequent code repetitions.
Flexibility	More flexible, can accept arguments and return values.	Less flexible, does not return values.

2. Explain CALL and RET instructions.

CALL
Purpose: Invokes a subroutine by transferring control to a specified address and pushes the return address onto the stack, allowing the program to resume execution after the subroutine completes.

RET
Purpose: Returns control from a subroutine to the calling code by popping the return address from the stack, enabling the program to continue execution from the point immediately following the CALL instruction.