PROCEDURES

Procedures and Macros:

- When we need to use a group of instructions several times throughout a program there are two ways we can avoid having to write the group of instructions each time we want to use them.

1. One way is to write the group of instructions as a separate **procedure.**

2. Another way we can use **macros.**

Procedures:

- The procedure is a group of instructions stored as a separate program in the memory and it is called from the main program whenever required using CALL instruction.
- For calling the procedure we have to store the return address (next instruction address followed by CALL) onto the stack.
- At the end of the procedure RET instruction used to return the execution to the next instruction in the main program by retrieving the address from the top of the stack.

PROCEDURES

- Machine codes for the procedure instructions put only once in memory.
- The procedure can be defined anywhere in the program using assembly directives PROC and ENDP.

The four major ways of passing parameters to and from a procedure are:

1. In registers
2. In dedicated memory location accessed by name
3. With pointers passed in registers
4. With the stack

The type of procedure depends on where the procedure is stored in the memory.

- If it is in the same code segment where the main program is stored the it is called near procedure otherwise it is referred to as far procedure.

PROCEDURES

- For near procedure CALL instruction pushes only the IP register contents on the stack, since CS register contents remains unchanged for main program.
- But for Far procedure CALL instruction pushes both IP and CS on the stack.

Syntax:

Procedure name PROC near

instruction 1

instruction 2

RET

Procedure name ENDP

Example:

near procedure:

```
ADD2 PROC near
ADD AX,BX
RET        ADD2 ENDP
```

PROCEDURES

far procedure:

Procedures segment

Assume CS : Procedures

ADD2 PROC far

ADD AX,BX

RET ADD2 ENDP

Procedures ends

Depending on the characteristics the procedures are two types

1. Re-entrant Procedures

2. Recursive Procedures

Reentrant Procedures: The procedure which can be interrupted, used and "reentered" without losing or writing over anything.

Recursive Procedure: A recursive procedure is procedure which calls itself.

Macros:

- A **macro is a group of repetitive instructions in a program which are codified only once and can be used as many** times as necessary.
- A macro can be defined anywhere in program using the directives **MACRO and ENDM**
- Each time we call the macro in a program, the assembler will insert the defined group of instructions in place of the call.
- The assembler generates machine codes for the group of instructions each time the macro is called.
- Using a macro avoids the overhead time involved in calling and returning from a procedure.

Syntax of macro:

macroname MACRO

instruction1

Instruction2 ENDM

PROCEDURES

Example#1:

```
Read MACRO
mov ah,01h
int 21h
ENDM
```

Example#2:

```
Display MACRO
mov dl, al
Mov ah,02h
int 21h
ENDM
```

PROCEDURES

Advantage of Procedure and Macros:

Procedures:

Advantages: The machine codes for the group of instructions in the procedure only have to be put once.

Disadvantages

- Need for stack
- Overhead time required to call the procedure and return to the calling program.

Macros:

Advantages: Macro avoids overhead time involving in calling and returning from a procedure.

Disadvantages: Generating in line code each time a macro is called is that this will make the program take up more memory than using a procedure.

PROCEDURES

Differences between Procedures and Macros:

PROCEDURES	MACROS
Accessed by CALL and RET mechanism during program execution	Accessed by name given to macro when defined during assembly
Machine code for instructions only put in memory once	Machine code generated for instructions each time called
Parameters are passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro
Procedures uses stack	Macro does not utilize stack
A procedure can be defined anywhere in program using the directives PROC and ENDP	A macro can be defined anywhere in program using the directives MACRO and ENDM
Procedures takes huge memory for CALL (3 bytes each time CALL is used) instruction	Length of code is very huge if macro's are called for more number of times

PROCEDURES

- Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size.
- Procedures are identified by a name.
- Following this name, the body of the procedure is described which performs a well-defined job.
- End of the procedure is indicated by a return statement.
- Syntax:

```
proc_name:
    procedure body
    ...
    ret
```

PROCEDURES

- The procedure is called from another function by using the CALL instruction.
- The CALL instruction should have the name of the called procedure as an argument.
- Syntax:
  - CALL proc_name

PROCEDURE Example

- A program to Add numbers and return their sum

```
section .text
    global _start
_start:
mov ecx,'4'
 sub    ecx, '0'
 mov   edx, '5'
 sub    edx, '0'
 call   sum      ;call sum procedure
 mov   [res], eax
 mov ecx, msg
```

PROCEDURE Example

```
mov edx, len
mov ebx,1          ;file descriptor (stdout)
mov eax,4          ;system call number (sys_write)
int 0x80           ;call kernel

mov ecx, res
mov edx, 1
mov ebx, 1         ;file descriptor (stdout)
mov eax, 4         ;system call number (sys_write)
int 0x80           ;call kernel

mov eax,1          ;system call number (sys_exit)
int 0x80           ;call kernel
```

PROCEDURE Example

```
sum:
    mov     eax, ecx
    add     eax, edx
    add     eax, '0'
    ret


section .data
msg db "The sum is:", 0xA,0xD
len equ $- msg
segment .bss
res resb 1
```