Gramin Shikshan Prasarak Mandal's

GRAMIN POLYTECHNIC

(ISO 9001:2008 Certified Institute)

Chapter 5- Procedure and Macro

By

Shinde G. B.

M.Tech. (Electronics Engineering)

5.1 Procedure: defining and calling procedure-PROC,ENDP ,FAR and NEAR directives; CALL and RET instructions,parametres passing methods, assembly language programs using procedure.
5.2 Macro: defining macros, MACRO and ENDM directives, MACRO with parameters, assembly language programs using MACROS

Unit Outcomes

- Apply the relevant parameters passing method in the given situation.
- Develop an assembly language program using the relevant procedure for the given problem.
- Develop an assembly language program using MACROS for the given situation.
- Compare procedure and macro on the basis of the given parametres.

Procedure:

- A large program is divided into number of independent tasks which can be easily designed and implemented .
- This process of splitting is called as modular programming.
- The repeated group of instructions in a program can be organized as subprogram.
- This program is called as subroutine or procedures in ALP.
- That is procedure is a set of program statements that can be processed independently and which are used again and again.

1. Save return address
2. Procedure call
3. Execute procedure.
4. Return.

- **Advantages:**

1. Simple modular programming.
2. Reduces work and time.
3. overall Size reduction of code
4. Reuse of procedure many times.

Directives for procedure.

- PROC
- ENDP

PROC directive

- It indicates the beginning of a procedure and follows with the name of the procedure.
- Procedure may be NEAR or FAR
- The terms NEAR or FAR follows the PROC directive indicating the type of the procedure.
- If it is not mentioned assembler assumes NEAR as a type specifier.
- The directive PROC is used with the directive ENDP to enclose the procedure code.

- Procedure name PROC [NEAR/FAR]

Example:

```
ADD  PROC     NEAR
       -------
       -------
       -------
       -------
ADD ENDP
```

- Procedure can be called by using CALL instruction.

ENDP:

This directive informs the assembler that end of the procedure.

General form:

procedure_name ENDP

Procedure CALL

- The call instruction is used to transfer program control to a subprogram or a procedure by storing the return address on the stack.
- The call can be near call(inter-segment) or far call(intra-segment).
- Near call refers to a procedure call which is in the same code segment
- Far call refers to a procedure call which is in the different code segment.
- Syntax: CALL procedure_name

Procedure return(RET):

- The instruction return is used to transfer program control from the procedure back to the calling program(main program)
- It may be NEAR RET(inter segment return) or FAR RET(intra segment return)
- Syntax: RET

Write an ALP to perform arithmetic operations using procedure.

```
.model small
.data
Num1 db 10h
Num2 db 20h
Add db ?
Sub db ?
.code
Mov ax,@data
Mov ds,ax
Call addition       ; call procedure for addition
Call subtraction    ; call procedure for subtraction
```

```
Addition proc        ; procedure for addition
Mov ax,num1
Add ax,num2
Mov add,ax
Ret                  ;return from procedure
Endp
subtraction proc     ; procedure for subtraction
Mov ax,num1
Sub ax,num2
Mov sub,ax
Ret                  ;return from procedure
Endp                 ; end of the procedure
 ends
```

Write an ALP to perform arithmetic operations using procedure.

```
.model small
.data
Num1 db 10h
Num2 db 20h
Multi  ?
division db ?
.code
Mov ax,@data
Mov ds,ax
Call addition        ; call procedure for addition
Call subtraction   ; call procedure for subtraction
```

```
Addition proc        ; procedure for addition
Mov ax,num1
Add ax,num2
Mov add,ax
Ret                   ;return from procedure
Endp
subtraction proc     ; procedure for subtraction
Mov ax,num1
Sub ax,num2
Mov sub,ax
Ret                   ;return from procedure
```

Write an ALP to add two BCD numbers using procedure

```
.model small
.data
Num1 db 50h
Num2 db 30h
result db ?
carry db ?
.code
Mov ax,@data
Mov ds,ax
Call addition
Ends
end
```

```
Addition proc
mov al,num1
Add al,num2
Daa
Jnc dn
Inc carry
Dn:
mov result,al
Ret
endp
```

Write an ALP to find smallest number from an array using procedure.

```
.model small
.data
        Array dw 50h,30h,60h,70,20h
        Small db 0
.code
        mov ax,@data
        mov ds,ax
        Call smallest   ;call procedure to find smallest
        mov small,ax
        mov ah,4ch
```

```
Smallest proc
        mov cx,5
        mov si,offset array
        mov ax,[si]
        Dec cx
Up:
        Inc si
        cmp ax,[si]
        Jc next
        mov ax,[si]
Next:
        Loop up
        ret
        endp
        ends
```

Write an ALP for sum of series of 10 numbers using procedure.

```
.model small
.data
        Num db  1,2,3,4,5,6,7,8,9,10
        Result db 0
        Count db 10
 .code
      Mov ax,@data
      Mov ds,ax
      Call sum
```

```
Sum proc
        Mov cl,count
        Mov si,offset num
        Mov al,0
   up: Add al,[si]
        Inc si
        Dec cl
        Jnz up
        Ret
        Endp
```

Program using procedure for the given operation z=(A+B)*(C+D)

Algorithm:

- Initialize data segment
- Load register al with a
- Load register bl with b
- Call procedure for addition
- Store result for addition.
- Load register al with c
- Load register bl with d
- Call procedure for addition
- Multiply result of 2 sum
- Store final result

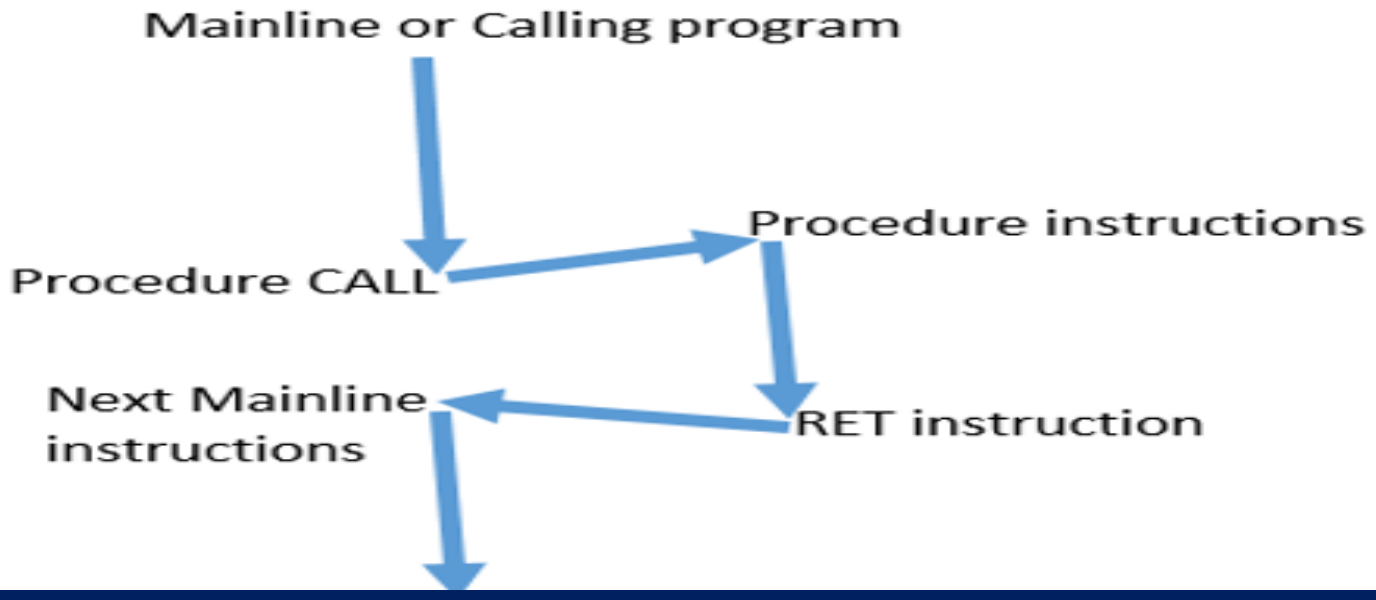
```
.model small
.data
 a db 1
 b db 2
 c db 3
 d db 4
 Sum1 db ?
 Sum2 dw ?
.code
 Mov ax,@data
 Mov ds,ax
 Mov al,a
 Mov bl,b
```

```
Call byte
        Mov sum1,al
        Mov al,c
        Mov bl,d
Call byte
        Mul sum1
        Mov z,ax
Byte proc
        Add al,bl
        Ret
        endp
        ends
        end
```

Recursive and Re-entrant procedure

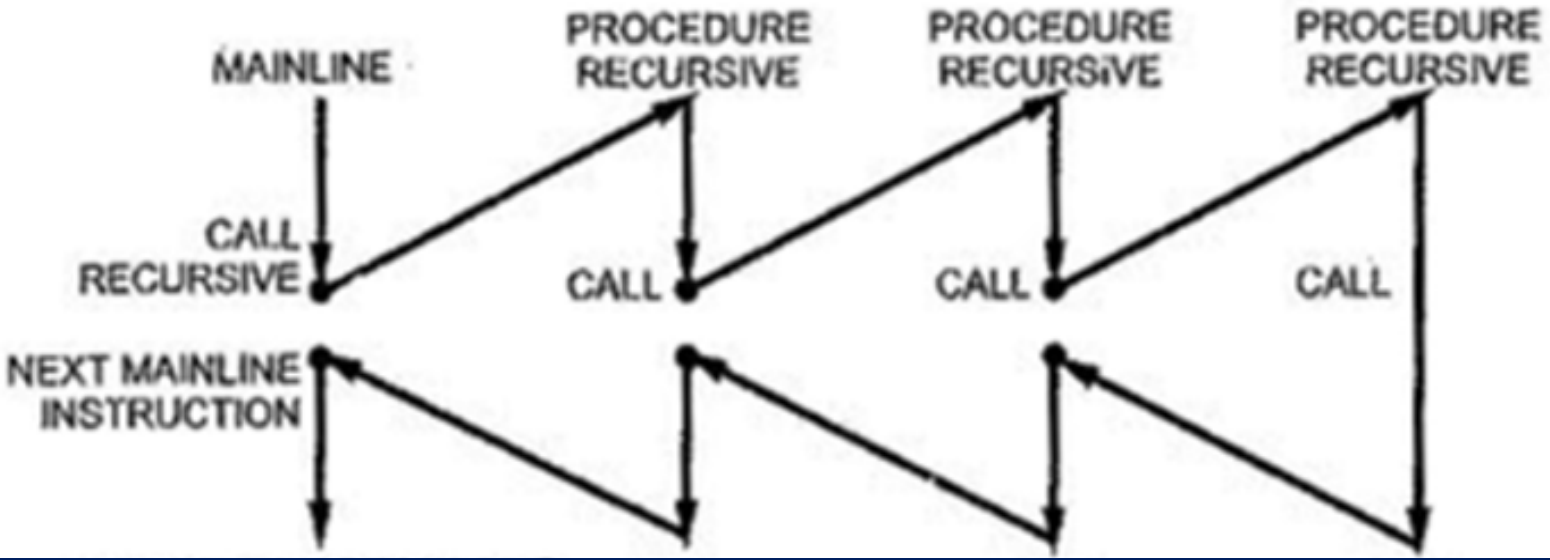
As we all know that a procedure is a set of instruction written separately which can be used any time in the code when required.

A normal procedure execution includes calling of the procedure, shifting the control of the processor to the procedure, and then returning the cont



1) Recursive procedures

A **recursive procedure** is a procedure which calls itself. This results in the procedure call to be generated from within the procedures again and again. This can be understood as follows:

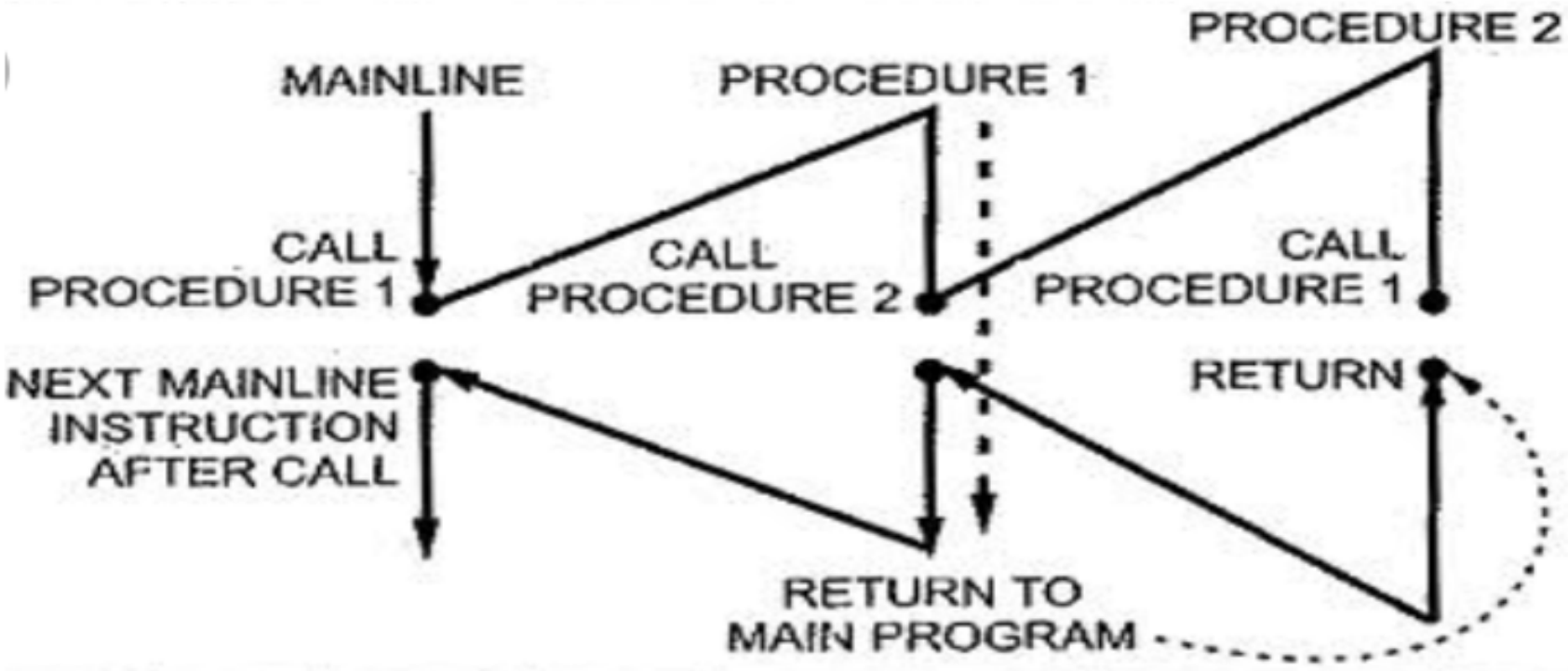


- The **recursive procedures** keep on executing until the termination condition is reached.
- The recursive procedures are very effective to use and to implement but they take a large amount of stack space and the linking of the procedure within the procedure takes more time as well as puts extra load on the processor.

2) Re-entrant procedures

The **re-entrant procedure** is a very special kind of procedure. In such kind of procedure, procedure 1 is called the mainline program, then procedure 2 is called form procedure 1 and then again procedure 1 is called form procedure 2.

- This can be well understood from the following diagram:


MAINLINE
PROCEDURE 1
PROCEDURE 2
CALL PROCEDURE 1
CALL PROCEDURE 2
CALL PROCEDURE 1
NEXT MAINLINE INSTRUCTION AFTER CALL
RETURN
RETURN TO MAIN PROGRAM

- This is called a **re-entrant procedure** because a procedure is re-entering into itself form another procedure which is also present inside its own body.
- The **re-entrant procedure** occurs in the following three conditions: when the procedure is undergoing recursion, when multi-threading is being implemented inside a program or when some interruption is being generated.
- Like the recursive procedures, it is important to have a termination condition for the procedures in the re-entrant procedures also, else we can face machine halts due to infinite procedure calls.

NEAR CALL	FAR CALL
Near call refers to a procedure which is in the same code memory	Far call refers to a procedure which is in the different code segment
It is also called as intra segment call	It is also called as inter segment call
Near call replaces old IP with the New IP.	Near call replaces old CS:IP pair with the new CS:IP pair.
The value of IP is pushed on to the stack	The value of CS:IP pair are pushed on to the stack
Less stack locations are required	More stack locations are required

MACROS

- In ALP small sequence of codes of the same pattern are repeated frequently at different places which performs the same operation on the different data.
- Such repeated code can be written separately as macro.
- When the processor finds the macro name in the source code the block of code associated with the macro name is substituted or expanded at the point of call known as macro expansion.

- Macro should be used when its body has a few program statements.
- The process of defining macro and using them to simplify the programming process is known as macro programming.

Advantages:

- Simplify and reduce the amount of repetitive coding.
- Reduces errors caused by the repetitive coding.
- Makes program more readable.
- execution time is less as compared with the procedure.

Procedure	macro
Procedure is a set of program statement that can be processed independently and reused again and again	A macro is a set of program statement that can be used again and again by using macro name
The machine code for the group of instruction needs to be loaded into main memory only once	Machine code for the group of instruction in the MACRO needs to be loaded into main memory whenever macro is used
Less main memory is required	More main memory is required
Uses call and ret instruction to call procedure which increase the overhead execution time	Avoids overhead execution time

Directives for MACRO

- MACRO
- ENDM
- INCLUDE
- LOCAL
- PURGE

MACRO :

- The directive MACRO informs the assembler the beginning of a macro.
- It consist of name of a macro followed by keyword MACRO .
- **Syntax:**

macro _name MACRO

ENDM:

- This directive informs the assembler that the end of the macro.
- **Syntax:**

ENDM

INCLUDE:

- The directive include informs the assembler to include the statements defined in the include file.
- The name of the include file follows the statement INCLUDE.
- It is used to place all the data and frequently used macros in to a file known as include file.

Syntax:

- INCLUDE<file path specification with file name>

PURGE

- The directive INCLUDE causes the assembler to include all the macro definitions that are in the specified library.
- Suppose library consist number of files but the program require only one then PURGE directive enable to delete unwanted files.

Syntax:

PURGE file name

- **Program to add two numbers using macro**

```
.model small
Add_num macro no1,no2,result
mov ax,no1
add ax,no2
mov result,ax
endm
.data
 num1 dw 1234h
 num2 dw 4325h
 res dw ?
.code
 Mov ax,@data
 Mov ds,ax
 Add_num num1,num2,res
 ends
 end
```

Thank You!!!