Chapter - 6) Procedure & Macro in Assembly Language Programming

Procedures: The process of splitting a large program into small tasks and designing them independently is known as modular programming. Large programs are more prone to error and it is difficult to locate and isolate errors. Therefore a repeated group of instruction in a program can be organised as subprograms / procedures.

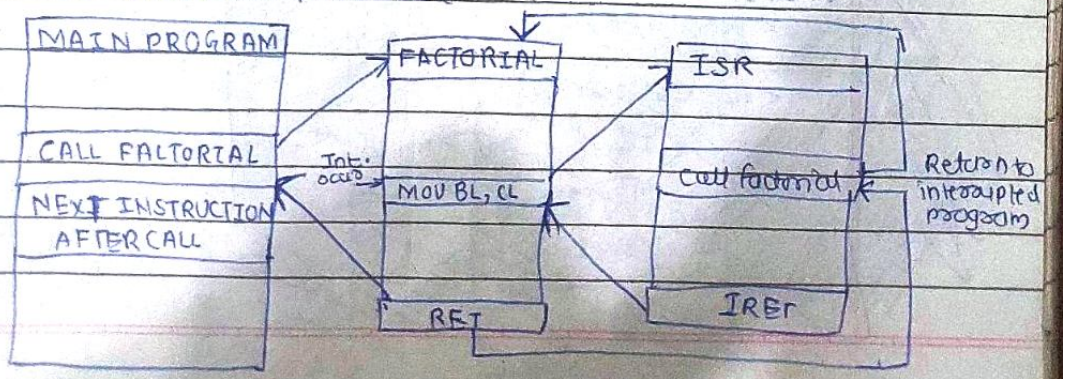
A procedure is a set of program statements that can be processed independently and reuse again and again.

Advantages: ① Simple modular programming
② Reduced work load and development time
③ Debugging of program and procedure are easier.
④ Reduction in size of main program

Disadvantages: ① Extra code required to integrate procedures i.e CALL & RET instructions.
② Execution time is more

Defining procedures near / far: The assembler directives PROC and ENDP are used to define a procedure. The directive PROC indicates the beginning of procedure and directive ENDP indicates end of the procedure to the assembler. These both directives must enclose procedure code that defines the subroutine. The procedure must be defined within the code segment only.

A procedure must be written in such a way that it can be interrupted, used and re-entered without losing or writing over anything. Such procedure is called as re-entrant procedure.



200

A recursive procedure is a procedure which calls within itself and is used to work with complex data structures.

Re-entrant	Recursive
① The procedure which can be interrupted used and re-entered without losing or writing over anything.	① It is the procedure which calls itself
② The flow of control could be interrupted by a hardware interrupt and transferred to an interrupt service routine	② The flow of control could be caused by CALL instruction and transferred to users procedure.

Procedure for directives :

a) PROC b) ENDP

▶ PROC :- The directive PROC indicates the beginning of the procedure and must follow the procedure name. The term FAR and NEAR follows the procedure directive indicating the type of procedure. If term is not specified, then by default its NEAR

General form :

Procedure_Name PROC [NEAR/FAR] — (Refer pg. 130)

▶ ENDP

201

Procedure Call [CALL instruction] — (Refer pg 94)

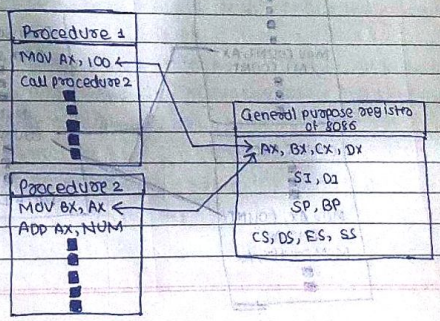
NEAR CALL	FAR CALL
① A near call refers to a procedure call which is in the same code segment as CALL instruction	① A far call refers to a procedure which is in different code segment as CALL instructions
② Also called as Intra-Segment call	② Also called as Inter-segment call
③ It replaces old IP with new IP	③ It replaces old CS:IP with new CS:IP pairs.
④ The old value of IP is pushed onto the stack	④ The value of the old CS:IP pair are pushed onto the stack
⑤ Less stack locations required	⑤ More stack locations required

Procedure return [RET instruction] — (Refer pg 95-96)

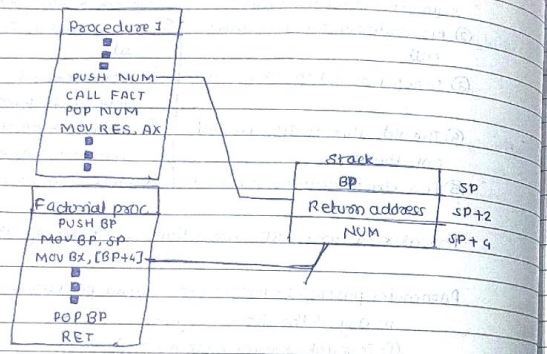
Parameter passing :- Parameters can be passed between procedures in any of the three ways :

① Through general purpose registers
② In an argument list
③ On the stack

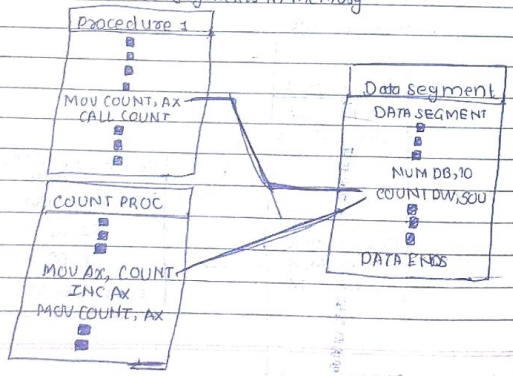
① Passing parameters through the register : The processor does not save the state of the general purpose registers on procedure calls.



② Parameter passing on the stack: To pass a large number of parameters to the called procedure, the parameters can be placed onto the stack in a stack frame for calling the procedure.



③ Passing parameters in an argument list: An alternate method of passing a larger numbers of parameters to the called procedure is to place in an argument list in one of the data segments in memory



Macros: A macro is defined as a single instruction that expands automatically into a set of instructions to perform a particular task.

Advantages:
① Simplify and reduce the amount of repeatative coding
② Reduces errors caused by repetative coding.
③ Makes program more readable
④ Execution time is less as compared to procedure as no extra instructions are required.

Defining Macros: For macros that you want to include in your program, you must first define them. The assembler directive MACRO indicates to the assembler the beginning of MACRO and ENDM indicate the end of the macro to the assembler.

Directives for macros:
(a) MACRO (b) ENDM (c) LOCAL
(d) INCLUDE (e) PURGE

- (a) MACRO → (Refer pg 131)
- (b) ENDM → (Refer pg 131)
- (c) LOCAL → Some macro requires that you define data item and instruction labels within the macro definition. The duplicate name of MACROS would confuse the assembler to generate errors messages.
  To ensure each macro name is unique, code the LOCAL directive immidiately after the MACRO statements, even before comments
  General form:
  LOCAL dummy_1, dummy_2 ....
- (d) INCLUDE → (Refer pg 134)

PAGE NO. 204
DATE

▶ PURGE :- The PURGE directive enable you to delete the unwanted macros from the currrent assembly

Ex:- INCLUDE D:\TASM\MACRO.LIB
PURGE DISP, DISP_8BIT_NUM

Macro arguments :- To make macros flexible, you can define names in it as dummy arguments

Ex:- .MODEL SMALL
.DATA
STR_S DB 'COMPUTER$'
STR_D DB 10 DUP(0)
COUNT DB 8
.CODE
::
::
STRREV STR_S, STR_D, COUNT
::
::
::
ENDS
END

In above example, the value of STR_S, STR_D and COUNT will be substituted to MACRO ARGUMENT STR1, STR2 and COUNT respectively.

Procedure	Macro
① A procedure is a set of the program statements that can be processed independently and reuse again and again	① A macro is a set of the program statements that can be used again and again and hence MACRO is called as open subroutine
② The machine code for the group of instructions in the procedures need to be loaded in main memory	② A machine code in the group of instructions in MACRO needs to be loaded in main memory whenever MACRO is used.
③ Less main memory required	③ More main memory required

PAGE NO. 205
DATE

Procedure	MACRO
④ CALL and RET instructions are used to call procedure which increase the overhead execution time involved in calling and returning of procedures	④ The MACRO avoids the overhead execution time involved in calling and returning from a macro as macro does not require CALL and RET instructions
⑤ The procedure needs the stack compulsorily	⑤ The macros does not need stack compulsorily.

Programming using procedures:
Program to perform arithmetic operations such as addition, subtraction, multiplication and division:

```
.MODEL SMALL
.DATA
NUM1 DW 5432H
NUM2 DW 99H
RES_ADD DW ?
RES_SUB DW ?
RES_MUL DW ?
RES_QUO DW ?
RES_REM DW ?
.CODE
MOV AX, @DATA
MOV DS, AX
CALL ADD_NUM
CALL SUB_NUM
CALL MUL_NUM
CALL DIV_NUM
MOV AH, 4CH
INT 21H
ADD_NUM PROC
        MOV AX, NUM1
        ADD AX, NUM2
        MOV RES_ADD, AX
        RET
        ENDP
```

```
SUB_NUM PROC
        MOV AX, NUM1
        SUB AX, NUM2
        MOV RES_ADD, AX
        RET
        ENDP
MUL_NUM PROC
        MOV AX, NUM1
        MUL NUM2
        MOV WORD_PTR RES_MUL, AX
        MOV WORD PTR RES MUL+2, DX
        RET
        ENDP
DIV_NUM PROC
        MOV AX, NUM1
        CWD
        DIV NUM2
        MOV RES_QUO, AX
        MOV RES_REM, DX
        RET
        ENDP
ENDS
END
```

206

Prog. to arrange numbers in array in ascending order using procedure

```
.MODEL SMALL
.DATA
    ARRAY DW 12H, 11H, 21H, 9H, 19H
.CODE
    MOV AX, @DATA
    MOV DS, AX
    CALL ASC_ORDER
    MOV AH, 4CH
    INT 21H
ASC_ORDER PROC
    MOV BX, 5
UP1: LEA SI, ARRAY
    MOV CX, 4
UP: MOV AX, [SI]
    CMP AL, [SI+2]
    JC DN
    XCHG AX, [SI+2]
    XCHG AX, [SI]
DN: ADD SI, 2
    LOOP UP
    DEC BX
    JNZ UP1
    RET
    ENDP
ENDS
END
```

Prog. to arrange numbers in array in descending order using procedure

```
.MODEL SMALL
.DATA
    ARRAY DW 12H, 11H, 21H, 9H, 19H
.CODE
    MOV AX, @DATA
    MOV DS, AX
    CALL DSC_ORDER
    MOV AH, 4CH
    INT 21H
DSC_ORDER PROC
    MOV BX, 5
UP1: LEA SI, ARRAY
    MOV CX, 4
UP: MOV AX, [SI]
    CMP AL, [SI+2]
    JNC DN
    XCHG AX, [SI+2]
    XCHG AX, [SI]
DN: ADD SI, 2
    LOOP UP
    DEC BX
    JNZ UP1
    RET
    ENDP
ENDS
END
```

207

Prog. to find smallest number from the array using procedure

```
.MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 2H
    SMALL DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    CALL SMALLEST
    MOV SMALL, AX
    MOV AH, 4CH
    INT 21H
SMALLEST PROC
    MOV CX, 5
    MOV SI, OFFSET ARRAY
    MOV AX, [SI]
    DEC CX
UP: INC SI
    INC SI
    CMP AX, [SI]
    JC NEXT
    MOV AX, [SI]
NEXT: LOOP UP
    RET
    ENDP
    ENDS
    END
```

Prog. to find largest number from the array using procedure

```
.MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 2H
    LARGE DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    CALL LARGEST
    MOV LARGE, AX
    MOV AH, 4CH
    INT 21H
LARGEST PROC
    MOV CX, 5
    MOV SI, OFFSET ARRAY
    MOV AX, [SI]
    DEC CX
UP: INC SI
    INC SI
    CMP AX, [SI]
    JNC NEXT
    MOV AX, [SI]
NEXT: LOOP UP
    RET
    ENDP
    ENDS
    END
```

PAGE NO. 208
DATE

Prog for addition of series of 8 bit numbers using procedure

```
.MODEL SMALL
.DATA
LIST DB 1,2,3,4,5,6,7,8,9,10
RES DB 0
COUNT DB 10
.CODE
MOV AX, @DATA
MOV DS, AX
CALL SUM
SUM PROC
MOV CL, COUNT
MOV SI, OFFSET LIST
MOV AL, 0
UP: ADD AL, [SI]
INC SI
DEC CL
JNZ UP
RET
ENDP
ENDS
END
```

Prog for performing using procedure
Z = (A+B) * (C+D)

```
.MODEL SMALL
.DATA
A DB 1
B DB 2
C DB 3
D DB 4
Z DW ?
ASUM DB ?
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AL, A
MOV BL, B
CALL ADD_BYTE
MOV ASUM, AL
MOV AL, C
MOV BL, D
CALL ADD_BYTE
MUL ASUM
MOV Z, AX
ADD_BYTE PROC
ADD AL, BL
RET
ENDP
ENDS
END
```

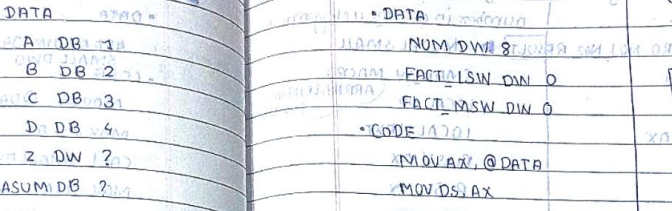
→ subroutine

PAGE NO. 209
DATE

Procedure to find factorial of numbers

```
.MODEL SMALL
.DATA
NUM DW 8
FACT_LSW DW 0
FACT_MSW DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
CALL FACT
FACT PROC
PUSH AX
PUSH BX
PUSH DX
MOV AX, NUM
MOV BX, AX
DEC BX
UP: MUL BX
MOV FACT_LSW, AX
MOV FACT_MSW, DX
DEC BX
CMP BX, 0
JNZ UP
POP DX
POP BX
POP AX
RET
ENDP
ENDS
END
```

Prog to multiply 8 bit numbers using procedure




```
.MODEL SMALL
.DATA
NUM1 DW 99H
NUM2 DW 99H
RES_MUL DD ?
.CODE
MOV AX, @DATA
MOV DS, AX
CALL MUL_NUM
MOV AH, 4CH
INT 21H
MUL_NUM PROC
MOV AX, NUM1
MUL NUM 2
MOV WORD PTR RES_MUL, AX
MOV WORD PTR RES_MUL+2, DX
RET
ENDP
ENDS
END
```

PAGE NO. 210
DATE

Programming using Macros:

Program to add 2 nos 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 4321H
RES DW ?
.CODE
MOV AX, @DATA
MOV DS, AX
ADD_NUM NUM1, NUM2, RES
ENDS
END
```

Program to display string

```
.MODEL SMALL
DISP MACRO MSG
PUSH AX
PUSH DX
MOV AH, 09H
MOV DX, OFFSET MSG
INT 21H
POP DX
POP AX
ENDM
.DATA
STR DB 'My name is ABC$'
.CODE
MOV AX, @DATA
MOV DS, AX
DISP STR
MOV AH, 4CH
INT 21H
ENDS END
```

Program to find smallest number in array using macro

```
.MODEL SMALL
SMALL_W MACRO ARRAY, LENGTH, SMALLEST
LOCAL UP
LOCAL NEXT
PUSH AX
PUSH CX
PUSH SI
MOV CL, LENGTH
MOV SI, OFFSET ARRAY
MOV AX, [SI]
DEC CL
UP: ADD SI, 2
CMP AX, [SI]
JC NEXT
MOV AX, [SI]
NEXT: DEC CL
JNZ UP
MOV SMALLEST, AX
POP SI
POP CX
POP AX
ENDM
.DATA
LIST_1 DW 12H, 19H, 98H, 56H, 72H
LIST_2 DW 65H, 75H, 85H, 01H, 45H
COUNT DB 5
SMALL_1 DW ?
SMALL_2 DW ?
.CODE
MOV AX, @DATA
MOV DS, AX
SMALL_W LIST_1, COUNT, SMALL_1
SMALL_W LIST_2, COUNT, SMALL_2
MOV AH, 4CH
INT 21H
ENDS END
```

PAGE NO. 211
DATE

Using Macros, write ALP to solve $P = x^2 + y^2$ where x and y are 8 bit number

```
.MODEL SMALL
SQR_NUM MACRO NO1, SQR
MOV AL, NO1
MUL AL, NO1
MOV SQR, AX
ENDM
.DATA
X DB 34H
Y DB 21H
SX DW ?
SY DW ?
P DW ?
.CODE
MOV AX, @DATA
MOV DS, AX
SQR_NUM X, SX
SQR_NUM Y, SY
MOV AX, SX
ADD AX, SY
MOV P, AX
ENDS
END
```