Stacks -- Procedures -- Macros

UCS1502

MICROPROCESSORS AND INTERFACING

Ms. S. Angel Deborah
AP/CSE

Learning Objectives

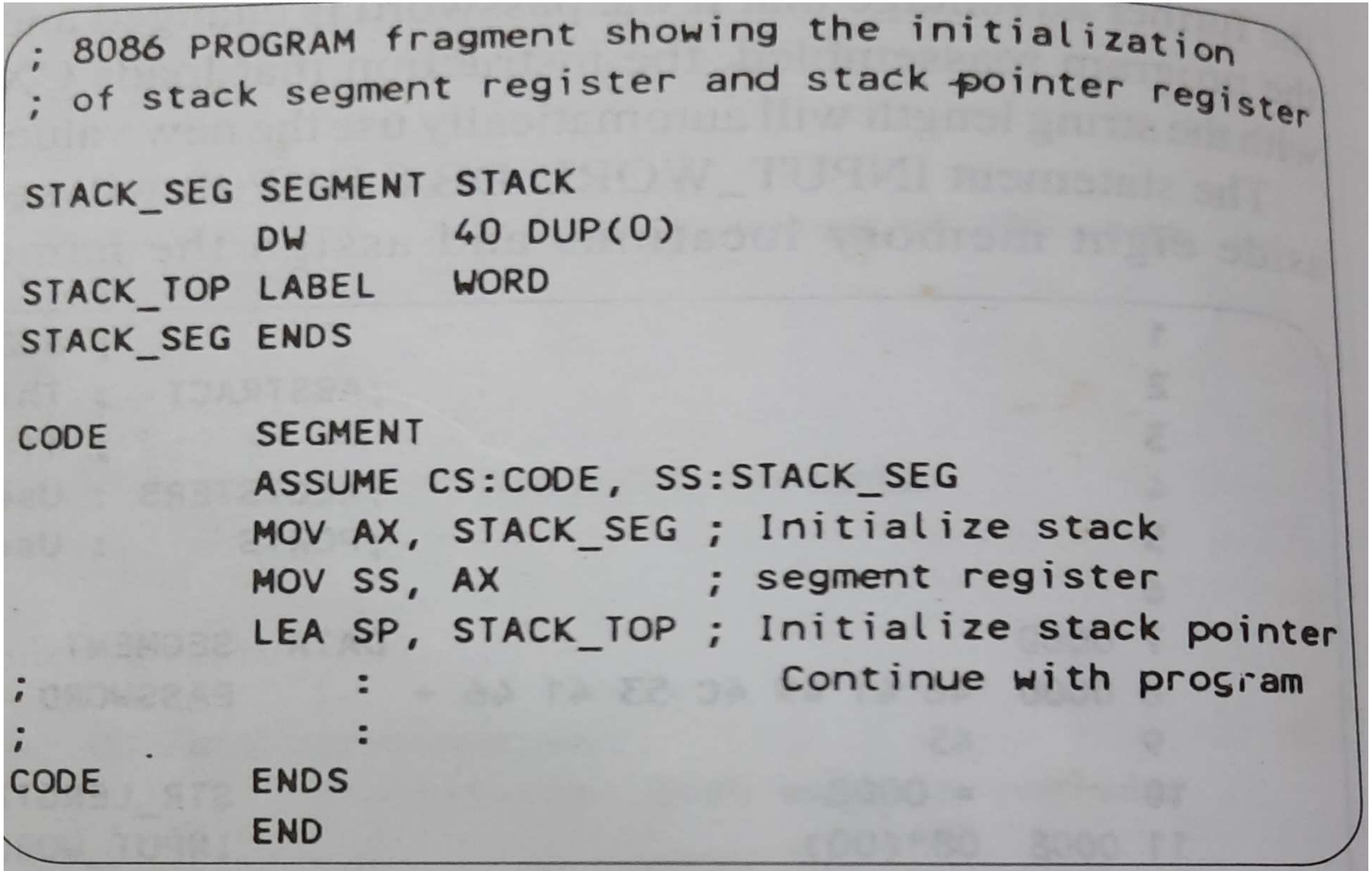
1. To understand the stack of 8086
2. To understand the difference between procedures and macros

Overview

- Stack
- Procedure
- Types of Procedures
- Advantage and disadvantages
- Macro

Stack -Introduction

- A section of memory is set aside for storing return addresses.
- It is also used to save the contents of registers for the calling program while a procedure executes.
- Used to hold data or addresses that will be acted upon by a procedure.
- Stack segment register holds the segment base address of the stack segment. Its size is 64K
- Stack Pointer register is used to hold the offset of the last word written on the stack.



```
; 8086 PROGRAM fragment showing the initialization
; of stack segment register and stack pointer register

STACK_SEG SEGMENT STACK
          DW      40 DUP(0)
STACK_TOP LABEL   WORD
STACK_SEG ENDS

CODE      SEGMENT
          ASSUME CS:CODE, SS:STACK_SEG
          MOV AX, STACK_SEG ; Initialize stack
          MOV SS, AX        ; segment register
          LEA SP, STACK_TOP ; Initialize stack pointer
;             :               Continue with program
;             :
CODE      ENDS
          END
```

Fig. 5.8 *Required program additions when using a stack.*

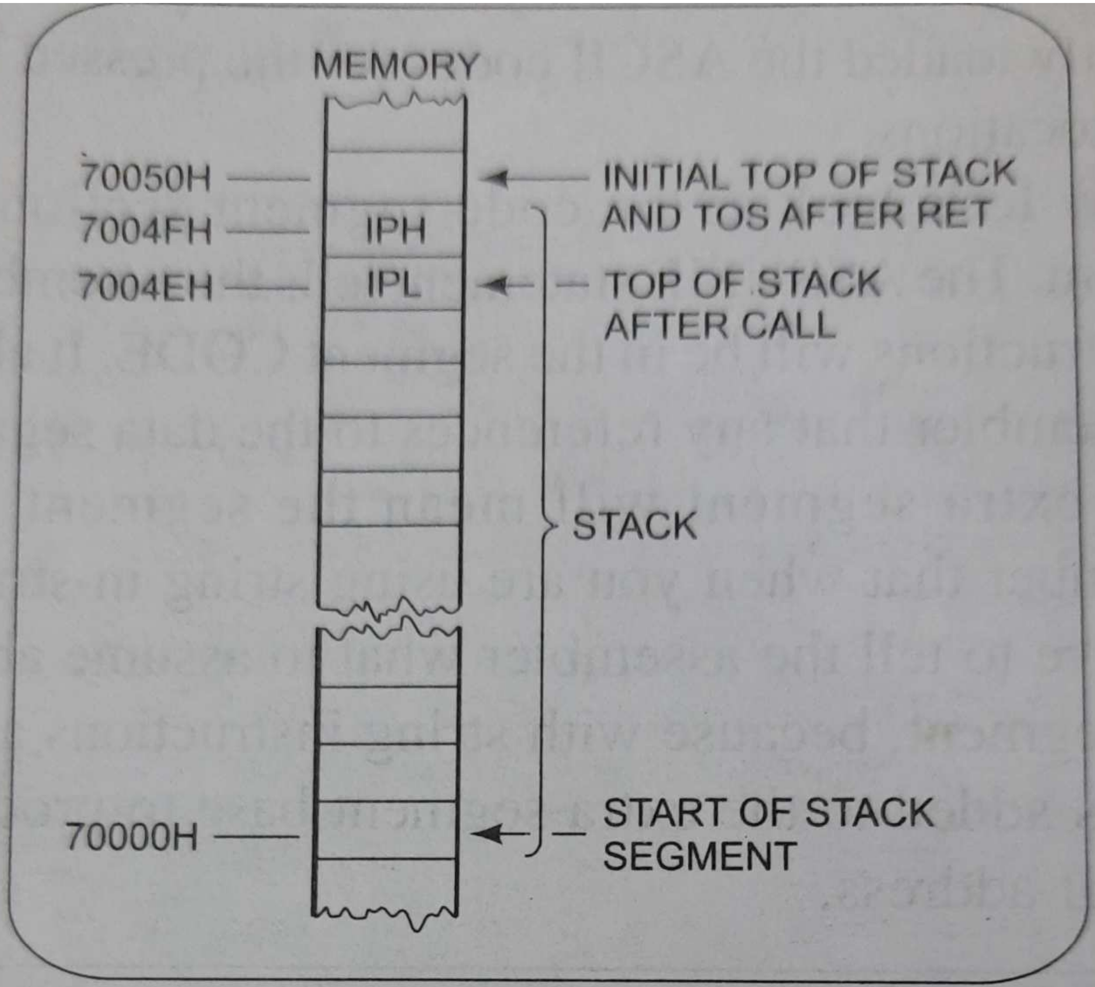
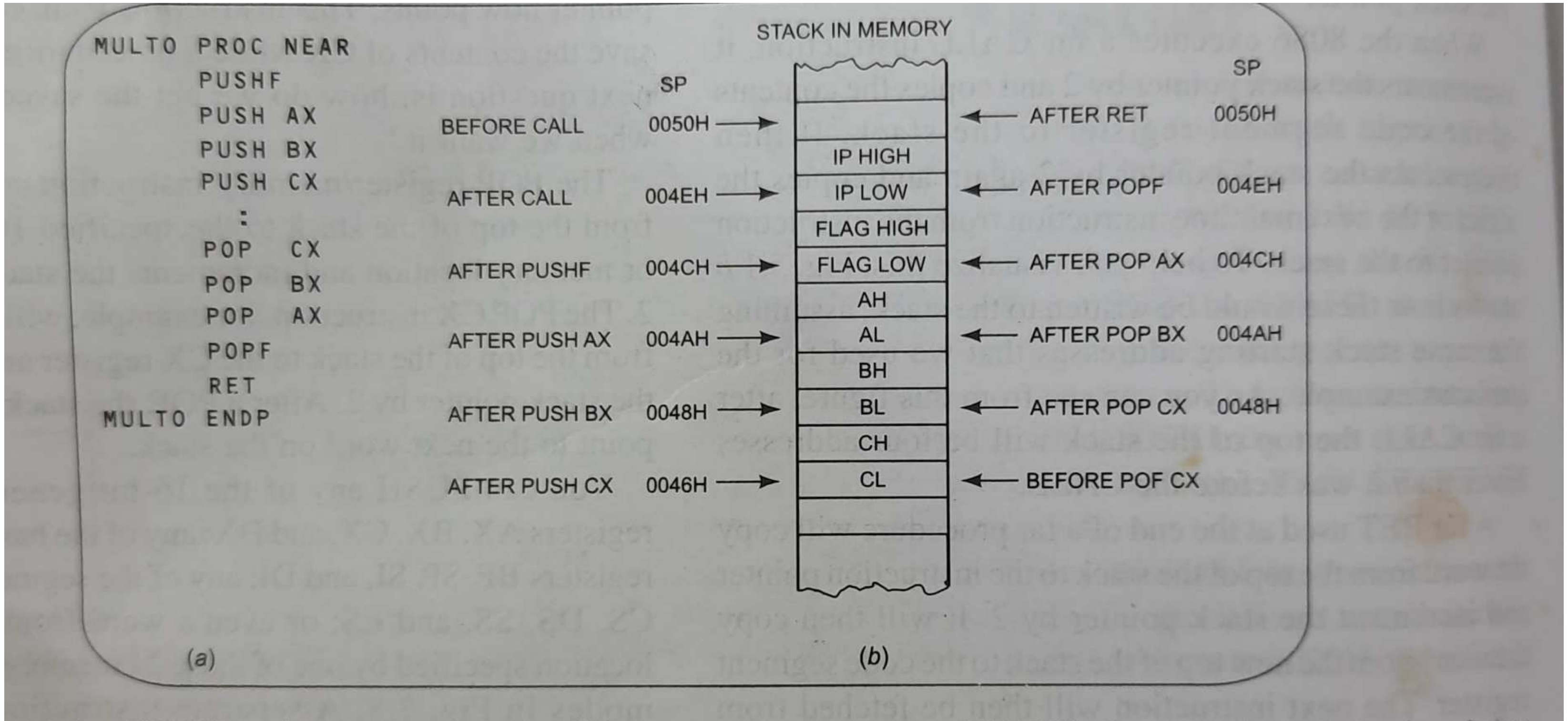


Fig. 5.7 *Stack diagram showing how the return address is pushed onto the stack by CALL.*



```
MULTO PROC NEAR
      PUSHF
      PUSH  AX
      PUSH  BX
      PUSH  CX
        :
      POP   CX
      POP   BX
      POP   AX
      POPF
      RET
MULTO ENDP
```

Using PUSH and POP instructions. (a) Instruction sequence, (b) Effect on stack and stack pointer.

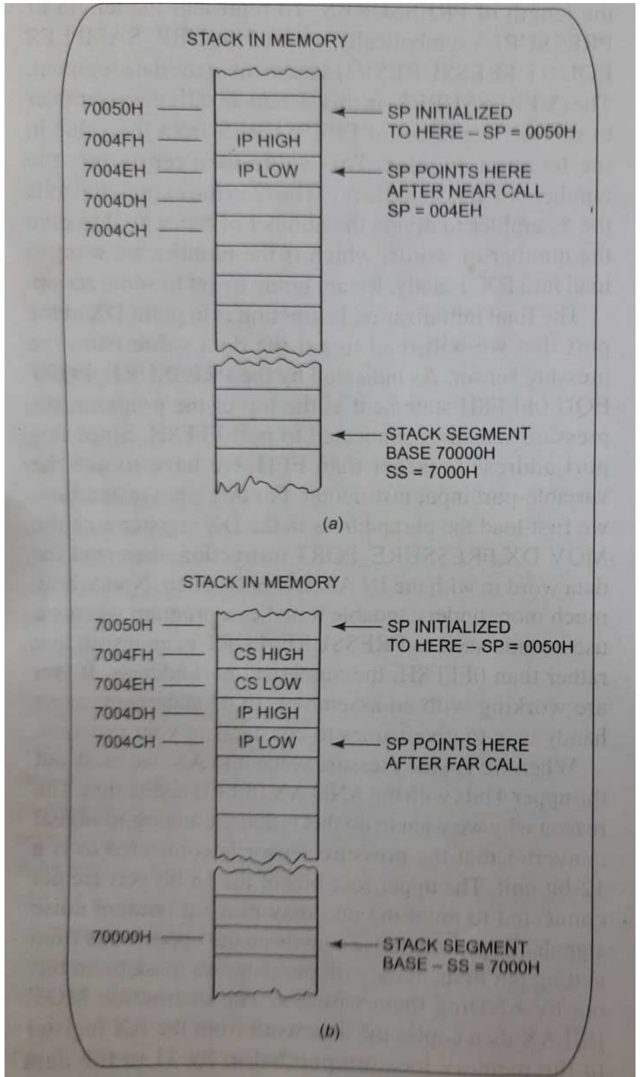
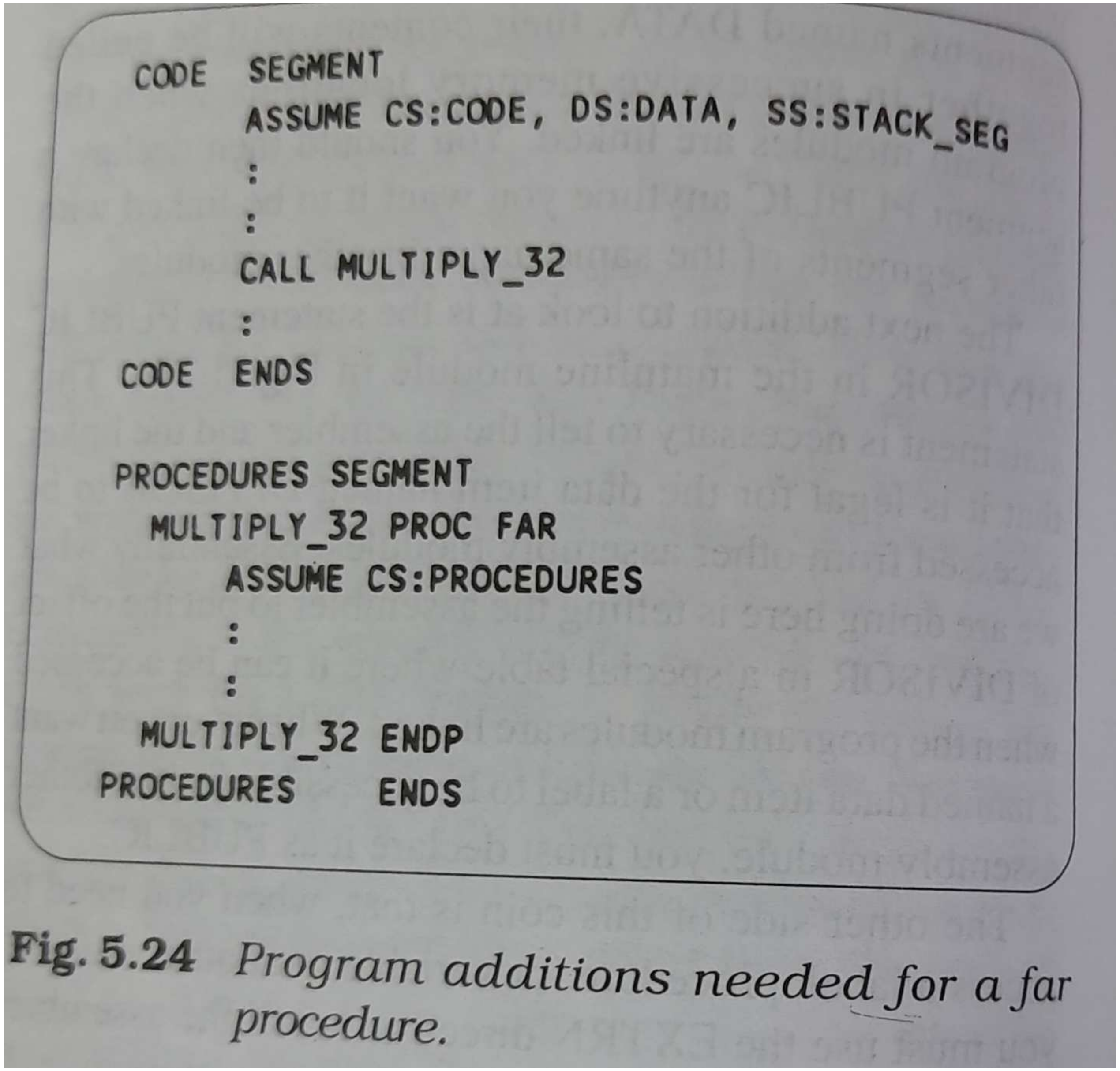

STACK IN MEMORY
70050H
7004FH
7004EH
7004DH
7004CH
IP HIGH
IP LOW
SP INITIALIZED TO HERE – SP = 0050H
SP POINTS HERE AFTER NEAR CALL SP = 004EH
STACK SEGMENT BASE 70000H SS = 7000H
(a)
STACK IN MEMORY
70050H
7004FH
7004EH
7004DH
7004CH
70000H
CS HIGH
CS LOW
IP HIGH
IP LOW
SP INITIALIZED TO HERE – SP = 0050H
SP POINTS HERE AFTER FAR CALL
STACK SEGMENT BASE – SS = 7000H
(b)

Fig. 5.11 *Stack diagram for program in Figure 5.10. (a) For near CALL, (b) For far CALL.*

Procedure

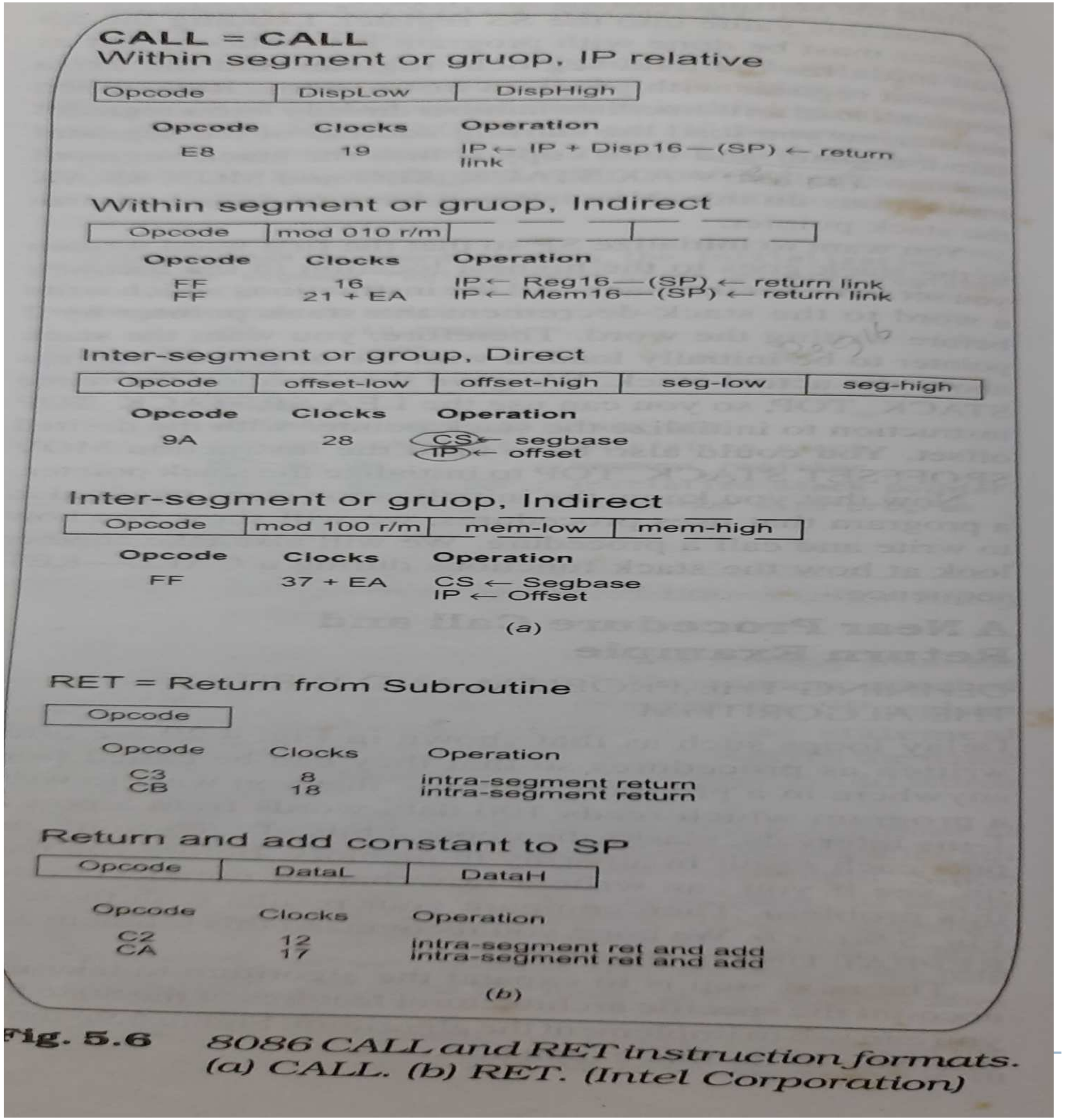
- To avoid writing the sequence of instructions in the program each time, we write a sequence of a separate subprogram called procedure.
- Direct Within-Segment Near call
- Indirect Within-Segment Near call
- Inter-segment Far call



```
CODE  SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STACK_SEG
      :
      :
      CALL MULTIPLY_32
      :
CODE  ENDS

PROCEDURES SEGMENT
  MULTIPLY_32 PROC FAR
      ASSUME CS:PROCEDURES
      :
      :
  MULTIPLY_32 ENDP
PROCEDURES     ENDS
```

Fig. 5.24 *Program additions needed for a far procedure.*



CALL = CALL

Within segment or gruop, IP relative

Opcode	DispLow	DispHigh

Opcode	Clocks	Operation
E8	19	IP ← IP + Disp16 —(SP) ← return link

Within segment or gruop, Indirect

Opcode	mod 010 r/m					

Opcode	Clocks	Operation
FF	16	IP ← Reg16—(SP) ← return link
FF	21 + EA	IP ← Mem16—(SP) ← return link

Inter-segment or group, Direct

Opcode	offset-low	offset-high	seg-low	seg-high

Opcode	Clocks	Operation

9A	28	CS ← segbase
		IP ← offset

Inter-segment or gruop, Indirect

Opcode	mod 100 r/m	mem-low	mem-high

Opcode	Clocks	Operation

FF	37 + EA	CS ← Segbase
		IP ← Offset

(*a*)

RET = Return from Subroutine

Opcode

Opcode	Clocks	Operation
C3	8	intra-segment return
CB	18	intra-segment return

Return and add constant to SP

Opcode	DataL	DataH

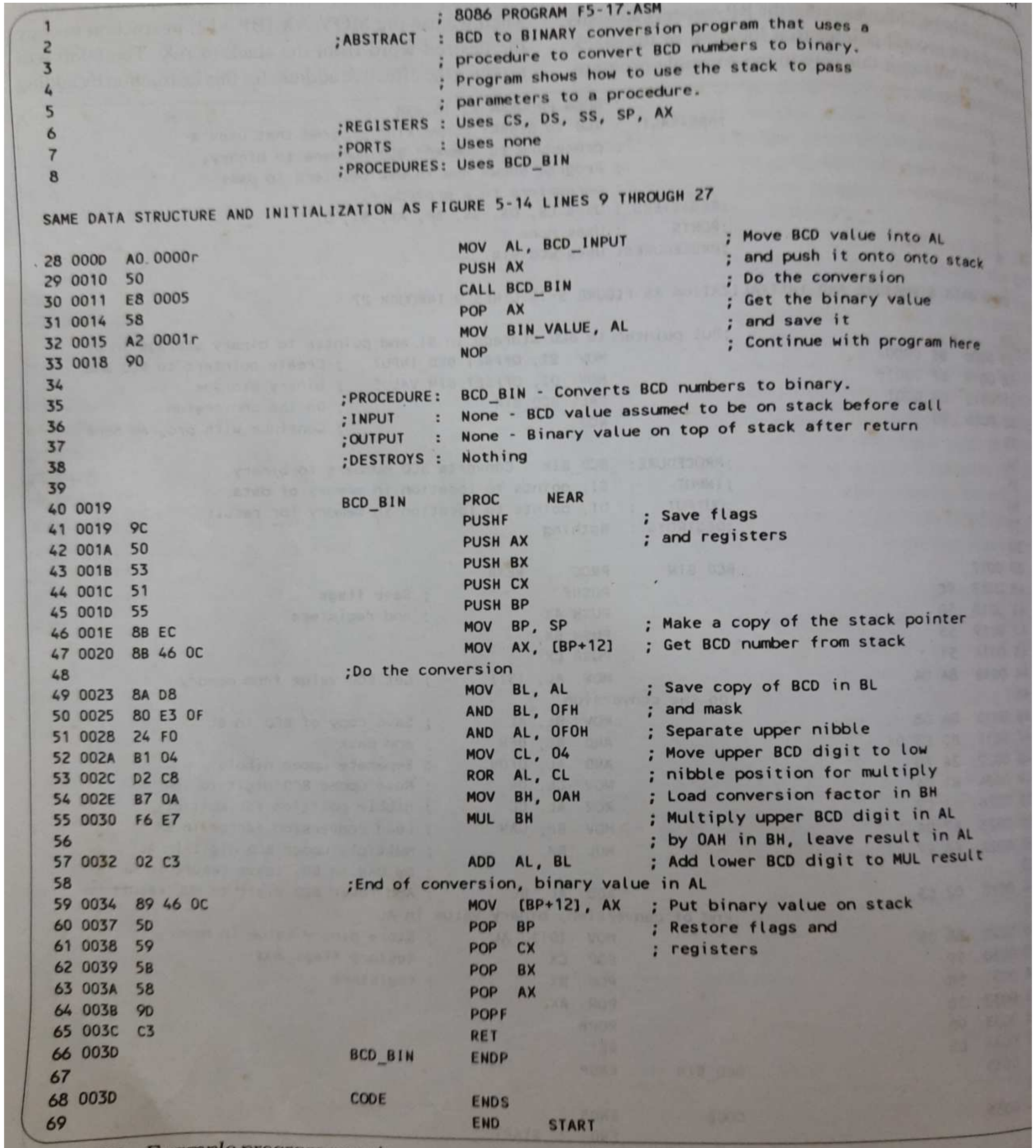
Opcode	Clocks	Operation
C2	12	intra-segment ret and add
CA	17	intra-segment ret and add

(*b*)

Fig. 5.6 *8086 CALL and RET instruction formats. (a) CALL. (b) RET. (Intel Corporation)*

Passing parameters to and from procedures

- Four ways
  - In Registers
  - In dedicated memory locations accessed by name
  - With pointers passed in registers
  - With the stack



```
 1                                 ; 8086 PROGRAM F5-17.ASM
 2                    ;ABSTRACT   : BCD to BINARY conversion program that uses a
 3                                ; procedure to convert BCD numbers to binary.
 4                                ; Program shows how to use the stack to pass
 5                                ; parameters to a procedure.
 6                    ;REGISTERS  : Uses CS, DS, SS, SP, AX
 7                    ;PORTS      : Uses none
 8                    ;PROCEDURES: Uses BCD_BIN

SAME DATA STRUCTURE AND INITIALIZATION AS FIGURE 5-14 LINES 9 THROUGH 27

28 000D  A0 0000r                  MOV  AL, BCD_INPUT        ; Move BCD value into AL
29 0010  50                        PUSH AX                   ; and push it onto onto stack
30 0011  E8 0005                   CALL BCD_BIN              ; Do the conversion
31 0014  58                        POP  AX                   ; Get the binary value
32 0015  A2 0001r                  MOV  BIN_VALUE, AL        ; and save it
33 0018  90                        NOP                       ; Continue with program here
34
35                    ;PROCEDURE:  BCD_BIN - Converts BCD numbers to binary.
36                    ;INPUT    :  None - BCD value assumed to be on stack before call
37                    ;OUTPUT   :  None - Binary value on top of stack after return
38                    ;DESTROYS :  Nothing
39
40 0019               BCD_BIN      PROC    NEAR
41 0019  9C                        PUSHF                  ; Save flags
42 001A  50                        PUSH AX                ; and registers
43 001B  53                        PUSH BX
44 001C  51                        PUSH CX
45 001D  55                        PUSH BP
46 001E  8B EC                     MOV  BP, SP            ; Make a copy of the stack pointer
47 0020  8B 46 0C                  MOV  AX, [BP+12]       ; Get BCD number from stack
48                    ;Do the conversion
49 0023  8A D8                     MOV  BL, AL            ; Save copy of BCD in BL
50 0025  80 E3 0F                  AND  BL, 0FH           ; and mask
51 0028  24 F0                     AND  AL, 0F0H          ; Separate upper nibble
52 002A  B1 04                     MOV  CL, 04            ; Move upper BCD digit to low
53 002C  D2 C8                     ROR  AL, CL            ; nibble position for multiply
54 002E  B7 0A                     MOV  BH, 0AH           ; Load conversion factor in BH
55 0030  F6 E7                     MUL  BH                ; Multiply upper BCD digit in AL
56                                                        ; by 0AH in BH, leave result in AL
57 0032  02 C3                     ADD  AL, BL            ; Add lower BCD digit to MUL result
58                    ;End of conversion, binary value in AL
59 0034  89 46 0C                  MOV  [BP+12], AX      ; Put binary value on stack
60 0037  5D                        POP  BP               ; Restore flags and
61 0038  59                        POP  CX               ; registers
62 0039  5B                        POP  BX
63 003A  58                        POP  AX
64 003B  9D                        POPF
65 003C  C3                        RET
66 003D               BCD_BIN      ENDP
67
68 003D               CODE         ENDS
69                                 END     START
```

Fig. 5.17 *Example program passing parameters on the stack.*

Reentrant Procedure

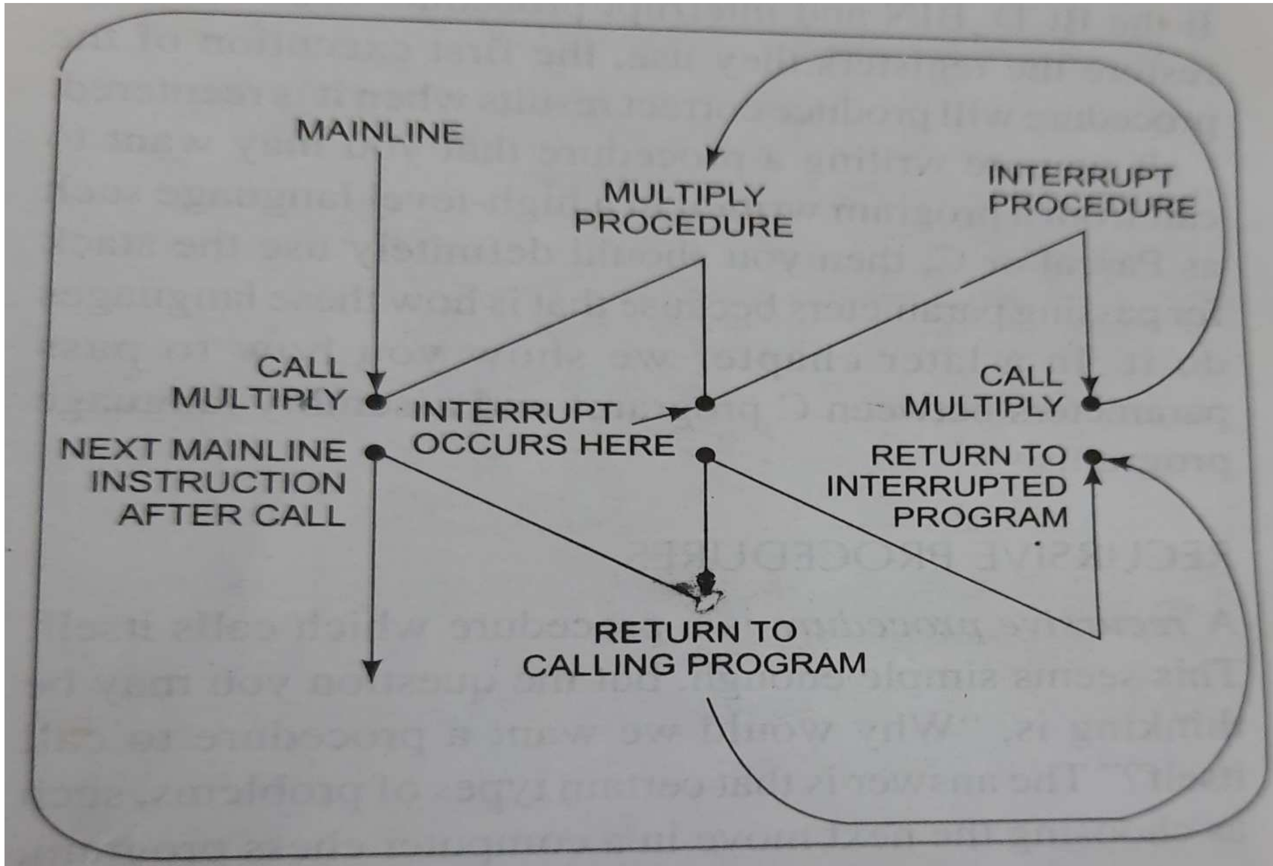
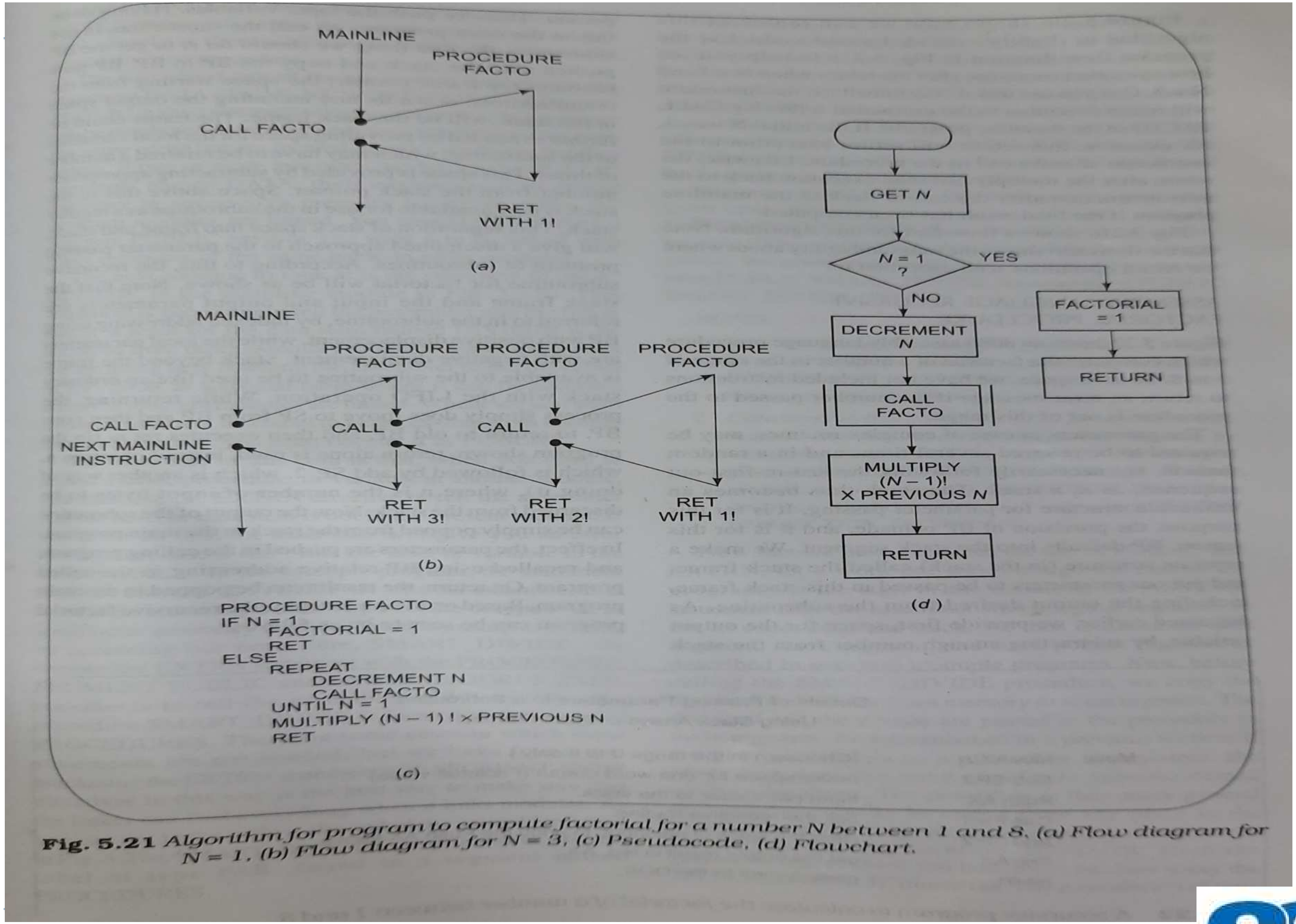


Fig. 5.20 *Program execution flow for reentrant procedure.*

Recursive Procedure



```
PROCEDURE FACTO
IF N = 1
    FACTORIAL = 1
    RET
ELSE
    REPEAT
        DECREMENT N
        CALL FACTO
    UNTIL N = 1
    MULTIPLY (N – 1) ! × PREVIOUS N
    RET
```

(c)

Fig. 5.21 *Algorithm for program to compute factorial for a number N between 1 and 8. (a) Flow diagram for N = 1, (b) Flow diagram for N = 3, (c) Pseudocode, (d) Flowchart.*

Advantage and disadvantage of procedure

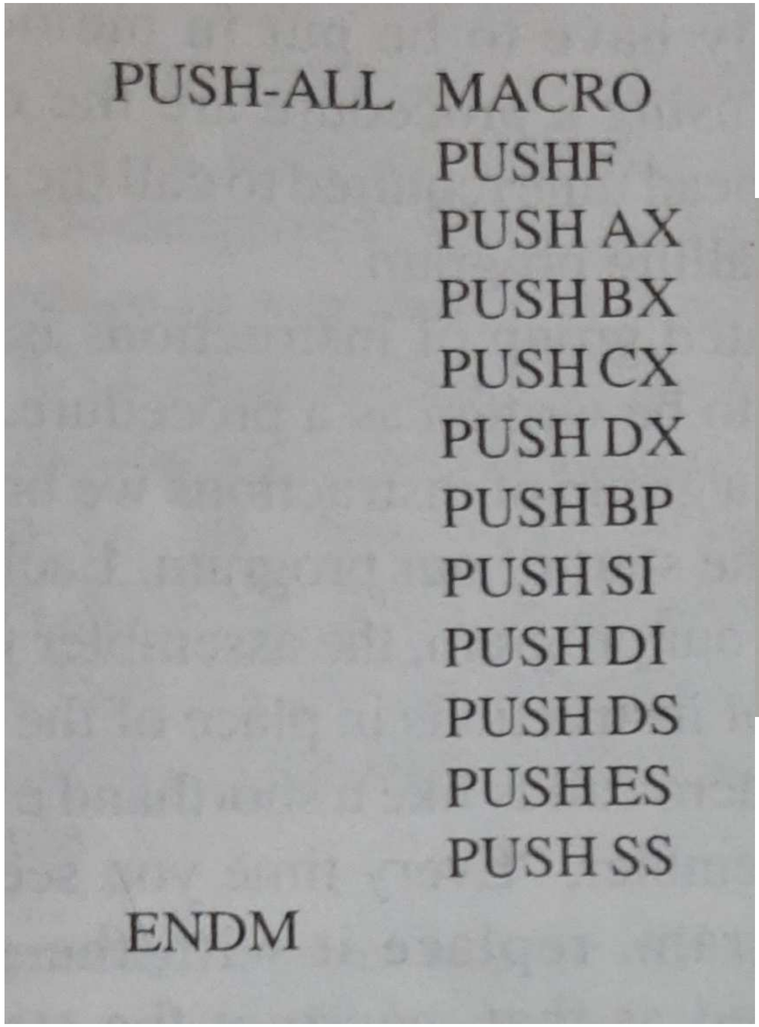
- Advantage:
  - Machine codes for the group of instructions in procedure should be put in memory only once
- Disadvantage:
  - Need for a stack
  - Overhead time to call and return

Macro

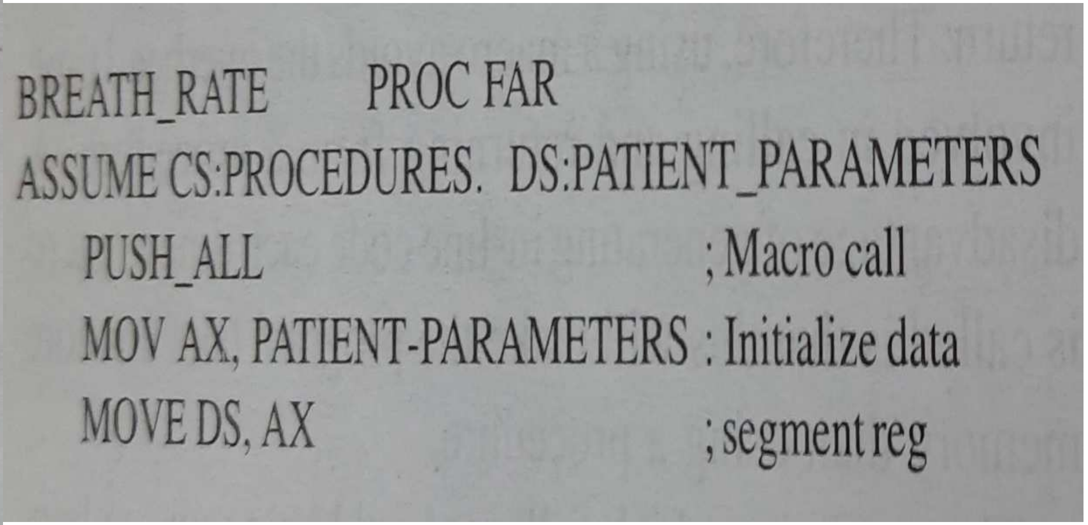
- A macro is a group of instructions we bracket and give a name at the start of our program
- Each time we call it, the assembler will insert it in the program.

Advantage and disadvantage of macro

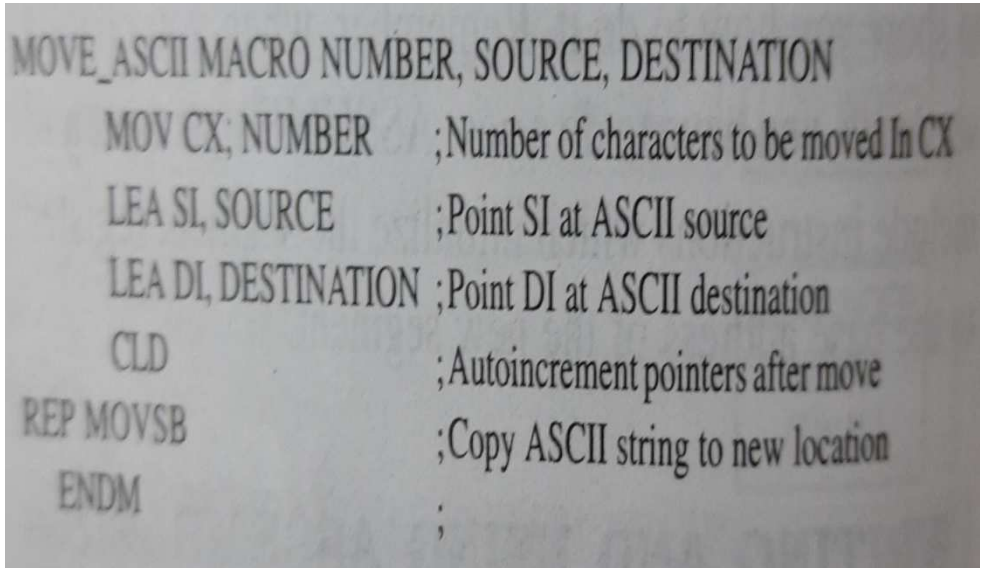
- Advantage:
  - No need for a stack
  - No Overhead time to call and return
- Disadvantage:
  - Machine codes for the group of instructions in procedure should be put in memory multiple times.



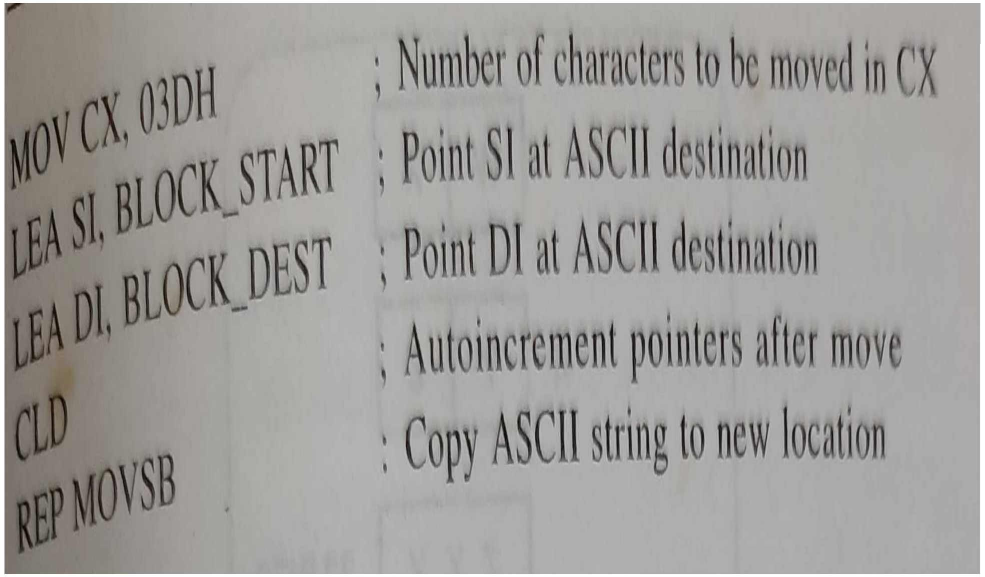
```
PUSH-ALL   MACRO
           PUSHF
           PUSH AX
           PUSH BX
           PUSH CX
           PUSH DX
           PUSH BP
           PUSH SI
           PUSH DI
           PUSH DS
           PUSH ES
           PUSH SS
 ENDM
```



```
BREATH_RATE      PROC FAR
ASSUME CS:PROCEDURES.  DS:PATIENT_PARAMETERS
    PUSH_ALL                           ; Macro call
    MOV AX, PATIENT-PARAMETERS : Initialize data
    MOVE DS, AX                        ; segment reg
```



```
MOVE_ASCII MACRO NUMBER, SOURCE, DESTINATION
      MOV CX; NUMBER      ;Number of characters to be moved In CX
      LEA SI, SOURCE      ;Point SI at ASCII source
      LEA DI, DESTINATION ;Point DI at ASCII destination
      CLD                 ;Autoincrement pointers after move
REP MOVSB                 ;Copy ASCII string to new location
   ENDM                   ;
```



```
MOV CX, 03DH            ; Number of characters to be moved in CX
LEA SI, BLOCK_START     ; Point SI at ASCII destination
LEA DI, BLOCK_DEST      ; Point DI at ASCII destination
CLD                     ; Autoincrement pointers after move
REP MOVSB               : Copy ASCII string to new location
```

Check your understanding?

- What is the status of SP when PUSH is executed?
- What is the difference between procedure and macro?
- What increases the overhead time to call and return?

Summary

- Stack
- Procedure
- Types of Procedures
- Advantage and disadvantages
- Macro

Reference

- Doughlas V Hall, "Microprocessors and Interfacing, Programming and Hardware", TMH, 2012.

Thank you