8086 Stack, Procedures

Md. Shahidul Salim

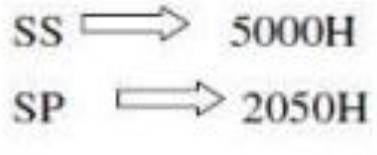
Farhan Sadaf

Stack

- The 8086 microprocessor has a dedicated area in memory called the stack, which is used for temporary storage of data during program execution.
- It operates on a **Last-In-First-Out (LIFO)** principle, meaning that the most recently stored data is the first to be retrieved.
- The **stack pointer (SP)** is a 16-bit register that points to the current top of the stack. It contains the offset address of the memory location in the stack segment.
- **Stack Segment (SS)** register contains the base address of the stack segment in the memory.
- The stack segment, like any other segment, may have a memory block of a maximum of 64 Kbytes locations, and thus may overlap with any other segments.

Stack (Cont.)

- The Stack Segment register (SS) and Stack pointer register (SP) together address the stack-top.



- For a selected value of SS, the maximum value of SP=FFFFH and the segment can have maximum of 64K locations.
- If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack.
- After successive push operations, when the stack pointer contains 0000H, any attempt to further push the data to the stack will result in stack overflow.

Stack (Cont.)

- Stack is used by **CALL** instruction to keep return address for procedure, **RET** instruction gets this value from the stack and returns to that offset.
- Quite the same thing happens when **INT** instruction calls an interrupt, it stores in stack flag register, code segment and offset.
- **IRET** instruction is used to return from interrupt call.

Stack Operations

1. **PUSH**
   - Stores a 16 bit value in the stack. Stack pointer (SP) is decremented by 2, for every PUSH operation.
   - E.g. PUSH AX means SP=SP-2 and AX->[SP].

```
Syntax for PUSH instruction:

PUSH REG
PUSH SREG
PUSH memory
PUSH immediate (Only works on 80186 CPU and later)

REG: AX, BX, CX, DX, DI, SI, BP, SP.
SREG: DS, ES, SS, CS.
memory: [BX], [BX+SI+7], 16 bit variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...
```

Stack Operations (Cont.)

2. POP

- Gets 16 bit value from the stack. Stack pointer (SP) is incremented by 2, for every POP operation.
- E.g. POP AX means [SP]->AX and SP=SP+2.

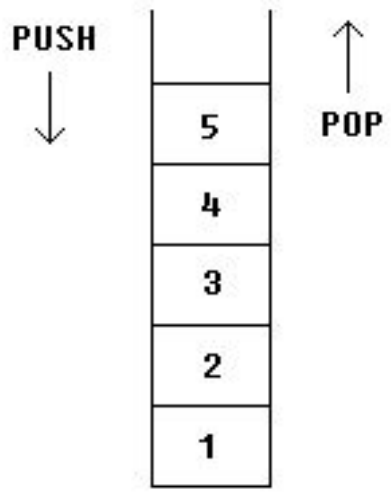
```
Syntax for POP instruction:

POP REG
POP SREG
POP memory

REG: AX, BX, CX, DX, DI, SI, BP, SP.
SREG: DS, ES, SS, (except CS).
memory: [BX], [BX+SI+7], 16 bit variable, etc...
```

Stack Operations (Cont.)

3. **PUSHF and POPF**
   - These instructions are used to push and pop the flags register (FLAGS) onto and from the stack.
   - They are often used when preserving and restoring the processor's status during subroutine calls or context switches.

Stack Operations (Cont.)

- If we push these values one by one into the stack: **1, 2, 3, 4, 5** the first value that we will get on pop will be **5,** then **4, 3, 2,** and only then **1**.



- It is very important to do equal number of **PUSH**s and **POP**s, otherwise the stack maybe corrupted and it will be impossible to return to operating system.

Stack Examples

- **PUSH** and **POP** instruction are especially useful because we don't have too much registers to operate with, so here is a trick:
  - Store original value of the register in stack (using PUSH).
  - Use the register for any purpose.
  - Restore the original value of the register from stack (using POP).

```
ORG     100h

MOV     AX, 1234h
PUSH    AX          ; store value of AX in stack.

MOV     AX, 5678h   ; modify the AX value.

POP     AX          ; restore the original value of AX.

RET
END
```

Stack Examples (Cont.)

- Another use of the stack is for exchanging the values.

```
ORG     100h

MOV     AX, 1212h   ; store 1212h in AX.
MOV     BX, 3434h   ; store 3434h in BX


PUSH    AX          ; store value of AX in stack.
PUSH    BX          ; store value of BX in stack.

POP     AX          ; set AX to original value of BX.
POP     BX          ; set BX to original value of AX.

RET
END
```

Procedures

- In a program, we very frequently face situations where there is a need to perform the same set of task again and again. So, for that instead of writing the same sequence of instructions, again and again, they are written separately in a subprogram. This subprogram is called a procedure.

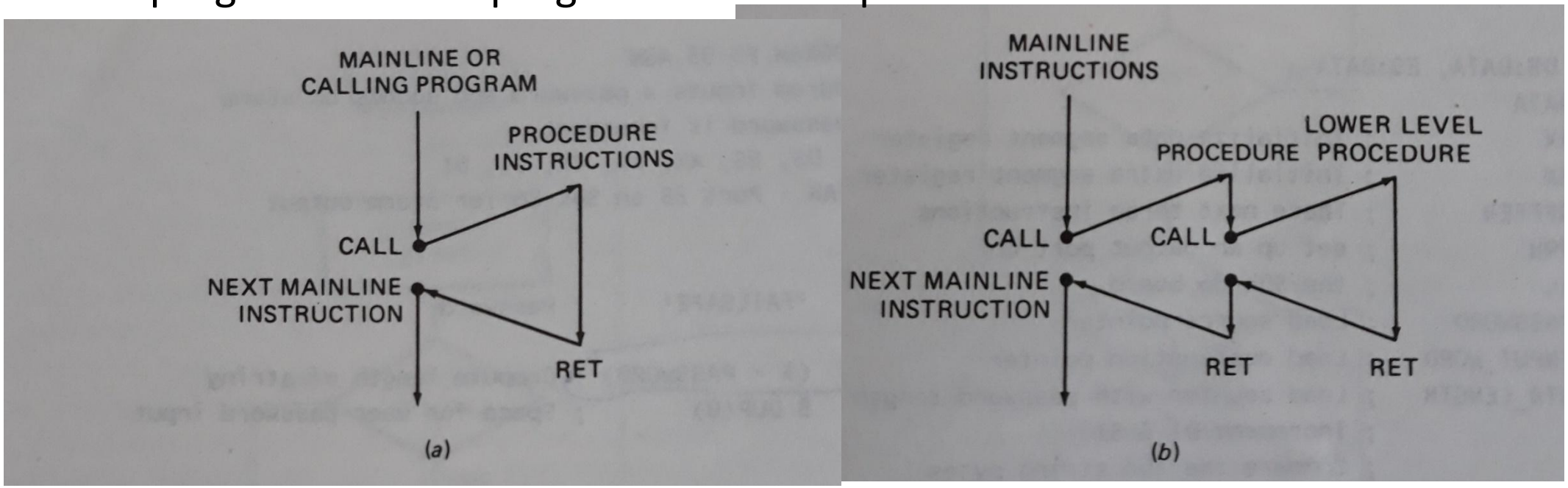
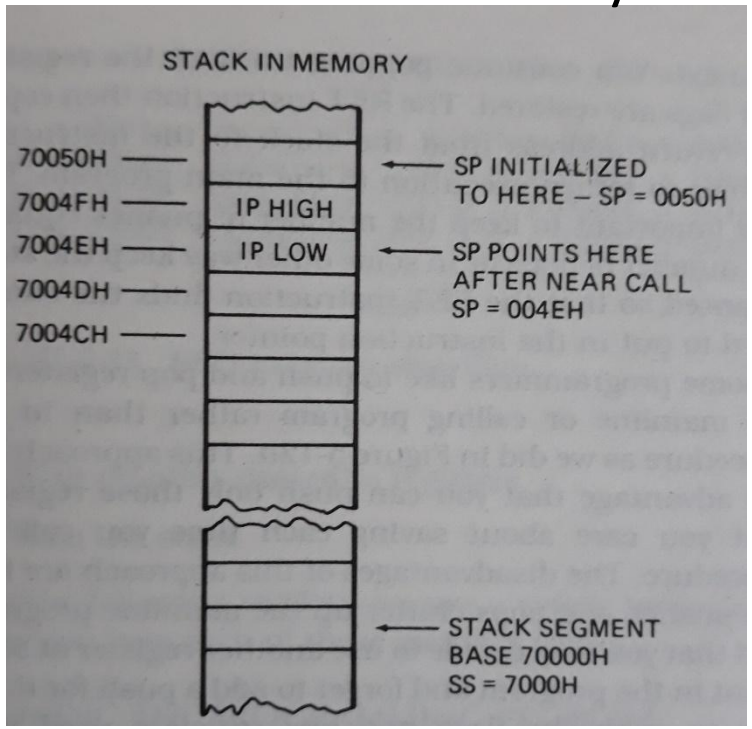


FIGURE 5-4 Program flow to and from procedures. (*a*) Single procedures. (*b*) Nested procedures.

Procedure Types

1. **Near Call or Intra-segment call**
   - A near call refers a procedure which is in the same code segment.
   - Only Instruction Pointer (IP) contents will be changed in NEAR procedure.
   - Near calls are more efficient in terms of execution time because they do not involve changing the code segment register.

```
SP <- SP-2
IP -> stores onto stack

IP <- starting address of a procedure
```

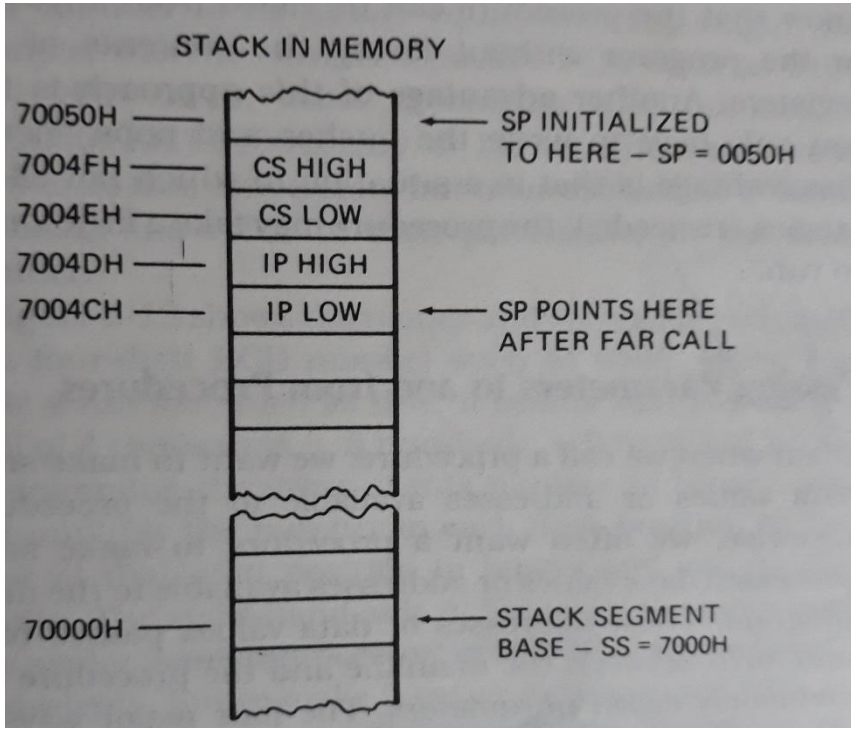


Procedure Types (Cont.)

2. Far Call or Inter-segment call

- A Far call refers a procedure which is in different code segment.
- In this case both Instruction Pointer (IP) and the Code Segment (CS) register content will be changed.

```
SP <- sp-2
cs contents -> stored on stack

SP <- sp-2
IP contents -> stored on stack

CS <- Base address of segment having procedure
IP <- address of first instruction in procedure
```



Procedure Syntax

- **PROC** is a keyword to define that the set of instructions enclosed by the given name is a procedure.
- The **ENDP** keyword defines that the body of the procedure has been ended.

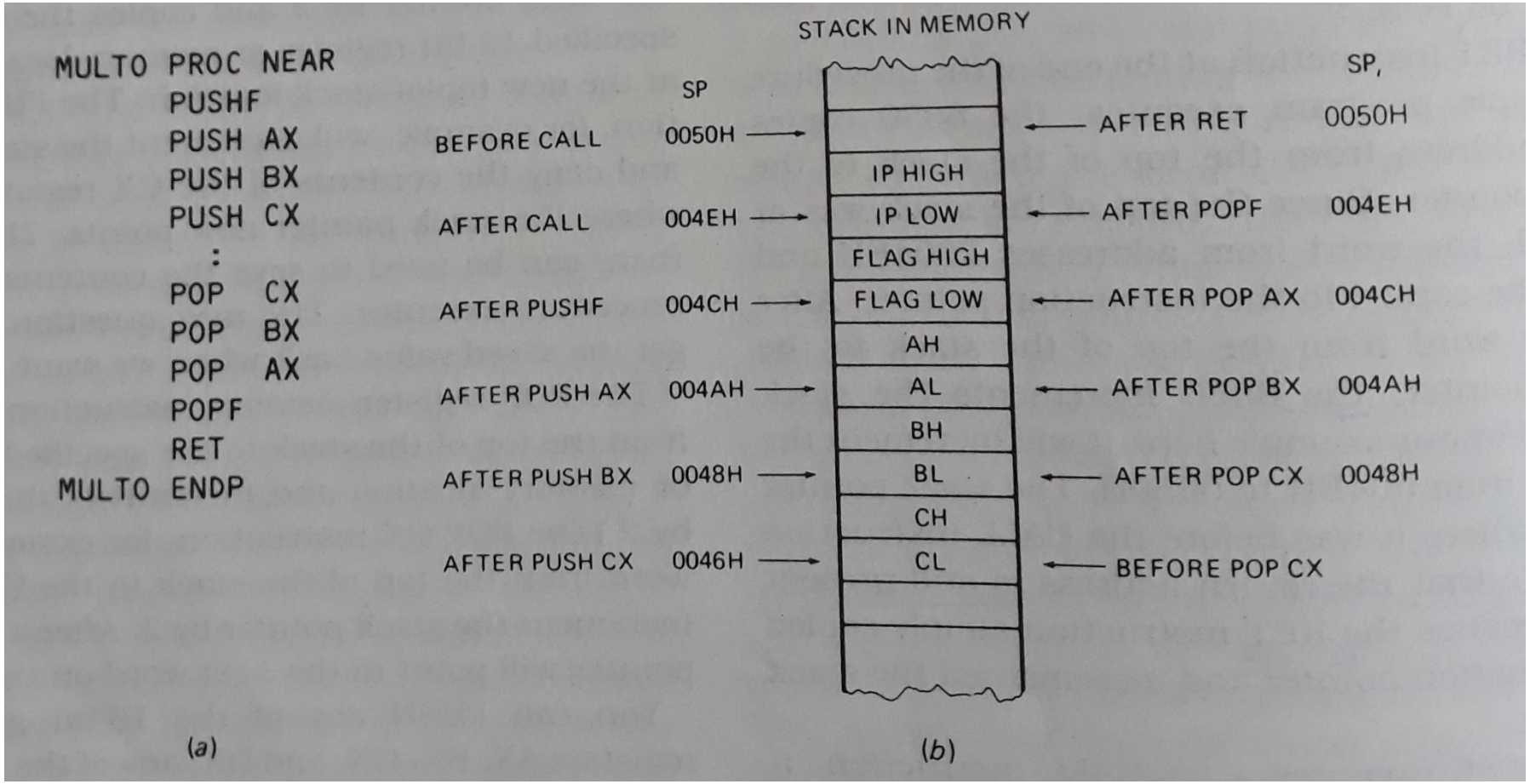
```
procedure_name PROC  [NEAR / FAR]
    Instruction 1
    Instruction 2
    - - - - - - - - - - -
    - - - - - - - - - - -
    Instruction n
procedure_name ENDP
```

ret

- The procedure will be executed whenever a **CALL** to the procedure is made.

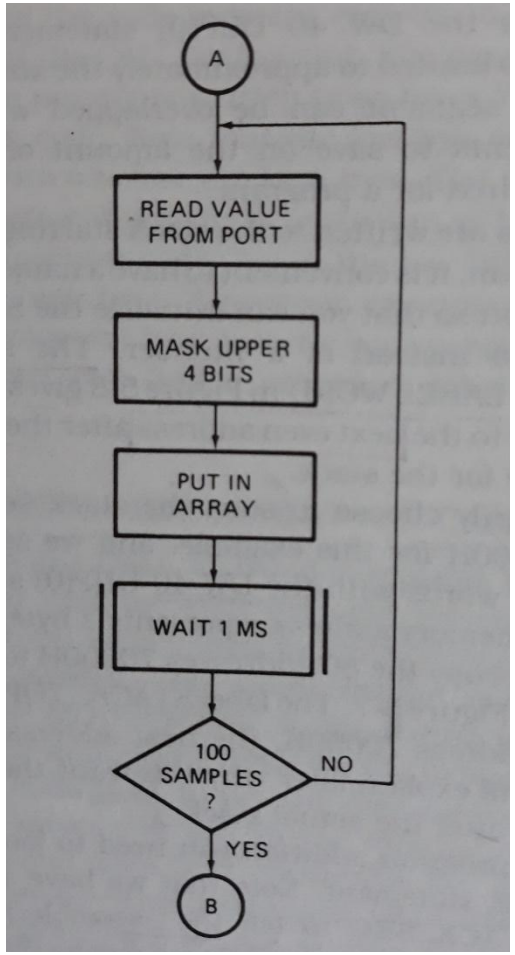
```
CALL procedure_name
```

Procedure Examples



Procedure Examples (Cont.)

- Read 100 samples of data at 1-ms interval



Procedure Examples (Cont.)

- Read 100 samples of data at 1-ms interval (Cont.)

```
PRESSURE_PORT   EQU  0FFF8H

DATA      SEGMENT
     PRESSURES       DW   100  DUP(0)       ; Set up array of 100 words
     NBR_OF_SAMPLES  EQU  (($-PRESSURES)/2)
DATA      ENDS

STACK_SEG  SEGMENT
           DW   40  DUP(0)     ; set stack length of 40 words
STACK_TOP  LABEL    WORD
STACK_SEG  ENDS
```

```
CODE       SEGMENT
           ASSUME  CS:CODE, DS:DATA, SS:STACK_SEG
START:     MOV  AX, DATA             ; Initialize data segment register
           MOV  DS, AX
           MOV  AX, STACK_SEG        ; Initialize stack segment register
           MOV  SS, AX
           MOV  SP, OFFSET STACK_TOP ; Intialize stack pointer to top of stack

           LEA  SI, PRESSURES        ; Point SI to start of array
           MOV  BX, NBR_OF_SAMPLES   ; Load BX with number of samples
           MOV  DX, PRESSURE_PORT    ; Point DX at input port
NEXT_VALUE:IN   AX, DX               ; Read data from port
           AND  AX, 0FFFH            ; Mask upper 4 bits
           MOV [SI],AX               ; Store data word in array
           CALL WAIT_1MS             ; Delay 1 ms
           INC  SI                   ; Point SI at next location in array
           INC  SI
           DEC  BX                   ; Decrement sample counter
           JNZ  NEXT_VALUE           ; Repeat until 100 samples done
STOP:      NOP

WAIT_1MS   PROC     NEAR
           MOV  CX, 23F2H            ; Load delay constant into CX
HERE:      LOOP HERE                 ; Loop until CX = 0
           RET
WAIT_1MS   ENDP

CODE       ENDS
           END
           END
```

References

- https://www.brainkart.com/article/memory-Stacks-in-8086-Microprocessor_7851/
- https://faculty.kfupm.edu.sa/COE/shazli/coe205/Help/asm_tutorial_09.html#:~:text=there%20are%20two%20instructions%20that,bit%20value%20from%20the%20stack.&text=REG%3A%20AX%2C%20BX%2C%20CX,%2C%20SI%2C%20BP%2C%20SP
- https://www.includehelp.com/embedded-system/procedures-in-the-8086-microprocessor.aspx
- https://www.snjb.org/polytechnic/up-images/downloads/chapter%206-MAPupFile_058d4fa990ababaa.pdf