Chapter 8

Stack

Stack is a one dimensional data structure.
Items are added and removed in last-in first-out manner.
The most recent item is called top of the stack (TOS).

Declaring stack segment

```
.STACK 100H
```

When the program is assembled and loaded in memory,

- SS register contains the segment number of the stack segment.
- SP register is initialized to 100h for the preceding stack declaration. This indicates that the stack is empty. When the stack is not empty, SP contains the offset address of the TOS.

PUSH and PUSHF

PUSH source (e.g. PUSH AX)
source = 16 bit register or memory word

Execution of PUSH causes the following to happen:

- SP is decreased by 2.
- A copy of the source content is pushed to the address specified by SS:SP. The source is unchanged

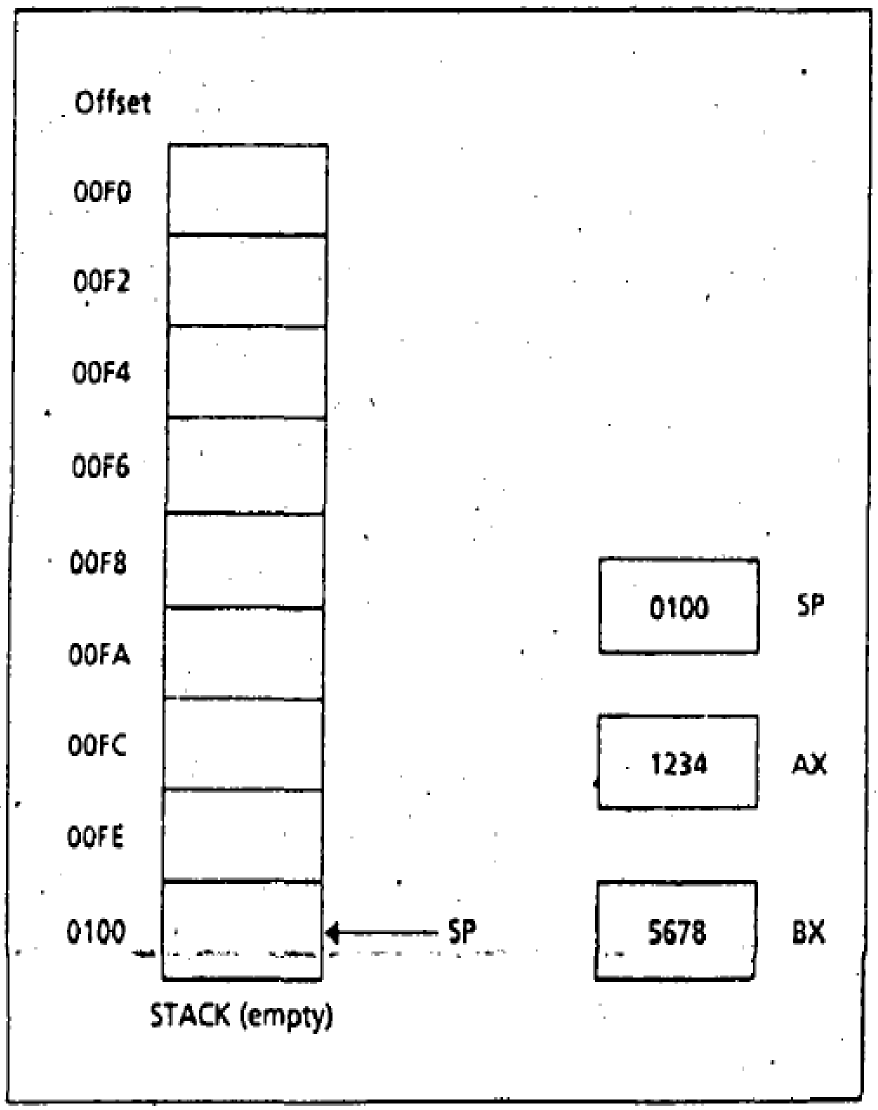
PUSHF

This instruction pushes the contents of the FLAGS register onto the stack.

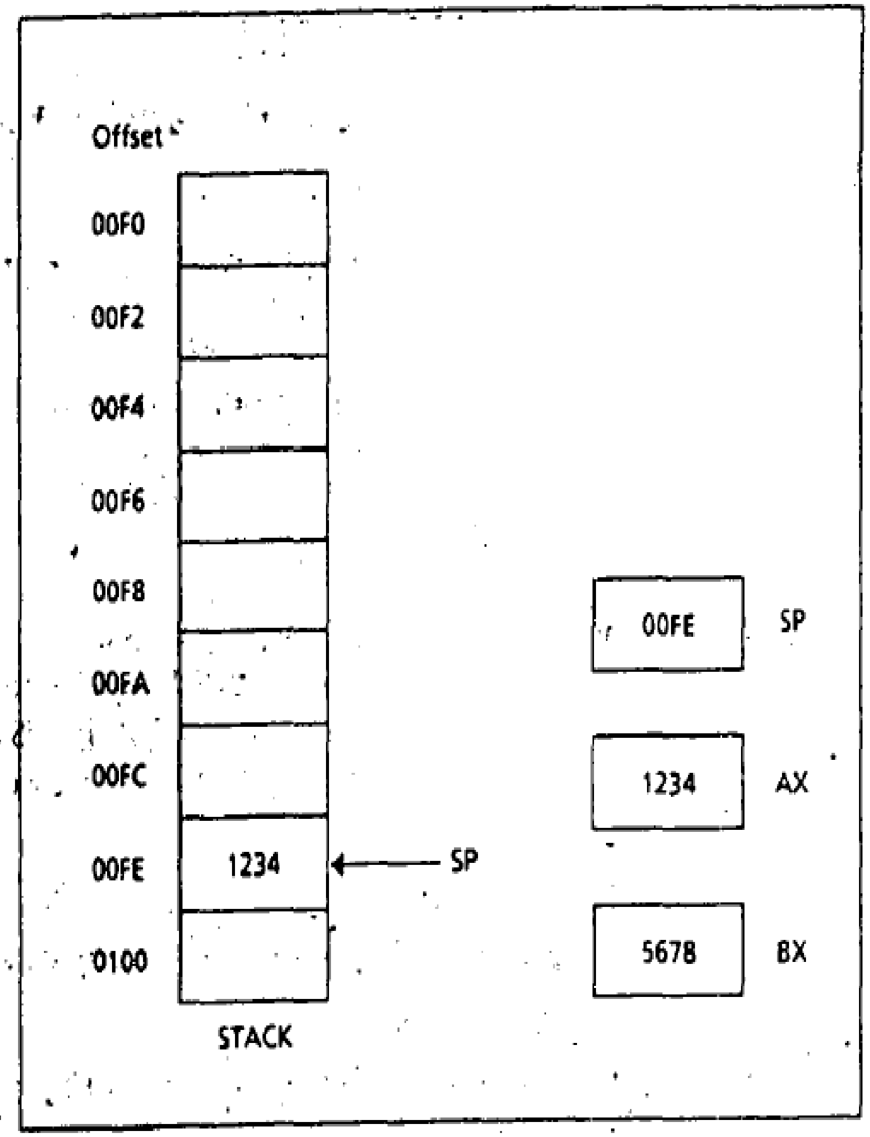
There is no effect of PUSH/PUSHF on the flags.

Illegal: PUSH DL (Byte instruction is illegal)
            PUSH 2 (Push of immediate data is illegal)

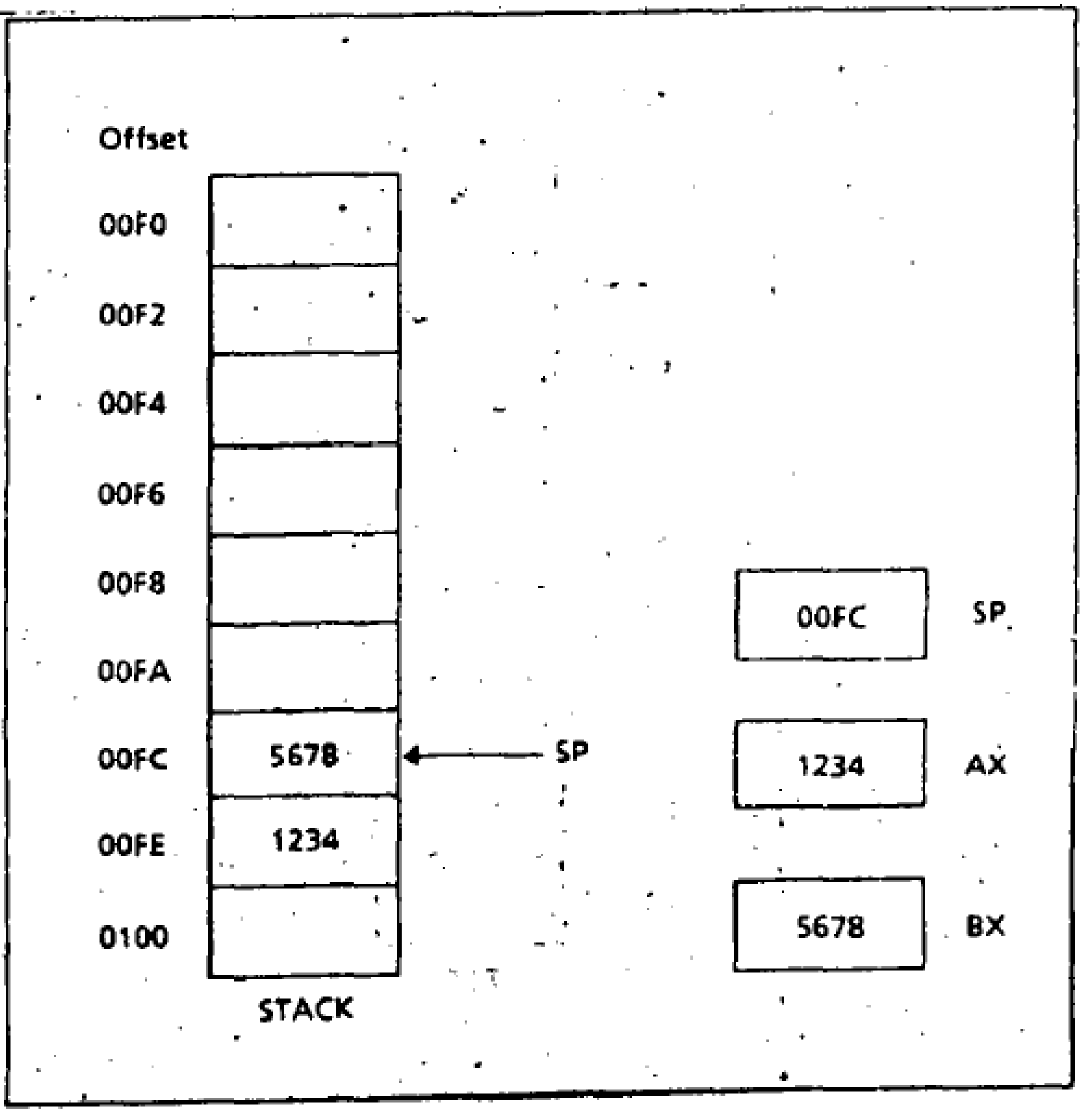
Empty Stack


Offset
00F0
00F2
00F4
00F6
00F8
00FA
00FC
00FE
0100
SP
STACK (empty)
0100
SP
1234
AX
5678
BX

After PUSH AX



After PUSH BX



POP and POPF

POP destination
destination = 16 bit register (except IP) or memory word

Execution of POP:

- The content of SS:SP (TOS) is moved to the destination.
- SP is increased by 2.

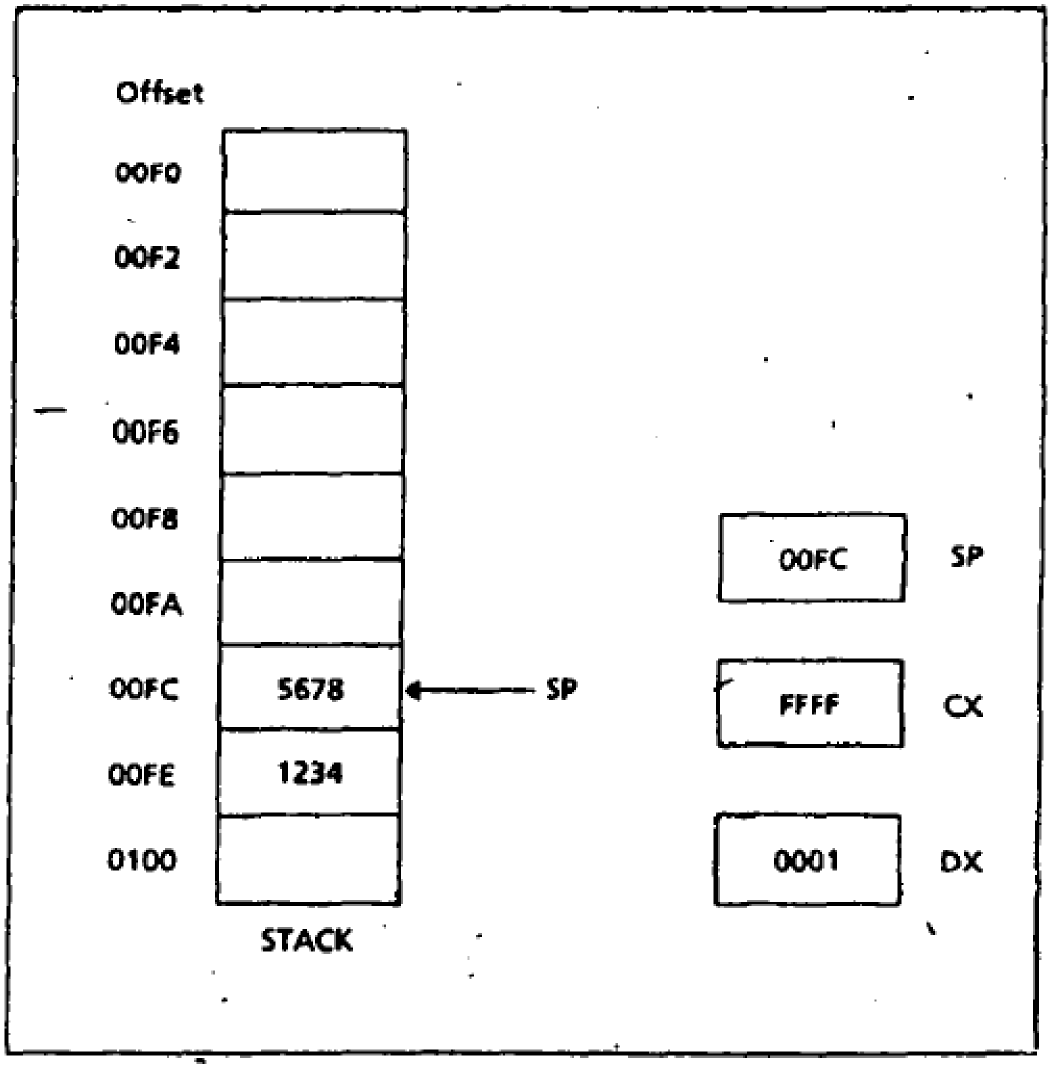
POPF

This instruction pops the TOS into the FLAGS register.

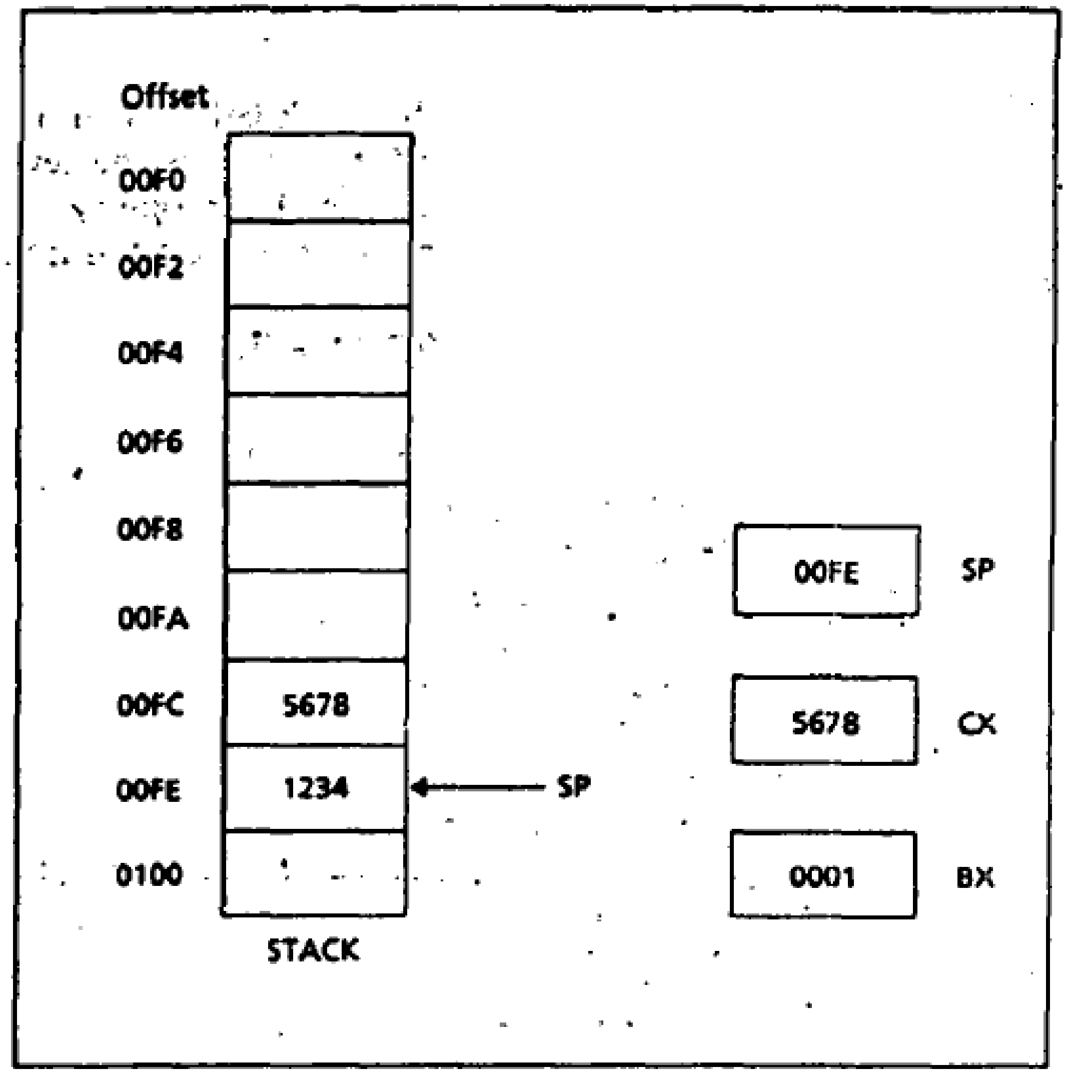
There is no effect of POP/POPF on the flags.

Illegal: POP DL (Byte instruction is illegal)

Before POP

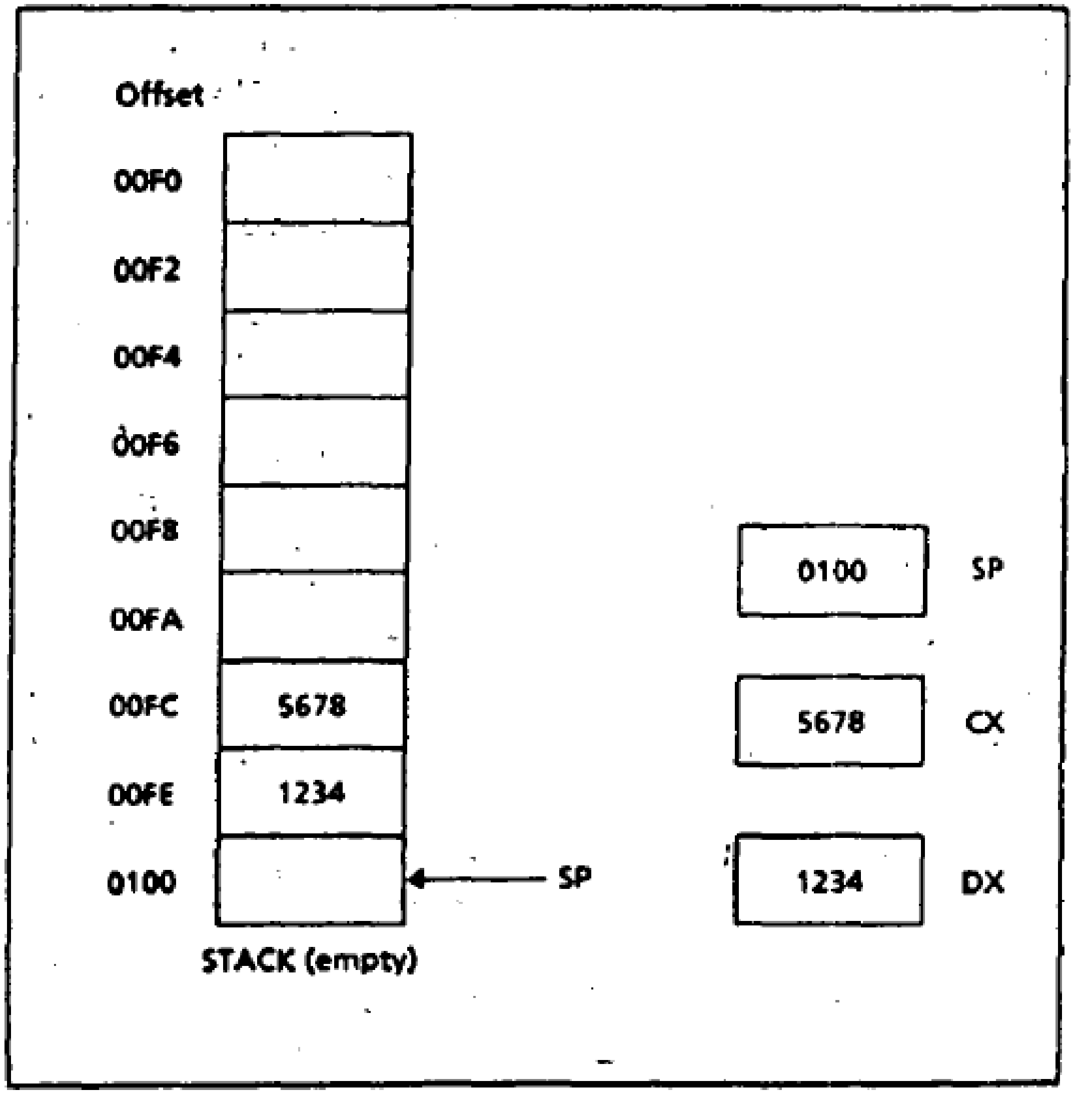

Offset
00F0
00F2
00F4
00F6
00F8
00FA
00FC
5678
SP
00FE
1234
0100
STACK
00FC
SP
FFFF
CX
0001
DX

After POP CX



Offset
00F0
00F2
00F4
00F6
00F8
00FA
00FC
5678
00FE
1234
SP
0100
STACK
00FE
SP
5678
CX
0001
BX

After POP DX



Offset
00F0
00F2
00F4
00F6
00F8
00FA
00FC
5678
00FE
1234
0100
SP
STACK (empty)
0100
SP
5678
CX
1234
DX

Procedure declaration

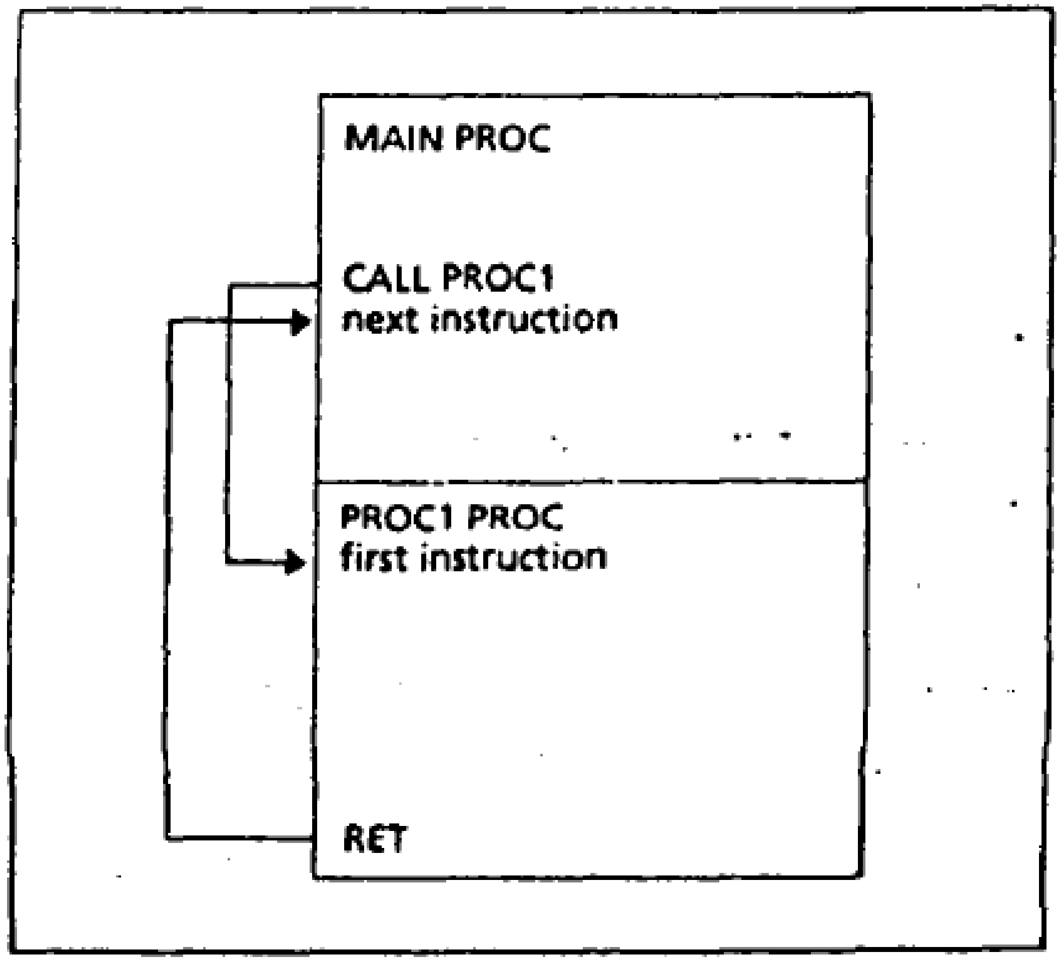
```
name PROC type
        ;body of the procedure

RET
name ENDP
```

name = user defined name of the procedure
type = (optional) NEAR or FAR

If type is omitted, NEAR is assumed.

NEAR means caller and called procedures are in the same segment. Far means caller and called procedures are in different segment.

Procedure call and return



CALL

Direct procedure call:
CALL name
name = name of the procedure

Indirect procedure call:
CALL address
address = a register or memory location containing the address of a procedure.

Execution of CALL

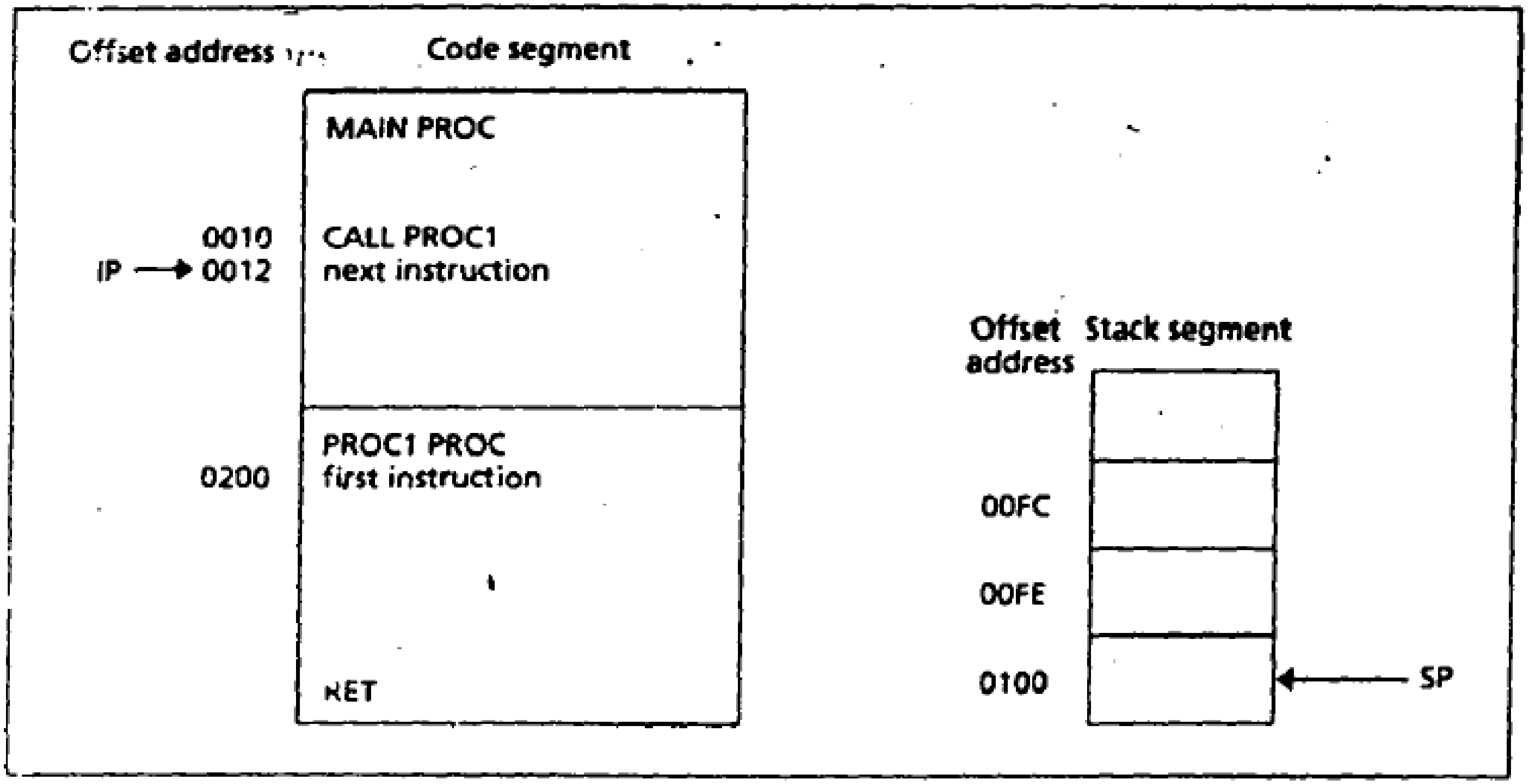
- Return address of the caller is saved on the stack.

  return address = offset of the next instruction after the CALL statement in the caller procedure.
- IP gets the offset address of the first instruction of called procedure.

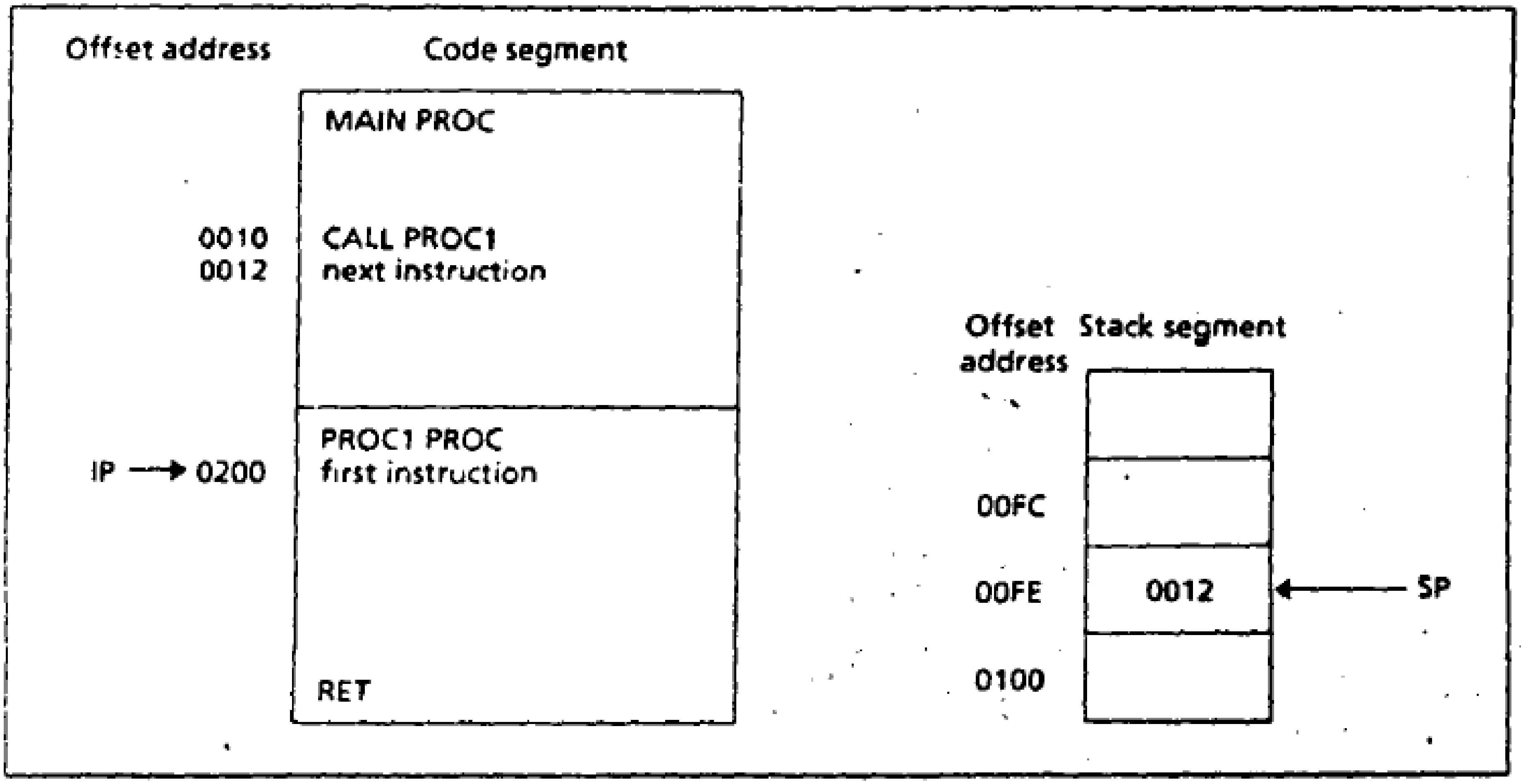
  CS:IP = segment:offset of the first instruction of the called procedure.

  Control goes to the called procedure.

Before CALL


Offset address
Code segment
MAIN PROC
0010
CALL PROC1
IP
0012
next instruction
0200
PROC1 PROC
first instruction
RET
Offset address
Stack segment
00FC
00FE
0100
SP

After CALL



Offset address
Code segment
MAIN PROC
0010
CALL PROC1
0012
next instruction
IP
0200
PROC1 PROC
first instruction
RET
Offset address
Stack segment
00FC
00FE
0012
0100
SP

RET

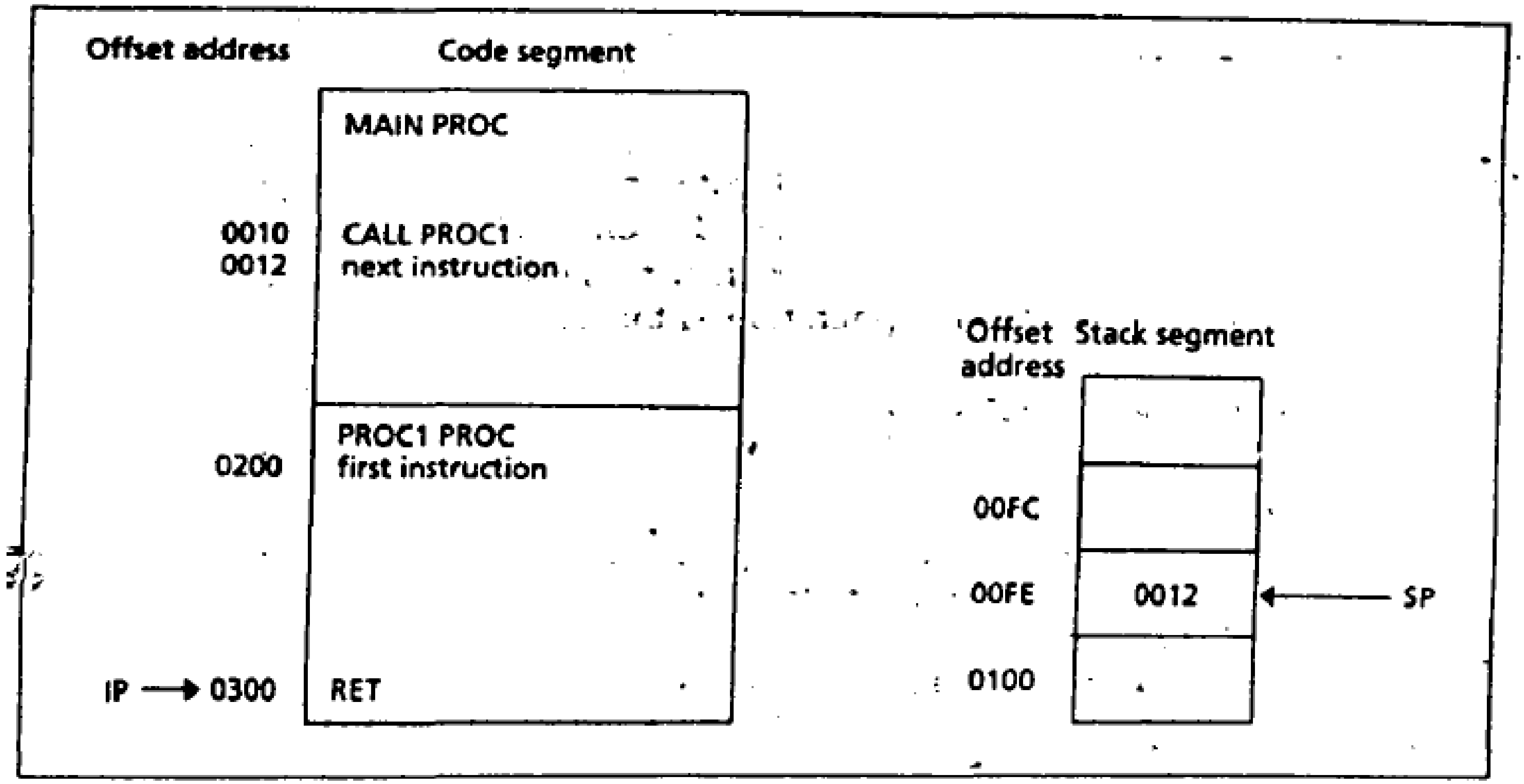
RET pop_value

pop_value = (optional)

Execution of RET:

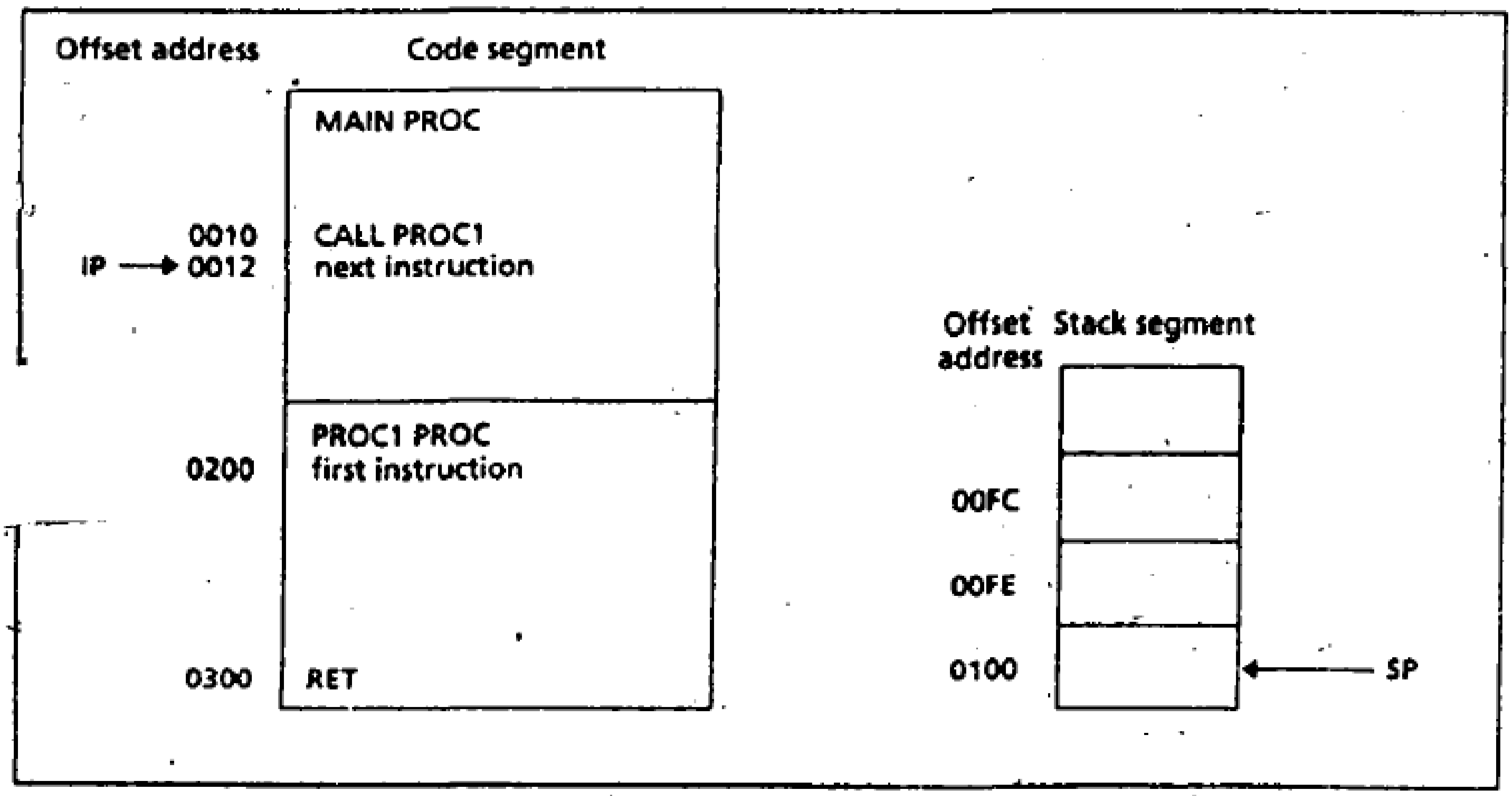
- IP gets the value of TOS. So, CS:IP = segment:offset of the return address. Thus, control goes back to the caller program

Before RET



Offset address
Code segment
MAIN PROC
0010
CALL PROC1
0012
next instruction
0200
PROC1 PROC
first instruction
IP
0300
RET
Offset address
Stack segment
00FC
00FE
0012
SP
0100

After RET



Offset address
Code segment
MAIN PROC
0010
CALL PROC1
IP
0012
next instruction
0200
PROC1 PROC
first instruction
0300
RET
Offset address
Stack segment
00FC
00FE
0100
SP