The Stack and Introduction to Procedures

The Stack

- The stack segment of a program is used for temporary storage of data and addresses
- A stack is a one-dimensional data structure
- Items are added to and removed from one end of the structure using a "Last In - First Out" technique (LIFO)
- The statement .STACK 100H in your program sets aside a block of 256 bytes of memory to hold the stack
- PUSH and POP instructions that add and remove words from the stack

The Stack (cont'd)

.STACK 100H

When the program assembled and loaded in the memory:

- SS will contain the segment number of stack segment.
- SP is initialized to 100H, which is represent the empty stack position
- When the stack is not empty, SP contain the offset address of the top of the stack.

If stack is empty, SP has a value of 100H; otherwise it has a value between 0000-00FEH

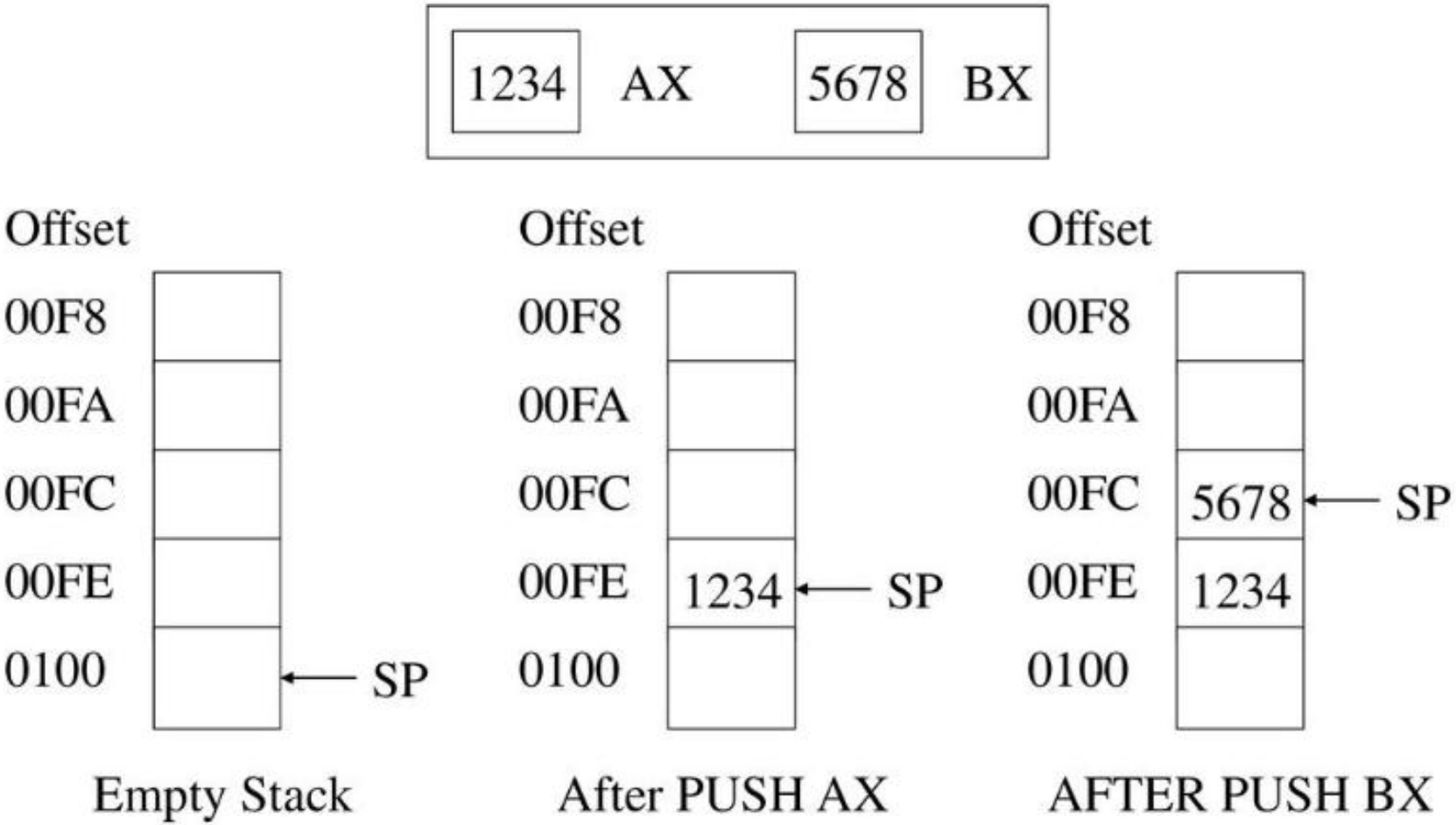
The PUSH Instruction

- To add a new word to the stack we PUSH it on.
- Syntax:

  PUSH source

  16-bit register or memory location
- After execution of PUSH:
  - SP is decremented by 2.
  - A copy of the source content is moved to the address specified by SS:SP.
  - The source is unchanged.

The PUSH Instruction



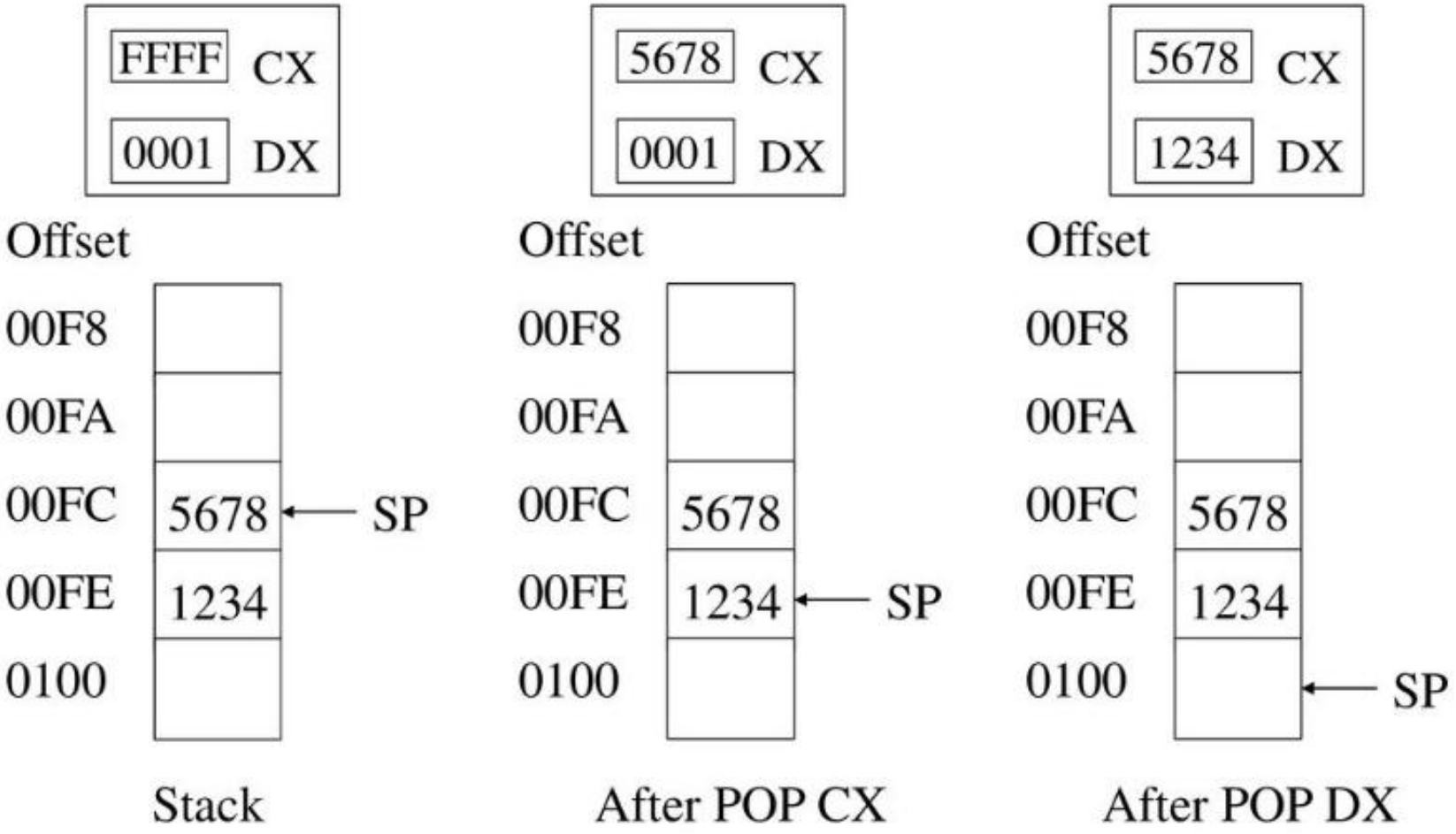
The POP Instruction

- To add a new word to the stack we POP it on.
- Syntax:

  POP destination

  16-bit register or memory location
- After execution of POP:
  - The content of SS:SP (the top of the stack)is moved to the destination
  - SP is increased by 2.

The POP Instruction



PUSH and POP Instruction

- ***Restrictions:***
    1. PUSH and POP work only with words
    2. Byte and immediate data operands are illegal

Example

- AX = 3245H
- BX = 1234H
- CX = ABCDH
- SP = FEH

```
PUSH AX
PUSH CX
POP BX
```

AX =?
BX =?
CX =?
SP =?

FLAGS Register and Stack

- PUSHF
  - pushes (copies) the contents of the FLAGS register onto the stack. It has no operands
- POPF
  - pops (copies) the contents of the top word in the stack to the FLAGS register. It has no operands

- **NOTES:**
  - PUSH, POP, and PUSHF do not affect the flags.
  - POPF could theoretically change all the flags because it resets the FLAGS REGISTER to some original value that you have previously saved with the PUSHF instruction

Exercise

Write assembly code that uses the stack operations to swap the content of AX and DX.

```
PUSH AX
PUSH DX
POP AX
POP DX
```

Procedures

- Top-down program design
  - Decompose the original problem into a series of subproblems that are easier to solve than the original problem
- Subproblems in assembler language can be structured as a collection of procedures
- Main procedure contains the entry point to the program and can call one of the other procedures using a CALL statement
- It is possible for a called sub-procedure to call other procedures
- It is also possible for a called sub-procedure to call itself (recursion)!

Procedures (cont'd)

- When the instructions in a called procedure have been executed, the called procedure usually returns control to the calling procedure at the next sequential instruction after the CALL statement
- Programmers must devise a way to communicate between procedures, there are no parameter lists.
- When writing a procedure, do NOT PUSH or POP any registers in which you intend to return output!!

PROC Instruction

- PROC instruction establishes a procedure
- Procedure declaration syntax:

```
name        PROC
;  body of the procedure
           RET
name   ENDP
```

- name is a user-defined variable.
- RET instruction causes control to transfer back to the calling Procedure.
- Every procedure should have a RET coded somewhere within the procedure - usually the last instruction in a procedure

CALL Instruction

- A CALL instruction invokes a procedure
- SYNTAX: CALL name (direct CALL)

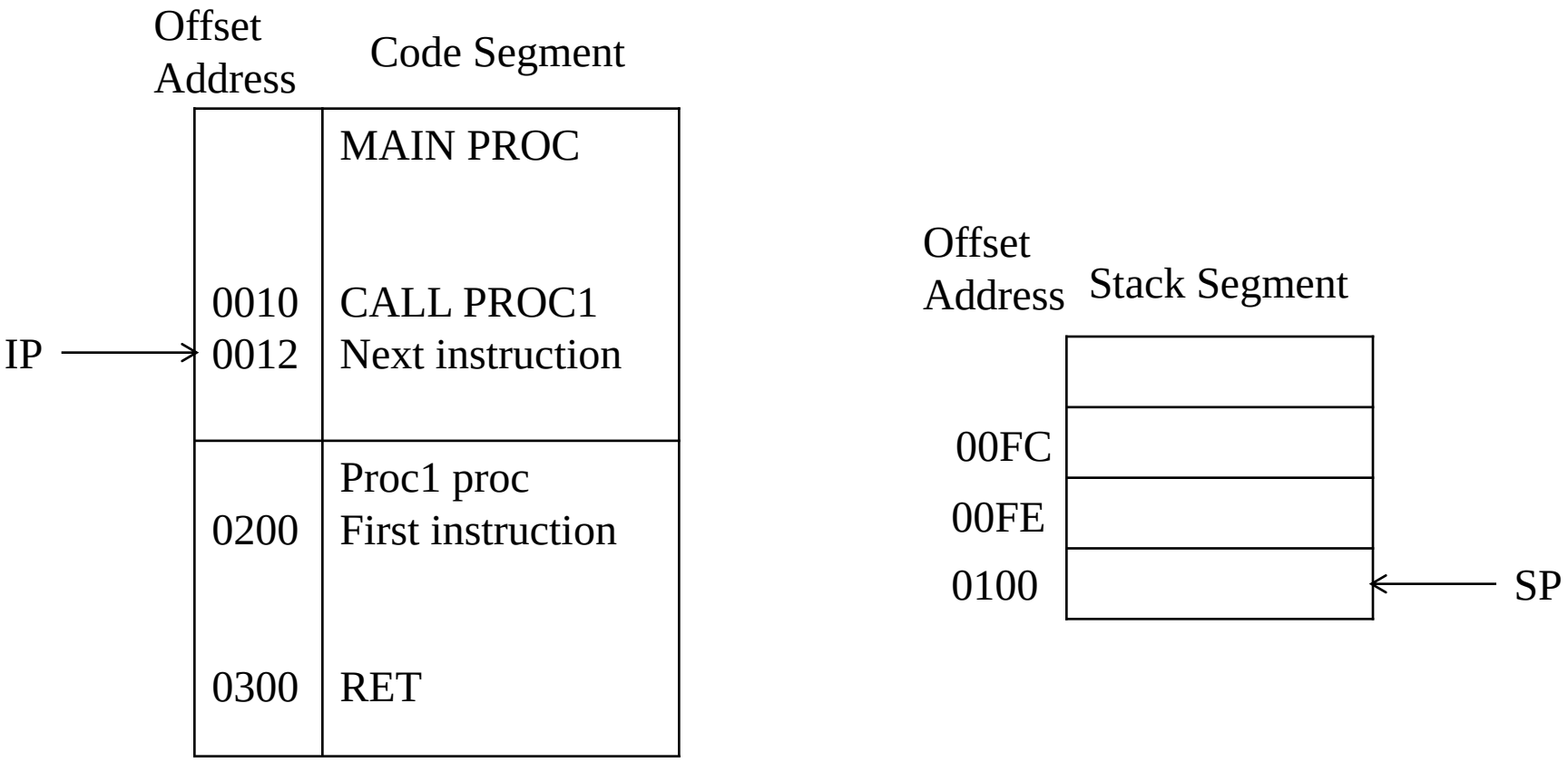
  where name is the name of a procedure.

- Executing a CALL instruction causes the following to happen:

  – The return address of the CALLing program which is in the IP register is pushed (saved) on the STACK. This saved address is the offset of the next sequential instruction after the CALL statement (CS:IP)

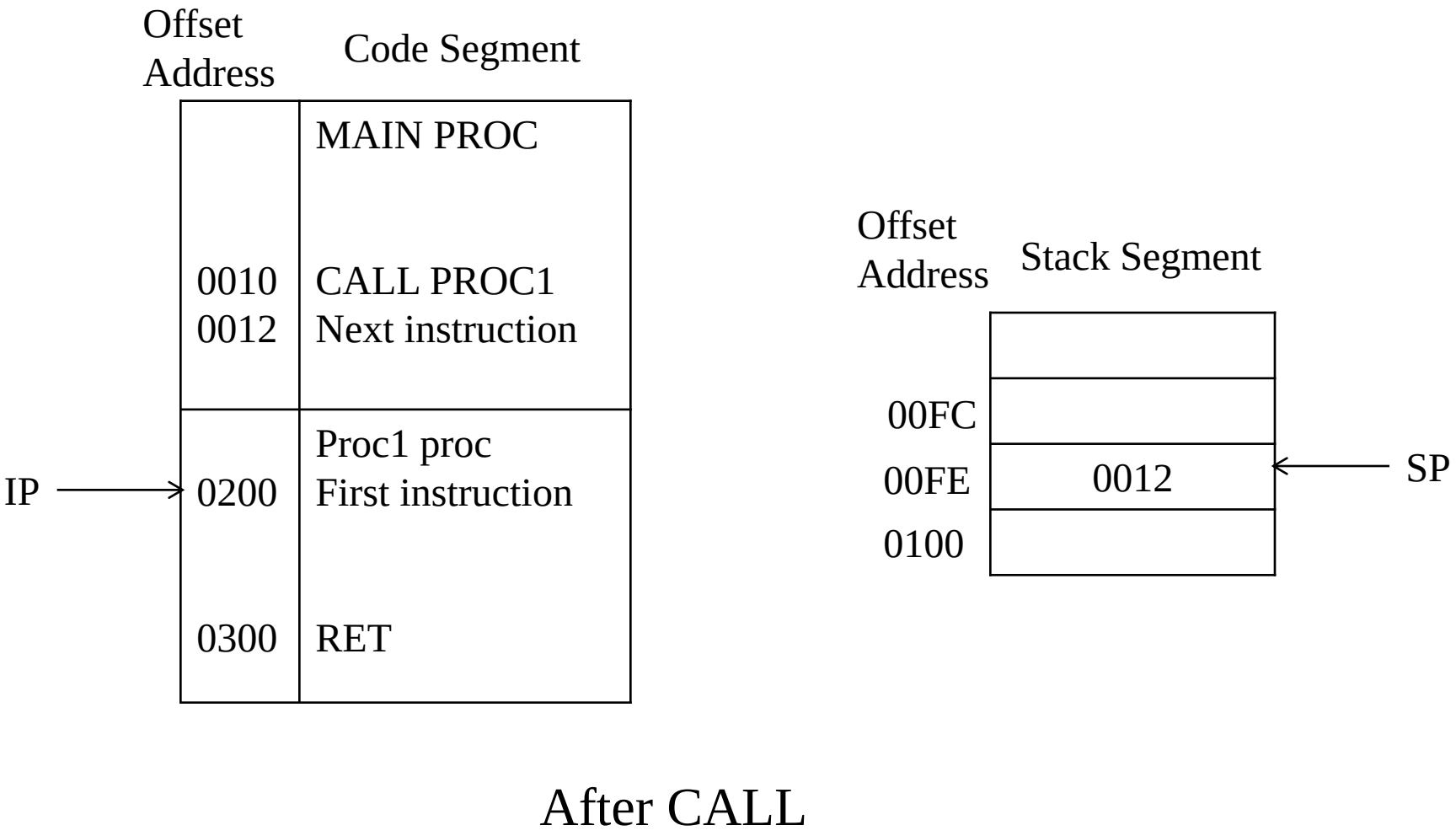
  – The IP then gets the offset address of the first instruction in the procedure

CALL Instruction



Before CALL

CALL Instruction

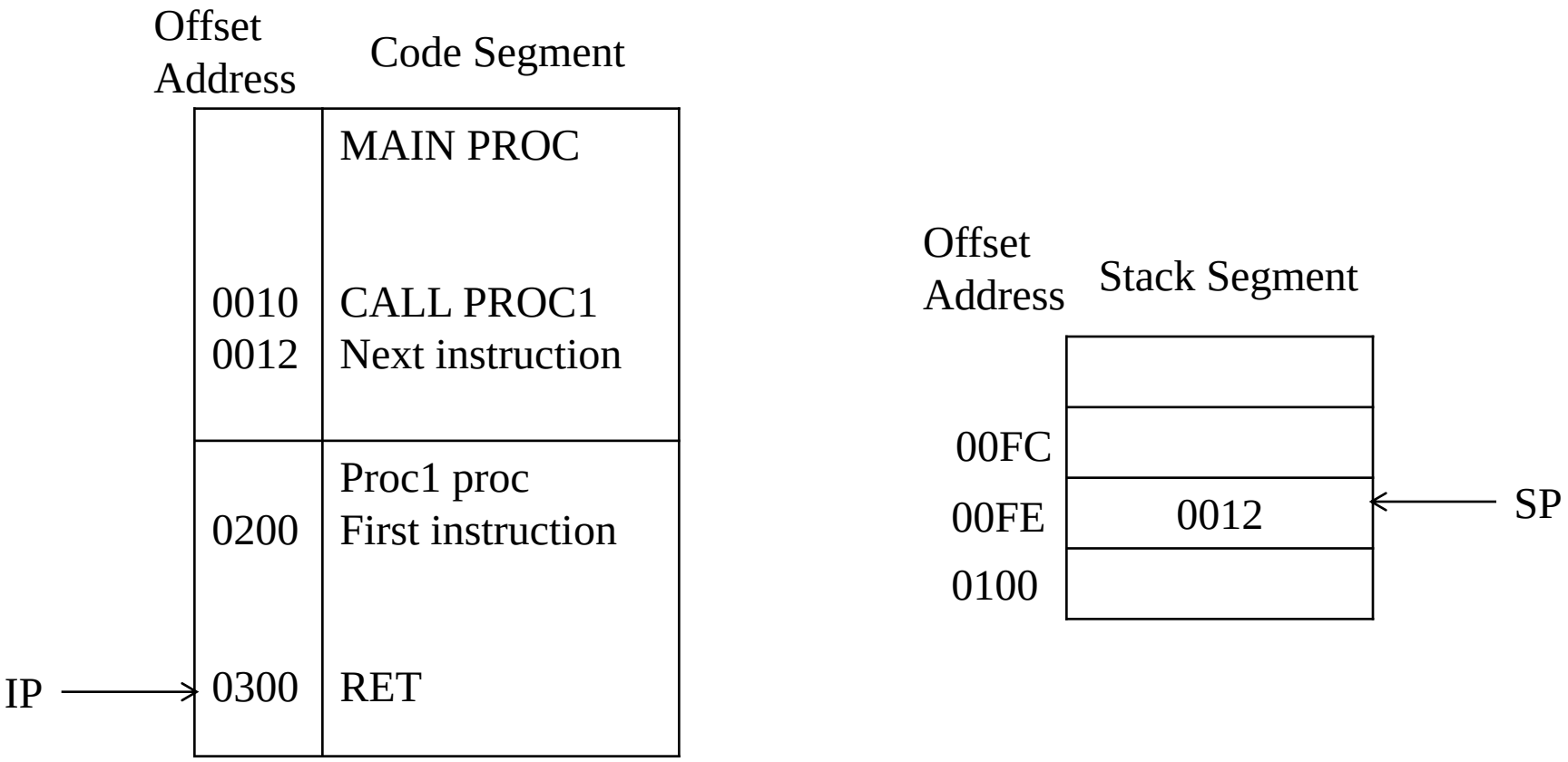


After CALL

RET Instruction

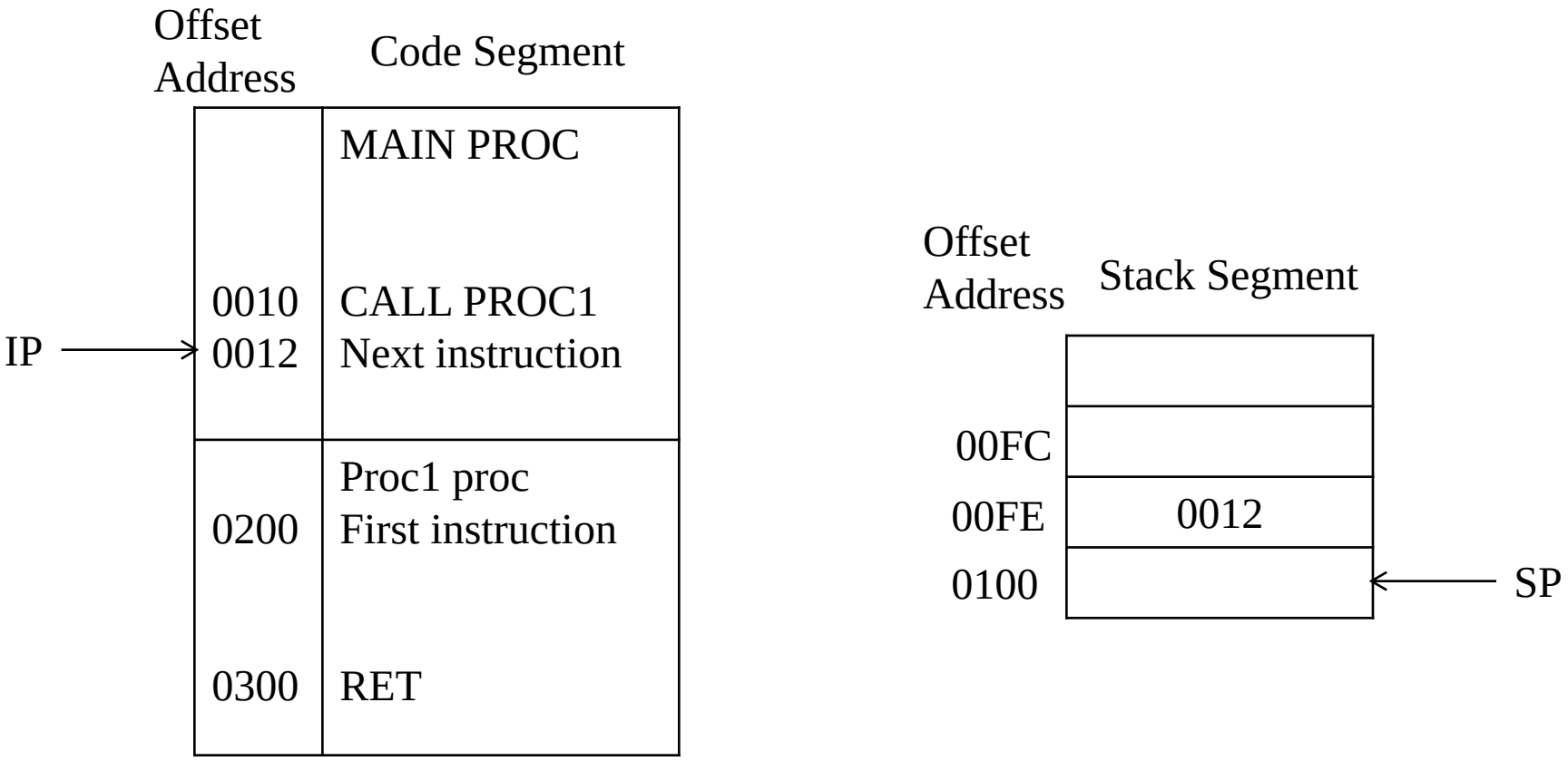
- RET statement cause the stack to be popped into IP. Procedures typically end with a RET statement.
- Syntax: RET
- Once the RET is executed, CS:IP now contains the segment offset of the return address and control returns to the calling program
- In order for the return address to be accessible, each procedure must ensure that the return address is at the top of the stack when the RET instruction is executed.

CALL Instruction



Before RET

CALL Instruction



After RET

THANK YOU