Chapter-8

The Stack and Introduction to Procedures

Overview

The stack segment of a program is used for temporary storage of data and addresses. Stack is used to implement procedures.

8.1 The Stack

A stack is one-dimensional data structure. It is processed in a “last-in, first-out” manner. The most recent addition to the stack is called **top of the stack**.

A program must set aside a block of memory to hold the stack. For example,

```
.STACK   100H
```

When the program is assembled and loaded in memory, SS will contain the segment number of the stack segment. For the preceding stack declaration, SP, the stack pointer, is initialized to 100H. This represents the empty stack position. When the stack is not empty, SP contains the offset address of the top of the stack.

PUSH and PUSHF

To add a new word to the stack we **PUSH** it on. The syntax is

```
PUSH  source
```

where source is a 16-bit register or memory word. For example,

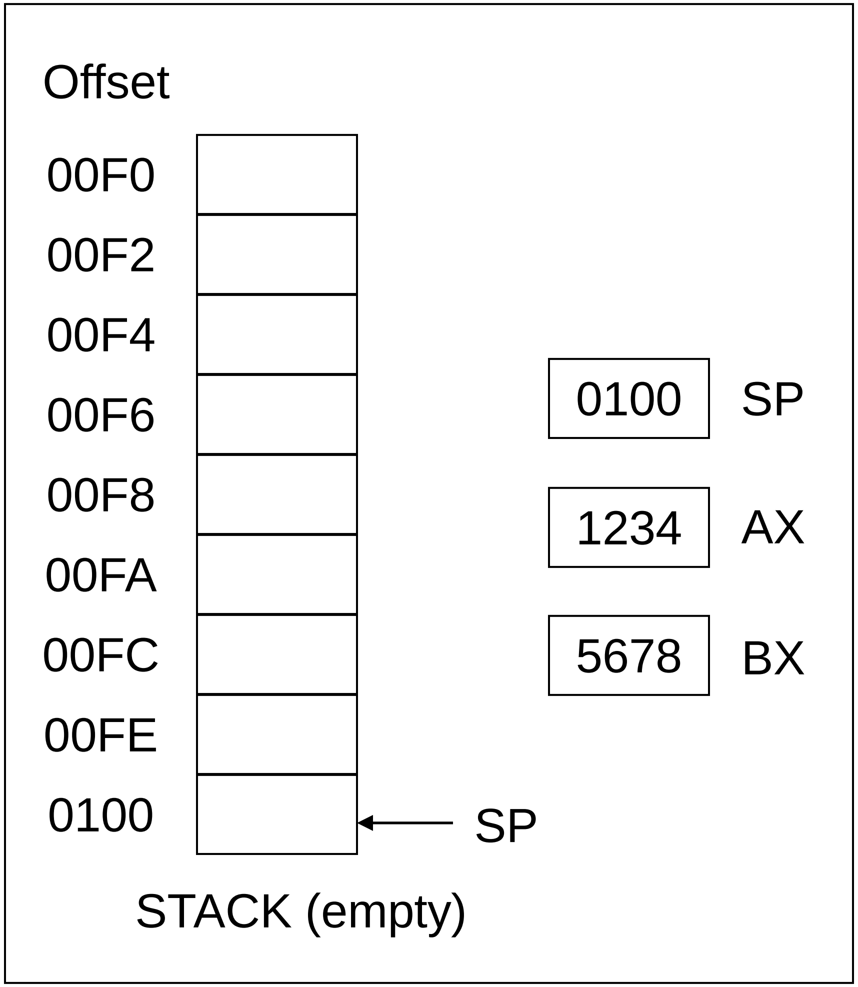
```
PUSH  AX
```

Execution of PUSH causes the following to happen:

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

The instruction **PUSHF**, which has no operands, pushes the contents of the FLAGS register onto the stack.

Figure 8.1A Empty Stack



Offset

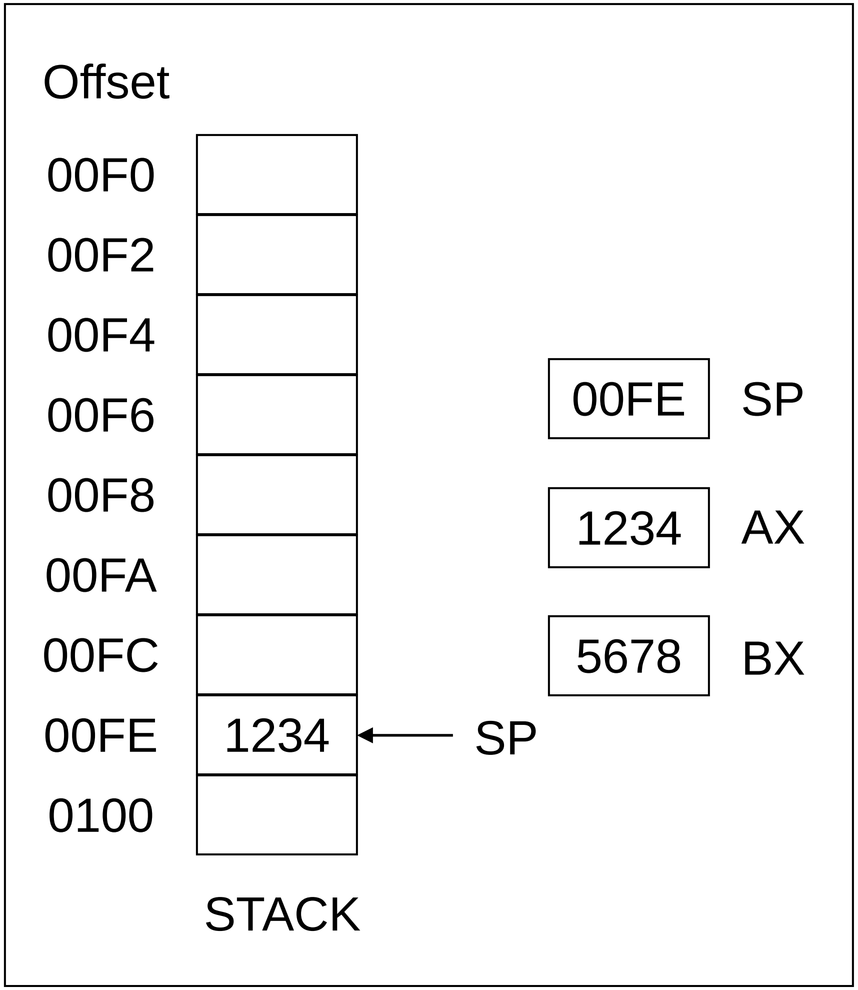
Offset	
00F0	
00F2	
00F4	
00F6	
00F8	
00FA	
00FC	
00FE	
0100	

← SP

STACK (empty)

0100	SP
1234	AX
5678	BX

Figure 8.1B After PUSH AX



Offset

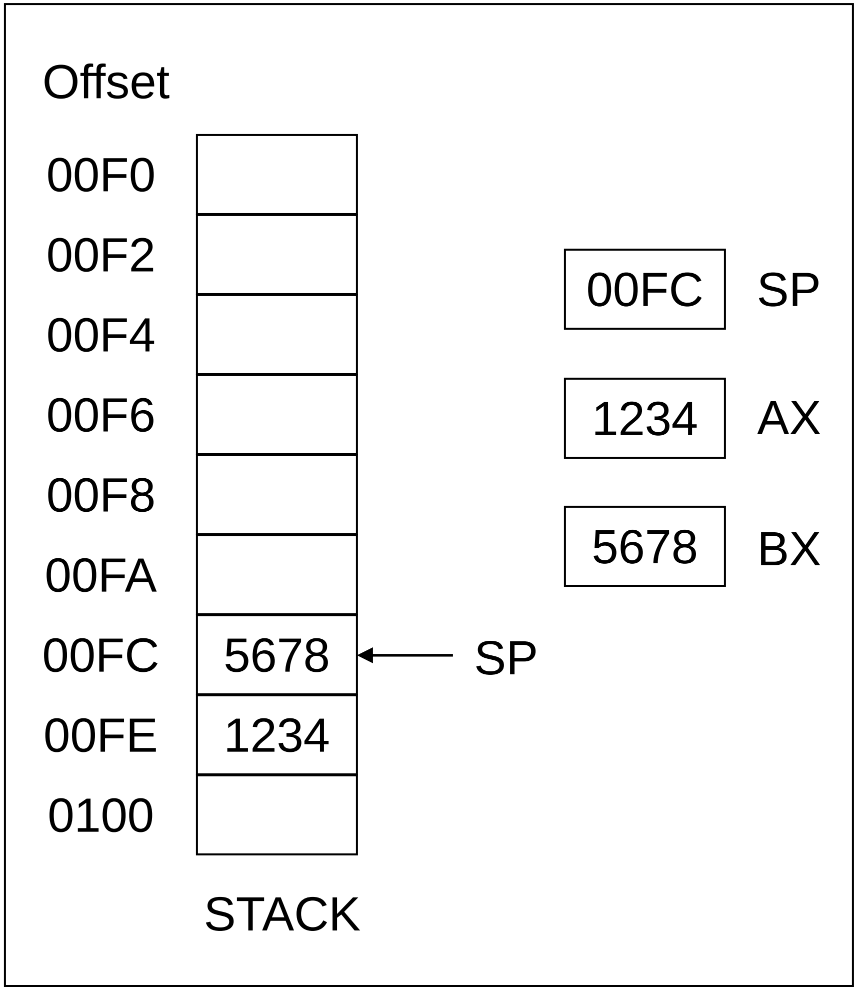
Offset	
00F0	
00F2	
00F4	
00F6	
00F8	
00FA	
00FC	
00FE	1234
0100	

← SP

STACK

00FE	SP
1234	AX
5678	BX

Figure 8.1C After PUSH BX



Offset

00F0

00F2

00F4

00F6

00F8

00FA

00FC 5678 ← SP

00FE 1234

0100

STACK

00FC SP

1234 AX

5678 BX

Initially, SP contains the offset address of the memory location immediately following the stack segment.

POP and POPF

To remove top item from the stack we **POP** it. The syntax is

POP destination

where destination is a 16-bit register (except IP) or memory word. For example,

POP BX

Execution of POP causes the following to happen:

1. The content of SS:SP (the top of the stack) is moved to the destination.
2. SP is increased by 2.

Note that PUSH and POP are word operations, so a byte instruction such as

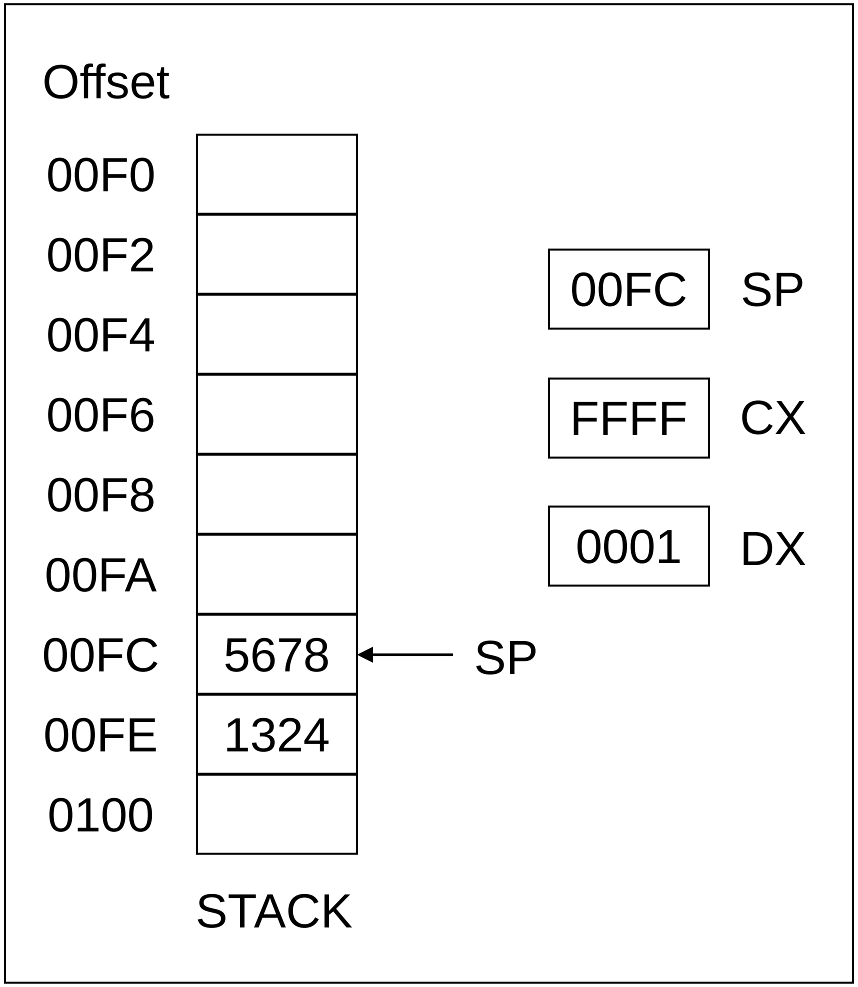
Illegal: PUSH DL

Is illegal. So is a push of immediate data, such as

Illegal: PUSH 2

Note: an immediate data push is legal for the 80186/80486 processors.

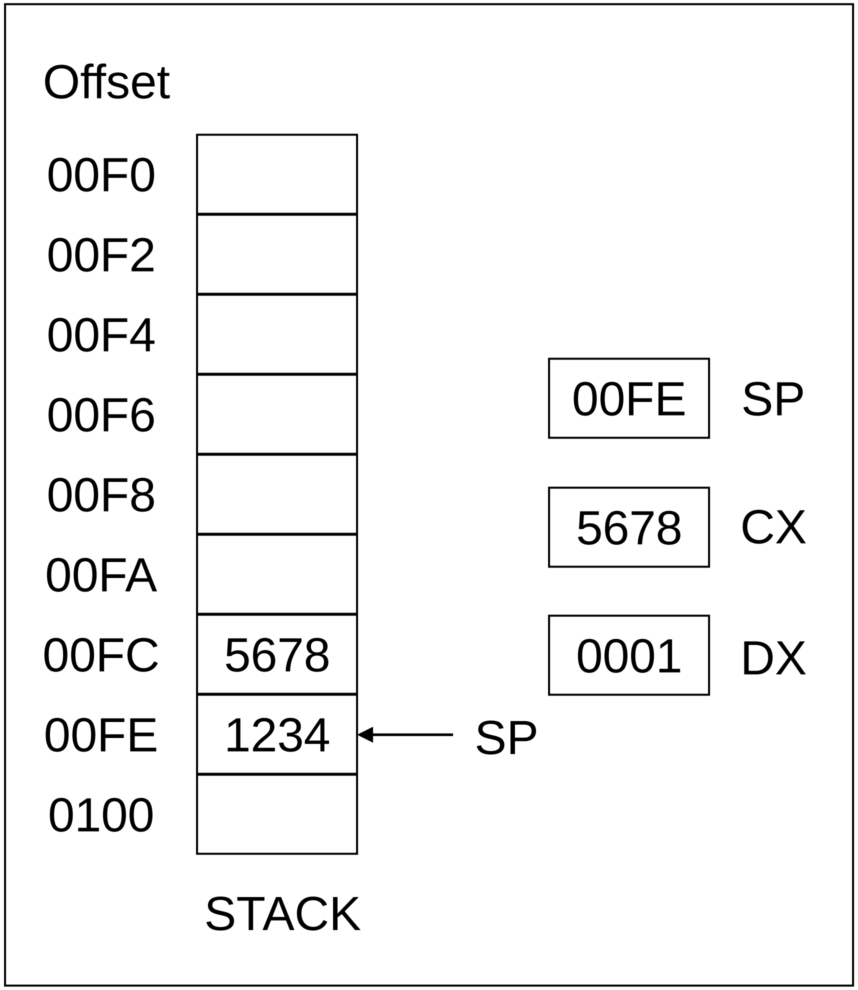
Figure 8.2A Before POP



Offset	
00F0	
00F2	
00F4	
00F6	
00F8	
00FA	
00FC	5678 ← SP
00FE	1324
0100	
	STACK

00FC	SP
FFFF	CX
0001	DX

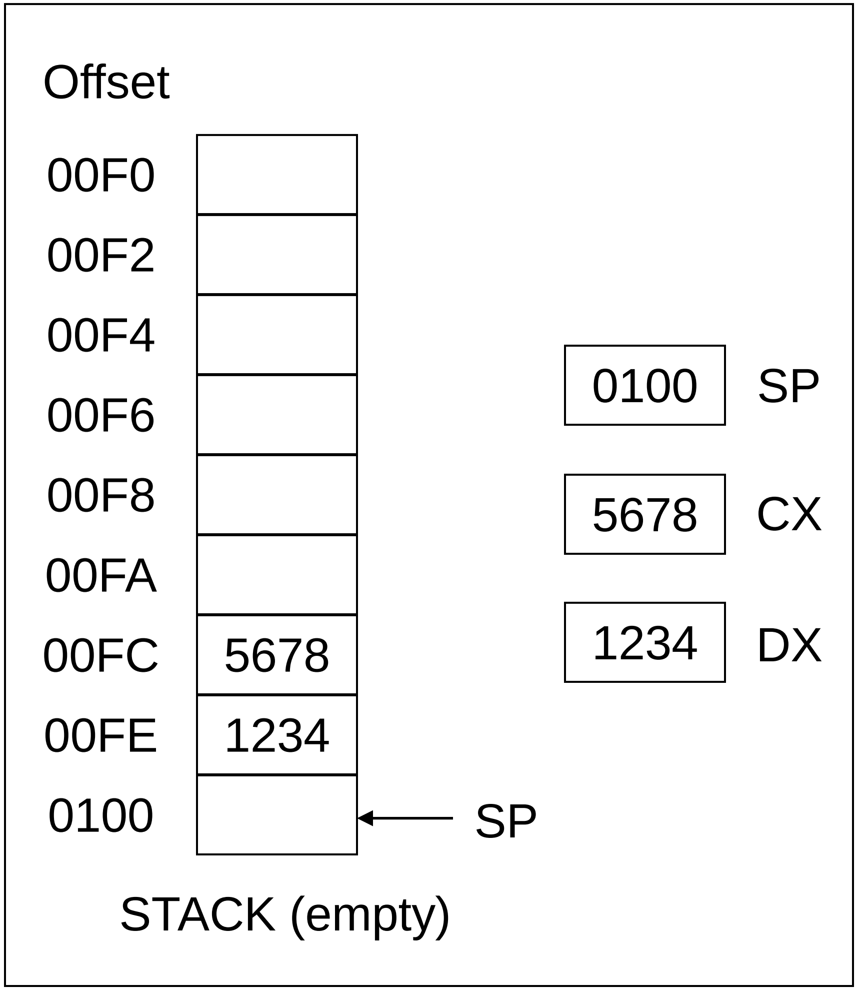
Figure 8.2B After POP CX



Offset	
00F0	
00F2	
00F4	
00F6	
00F8	
00FA	
00FC	5678
00FE	1234 ← SP
0100	
	STACK

00FE	SP
5678	CX
0001	DX

Figure 8.2C After POP DX



Offset	
00F0	
00F2	
00F4	
00F6	
00F8	
00FA	
00FC	5678
00FE	1234
0100	

0100	SP
5678	CX
1234	DX

STACK (empty)

8.2 A Stack Application

```
TITLE  PGM8_1: REVERSE INPUT
.MODEL  SMALL
.STACK  100H
.CODE
MAIN  PROC
MOV AH, 2
MOV DL, '?'
INT 21H
XOR CX, CX
MOV AH, 1
INT 21H
WHILE_:
CMP AL, 0DH
JE  END_WHILE
PUSH AX
INC CX
INT 21H
JMP WHILE_
END_WHILE:
MOV AH,2
MOV DL, 0DH
INT 21H
MOV DL, 0AH
INT 21H
JCXZ EXIT
TOP:
POP DX
INT 21H
LOOP TOP
EXIT:
MOV AH, 4CH
INT 21H
MAIN ENDP
END MAIN
```

8.3 Terminology of Procedures

An assembly language program can be structured as a collection of procedures. One of the procedures is the main procedure, and it contains the entry point to the program.

When one procedure calls another, control transfers to the called procedure.

Procedure Declaration

The syntax of procedure declaration is the following:

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

Name is the user-defined name of the procedure. The optional type is **NEAR** or **FAR** (if type is omitted, NEAR is assumed).

NEAR Type Procedure

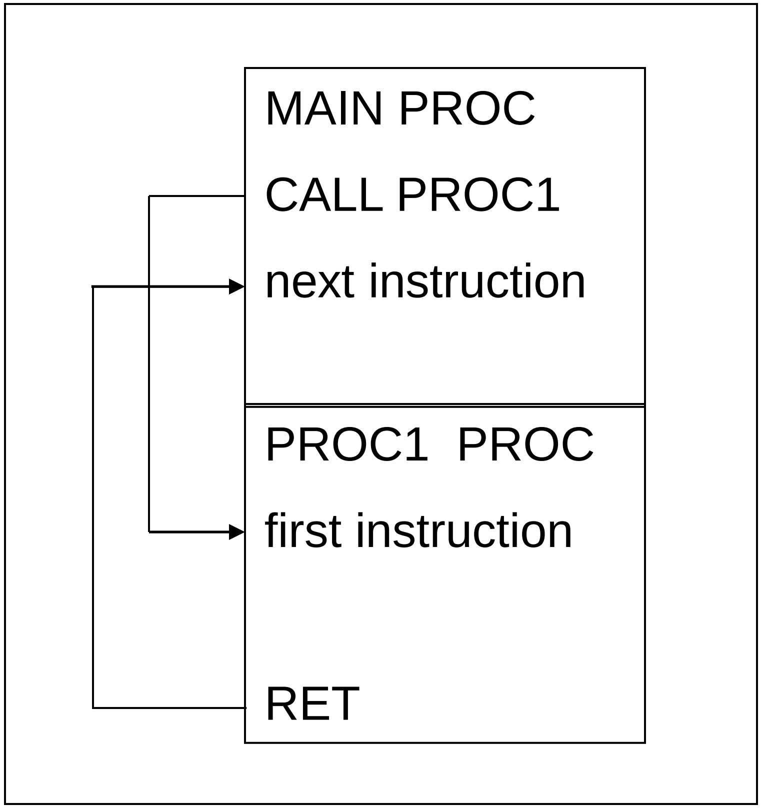
For **NEAR** type procedure the statement that calls the procedure is in the same segment as the procedure itself.

FAR Type Procedure

For **FAR** type procedure the calling statement is in a different segment than the called procedure.

Figure 8.3

Procedure Call and Return



RET

The **RET** (return) instruction causes control transfer back to the calling procedure. Every procedure (except the main procedure) should have a RET someplace; usually it’s the last statement in the procedure.

Communication Between Procedures

Unlike high-level language procedures, assembly language procedures do not have parameter lists, so it’s up to the programmer to devise a way for procedures to communicate. For example, if there are only a few input and output values, they can be placed in registers.

8.4 CALL and RET

To invoke a procedure, the **CALL** instruction is used. There are two kinds of procedure calls, **direct** and **indirect**.

Direct Procedure Call

The syntax of direct procedure call is

CALL name

where name is the name of a procedure.

Indirect Procedure Call

The syntax of an indirect procedure call is

CALL address_expression

where address_expression specifies a register or memory location containing the address of a procedure.

Executing a CALL instruction causes the following to happen:

1. The return address to the calling program is saved on the stack. This is the offset (IP) of the next instruction after the CALL statement.

2. IP gets the offset address of the first instruction of the procedure.

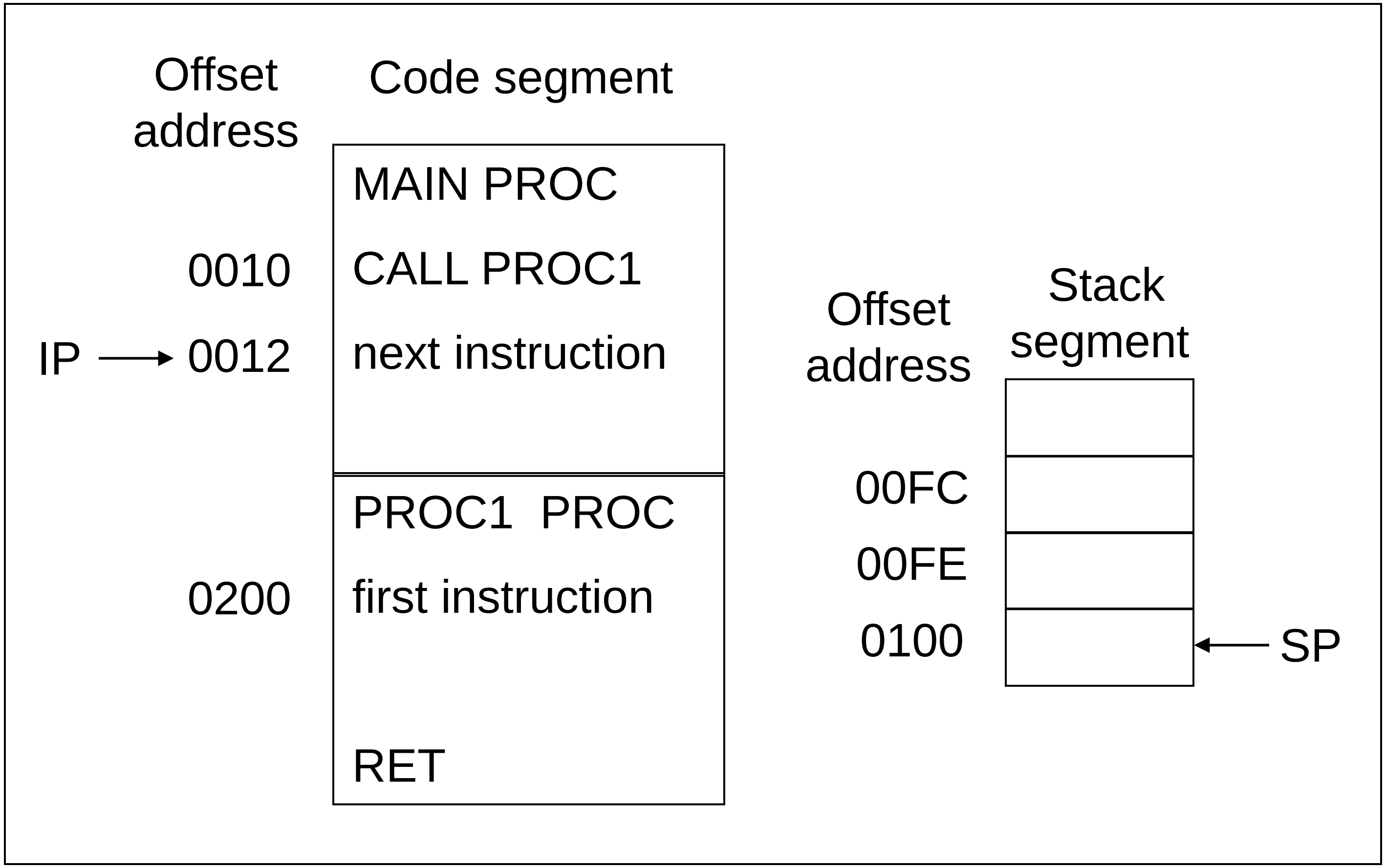
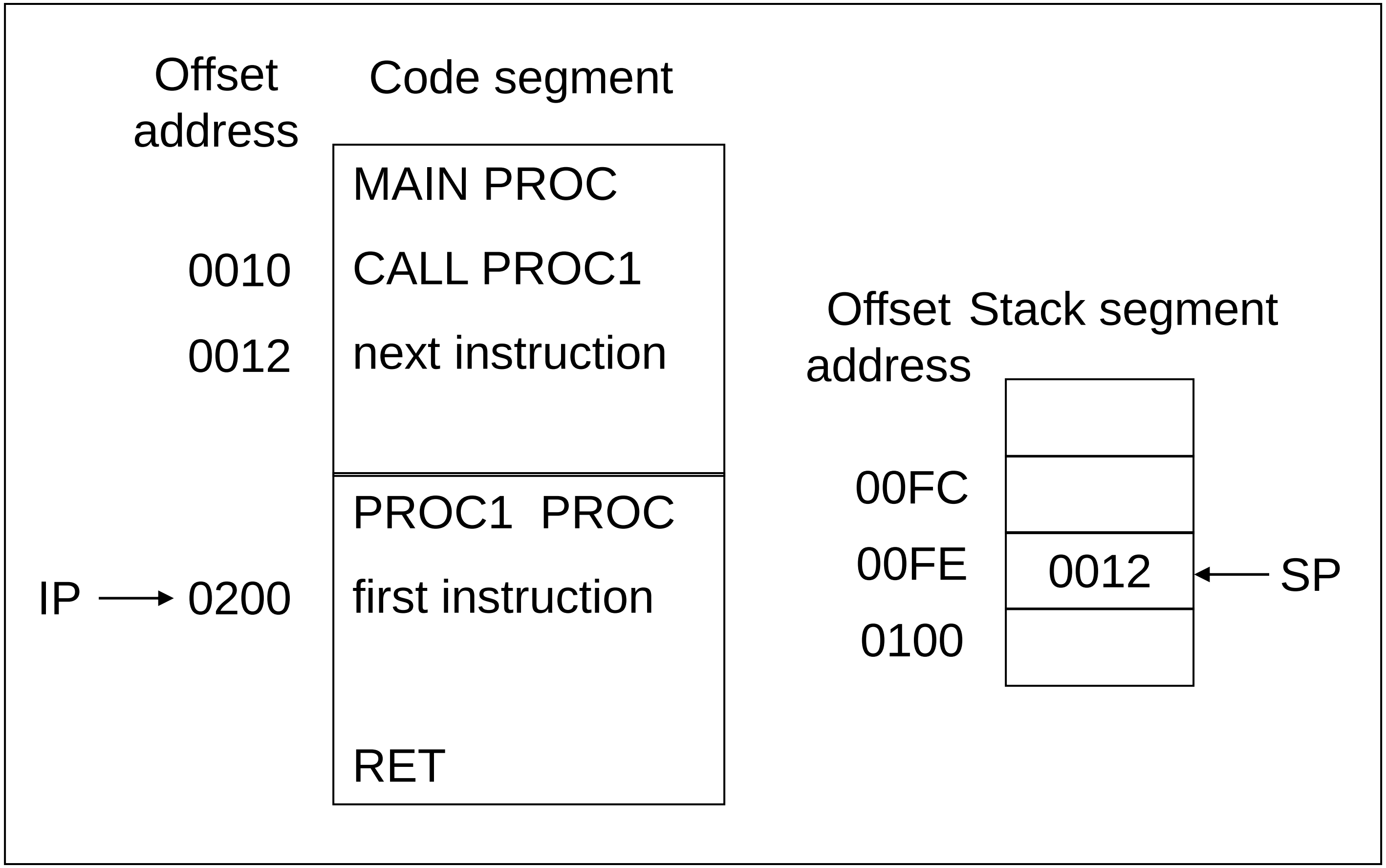


Figure 8.4A Before CALL



Offset address

Code segment

MAIN PROC

0010 CALL PROC1

0012 next instruction

PROC1 PROC

IP → 0200 first instruction

RET

Offset address

Stack segment

00FC

00FE 0012 ← SP

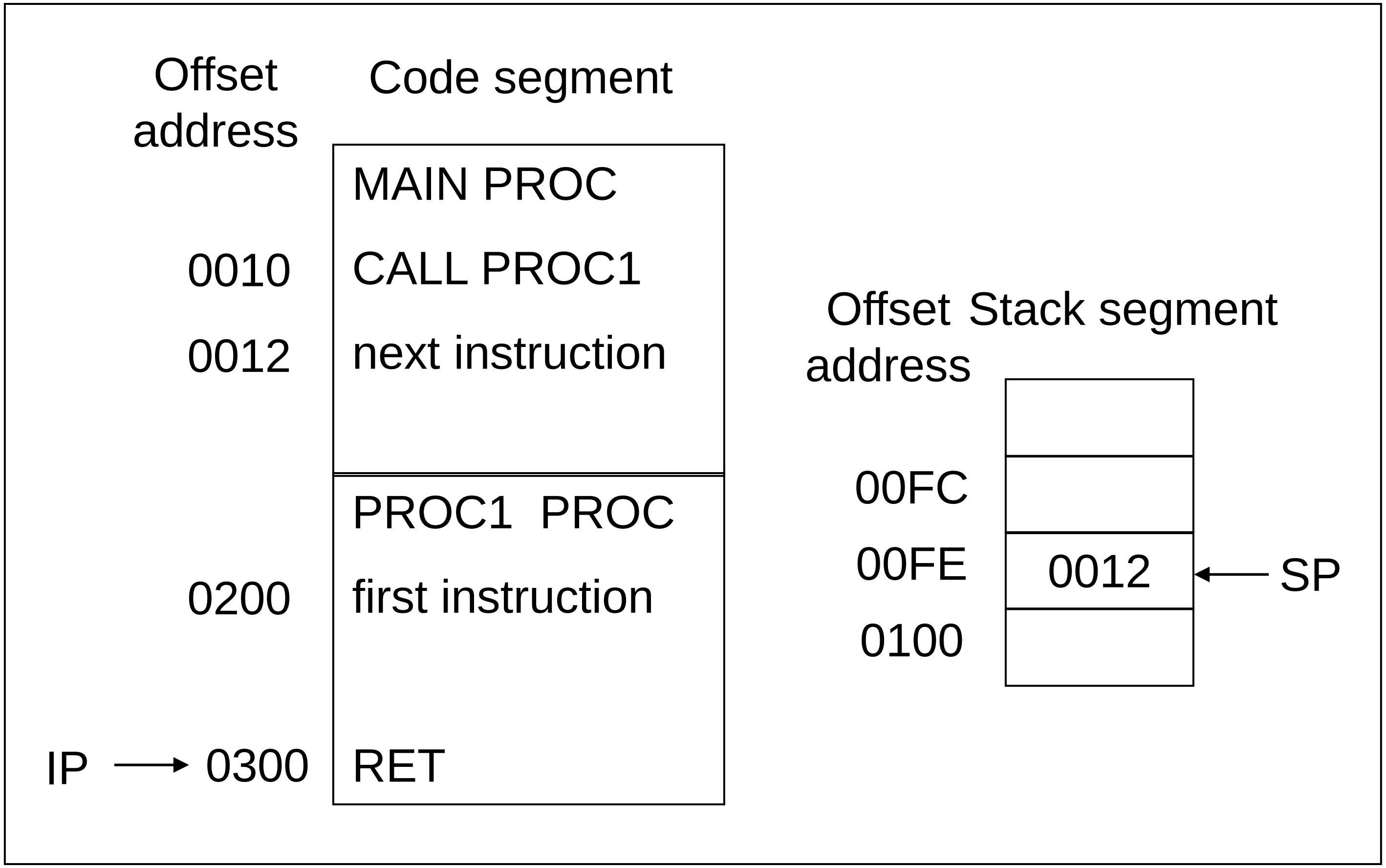
0100

Figure 8.4B After CALL

To return from a procedure, the following instruction is executed:

RET pop_value

The integer argument pop_value is optional. Execution of RET causes the stack to be popped into IP. If a pop_value N is specified then an N bytes from the stack.



Offset address

Code segment

MAIN PROC

0010 CALL PROC1

0012 next instruction

PROC1 PROC

0200 first instruction

IP → 0300 RET

Offset address

Stack segment

00FC

00FE 0012 ← SP

0100

Figure 8.5A Before RET

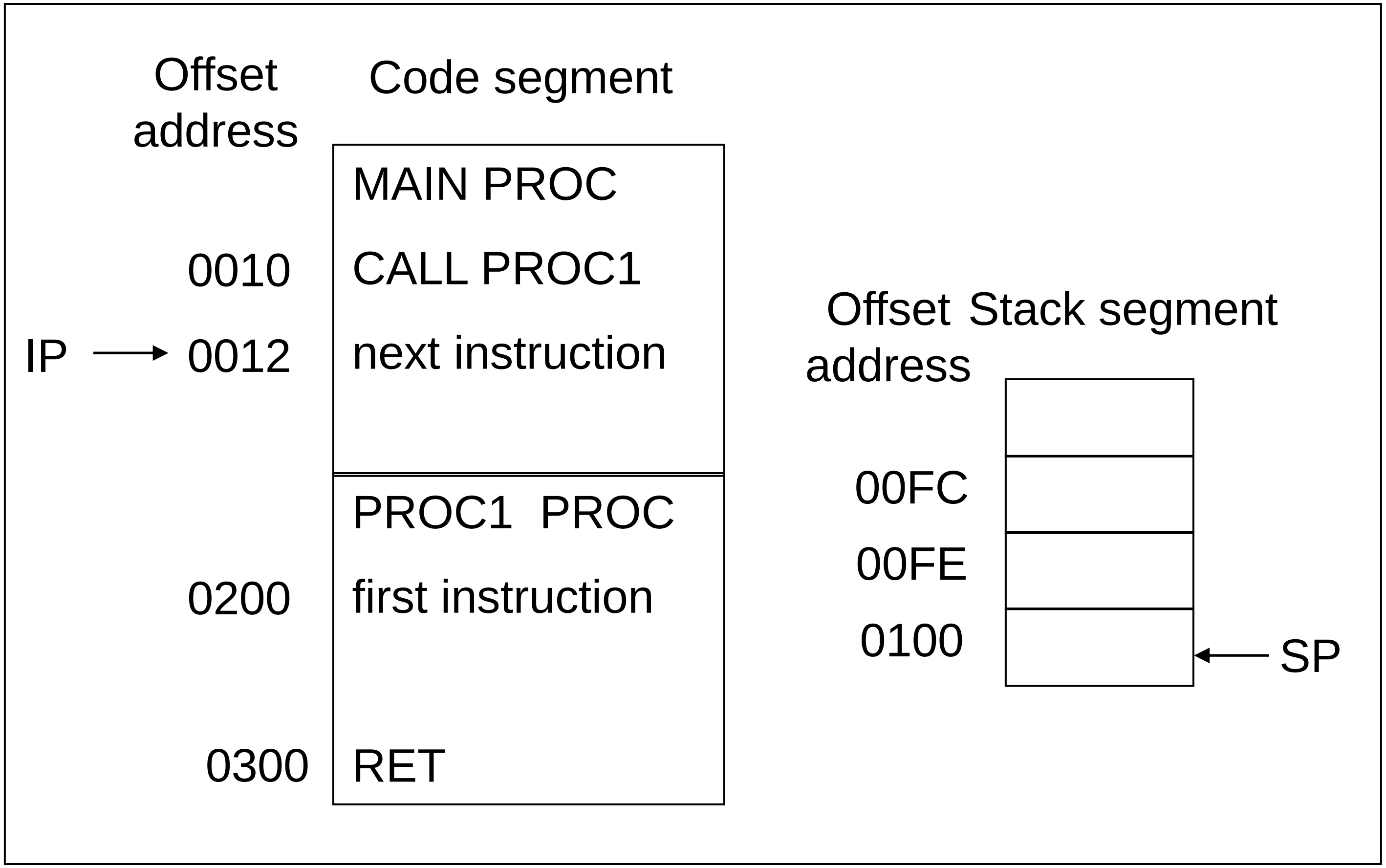


Figure 8.5B After RET

8.5 An Example of a Procedure

TITLE PGM8_2: MULTIPLICATION BY ADD AND SHIFT

```
.MODEL  SMALL
.STACK  100H
.CODE
MAIN  PROC
CALL  MULTIPLY
MOV  AH, 4CH
INT  21H
MAIN  ENDP
MULTIPLY  PROC
PUSH  AX
PUSH  BX
XOR  DX, DX
REPEAT:
TEST  BX, 1
JZ  END_IF
ADD  DX, AX
END_IF:
SHL  AX, 1
SHR  BX, 1
JNZ  REPEAT
POP  BX
POP  AX
RET
MULTIPLY  ENDP
END  MAIN
```