2. NEAR JUMP, which is the default, has the format "JMP label". This is a near jump (within the current code segment) and has the opcode E9. The target address can be any of the addressing modes of direct, register, register indirect, or memory indirect:

(a) Direct JUMP is exactly like the short jump explained earlier, except that the target address can be anywhere in the segment within the range +32767 to -32768 of the current IP.

(b) Register indirect JUMP; the target address is in a register. For example, in "JMP BX", IP takes the value BX.

(c) Memory indirect JMP; the target address is the contents of two memory locations pointed at by the register. Example: "JMP [DI]" will replace the IP with the contents of memory locations pointed at by DI and DI+ 1.

3. FAR JUMP which has the format "JMP FAR PTR label". This is a jump out of the current code segment, meaning that not only the IP but also the CS is replaced with new values.

CALL statements

Another control transfer instruction is the CALL instruction, which is used to call a procedure. CALLs to procedures are used to perform tasks that need to be performed frequently. This makes a program more structured. The target address could be in the current segment, in which case it will be a NEAR call or outside the current CS segment, which is a FAR call. To make sure that after execution of the called subroutine the microprocessor knows where to come back, the microprocessor automatically saves the address of the instruction following the call on the stack. It must be noted that in the NEAR call only the IP is saved on the stack, and in a FAR call both CS and IP are saved. When a subroutine is called, control is transferred to that subroutine and the processor saves the IP (and CS in the case of a FAR call) and begins to fetch instructions from the new location. After finishing execution of the subroutine, for control to be transferred back to the caller, the last instruction in the called subroutine must be RET (return). In the same way that the assembler generates different opcode for FAR and NEAR calls, the opcode for the RET instruction in the case of NEAR and FAR is different, as well. For NEAR calls, the IP is restored; for FAR calls, both CS and IP are restored. This will ensure that control is given back to the caller. As an example, assume that SP = FFFEH and the following code is a portion of the program unassembled in DEBUG:

```
12B0:0200 BB1295  MOV BX,9512
12B0:0203 E8FA00  CALL 0300
12B0:0206 B82F14  MOV AX,142F
```

Since the CALL instruction is a NEAR call, meaning that it is in the same code segment (different IP, same CS), only IP is saved on the stack. In this case, the IP address of the instruction after the call is saved on the stack as shown in Figure 2-5. That IP will be 0206, which belongs to the "MOV AX,142F" instruction.

The last instruction of the called subroutine must be a RET instruction which directs the CPU to POP the top 2 bytes of the stack into the IP and resume executing at offset address 0206. For this reason, the number of PUSH and POP instructions (which alter the SP) must match. In other words, for every PUSH there must be a POP.

```
12B0:0300 53  PUSH BX
12B0:0301 ... ..... ..
..............  ... ......  ..
12B0:0309 5B  POP BX
12B0:030A C3  RET
```

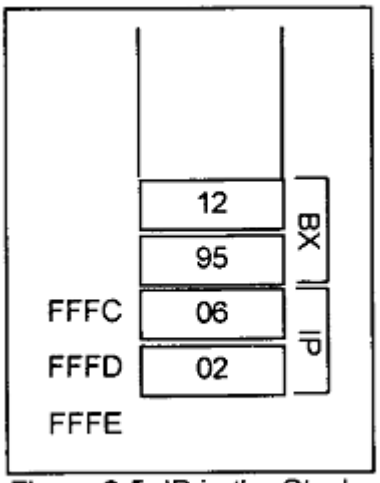

Figure 2-5. IP in the Stack

2.2 Assembly language subroutines

In Assembly language programming it is common to have one main program and many subroutines to be called from the main program. This allows you to make each subroutine into a separate module. Each module can be tested separately and then brought together, as will be shown next chapters. The main program is the entry point from DOS and is FAR, as explained earlier, but the subroutines called within the main program can be FAR or NEAR.

Remember that NEAR routines are in the same code segment, while FAR routines are outside the current code segment. If there is no specific mention of FAR after the directive PROC, it defaults to NEAR, as shown in Figure 2-6. From now on, all code segments will be written in that format.

Rules for names in Assembly language

By choosing label names that are meaningful, a programmer can make a program much easier to read and maintain. There are several rules that names must follow. **First,** each label name must be unique. The names used for labels in Assembly

language programming consist of alphabetic letters in both upper and lower case, the digits 0 through 9, and the special characters question mark (?), period (.), at (@), underline (_), and dollar sign ($). The first character of the name must be an alphabetic character or special character. It cannot be a digit. The period can only be used as the first character, but this is not recommended since later versions of MASM have several reserved words that begin with a period. Names may be up to 31 characters long.

```
;-------------------------------------------
        .CODE
MAIN    PROC  FAR          ;THIS IS THE ENTRY POINT FOR DOS
        MOV   AX,@DATA
        MOV   DS,AX
        CALL  SUBR1
        CALL  SUBR2
        CALL  SUBR3
        MOV   AH,4CH
        INT   21H
MAIN    ENDP
;-------------------------------------------
SUBR1   PROC
        ...
        ...
        RET
SUBR1   ENDP
;-------------------------------------------
SUBR2   PROC
        ...
        ...
        RET
SUBR2   ENDP
;-------------------------------------------
SUBR3   PROC
        ...
        ...
        RET
SUBR3   ENDP
;-------------------------------------------
        END   MAIN         ;THIS IS THE EXIT POINT
```

Figure 2-6. Shell of Assembly Language Subroutines