7.1 HLT (Halt Processing):

The HLT instruction causes the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The different ways to get the processor out of the halt state are with an interrupt signal on the INTR pin, an interrupt signal on the NMI pin, or a reset signal on the RESET input or during a DMA operation. This instruction normally appears in a program to wait for an interrupt. It often synchronizes external hardware interrupts with the software system.

7.2 RET (Return):

The RET instruction removes a 16-bit number (near return) from the stack and places it into IP, or removes a 32-bit number (far return) and places it into IP and CS. The near and far return instructions are both defined in the procedure's PROC directive, which automatically selects the proper return instruction. When IP or CS are changed, the address of the next instruction is at a new memory location. This new location is the address of the instruction that immediately follows the most recent CALL to a procedure. Figure 6–8 shows how the CALL instruction links to a procedure and how the RET instruction returns in the 8086 operating in the real mode. There is one other form of the return instruction, which adds a number to the contents of the stack pointer (SP) after the return address is removed from the stack.

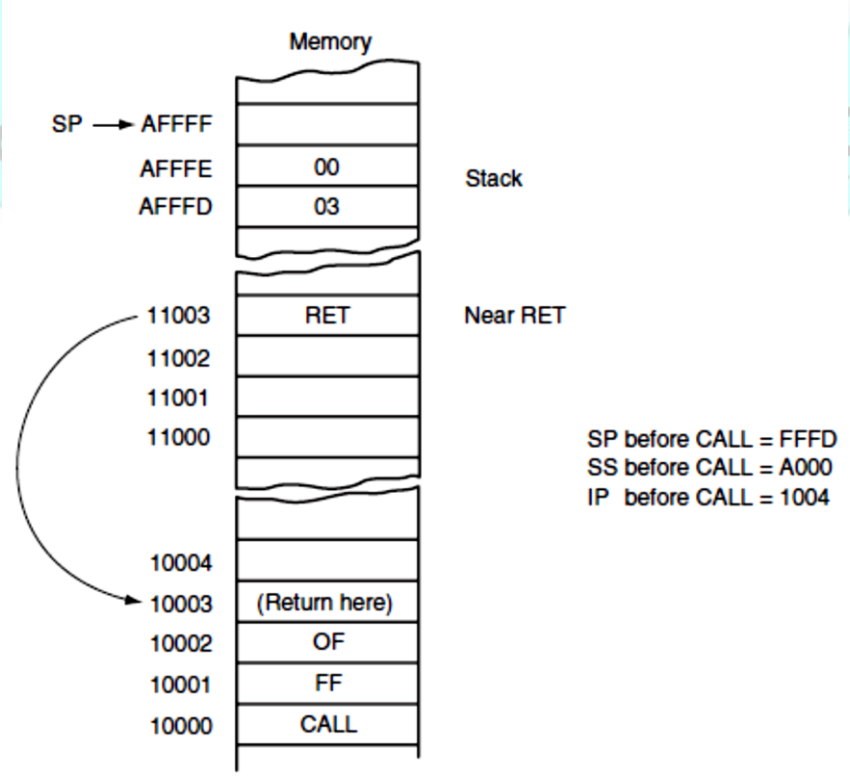

Figure (7-1): The effect of a near return instruction on the stack and instruction pointer.

7.3 NOP (Perform No Operation):

When the microprocessor encounters a no operation instruction (NOP), it takes a short time to execute. This instruction simply uses up three clock cycles and increments the instruction pointer to point to the next instruction. The NOP instruction can be used to increase the delay of a delay loop. When hand coding, a NOP can also be used to hold a place in a program for an instruction that will be added later. NOP does not affect any flag.

7.4 FLAG Manipulation Instructions:

STC (Set Carry FLAG): This instruction sets the carry flag to 1. It does not affect any other flag.

CLC (Clear Carry FLAG): This instruction resets the carry flag to 0. It does not affect any other flag.

CMC (Complement Carry FLAG ($\overline{CF}$)): This instruction complements the carry flag. It does not affect any other flag.

STD (Set Direction FLAG): This instruction sets the direction flag to 1. It does not affect any other flag.

CLD (Clear Direction FLAG): This instruction resets the direction flag to 0. It does not affect any other flag.

STI (Set Interrupt FLAG): Setting the interrupt flag to a 1 enables the INTR interrupt input of the 8086. The instruction will not take affect until the next instruction after STI. When the INTR input is enabled, an interrupt signal on this input will then cause the 8086 to interrupt program execution, push the return address and flags on the stack, and execute an interrupt service procedure. An IRET instruction at the end of the interrupt service procedure will restore the return address and flags that were pushed onto the stack and return execution to the interrupted program. STI does not affect any other flag.

CLI (Clear Interrupt FLAG): This instruction resets the interrupt flag to 0. If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input. The CLI instructions, however, has no effect on the non-maskable interrupt input, NMI. It does not affect any other flag.

LAHF (Copy Low Byte of FLAG Register to AH Register): The LAHF instruction copies the low-byte of the 8086 flag register to AH register. It can then be pushed onto the stack along with AL by a PUSH AX instruction. LAHF does not affect any flag.

SAHF (Copy AH Register to Low Byte of FLAG Register): The SAHF instruction replaces the low-byte of the 8086 flag register with a byte from the AH register. SAHF changes the flags in lower byte of the flag register.

7.5 Stack Related Instructions:

PUSH–PUSH Source: The 8086 PUSH instruction always transfers 2 bytes of data to the stack. The PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment to which the stack pointer points. The source of the word can be general-purpose register, segment register, or memory. The stack segment register and the stack pointer must be initialized before this instruction can be used. PUSH can be used to save data on the stack so that it will not destroyed by a procedure. This instruction does not affect any flag.

PUSH BX	Decrement SP by 2, copy BX to stack.
PUSH DS	Decrement SP by 2, copy DS to stack.
PUSH BL	Illegal; must push a word
PUSH TABLE [BX]	Decrement SP by 2, and copy word from memory in DS at EA = TABLE + [BX] to stack

POP–POP Destination: The POP instruction performs the inverse operation of a PUSH instruction. The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the instruction. The destination can be a general-purpose register, a segment register or a memory location. The data in the stack is not changed. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to point to the next word on the stack. The POP instruction does not affect any flag.

POP DX	Copy a word from top of stack to DX; increment SP by 2
POP DS	Copy a word from top of stack to DS; increment SP by 2
POP TABLE [DX]	Copy a word from top of stack to memory in DS with EA = TABLE + [BX]; increment SP by 2.

PUSHF (Push FLAG Register to Stack): The PUSHF instruction decrements the stack pointer by 2 and copies a word in the flag register to two memory locations in stack pointed to by the stack pointer. The stack segment register is not affected. This instruction does to affect any flag.

POPF (POP Word from Top of Stack to FLAG Register): The POPF instruction copies a word from two memory locations at the top of the stack to the flag register and increments the stack pointer by 2. The stack segment register and word on the stack are not affected. This instruction does to affect any flag.

7.6 Input-Output Instructions:

IN–IN Accumulator, Port: The IN instruction copies data from a port to the AL or AX register. If an 8-bit port is read, the data will go to AL. If a 16-bit port is read, the data will go to AX.

The IN instruction has two possible formats, fixed port and variable port. For fixed port type, the 8-bit address of a port is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

```
IN AL, C8H        Input a byte from port C8H to AL
IN AX, 34H        Input a word from port 34H to AX
```

For the variable-port form of the IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16-bit register, the port address can be any number between 0000H and FFFFH. Therefore, up to 65,536 ports are addressable in this mode.

```
MOV DX, 0FF78H    Initialize DX to point to port
IN AL, DX         Input a byte from 8-bit port 0FF78H to AL
IN AX, DX         Input a word from 16-bit port 0FF78H to AX
```

The variable-port IN instruction has advantage that the port address can be computed or dynamically determined in the program. Suppose, for example, that an 8086-based computer needs to input data from 10 terminals, each having its own port address. Instead of having a separate procedure to input data from each port, you can write one generalized input procedure and simply pass the address of the desired port to the procedure in DX. The IN instruction does not change any FLAG bit.

OUT–OUT Port, Accumulator: The OUT instruction copies a byte from AL or a word from AX to the specified port. The OUT instruction has two possible forms, fixed port and variable port.

For the fixed port form, the 8-bit port address is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

```
OUT 3BH, AL       Copy the content of AL to port 3BH
OUT 2CH, AX       Copy the content of AX to port 2CH
```

For variable port form of the OUT instruction, the content of AL or AX will be copied to the port at an address contained in DX. Therefore, the DX register must be loaded with the desired port address before this form of the OUT instruction is used.

```
MOV DX, FFF8H     Load desired port address in DX
OUT DX, AL        Copy content of AL to port FFF8H
OUT DX, AX        Copy content of AX to port FFF8H
```

The OUT instruction does not affect any FLAG bit.