ieved before returning to the
his chapter includes two illustrative pro-
The first illustrates the use of the stack-related
ions to examine and manipulate the flags;
second illustrates the subroutine technique
ffic-signal controller.

TIVES

ne the stack, the stack pointer (register), and
rogram counter, and describe their uses.
ain how information is stored and retrieved
the stack using the instructions PUSH and
and the stack pointer register.

- ☐ Explain how
the program counter and the
the contents of the stack pointer register when a
subroutine is called.
- ☐ List and explain conditional Call and Return in-
structions.
- ☐ Illustrate the concepts in the following subrou-
tines: multiple calling, nesting, and common end-
ing.
- ☐ Compare similarities and differences between
PUSH/POP and CALL/RET instructions.

STACK

The **stack** in an 8085 microcomputer system can be described as a set of memory locations in the R/W memory, specified by a programmer in a main program. These memory locations are used to store binary information (bytes) temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP, which loads a 16-bit memory address in the stack pointer register of the microprocessor. Once the stack location is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register. For example, if the stack pointer register is loaded with the memory address 2099H (LXI SP,2099H), the storing of data bytes begins at 2098H and continues in reversed numerical order (decreasing memory addresses such as 2098H, 2097H, etc.). Therefore, as a general practice, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The size of the stack is limited only by the available memory.

Data bytes in the register pairs of the microprocessor can be stored on the stack (two at a time) in reverse order (decreasing memory address) by using the instruction PUSH. Data bytes can be transferred from the stack to respective registers by using the instruction POP. The stack pointer register tracks the storage and retrieval of the information. Because two data bytes are being stored at a time, the 16-bit memory address in the stack pointer register is decremented by two; when data bytes are retrieved, the address is incremented by two. An address in the stack pointer register indicates that the next two memory locations (in descending numerical order) can be used for storage.

The stack is shared by the programmer and the microprocessor. The programmer can store and retrieve the contents of a register pair by using PUSH and POP instructions. Similarly, the microprocessor automatically stores the contents of the program counter when a subroutine is called (to be discussed in the next section). The instructions necessary for using the stack are explained below.

INSTRUCTIONS

Opcode	Operand	
LXI	SP,16-bit	☐ Load Stack Pointer ☐ Load the stack pointer register with a 16-bit address. The LXI instructions were discussed in Chapter 7
PUSH	Rp	Store Register Pair on Stack ☐ This is a 1-byte instruction ☐ It copies the contents of the specified register pair on the stack as described below ☐ The stack pointer register is decremented, and the contents of the high-order register (e.g., register B) are copied in the location shown by the stack pointer register ☐ The stack pointer register is again decremented, and the contents of the low-order register (e.g., register C) are copied in that location ☐ The operands B, D, and H represent register pairs BC, DE, and HL, respectively ☐ The operand PSW represents Program Status Word, meaning the contents of the accumulator and the flags
PUSH	B	
PUSH	D	
PUSH	H	
PUSH	PSW	
POP	Rp	Retrieve Register Pair from Stack ☐ This is a 1-byte instruction ☐ It copies the contents of the top two memory locations of the stack into the specified register pair ☐ First, the contents of the memory location indicated by the stack pointer register are copied into the low-order register (e.g., register L), and then the stack pointer register is incremented by 1 ☐ The contents of the next memory location are copied into the high-order register (e.g., register H), and the stack pointer register is again incremented by 1
POP	B	
POP	D	
POP	H	
POP	PSW	

All three of these instructions belong to the data transfer (copy) group; thus, the contents of the source are not modified, and no flags are affected.

Exam

In the following set of instructions (illustrated in Figure 9.1), the stack pointer is initialized, and the contents of register pair HL are stored on the stack by using the PUSH instruction. Register pair HL is used for the delay counter (actual instructions are not

9.2 SUBROUTINE

A **subroutine** is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program. For example, if a time delay is required between three successive events, three delays can be written in the main program. To avoid repetition of the same delay instructions, the subroutine technique is used. Delay instructions are written once, separately from the main program, and are called by the main program when needed.

The 8085 microprocessor has two instructions to implement subroutines: CALL (call a subroutine), and RET (return to main program from a subroutine). The CALL instruction is used in the main program to call a subroutine, and the RET instruction is used at the end of the subroutine to return to the main program. When a subroutine is called, the contents of the program counter, which is the address of the instruction following the CALL instruction, is stored on the stack and the program execution is transferred to the subroutine address. When the RET instruction is executed at the end of the subroutine, the memory address stored on the stack is retrieved, and the sequence of execution is resumed in the main program. This sequence of events is illustrated in Example 9.3.

INSTRUCTIONS

Opcode	Operand	
CALL	16-bit memory address of a subroutine	Call Subroutine Unconditionally ☐ This is a 3-byte instruction that transfers the program sequence to a subroutine address ☐ Saves the contents of the program counter (the address of the next instruction) on the stack ☐ Decrements the stack pointer register by two ☐ Jumps unconditionally to the memory location specified by the second and third bytes. The second byte specifies a line number and the third byte specifies a page number ☐ This instruction is accompanied by a return instruction in the subroutine
RET		Return from Subroutine Unconditionally ☐ This is a 1-byte instruction ☐ Inserts the two bytes from the top of the stack into the program counter and increments the stack pointer register by two ☐ Unconditionally returns from a subroutine

The conditional Call and Return instructions will be described later in the chapter.

LIST OF SUBROUTINES CALLED

If a subroutine is calling other subroutines, the user should be provided with a list. The user can check what parameters need to be passed to various subroutines and what registers are modified in the process.

9.3 RESTART, CONDITIONAL CALL, AND RETURN INSTRUCTIONS

In addition to the unconditional CALL and RET instructions, the 8085 instruction set includes eight Restart instructions and eight conditional Call and Return instructions.

9.31 Restart (RST) Instructions

RST instructions are 1-byte Call instructions that transfer the program execution to a specific location on page 00H. They are executed the same way as Call instructions. When an RST instruction is executed, the 8085 stores the contents of the program counter (the address of the next instruction) on the top of the stack and transfers the program to the Restart location. These instructions are generally used in conjunction with the interrupt process discussed in Chapter 12. These instructions are listed here to emphasize that they are Call instructions and not necessarily always associated with the interrupts. The list of eight RST instructions is as follows:

RST 0	Call 0000H	RST 4	Call 0020H
RST 1	Call 0008H	RST 5	Call 0028H
RST 2	Call 0010H	RST 6	Call 0030H
RST 3	Call 0018H	RST 7	Call 0038H

9.32 Conditional Call and Return Instructions

The conditional Call and Return instructions are based on four data conditions (flags): Carry, Zero, Sign, and Parity. The conditions are tested by checking the respective flags. In case of a conditional Call instruction, the program is transferred to the subroutine if the condition is met; otherwise, the main program is continued. In case of a conditional Return instruction, the sequence returns to the main program if the condition is met; otherwise, the sequence in the subroutine is continued. If the Call instruction in the main program is conditional, the Return instruction in the subroutine can be conditional or unconditional. The conditional Call and Return instructions are listed for reference.

CONDITIONAL CALL

CC	Call subroutine if Carry flag is set (CY = 1)
CNC	Call subroutine if Carry flag is reset (CY = 0)
CZ	Call subroutine if Zero flag is set (Z = 1)
CNZ	Call subroutine if Zero flag is reset (Z = 0)
CM	Call subroutine if Sign flag is set (S = 1, negative number)
CP	Call subroutine if Sign flag is reset (S = 0, positive number)

CPE	Call subroutine if Parity flag is set (P = 1, even parity)
CPO	Call subroutine if Parity flag is reset (P = 0, odd parity)

CONDITIONAL RETURN

RC	Return if Carry flag is set (CY = 1)
RNC	Return if Carry flag is reset (CY = 0)
RZ	Return if Zero flag is set (Z = 1)
RNZ	Return if Zero flag is reset (Z = 0)
RM	Return if Sign flag is set (S = 1, negative number)
RP	Return if Sign flag is reset (S = 0, positive number)
RPE	Return if Parity flag is set (P = 1, even parity)
RPO	Return if Parity flag is reset (P = 0, odd parity)

9.4 ADVANCED SUBROUTINE CONCEPTS

In Section 9.2, one type of subroutine (multiple-calling of a subroutine by a main program) was illustrated. Other types of subroutine techniques, such as nesting and multiple-ending, are briefly illustrated below.

9.41 Nesting

The programming technique of a subroutine calling another subroutine is called **nesting.** This process is limited only by the number of available stack locations. When a subroutine calls another subroutine, all return addresses are stored on the stack. Nesting is illustrated in Figure 9.15.

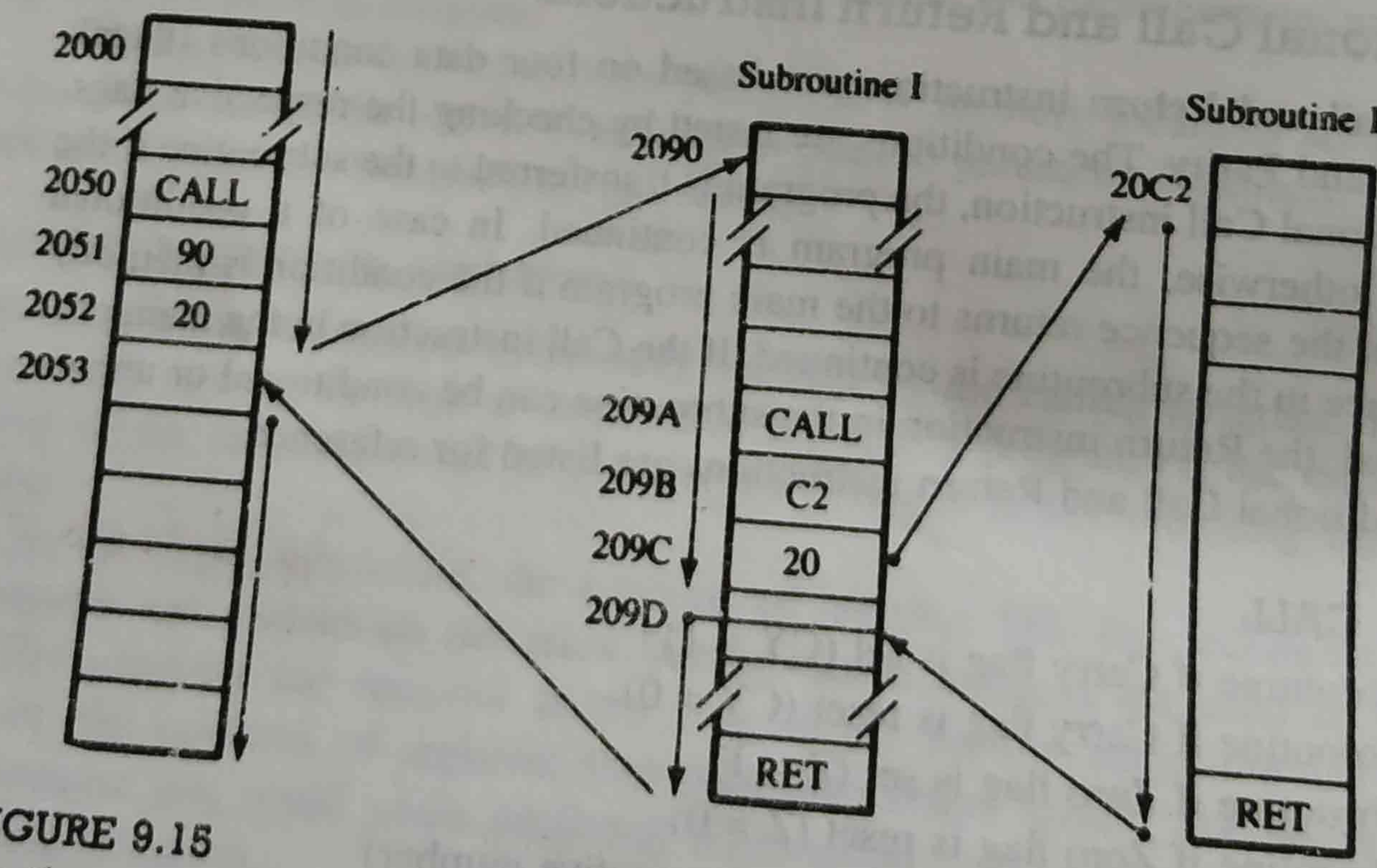
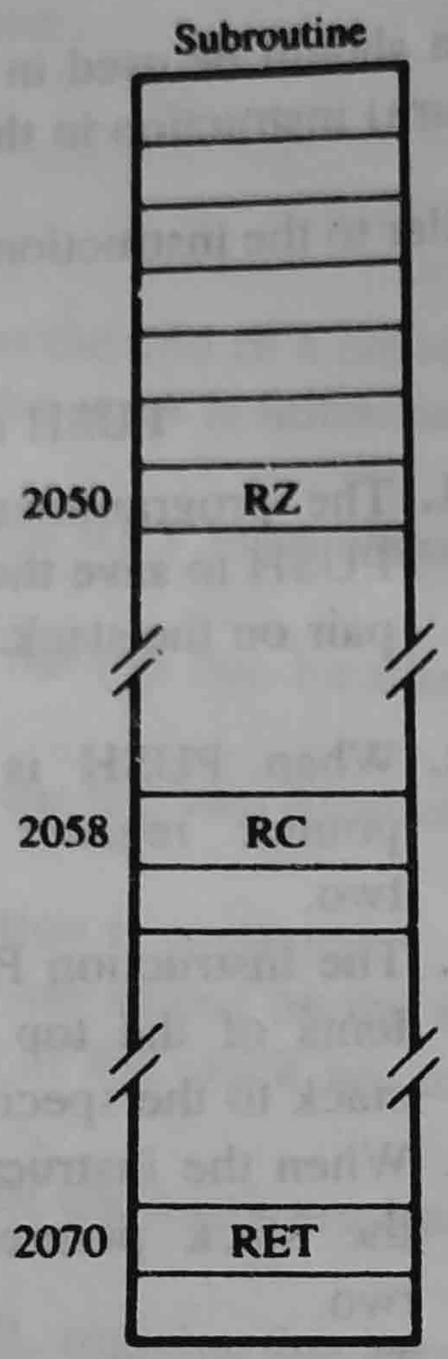


FIGURE 9.15
Nesting of Subroutines

FIGURE 9.16
Multiple-Ending Subroutine



The main program in Figure 9.15 calls the subroutine from location 2050H. The address of the next instruction, 2053H, is placed on the stack, and the program is transferred to the subroutine at 2090H. Subroutine I calls Subroutine II from location 209AH. The address 209DH is placed on the stack, and the program is transferred to Subroutine II. The sequence of execution returns to the main program, as shown in Figure 9.15.

9.42 Multiple-Ending Subroutines

Figure 9.16 illustrates three possible endings to one CALL instruction. The subroutine has two conditional returns (RZ—Return on Zero, and RC—Return on Carry) and one unconditional return (RET). If the Zero flag (Z) is set, the subroutine returns from location 2050H. If the Carry flag (CY) is set, it returns from location 2058H. If neither the Z nor the CY flag is set, it returns from location 2070H. This technique is illustrated in Chapter 10, Section 10.42.

SUMMARY

To implement a subroutine, the following steps are necessary:

1. The stack pointer register must be initialized, preferably at the highest memory loca-