```
                    ;Storing Result in Memory—Alternative to Output Display
18      32                      STA     XX70H       ;Store sum in memory
                                                    ;  XX70H
19      70
1A      XX
1B      76
1C      3E          OVRLOD:     MVI     A,FFH       ;Store overload signal in
1D      FF                                          ;  memory XX70
1E      32                      STA     XX70H
1F      70
20      XX
21      76                      HLT
```

```
XX60    28      ;Current Readings
  61    D8
  62    C2
  63    21
  64    24
  65    30
  66    2F
  67    19
  68    F2
  69    9F
```

PROGRAM DESCRIPTION AND OUTPUT

In this program, register C is used as a counter to count ten bytes. Register B is used to save the sum. The sign of the number is checked by verifying whether D_7 is 1 or 0. If the Carry flag is set to indicate the negative sign, the program rejects the number and goes to Block 5, Getting Ready for Next Operation.

The program should reject the data bytes D8, C2, F2, and 9F, and should add the rest. The answer displayed should be E5.

See Questions and Programming Assignments 32–40 at the end of this chapter.

7.5 LOGIC OPERATIONS: COMPARE

The 8085 instruction set has two types of Compare operations: CMP and CPI.

- ☐ CMP: Compare with Accumulator
- ☐ CPI: Compare Immediate (with Accumulator)

The microprocessor compares a data byte (or register/memory contents) with the contents of the accumulator by subtracting the data byte from (A), and indicates

whether the data byte is ≥\≤ (A) by modifying the flags. However, the contents are not modified.

7.5.1 Instructions

1. CMP R/M: Compare (Register or Memory) with Accumulator
   - ☐ This is a 1-byte instruction.
   - ☐ It compares the data byte in register or memory with the contents of the accumulator.
   - ☐ If (A) < (R/M), the CY flag is set and the Zero flag is reset.
   - ☐ If (A) = (R/M), the Zero flag is set and the CY flag is reset.
   - ☐ If (A) > (R/M), the CY and Zero flags are reset.
   - ☐ When memory is an operand, its address is specified by (HL).
   - ☐ No contents are modified; however, all remaining flags (S, P, AC) are affected according to the result of the subtraction.
2. CPI 8-bit: Compare Immediate with Accumulator
   - ☐ This is a 2-byte instruction, the second byte being 8-bit data.
   - ☐ It compares the second byte with (A).
   - ☐ If (A) < 8-bit data, the CY flag is set and the Zero flag is reset.
   - ☐ If (A) = 8-bit data, the Zero flag is set, and the CY flag is reset.
   - ☐ If (A) > 8-bit data, the CY and Zero flags are reset.
   - ☐ No contents are modified; however, all remaining flags (S, P, AC) are affected according to the result of the subtraction.

Example 7.11

Write an instruction to load the accumulator with the data byte 64H, and verify whether the data byte in memory location 2050H is equal to the accumulator contents. If both data bytes are equal, jump to memory location BUFFER.

Solution

Figure 7.17 illustrates Example 7.11.

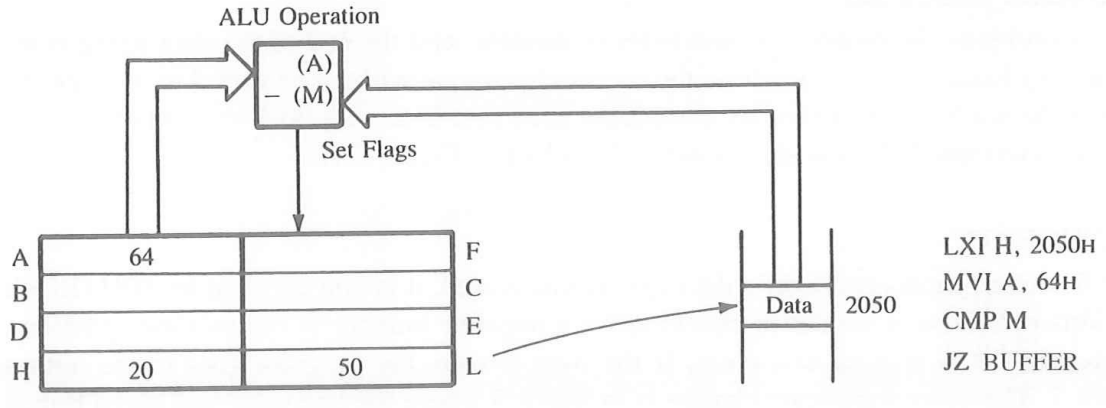


FIGURE 7.17
Compare Instructions

In these instructions, the instruction CMP M selects the memory location pointed out by the HL register (2050H) and compares the contents of that location with (A). If they are equal, the Zero flag is set, and the program jumps to location BUFFER.

Let us assume the data byte in memory location 2050H is 9AH. The microprocessor compares 64H with 9AH by subtracting 9AH from 64H as shown below.

(A) = 64H	0110 0100
	+
2's Complement of 9AH	0110 0110
CY	0 1100 1010
Complement CY	1 1100 1010

The result CAH will modify the flags as S = 1, Z = 0, and CY = 1; however, the original byte in the accumulator 64H will not be changed.

7.5.2 Illustrative Program: Use of Compare Instruction to Indicate End of Data String

PROBLEM STATEMENT

A set of current readings is stored in memory locations starting at XX50H. The end of the data string is indicated by the data byte 00H. Add the set of readings. The answer may be larger than FFH. Display the entire sum at PORT1 and PORT2 or store the answer in the memory locations XX70 and XX71H.

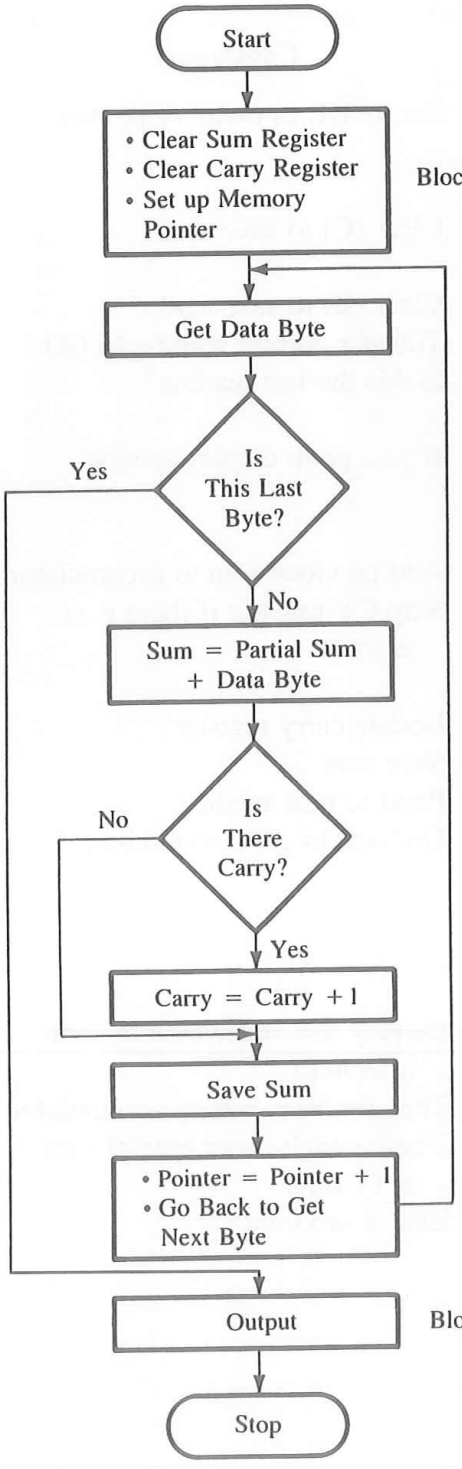
Data(H) 32, 52, F2, A5, 00

PROBLEM ANALYSIS

In this problem, the number of data bytes is variable, and the end of the data string is indicated by loading 00H. Therefore, the counter technique will not be useful to indicate the end of the readings. However, by comparing each data byte with 00H, the end of the data can be determined. This is shown in the flowchart in Figure 7.18.

FLOWCHART

The flowchart shows that after a data byte is transferred, it is first checked for 00H (Block 6). This operation is similar to checking for a negative number in the previous problem. However, this is also an exit point. If the byte is zero, the program goes to the output Block 7. The other significant change is in Block 5 where the unconditional Jump brings the sequence back into the program.



Mnemonics

```
LXI H,XX50H
MV1 C,00H
MOV B,C
NXTBYT: MOV A,M
CPI 00H
JZ DSPLAY
ADD C
JNC SAVE
INR B
SAVE: MOV C,A
INX H
JMP NXTBYT
DSPLAY: MOV A,C
OUT PORT1
MOV A,B
OUT PORT2
HLT
```

FIGURE 7.18
Flowchart for Compare Instruction to Check End of Data String

PROGRAM

Memory Address HI-LO	Machine Code	Label	Instruction Opcode	Operand	Comments
XX00	21	START:	LXI	H,XX50H	;Set up HL as memory pointer
01	50				
02	XX				
03	0E		MVI	C,00H	;Clear (C) to save sum
04	00				
05	41		MOV	B,C	;Clear (B) to save carry
06	7E	NXTBYT:	MOV	A,M	;Transfer current reading to (A)
07	FE		CPI	00H	;Is this the last reading?
08	00				
09	CA		JZ	DSPLAY	;If yes; go to display section
0A	16				
0B	XX				
0C	81		ADD	C	;Add previous sum to accumulator
0D	D2		JNC	SAVE	;Skip CY register if there is no
0E	11				; carry
0F	XX				
10	04		INR	B	;Update carry register
11	4F	SAVE:	MOV	C,A	;Save sum
12	23		INX	H	;Point to next reading
13	C3		JMP	NXTBYT	;Go back to get next reading
14	06				
15	XX				
		;Output Display			
16	79	DSPLAY:	MOV	A,C	
17	D3		OUT	PORT1	;Display low-order byte of sum
18	PORT1				; at PORT1
19	78		MOV	A,B	;Transfer carry bits to accumulator
1A	D3		OUT	PORT2	;Display high-order byte of sum
1B	PORT2				; at PORT2
1C	76		HLT		;End of program

```
                    ;Storing Result in Memory—Alternative to Output Display
16      21      DSPLAY:   LXI     H,XX70    ;Point index to XX70H location
17      70
18      XX
19      71                MOV     M,C       ;Store low-order byte of sum
                                            ;  in XX70H
1A      23                INX     H         ;Point index to XX71H
1B      70                MOV     M,B       ;Store high-order byte
1C      76                HLT               ;End of program

50      32      ;Data—Current Readings
51      52
52      F2
53      A5
54      00
```

PROGRAM DESCRIPTION AND OUTPUT

This program adds the first four readings, which results in the sum 21BH. The low-order byte 1BH is saved in register C, and the high-order byte 02H is stored in the carry register B. These are each displayed at different output ports by the OUT instruction or stored in the memory locations XX70 and XX71.

See Questions and Programming Assignments 41–56 at the end of the chapter.

7.5.3 Illustrative Program: Sorting

PROBLEM STATEMENT

A set of three readings is stored in memory starting at XX50H. Sort the readings in ascending order.

Data(H) 87, 56, 42

PROBLEM ANALYSIS

In this problem, three readings should be arranged in ascending order: 42, 56, and 87. The number of readings is kept small to simplify the explanation. However, the programming technique used here, called bubble sort, can be applied to a large set of data. Ordering data in a given sequence such as ascending, descending, or alphabetical order is a common application in programming.

The technique involved is comparing two bytes at a time and placing them in the proper sequence. For example, in our data set, we will compare the first two bytes (87H and 56H), and if the first byte is larger than the second byte, we will exchange their mem-

ory locations to arrange them in ascending order; otherwise, we will keep them in the same locations. We will follow the same procedure for the second and third bytes. The number of comparisons necessary is always $N - 1$ where N is the number of data bytes. Therefore, we need a counter of 2 for one complete comparison. Table 7.1 shows the memory locations and their contents after each pass (execution of one cycle). Two exchanges occur in the first pass, and the bytes are sequenced as 56, 42, and 87. In the second pass, only one exchange occurs. The microprocessor should continue these comparisons until no exchanges occur. We need to devise a technique or set up a reminder for the processor to recognize that no exchanges occur in a given pass. This is called setting a flag. In daily life, we write a note to remind ourselves. Similarly, we can use any register to write a binary note to the microprocessor. For example, we can write 1 (or any number such as FFH) in register D when an exchange takes place; otherwise, register D will be cleared.

PROGRAM

```
START:    LXI H, XX50H     ;Set up HL as a memory pointer for bytes
          MVI D, 00        ;Clear register D to set up a flag
          MVI C, 02        ;Set register C for comparison count
CHECK:    MOV A, M         ;Get data byte
          INX H            ;Point to next byte
          CMP M            ;Compare bytes
          JC NXTBYT        ;If (A) < second byte, do not exchange
          MOV B, M         ;Get second byte for exchange
          MOV M, A         ;Store first byte in second location
          DCX H            ;Point to first location
          MOV M, B         ;Store second byte in first location
          INX H            ;Get ready for next comparison
          MVI D, 01        ;Load 1 in D as a reminder for exchange
NXTBYT:   DCR C            ;Decrement comparison count
          JNZ CHECK        ;If comparison count ≠ 0, go back
          MOV A, D         ;Get flag bit in A
          RRC              ;Place flag bit D0 in Carry
          JC START         ;If flag is 1, exchange occurred
                           ;Start the next pass
          HLT              ;End of sorting
```

TABLE 7.1
Execution of Sort Program

Memory Locations	Initial Bytes	First Pass	Second Pass	Third Pass
XX50	87	56	42	42
XX51	56	42	56	56
XX52	42	87	87	87
Reg. D	0	1	1	0

PROGRAM DESCRIPTION AND OUTPUT

During the initialization phase, register HL is set up as a pointer where readings are stored; register D is cleared to set up a flag when an exchange occurs; and register C is initialized for a count of two because we have three data bytes to sort. Next is the comparison phase. The programs gets the first byte (87H) in A and compares that with the second byte (56H). In this comparison, the carry flag is reset; therefore, the next set of instructions places 56H in location XX50H and 87H in location XX51H. The flag in register D is set, the comparison counter in register C is decremented by 1, and the program returns to location CHECK for the next comparison. In this second comparison, 87H is compared with 42H, and again the bytes are exchanged. The flag (reminder) in D is set again, but now the comparison counter is zero. Therefore, the program falls through the first loop and checks the status of the flag in D by rotating the flag bit into Carry. Because Carry is set, the program goes back to the beginning and starts the process again from location XX50H. In the second pass, 56H and 42H are exchanged; therefore, the process is repeated a third time. In this pass, every comparison generates a carry. The program jumps to location NXTBYT, and the flag in register D remains zero. Thus, the program places the bytes in ascending order: 42H, 56H, and 87H.

7.6 DYNAMIC DEBUGGING

After you have completed the steps in the process of static debugging (described in the previous chapter), if the program still does not produce the expected output, you can attempt to debug the program by observing the execution of instructions. This is called **dynamic debugging.**

7.6.1 Tools for Dynamic Debugging

In a single-board microcomputer, techniques and tools commonly used in dynamic debugging are

- ☐ Single Step
- ☐ Register Examine
- ☐ Breakpoint

Each will be discussed below; the Single-Step and Register Examine keys were discussed briefly in the previous chapter.

SINGLE STEP

The Single-Step key on a keyboard allows you to execute one instruction at a time, and to observe the results following each instruction. Generally, a single-step facility is built with a hard-wired logic circuit. As you push the Single-Step key, you will be able to observe addresses and codes as they are executed. With the single-step technique you will be able to spot