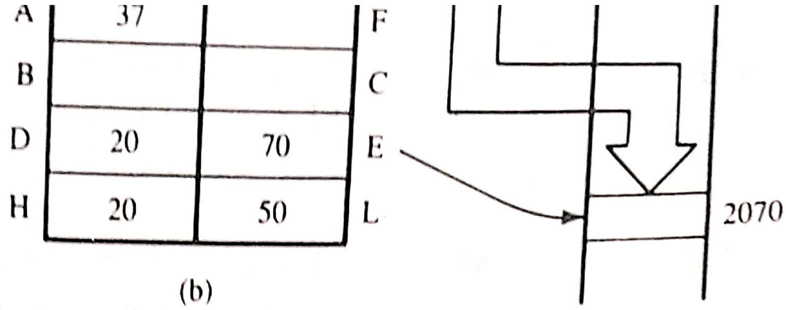


(b)

## PROGRAM EXECUTION AND OUTPUT

To execute the program, substitute the page number of your system's R/W memory in place of XX, enter the program and the data, and execute it. To verify the proper execution, check the memory locations from XX70H to XX7FH.

Let us assume the system R/W user memory starts at 2000H. Figure 7.9(a) shows how the contents of the memory location 2050H are copied into the accumulator by the instruction MOV A,M; the HL register points to location 2050 and instruction MOV A,M copies 37H into A. Figure 7.9(b) shows that the DE register points to the location 2070H and the instruction STAX D copies (A) into the location 2070H.

See Questions and Assignments 5–21 at the end of this chapter.

## ARITHMETIC OPERATIONS RELATED TO MEMORY

In the last chapter, the arithmetic instructions concerning three arithmetic tasks—Add, Subtract, and Increment/Decrement—were introduced. These instructions dealt with microprocessor register contents or numbers. In this chapter, instructions concerning the arithmetic tasks related to memory will be introduced:

ADD M/SUB M: Add/Subtract the contents of a memory location to/from the contents of the accumulator.

INR M/DCR M: Increment/Decrement the contents of a memory location.

## 7.31 Instructions

The arithmetic instructions referenced to memory perform two tasks: one is to copy a byte from a memory location to the microprocessor, and the other is to perform the arithmetic operation. These instructions (other than INR and DCR) implicitly assume that one of the operands is (A); after an operation, the previous contents of the accumulator are replaced by the result. All flags are modified to reflect the data conditions (see the exceptions: INR and DCR).

| Opcode | Operand | |
|---|---|---|
| ADD | M | Add Memory<br>☐ This is a 1-byte instruction<br>☐ It adds (M) to (A) and stores the result in A<br>☐ The memory location is specified by the contents of HL register |
| SUB | M | Subtract Memory<br>☐ This is a 1-byte instruction<br>☐ It subtracts (M) from (A) and stores the result in A<br>☐ The memory location is specified by (HL) |
| INR | M | This is a 1-byte instruction<br>☐ It increments the contents of a memory location by 1, not the memory address<br>☐ The memory location is specified by (HL)<br>☐ All flags except the Carry flag are affected |
| DCR | M | This is a 1-byte instruction<br>☐ It decrements (M) by 1<br>☐ The memory location is specified by (HL)<br>☐ All flags except the Carry flag are affected |

**Example**

Write instructions to add the contents of the memory location 2040H to (A), and subtract the contents of the memory location 2041H from the first sum. Assume the accumulator has 30H, the memory location 2040H has 68H, and the location 2041H has 7FH.

**Solution**

Before asking the microprocessor to perform any memory-related operations, we must specify the memory location by loading the HL register pair. In the example illustrated in Figure 7.10, the contents of the HL pair 2040H specify the memory location. The instruction ADD M adds 68H, the contents of memory location 2040H, to the contents of

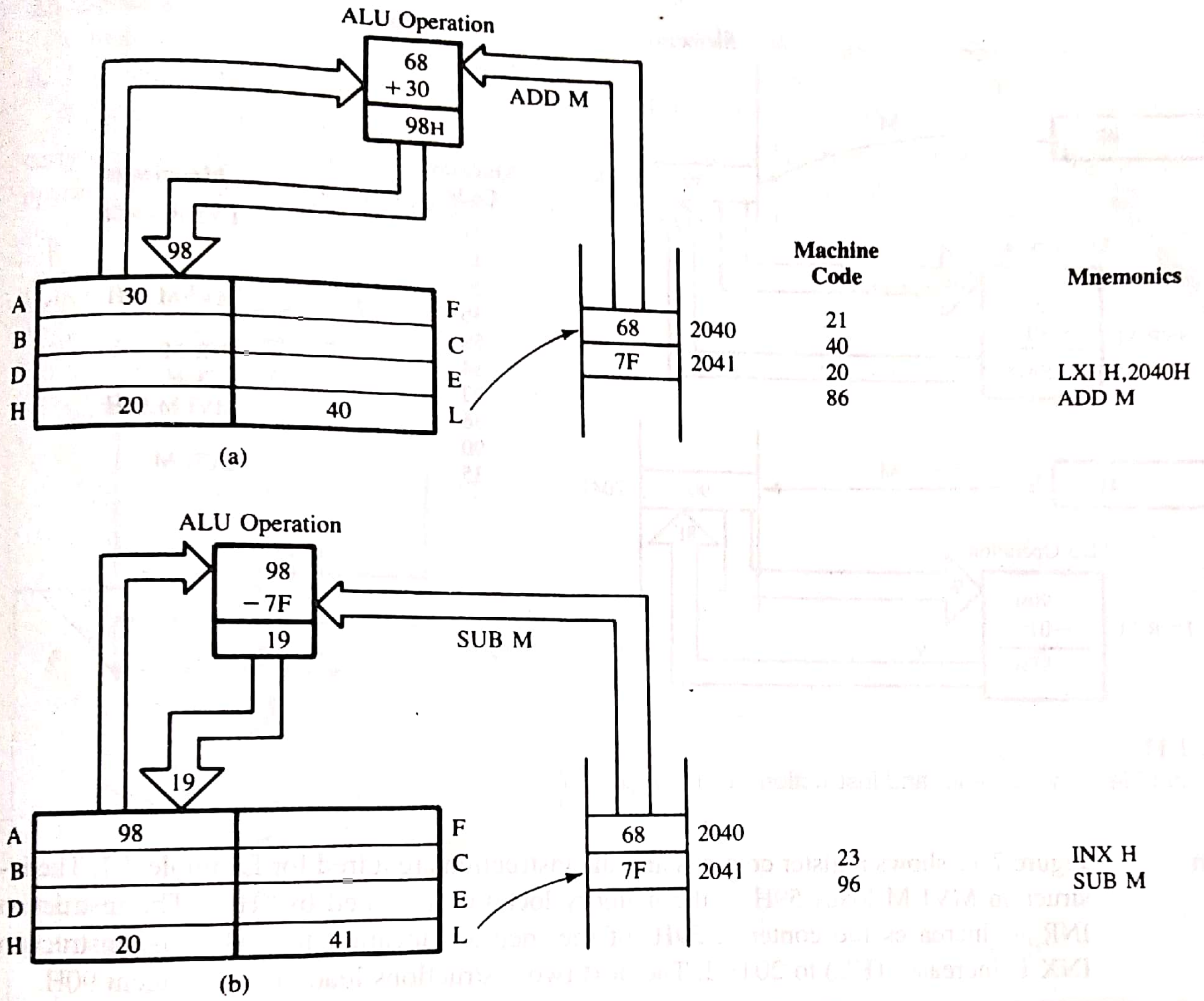


**FIGURE 7.10**
Register and Memory Contents and Instructions for Example 7.6

the accumulator (30H). The instruction INX H points to the next memory location, 2041H, and the instruction SUB M subtracts the contents (7FH) of memory location 2041H from the previous sum.

Example 7.7

Write instructions to

1. load 59H in memory location 2040H, and increment the contents of the memory location.
2. load 90H in memory location 2041H, and decrement the contents of the memory location.

## 7.32 Illustrative Program: Addition with Carry

### PROBLEM STATEMENT

Six bytes of data are stored in memory locations starting at XX50H. Add all the data bytes. Use register B to save any carries generated, while adding the data bytes. Display the entire sum at two output ports, or store the sum at two consecutive memory locations XX70H and XX71H.

**Data(H)** A2, FA, DF, E5, 98, 8B

### PROBLEM ANALYSIS

This problem can be analyzed in relation to the general flowchart in Figure 7.3 as follows:

1. Because of the memory-related arithmetic instructions just introduced in this section, two blocks in the general flowchart—data acquisition and data processing—can be combined in one instruction.

After the addition, it is necessary to check whether that operation has gene[rated a] carry (Block 3A). If a carry is generated, the carry register is incremented by o[ne (Block] 3B); otherwise, it is bypassed. The instruction ADC (Add with Carry) is inapp[ropriate for] this operation. (See Appendix F for the description of the instruction ADC.)

**PROGRAM**

| Memory Address HI-LO | Machine Code | Label | Instructions Opcode | Instructions Operand | Comments |
|---|---|---|---|---|---|
| XX00 | AF | | XRA | A | ;Clear (A) to save sum |
| 01 | 47 | | MOV | B,A | ;Clear (B) to save carry |
| 02 | 0E | | MVI | C,06H | ;Set up register C as a counter |
| 03 | 06 | | | | |
| 04 | 21 | | LXI | H,XX50H | ;Set up HL as memory pointer |
| 05 | 50 | | | | |
| 06 | XX | | | | |
| 07 | 86 | NXTBYT: | ADD | M | ;Add byte from memory |
| 08 | D2 | | JNC | NXTMEM | ;If no carry, do not increment ; carry register |
| 09 | 0C | | | | |
| 0A | XX | | | | |
| 0B | 04 | | INR | B | ;If carry, save carry bit |
| 0C | 23 | NXTMEM: | INX | H | ;Point to next memory location |
| 0D | 0D | | DCR | C | ;One addition is completed; ; decrement counter |
| 0E | C2 | | JNZ | NXTBYT | ;If all bytes are not yet added, ; go back to get next byte |
| 0F | 07 | | | | |
| 10 | XX | | | | |
| | | ;Output Display | | | |
| 11 | D3 | | OUT | PORT1 | ;Display low-order byte of the ; sum at PORT1 |
| 12 | PORT1 | | | | |
| 13 | 78 | | MOV | A,B | ;Transfer carry to accumulator |
| 14 | D3 | | OUT | PORT2 | ;Display carry digits |
| 15 | PORT2 | | | | |
| 16 | 76 | | HLT | | ;End of program |

;Storing in Memory—Alternative to Output Display

| | | | | |
|---|---|---|---|---|
| 11 | 21 | LXI | H,XX70H | ;Point to the memory ; location to store answer |
| 12 | 70 | | | |
| 13 | XX | | | |
| 14 | 77 | MOV | M,A | ;Store low-order byte at XX70H |
| 15 | 23 | INX | H | ;Point to location XX71H |
| 16 | 70 | MOV | M,B | ;Store carry bits |
| 17 | 76 | HLT | | ;End of program |

## LOGIC OPERATIONS: ROTATE 7.4

In the last chapter, the logic instructions concerning the four operations AND, OR, Ex-OR, and NOT were introduced. This chapter introduces instructions related to rotating the accumulator bits. The opcodes are as follows:

- ☐ RLC: Rotate Accumulator Left
- ☐ RAL: Rotate Accumulator Left Through Carry
- ☐ RRC: Rotate Accumulator Right
- ☐ RAR: Rotate Accumulator Right Through Carry

### 7.41 Instructions

This group has four instructions; two are for rotating left and two are for rotating right. The differences between these instructions are illustrated in the following examples.

1. RLC: Rotate Accumulator Left
   - ☐ Each bit is shifted to the adjacent left position. Bit $D_7$ becomes $D_0$.
   - ☐ CY flag is modified according to bit $D_7$.

---

**Example 7.8**

Assume the accumulator contents are AAH and CY = 0. Illustrate the accumulator contents after the execution of the RLC instruction twice.

**Solution**

Figure 7.13 shows the contents of the accumulator and the CY flag after the execution of the RLC instruction twice. The first RLC instruction shifts each bit to the left by one position, places bit $D_7$ in bit $D_0$ and sets the CY flag because $D_7 = 1$. The accumulator byte AAH becomes 55H after the first rotation. In the second rotation, the byte is again AAH, and the CY flag is reset because bit $D_7$ of 55H is 0.

---

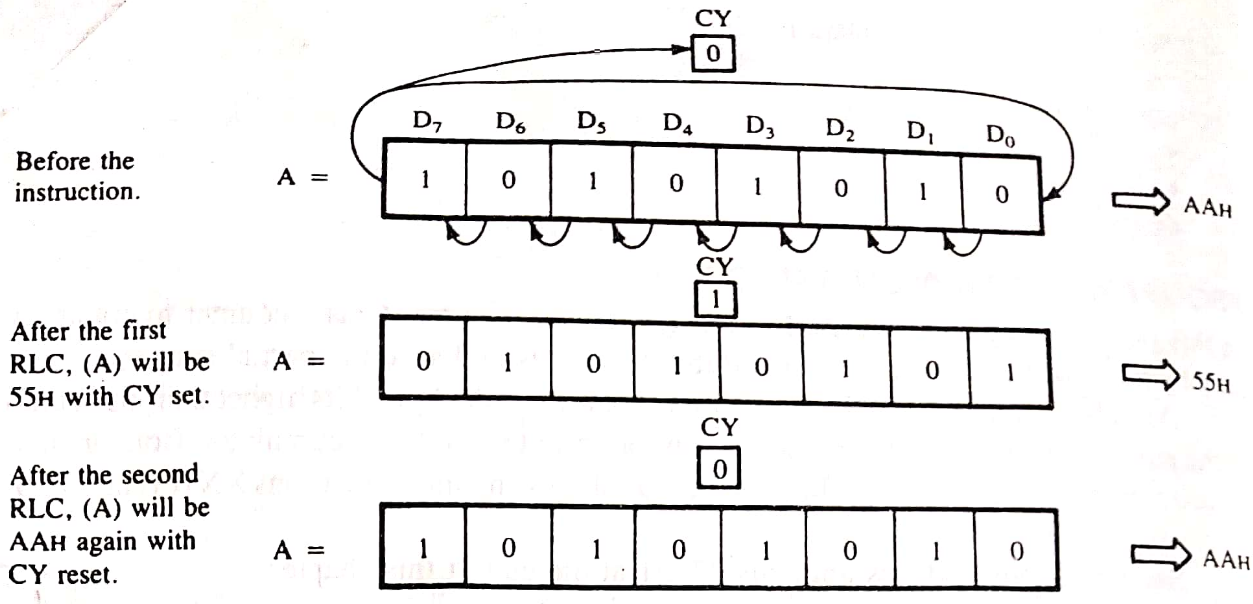


**FIGURE 7.13**
Accumulator Contents after RLC

2. RAL: Rotate Accumulator Left Through Carry
   - ☐ Each bit is shifted to the adjacent left position. Bit $D_7$ becomes the carry bit and the carry bit is shifted into $D_0$.
   - ☐ The Carry flag is modified according to bit $D_7$.

---

**Example 7.9** Assume the accumulator contents are AAH and CY = 0. Illustrate the accumulator contents after the execution of the instruction RAL twice.

**Solution** Figure 7.14 shows the contents of the accumulator and the CY flag after the execution of the RAL instruction twice. The first RAL instruction shifts each bit to the left by one position, places bit $D_7$ in the CY flag, and the CY bit in bit $D_0$. This is a 9-bit rotation; CY is assumed to be the ninth bit of the accumulator. The accumulator byte AAH becomes 54H after the first rotation. In the second rotation, the byte becomes A9H, and the CY flag is reset.

---

Examining these two examples, you may notice that the primary difference between these two instructions is that (1) the instruction RLC rotates through eight bits, and (2) the instruction RAL rotates through nine bits.

3. RRC: Rotate Accumulator Right
   - ☐ Each bit is shifted right to the adjacent position. Bit $D_0$ becomes $D_7$.
   - ☐ The Carry flag is modified according to bit $D_0$.
4. RAR: Rotate Accumulator Right Through Carry
   - ☐ Each bit is shifted right to the adjacent position. Bit $D_0$ becomes the carry bit, and the carry bit is shifted into $D_7$.

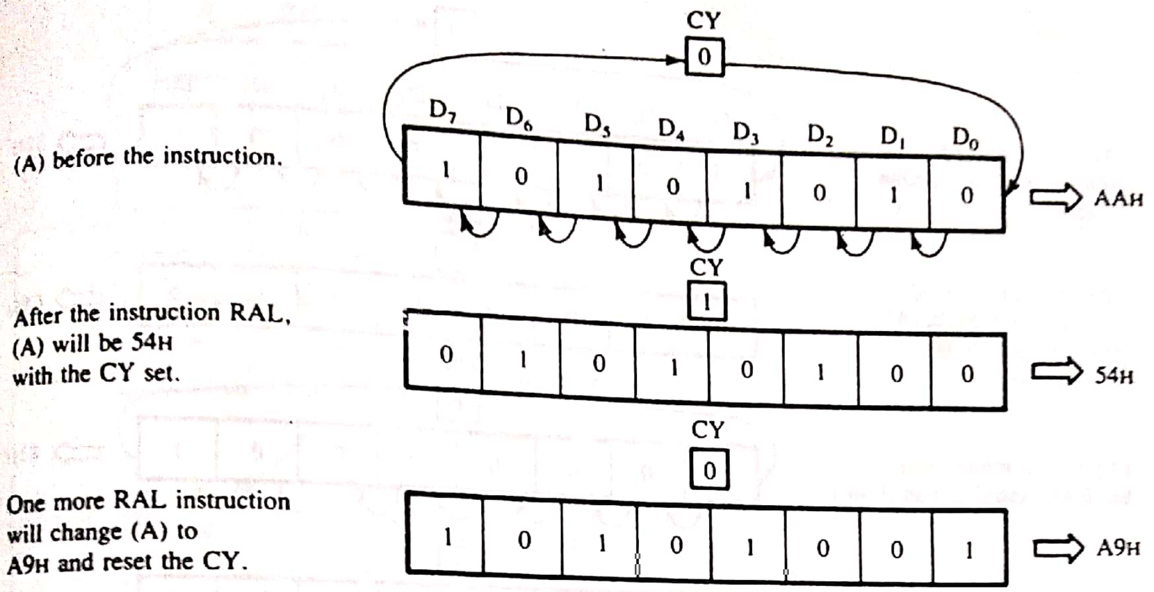


**FIGURE 7.14**
Accumulator Contents after RAL

---

**Example 7.10**

Assume the contents of the accumulator are 81H and CY = 0. Illustrate the accumulator contents after the RRC and RAR instructions.

**Solution**

Figure 7.15 shows the changes in the contents of the accumulator (81H) when the RRC instruction is used and when the RAR instruction is used. The 8-bit rotation of the RRC instruction changes 81H into C0H, and the 9-bit rotation of the RAR instruction changes 81H into 40H.

---

## APPLICATIONS OF ROTATE INSTRUCTIONS

The rotate instructions are primarily used in arithmetic multiply and divide operations and for serial data transfer.

For example, if (A) is 0000 1000 = 08H,

- By rotating 08H right: (A) = 0000 0100 = 04H
  This is equivalent to dividing by 2
- By rotating 08H left: (A) = 0001 0000 = 10H
  This is equivalent to multiplying by 2 ($10H = 16_{10}$)

However, these procedures are invalid when logic 1 is rotated left from $D_7$ to $D_0$ or vice versa. For example, if 80H is rotated left, it becomes 01H. Applications of serial data transfer are discussed in Chapter 16.

(A) will be rotated with the RRC instruction as shown.

After the execution of the instruction RRC, (A) will be $C0_H$ with the CY flag set.

(A) will be rotated with the RAR instruction as shown.

After the execution of the RAR instruction, (A) will be $40_H$ with the CY flag set.

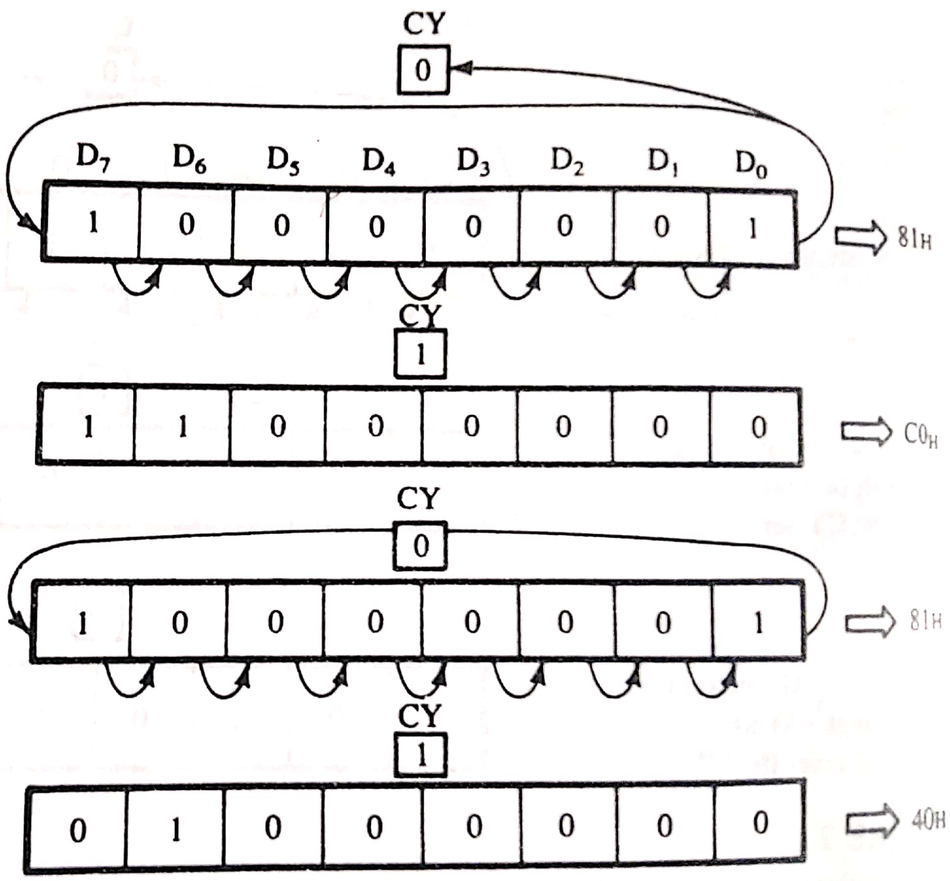


**FIGURE 7.15**
Rotate Right Instructions

| | |
|---|---|
| 68 | 19 |
| 69 | F2 |
| | 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 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.

### 7.51 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 location OUT1.

**Solution**

Figure 7.17 illustrates Example 7.11.

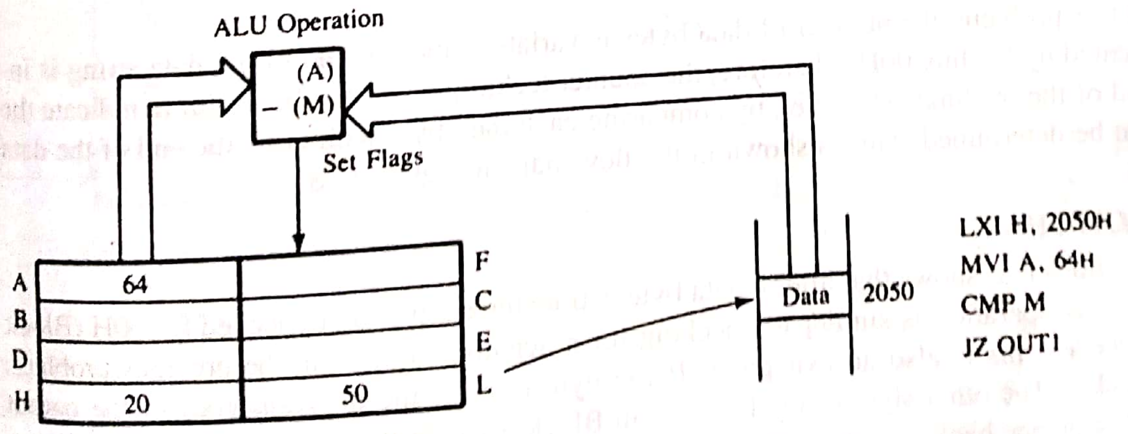


FIGURE 7.17
Compare Instructions

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.**

## 7.61 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

- ☐ incorrect addresses
- ☐ incorrect jump locations for loops
- ☐ incorrect data or missing codes

To use this technique effectively, you will have to reduce loop and delay counts to a minimum number. For example, in a program that transfers 100 bytes, it is meaningless to set the count to 100 and single-step the program 100 times. By reducing the count to two bytes, you will be able to observe the execution of the loop. (If you reduce the count to one byte, you may not be able to observe the execution of the loop.) By single-stepping the program, you will be able to infer the flag status by observing the execution of Jump instructions. The single-step technique is very useful for short programs.

**REGISTER EXAMINE**

The Register Examine key allows you to examine the contents of the microprocessor register. When appropriate keys are pressed, the monitor program can display the contents of the registers. This technique is used in conjunction either with the single-step or the breakpoint facilities discussed below.

After executing a block of instructions, you can examine the register contents at a critical juncture of the program and compare these contents with the expected outcomes.

**BREAKPOINT**

In a single-board computer, the breakpoint facility is, generally, a software routine that allows you to execute a program in sections. The breakpoint can be set in your program by using RST instructions. (See "Interrupts," Chapter 12.) When you push the Execute key, your program will be executed until the breakpoint, where the monitor takes over again. The registers can be examined for expected results. If the segment of the program is found satisfactory, a second breakpoint can be set at a subsequent memory address to debug the next segment of the program. With the breakpoint facility you can isolate the segment of the program with errors. Then that segment of the program can be debugged with the single-step facility. The breakpoint technique can be used to check out the timing loop, I/O section, and interrupts. (See Chapter 12 for how to write a breakpoint routine.)

## 7.62 Common Sources of Errors

Common sources of errors in the instructions and programs illustrated in this chapter are as follows:

1. Failure to clear the accumulator when it is used to add numbers.
2. Failure to clear the carry registers or keep track of a carry.
3. Failure to update a memory pointer or a counter.
4. Failure to set a flag before using a conditional Jump instruction.
5. Inadvertently clearing the flag before using a Jump instruction.
6. Specification of a wrong memory address for a Jump instruction.
7. Use of an improper combination of Rotate instructions.