5. The logic operations cannot be performed directly with the contents of two registers.
6. The individual bits in the accumulator can be set or reset using logic instructions.

See Questions and Assignments 20–29 at the end of this chapter.

6.4 BRANCH OPERATIONS

The **branch instructions** are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions. These instructions are the key to the flexibility and versatility of a computer.

The microprocessor is a sequential machine; it executes machine codes from one memory location to the next. Branch instructions instruct the microprocessor to go to a different memory location, and the microprocessor continues executing machine codes from that new location. The address of the new memory location is either specified explicitly or supplied by the microprocessor or by extra hardware. The branch instructions are classified in three categories:

1. Jump instructions
2. Call and Return instructions
3. Restart instructions

This section is concerned with applications of Jump instructions. The Call and Return instructions are associated with the subroutine technique and will be discussed in Chapter 9; Restart instructions are associated with the interrupt technique and will be discussed in Chapter 12.

The Jump instructions specify the memory location explicitly. They are 3-byte instructions: one byte for the operation code, followed by a 16-bit memory address. Jump instructions are classified into two categories: Unconditional Jump and Conditional Jump.

6.41 Unconditional Jump

The 8085 instruction set includes one unconditional Jump instruction. The unconditional Jump instruction enables the programmer to set up continuous loops.

INSTRUCTION

Opcode	Operand	Description
JMP	16-bit	Jump ☐ This is a 3-byte instruction ☐ The second and third bytes specify the 16-bit memory address. However, the second byte specifies the low-order and the third byte specifies the high-order memory address

For example, to instruct the microprocessor to go to the memory location 2000H, the mnemonics and the machine code entered will be as follows:

Machine Code	Mnemonics
C3	JMP 2000H
00	
20	

Note the sequence of the machine code. The 16-bit memory address of the jump location is entered in the reverse order, the low-order byte (00H) first, followed by the high-order byte (20H). The 8085 is designed for such a reverse sequence. The jump location can also be specified using a label. While writing a program, you may not know the exact memory location to which a program sequence should be directed. In that case, the memory address can be specified with a label (or a name). This is particularly useful and necessary for an assembler. However, you should not specify both a label and its 16-bit address in a Jump instruction. Furthermore, you cannot use the same label for different memory locations. The next illustrative program shows the use of the Jump instruction.

monitors the input port continuously. The output wi
sitions.

6.43 Conditional Jumps

Conditional Jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags. After logic and arithmetic operations, flip-flops (flags) are set or reset to reflect data conditions. The conditional Jump instructions check the flag conditions and make decisions to change or not to change the sequence of a program.

FLAGS

The 8085 flag register has five flags, one of which (Auxiliary Carry) is used internally. The other four flags used by the Jump instructions are

1. Carry flag
2. Zero flag
3. Sign flag
4. Parity flag

Two Jump instructions are associated with each flag. The sequence of a program can be changed either because the condition is present or because the condition is absent. For example, while adding the numbers you can change the program sequence either because the carry is present (JC = Jump On Carry) or because the carry is absent (JNC = Jump On No Carry).

INSTRUCTIONS

All conditional Jump instructions in the 8085 are 3-byte instructions; the second byte specifies the low-order (line number) memory address, and the third byte specifies the high-order (page number) memory address. The following instructions transfer the program sequence to the memory location specified under the given conditions:

Opcode	Operand	Description
JC	16-bit	Jump On Carry (if result generates carry and CY = 1)
JNC	16-bit	Jump On No Carry (CY = 0)
JZ	16-bit	Jump On Zero (if result is zero and Z = 1)
JNZ	16-bit	Jump On No Zero (Z = 0)
JP	16-bit	Jump On Plus (if D_7 = 0, and S = 0)
JM	16-bit	Jump On Minus (if D_7 = 1, and S = 1)
JPE	16-bit	Jump On Even Parity (P = 1)
JPO	16-bit	Jump On Odd Parity (P = 0)

All the Jump instructions are listed here for an overview. The Zero and Carry flags and related Jump instructions are used frequently. They are illustrated in the following examples.

6.44 Illustrative Program: Testing of the Carry Flag

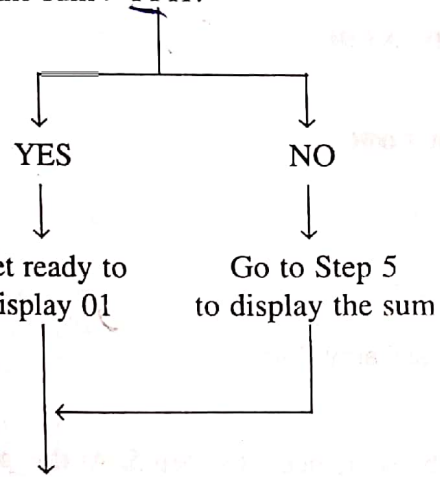
PROBLEM STATEMENT

Load the hexadecimal numbers 9BH and A7H in registers D and E, respectively, and add the numbers. If the sum is greater than FFH, display 01H at output PORT0; otherwise, display the sum.

PROBLEM ANALYSIS AND FLOWCHART

The problem can be divided into the following steps:

1. Load the numbers in the registers.
2. Add the numbers.
3. Check the sum.
   Is the sum > FFH?



4. Get ready to display 01 / Go to Step 5 to display the sum
5. Display.
6. End.

```
MVI  D, 9BH
MVI  E, A7H
MOV  A, D
ADD  E
JNC  DISPLAY
MVI  A, 01H
DISPLAY: OUT 00H
```

FLOWCHART AND ASSEMBLY LANGUAGE PROGRAM

The six steps listed above can be converted into a flowchart and assembly language program as shown in Figure 6.10.

Step 3 is a decision-making block. In a flowchart, the decision-making process is represented by a diamond shape. It is important to understand how this block is translated into the assembly language program. By examining the block carefully you will notice the following:

1. The question is: Is there a Carry?
2. If the answer is no, change the sequence of the program. In the assembly language this is equivalent to Jump On No Carry—JNC.

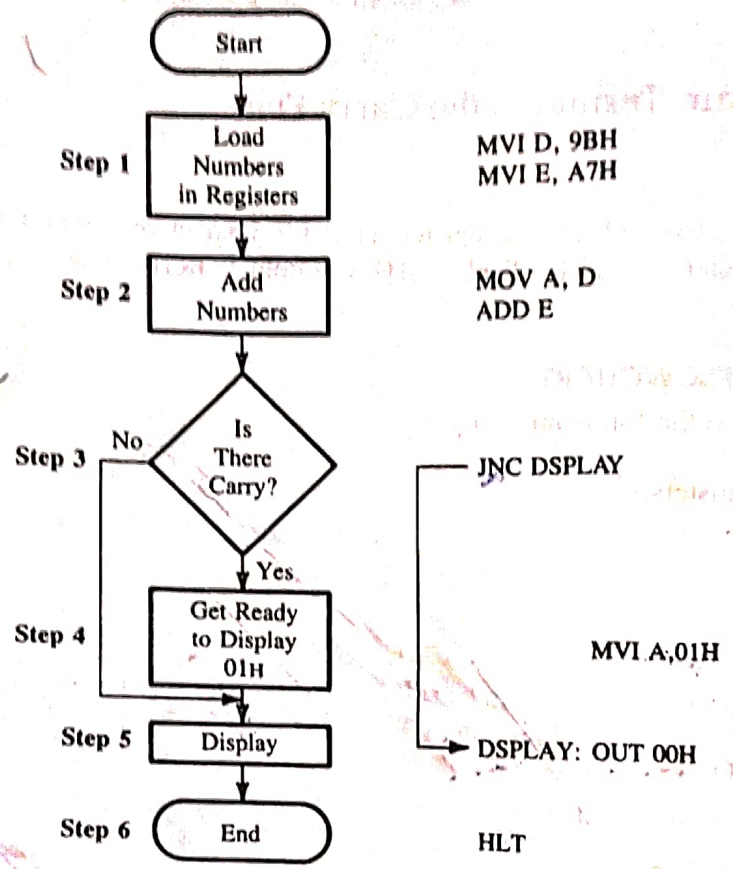


FIGURE 6.10
Flowchart and Assembly Language Program to Test Carry Flag

3. Now the next question is where to change the sequence—to Step 5. At this point the exact location is not known, but it is labeled DSPLAY.
4. The next step in the sequence is 4. Get ready to display byte 01H.
5. After completing the straight line sequence, translate Step 5 and Step 6: Display at the port and halt.

MACHINE CODE WITH MEMORY ADDRESSES

Assuming your R/W memory begins at 2000H, the preceding assembly language program can be translated as follows:

Memory Address	Machine Code	Label	Mnemonics
2000	16	START:	MVI D,9BH
2001	9B		
2002	1E		MVI E,A7H
2003	A7		

2004	7A		MOV A,D
2005	83		ADD E
2006	D2		JNC DSPLAY
2007	X		
2008	X		
2009	3E		MVI A,01H
200A	01		
200B	D3	DSPLAY:	OUT 00H
200C	00		
200D	76		HLT

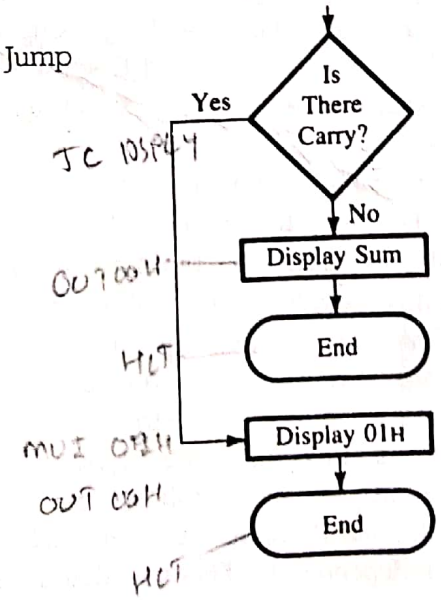
While translating into the machine code, we leave memory locations 2007H and 2008H blank because the exact location of the transfer is not known. What is known is that two bytes should be reserved for the 16-bit address. After completing the straight line sequence, we know the memory address of the label DSPLAY; i.e., 200BH. This address must be placed in the reversed order as shown:

2007	0B	Low-order: Line Number
2008	20	High-order: Page Number

USING THE INSTRUCTION JUMP ON CARRY (JC)

Now the question remains: Can the same problem be solved by using the instruction Jump On Carry (JC)? To use instruction JC, exchange the places of the answers YES and NO to the question: Is there a Carry? The flowchart will be as in Figure 6.11, and it shows that the program sequence is changed if there is a Carry. This flowchart has two end points; thus it will require a few more instructions than that of Figure 6.10. In this particular example, it is unimportant whether to use instruction JC or JNC, but in most cases the choice is made by the logic of a problem.

FIGURE 6.11
Flowchart for the Instruction Jump On Carry



WRITING ASSEMBLY LANGUAGE PROGRAMS

Communicating with a microcomputer—giving it commands to perform a task and watching it perform them—is exciting. However, one can be uneasy communicating in strange mnemonics and hexadecimal machine codes. This feeling is like the uneasiness one has when beginning to speak a foreign language. How do we learn to communicate with a microcomputer in its assembly language? By using a few mnemonics at a time such as the mnemonics for Read the switches and Display the data. This chapter has introduced a group of basic instructions that can command the 8085 microprocessor to perform simple tasks.

After we know a few instructions, how do we begin to write a program? Any program, no matter how large, begins with mnemonics. And, just as several persons contribute to the construction of a hundred-story building, so the writing of a large program is usually the work of a team. In addition, the 8085 instruction set contains only 74 different instructions, some of them used quite frequently.

In a hundred-story building, most of the rooms are similar. If one knows the basic fundamentals of constructing a room, one can learn how to tie these rooms together in a coherent structure. However, planning and forethought are critical. Before beginning to build a structure, an architectural plan must be drawn. Similarly, to write a program, one needs to draw up a plan of logical thoughts. A given task should be broken down into small units that can be built independently. This is called the **modular design approach.**

6.51 Getting Started

Writing a program is equivalent to giving specific *commands* to the microprocessor in a *sequence* to *perform a task.* The italicized words provide clues to writing a program. Let us examine these terms.

- ☐ *Perform a Task.* What is the task you are asking it to do?
- ☐ *Sequence.* What is the sequence you want it to follow?
- ☐ *Commands.* What are the commands (instruction set) it can understand?

These terms can be translated into steps as follows:

Step 1: Read the problem carefully.
Step 2: Break it down into small steps.
Step 3: Represent these small steps in a possible sequence with a flowchart—a plan of attack.
Step 4: Translate each block of the flowchart into appropriate mnemonic instructions.
Step 5: Translate mnemonics into the machine code.
Step 6: Enter the machine code in memory and execute. Only on rare occasions is a program successfully executed on the first attempt.
Step 7: Start troubleshooting (see Section 6.6, "Debugging a Program").

These steps are illustrated in the next section.

Debugging a program is similar to troubleshooting hardware, but it is much more difficult and cumbersome. It is easy to poke and pinch at the components in a circuit, but, in a program, the result is generally binary: either it works or it does not work. When it does not work, very few clues alert you to what exactly went wrong. Therefore, it is essential to search carefully for the errors in the program logic, machine codes, and execution.

The debugging process can be divided into two parts: static debugging and dynamic debugging.

Static debugging is similar to visual inspection of a circuit board; it is done by a paper-and-pencil check of a flowchart and machine code. **Dynamic debugging** involves observing the output, or register contents, following the execution of each instruction (the single-step technique) or of a group of instructions (the breakpoint technique). Dynamic debugging will be discussed in the next chapter.

6.61 Debugging Machine Code

Translating the assembly language to the machine code is similar to building a circuit from a schematic diagram; the machine code will have errors just as would the circuit board. The following errors are common:

1. Selecting a wrong code.
2. Forgetting the second or third byte of an instruction.
3. Specifying the wrong jump location.
4. Not reversing the order of high and low bytes in a Jump instruction.
5. Writing memory addresses in decimal, thus specifying wrong jump locations.

The program for controlling manufacturing processes listed in Section 6.53 has several of these errors. These errors must be corrected before entering the machine code in the R/W memory of your system.

PROGRAMMING TECHNIQUES: LOOPING, COUNTING, AND INDEXING

The programming examples illustrated in previous chapters are simple and can be solved manually. However, the computer surpasses manual efficiency when tasks must be repeated, such as adding a hundred numbers or transferring a thousand bytes of data. It is fast and accurate.

The programming technique used to instruct the microprocessor to repeat tasks is called **looping.** A loop is set up by instructing the microprocessor to change the sequence of execution and perform the task again. This process is accomplished by using Jump instructions. In addition, techniques such as counting and indexing (described below) are used in setting up a loop.

Loops can be classified into two groups:

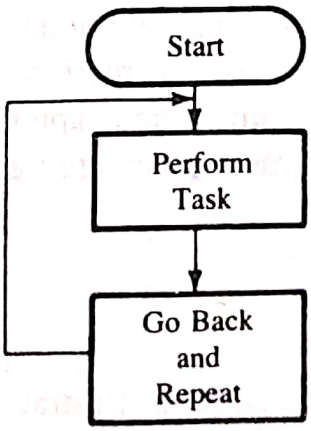
- ☐ Continuous loop—repeats a task continuously
- ☐ Conditional loop—repeats a task until certain data conditions are met

They are described in the next two sections.

7.11 Continuous Loop

A continuous loop is set up by using the unconditional Jump instruction shown in the flowchart (Figure 7.1).

FIGURE 7.1
Flowchart of a Continuous Loop



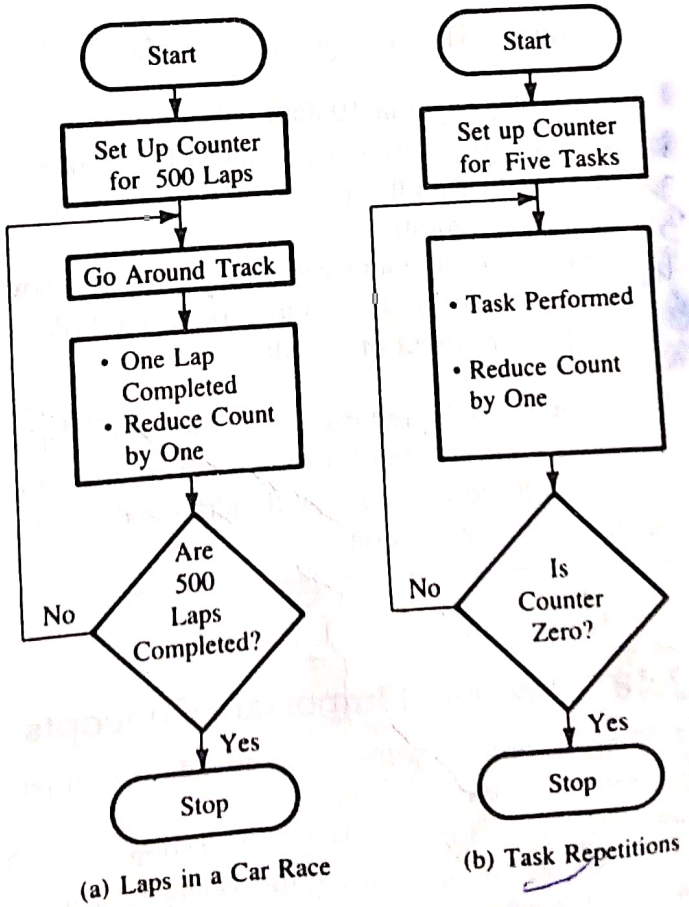
A program with a continuous loop does not stop repeating the tasks until the system is reset. Typical examples of such a program include a continuous counter (see Chapter 8, Section 8.2) or a continuous monitor system.

7.12 Conditional Loop

A conditional loop is set up by the conditional Jump instructions. These instructions check flags (Zero, Carry, etc.) and repeat the specified tasks if the conditions are satisfied. These loops usually include counting and indexing.

CONDITIONAL LOOP AND COUNTER

A counter is a typical application of the conditional loop. For example, how does the microprocessor repeat a task five times? The process is similar to that of a car racer in the Indy 500 going around the track 500 times. How does the racer know when 500 laps have been completed? The racing team manager sets up a counting and flagging method for the racer. This can be symbolically represented as in Figure 7.2(a). A similar approach is needed for the microprocessor to repeat the task five times. The microprocessor needs a counter, and when the counting is completed, it needs a flag. This can be accomplished with the conditional loop, as illustrated in the flowchart in Figure 7.2(b).



(a) Laps in a Car Race (b) Task Repetitions

FIGURE 7.2
Flowcharts to Indicate Number of Repetitions Completed

The computer flowchart of Figure 7.2(b) is translated into a program as follows:

1. Counter is set up by loading an appropriate count in a register.
2. Counting is performed by either incrementing or decrementing the counter.
3. Loop is set up by a conditional Jump instruction.
4. End of counting is indicated by a flag.

It is easier to count down to zero than to count up because the Zero flag is set when the register becomes zero. (Counting up requires the Compare instruction, which is introduced later.)

Conditional Loop, Counter, and Indexing Another type of loop includes indexing along with a counter. (*Indexing* means pointing or referencing objects with sequential numbers. In a library, books are arranged according to numbers, and they are referred to or sorted by numbers. This is called indexing.) Similarly, data bytes are stored in memory locations, and those data bytes are referred to by their memory locations.

7.2 ADDITIONAL DATA TRANSFER AND 16-BIT ARITHMETIC INSTRUCTIONS

The instructions related to the data transfer among microprocessor registers and the I/O instructions were introduced in the last chapter; this section introduces the instructions related to the data transfer between the microprocessor and memory. In addition, instructions for some 16-bit arithmetic operations are included because they are necessary for using the programming techniques introduced earlier in this chapter. The opcodes are as follows:

1. Loading 16-bit data in register pairs
   LXI Rp: Load Register Pair Immediate
2. Data transfer (copy) from memory to the microprocessor
   MOV R,M: Move (from memory to register)
   LDAX B/D: Load Accumulator Indirect
   LDA 16-bit: Load Accumulator Direct
3. Data transfer (copy) from the microprocessor to memory
   MOV M,R: Move (from register to memory)
   STAX B/D: Store Accumulator Indirect
   STA 16-bit: Store Accumulator Direct
4. Loading 8-bit data directly in memory register (location)
   MVI M,8-bit: Load 8-bit data in memory
5. Incrementing/Decrementing Register Pair
   INX Rp: Increment Register Pair
   DCX Rp: Decrement Register Pair

The instructions related to these operations are illustrated with examples in the following sections.

7.21 16-Bit Data Transfer to Register Pairs (LXI)

The LXI instructions perform functions similar to those of the MVI instructions, except that the LXI instructions load 16-bit data in register pairs and the stack pointer register. These instructions do not affect the flags.

INSTRUCTIONS

Opcode	Operand	
LXI	Rp, 16-bit	Load Register Pair
LXI	B, 16-Bit	☐ This is a 3-byte instruction
LXI	D,16-bit	☐ The second byte is loaded in the low-order register of the register pair (e.g., register C)
LXI	H,16-bit	☐ The third byte is loaded in the high-order register pair (e.g., register B)

LXI SP,16-bit

- There are four such instructions in the set as shown. The operands B, D, and H represent BC, DE, and HL registers, and SP represents the stack pointer register

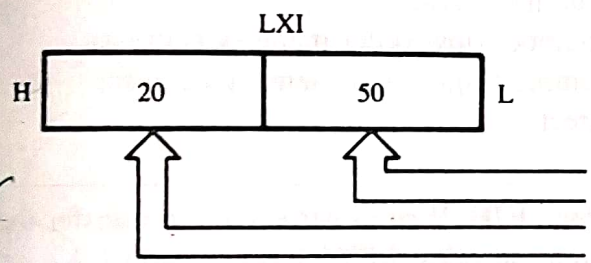
---

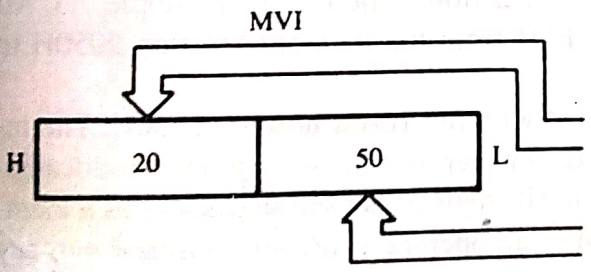
Example 7.2

Write instructions to load the 16-bit number 2050H in the register pair HL using LXI and MVI opcodes, and explain the difference between the two instructions.

Instructions Figure 7.4 shows the register contents and the instructions required for Example 7.2.

The LXI instruction is functionally similar to two MVI instructions. The LXI instruction takes three bytes of memory and requires ten clock periods (T-states). On the other hand, two MVI instructions take four bytes of memory and require 14 clock periods (T-states).





Machine Code	Mnemonics	Comments
21	LXI H,2050H	;Load HL registers
50*		;50H in L register and
20		;20H in H register
26	MVI H,20H	;Load 20H in register H
20		
2E	MVI L,50H	;Load 50H in register L
50		

*NOTE: The order of the LXI machine code is reversed in relation to the mnemonics; low-order byte first followed by the high-order byte. This is similar to Jump instructions.

FIGURE 7.4
Instructions and Register Contents for Example 7.2

7.22 Data Transfer (Copy) from Memory to the Microprocessor

The 8085 instruction set includes three types of memory transfer instructions; two use the indirect addressing mode and one uses the direct addressing mode. These instructions do not affect the flags.

1. MOV R,M: Move (from Memory to Register)
   - This is a 1-byte instruction

- ☐ It copies the data byte from the memory location into a register
- ☐ R represents microprocessor registers A, B, C, D, E, H, and L
- ☐ The memory location is specified by the contents of the HL register
- ☐ This specification of the memory location is indirect; it is called the indirect addressing mode

2. LDAX B/D: Load Accumulator Indirect

- ☐ This is a 1-byte instruction
- ☐ It copies the data byte from the memory location into the accumulator

LDAX B
LDAX D

- ☐ The instruction set includes two instructions as shown
- ☐ The memory location is specified by the contents of the registers BC or DE
- ☐ The addressing mode is indirect

3. LDA 16-bit: Load Accumulator Direct

- ☐ This is a 3-byte instruction
- ☐ It copies the data byte from the memory location specified by the 16-bit address in the second and third byte to the accumulator
- ☐ The second byte is a line number (low-order memory address)
- ☐ The third byte is a page number (high-order memory address)
- ☐ The addressing mode is direct

Example 7.3 The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to the accumulator using three different opcodes: MOV, LDAX, and LDA.

Solution

Figure 7.5 shows the register contents and the instructions required for Example 7.3. All of these three instructions copy the data byte F7H from the memory location 2050H to the accumulator.

In Figure 7.5(a), register HL is first loaded with the 16-bit number 2050H. The instruction MOV A,M uses the contents of the HL register as a memory pointer to location 2050H; this is the indirect addressing mode. The HL register is used frequently as a memory pointer because any instruction that uses M as an operand can copy from and into any one of the registers.

In Figure 7.5(b), the contents of register BC are used as a memory pointer to location 2050H by the instruction LDAX B. Registers BC and DE can be used as restricted memory pointers to copy the contents of only the accumulator into memory and vice versa; however, they cannot be used to copy the contents of other registers.

Figure 7.5(c) illustrates the direct addressing mode; the instruction LDA specifies the memory address 2050H directly as a part of its operand.

After examining all three methods, you may notice that the indirect addressing mode takes four bytes and the direct addressing mode takes three bytes. The question is: Why not just use the direct addressing mode?

If only one byte is to be transferred, the LDA instruction is more efficient. But for a block of memory transfer, the instruction LDA (three bytes) will have to be repeated for

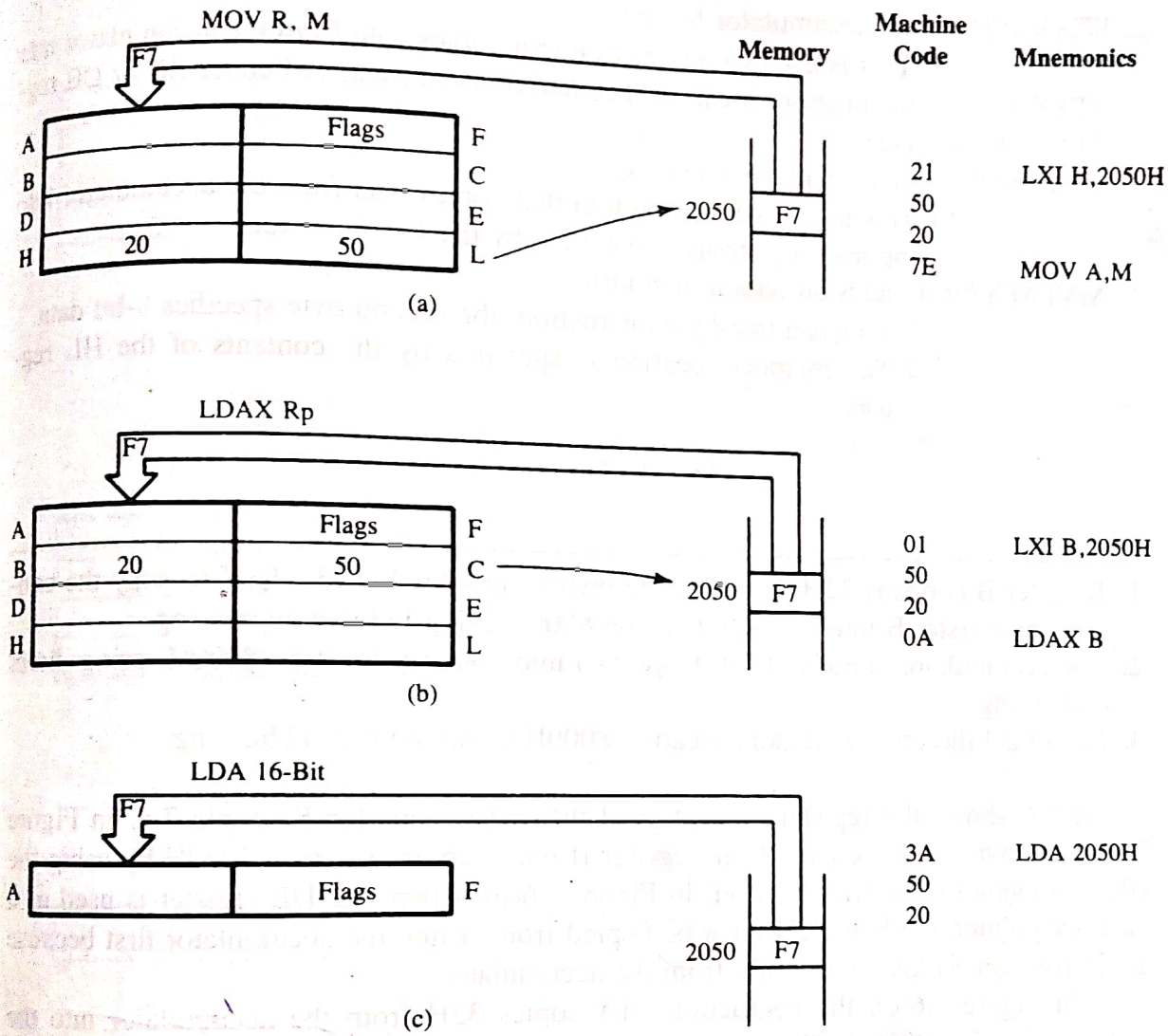


FIGURE 7.5
Instructions and Register Contents for Example 7.3

each memory. On the other hand, a loop can be set up with two other instructions, and the contents of a register pair can be incremented or decremented. This is further illustrated in Section 7.26.

7.23 Data Transfer (Copy) from the Microprocessor to Memory or Directly into Memory

The instructions for copying data from the microprocessor to a memory location are similar to those described in the previous section. These instructions are as follows:

1. MOV M,R: Move (from Register to Memory).
   - ☐ This is a 1-byte instruction that copies data from a register, R, into the memory location specified by the contents of HL registers

2. STAX B/D: Store Accumulator Indirect
   - ☐ This is a 1-byte instruction that copies data from the accumulator into the memory location specified by the contents of either BC or DE registers

   STAX B
   STAX D
3. STA 16-bit: Store Accumulator Direct
   - ☐ This is a 3-byte instruction that copies data from the accumulator into the memory location specified by the 16-bit operand.
4. MVI M,8-bit: Load 8-bit data in memory
   - ☐ This is a two-byte instruction; the second byte specifies 8-bit data
   - ☐ The memory location is specified by the contents of the HL register

---

1. Register B contains 32H. Illustrate the instructions MOV and STAX to copy the contents of register B into memory location 8000H using indirect addressing.
2. The accumulator contains F2H. Copy (A) into memory location 8000H, using direct addressing.
3. Load F2H directly in memory location 8000H using indirect addressing.

Figure 7.6 shows the register contents and the instructions for Example 7.4. In Figure 7.6(a), the byte 32H is copied from register B into memory location 8000H by using the HL as a memory pointer. However, in Figure 7.6(b), where the DE register is used as a memory pointer, the byte 32H must be copied from B into the accumulator first because the instruction STAX copies only from the accumulator.

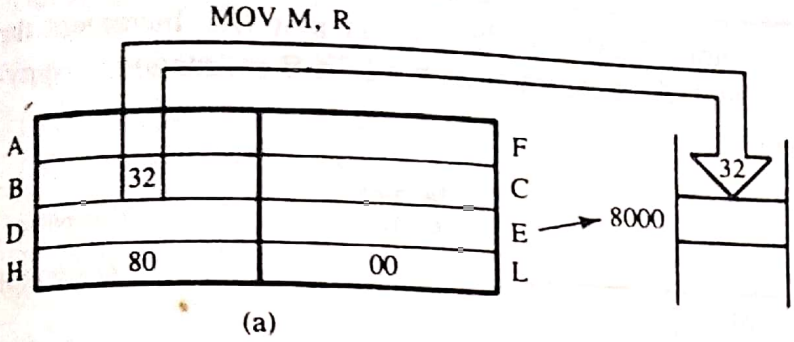
In Figure 7.6(c), the instruction STA copies 32H from the accumulator into the memory location 8000H. The memory address is specified as the operand; this is an illustration of the direct addressing mode. On the other hand, Figure 7.6(d) illustrates how to load a byte directly in memory location by using the HL as a memory pointer.

7.24 Arithmetic Operations Related to 16 Bits or Register Pairs

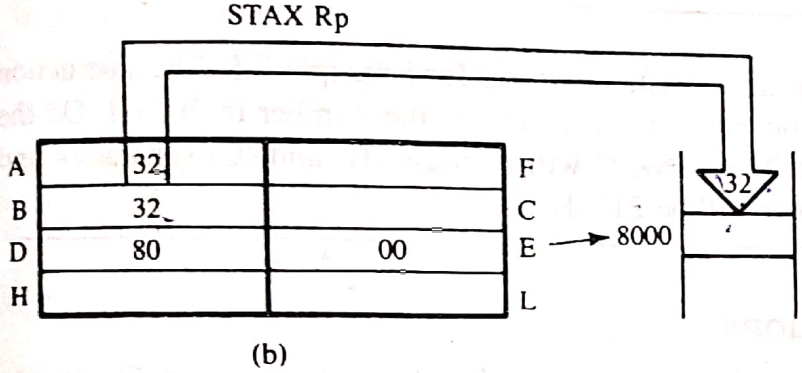
The instructions related to incrementing/decrementing 16-bit contents in a register pair are introduced below. These instructions do not affect flags.

1. INX Rp: Increment Register Pair
   - ☐ This is a 1-byte instruction
   - ☐ It treats the contents of two registers as one 16-bit number and increases the contents by 1
   - ☐ The instruction set includes four instructions, as shown

   INX B
   INX D
   INX H
   INX SP

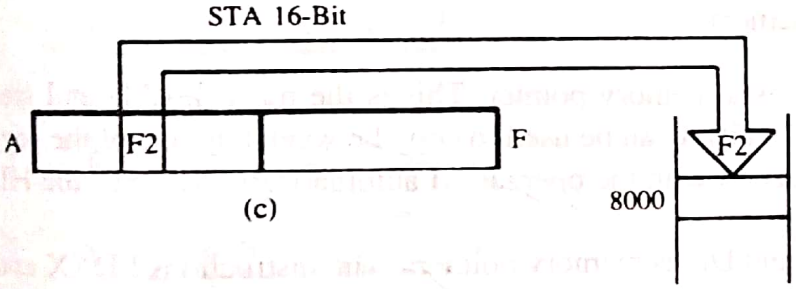


Machine Code	Mnemonics
21	LXI H,8000H
00	
80	
70	MOV M,B



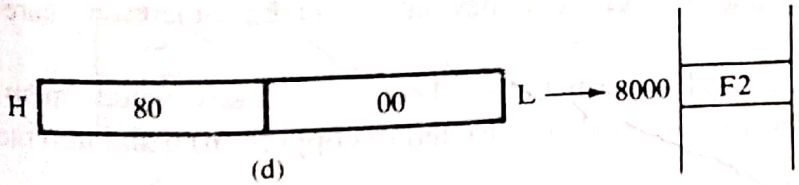
This instruction copies the contents of the accumulator into memory. Therefore, it is necessary first to copy (B) into A.

11	LXI D,8000H
00	
80	
78	MOV A,B
12	STAX D



This also requires the transfer of (B) to A.

32	STA 8000H
00	
80	



21	LXI H,8000H
00	
80	
36	MVI M,F2H
F2	

FIGURE 7.6
Instructions and Register Contents for Example 7.4

2. DCX Rp: Decrement Register Pair
   - ☐ This is a 1-byte instruction
   - ☐ It decreases the 16-bit contents of a register pair by 1
   - ☐ The instruction set includes four instructions, as shown

DCX B
DCX D
DCX H
DCX SP

7.25 Review of Instructions

In this section, we examined primarily how to copy data from the microprocessor into memory and vice versa. The 8085 instruction provides three methods of copying data between the microprocessor and memory:

1. Indirect addressing using HL as a memory pointer: This is the most flexible and frequently used method. The HL register can be used to copy between any one of the registers and memory. Any instruction with the operand M automatically assumes the HL is the memory pointer.
2. Indirect addressing using BC and DE as memory pointers: The instructions LDAX and STAX use BC and DE as memory pointers. However, this method is restricted to copying from and into the accumulator and cannot be used for other registers. In addition, the mnemonics (LDAX and STAX) are somewhat misleading; therefore, careful attention must be given to their interpretion.
3. Direct addressing using LDA and STA instructions: These instructions include memory address as the operand. This method is also restricted to copying from and into the accumulator.

In addition to the above data copy instructions, we discussed two instructions, INX and DCX, concerning the register pairs. The critical feature of these instructions is that they do not affect the flags.

7.26 Illustrative Program: Block Transfer of Data Bytes

PROBLEM STATEMENT

Sixteen bytes of data are stored in memory locations at XX50H to XX5FH. Transfer the entire block of data to new memory locations starting at XX70H.

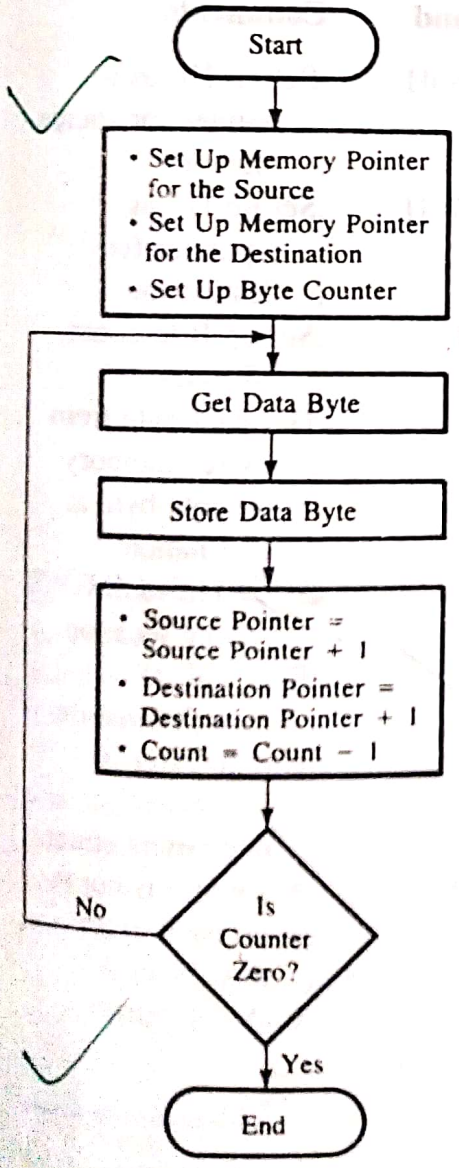
Data(H) 37, A2, F2, 82, 57, 5A, 7F, DA, E5, 8B, A7, C2, B8, 10, 19, 98

PROBLEM ANALYSIS

The problem can be analyzed in terms of the blocks suggested in the flowchart (Figure 7.8). The steps are as follows:

The flowchart in Figure 7.8 includes five blocks; these blocks are identified with numbers referring to the blocks in the generalized flowchart in Figure 7.3. This problem is not concerned with data manipulation (processing); therefore, the flowchart does not require Blocks 3 and 4 (data processing and temporary storage of partial results). The problem simply deals with the transferring of the data bytes from one location to another location in memory; therefore, the Store Data Byte block is equivalent to the Output block in the generalized flowchart.

Block 1 is the initialization block; this block sets up two memory pointers and one counter. Block 5 is concerned with updating the memory pointers and the counter. The



	Mnemonics	Steps
Block 1	LXI H,XX50H	1. Set up HL as a pointer for the source memory
	LXI D,XX70H	Set up DE as a pointer for the destination memory
	MVI B,10H	Set up B as byte counter
Block 2	NEXT: MOV A,M	2. Get data byte from the source memory
Block 7	STAX D	3. Store the data byte in destination memory
Block 5	INX H INX D DCR B	4. Get ready to transfer next byte
Block 6	JNZ NEXT	5. Go back to get next byte if byte counter ≠0
	HLT	

FIGURE 7.8
Flowchart for Block Transfer of Data Bytes

statements shown in the block appear strange if they are read as algebraic equations; however, they are not algebraic equations. The statement Pointer = Pointer + 1 means the new value is obtained by incrementing the previous value by one.

The statements in the flowchart correspond one-to-one with the mnemonics. In large programs, such details in the flowchart are impractical as well as undesirable. However, these details are included here to show the logic flow in writing programs. In Figure 7.8, some of the details can be eliminated very easily from the flowchart. For example, Blocks 2 and 7 can be combined in one statement; such as, Transfer Data Byte from Source to Destination. Similarly, Block 5 can be reduced to one statement; such as, Update memory Pointers and Counter.

PROGRAM

Memory Address HI-LO	Hex Code	Label	Instructions Opcode	Operand	Comments
XX00	21	START:	LXI	H,XX50H	;Set up HL as a
01	50				; pointer for source
02	XX				; memory
03	11		LXI	D,XX70H	;Set up DE as
04	70				; a pointer for
05	XX				; destination
06	06		MVI	B,10H	;Set up B to count
07	10				; 16 bytes
08	7E	NEXT:	MOV	A,M	;Get data byte from
					; source memory
09	12		STAX	D	;Store data byte at
					; destination
0A	23		INX	H	;Point HL to next
					; source location
0B	13		INX	D	;Point DE to
					; next destination
0C	05		DCR	B	;One transfer is
					; complete,
0D	C2		JNZ	NEXT	; decrement count
0E	08				;If counter is not 0,
0F	XX				; go back to transfer
10	76		HLT		; next byte
					;End of program
XX50	37				;Data
↓	↓				
XX5F	98				