

INSTRUCTION CLASSIFICATION

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform. The 8085 microprocessor includes the instruction set of its predecessor, the 8080A, plus two additional instructions.

5.21 The 8085 Instruction Set

The 8085 instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

DATA TRANSFER (COPY) OPERATIONS

This group of instructions copies data from a location called a source to another location, called a destination, without modifying the contents of the source. In technical manuals, the term *data transfer* is used for this copying function. However, the term *transfer* is misleading; it creates the impression that the contents of a source are destroyed when, in fact, the contents are retained without any modification. The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
<input type="checkbox"/> Between registers	Copy the contents of register B into register D.
<input type="checkbox"/> Specific data byte to a register or a memory location	Load register B with the data byte 32H.
<input type="checkbox"/> Between a memory location and a register	From the memory location 2000H to register B.
<input type="checkbox"/> Between an I/O device and the accumulator	From an input keyboard to the accumulator.

ARITHMETIC OPERATIONS

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

- ☐ **Addition**—Any 8-bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.
- ☐ **Subtraction**—Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's complement, and the results, if negative, are expressed in 2's complement. No two other registers can be subtracted directly.
- ☐ **Increment/Decrement**—The 8-bit contents of a register or a memory location can be incremented or decremented by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decremented by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

LOGICAL OPERATIONS

These instructions perform various logical operations with the contents of the accumulator.

- ☐ **AND, OR, Exclusive-OR**—Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, ORed, or Exclusive-ORed with the contents of the accumulator. The results are stored in the accumulator.

- **Rotate**—Each bit in the accumulator can be shifted either left or right to the next position.
- **Compare**—Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
- **Complement**—The contents of the accumulator can be complemented; all 0s are replaced by 1s and all 1s are replaced by 0s.

BRANCHING OPERATIONS

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

- **Jump**—Conditional jumps are an important aspect of the decision-making process in programming. These instructions test for a certain condition (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called *unconditional jump*.
- **Call, Return, and Restart**—These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

MACHINE CONTROL OPERATIONS

These instructions control machine functions such as Halt, Interrupt, or do nothing.

5.22 Review of the 8085 Operations

The microprocessor operations related to data manipulation can be summarized in four functions:

1. copying data
2. performing arithmetic operations
3. performing logical operations
4. testing for a given condition and altering the program sequence

Some important aspects of the instruction set are noted below:

1. In data transfer, the contents of the source are not destroyed; only the contents of the destination are changed. The data copy instructions do not affect the flags.
2. Arithmetic and logical operations are performed with the contents of the accumulator, and the results are stored in the accumulator (with some exceptions). The flags are affected according to the results.
3. Any register including memory can be used for increment and decrement.
4. A program sequence can be changed either conditionally or by testing for a given data condition.

Mnemonic → combin of letters to suggest the opⁿ of instrⁿ

INSTRUCTION AND DATA FORMAT

An **instruction** is a command to the microprocessor to perform a given task on specified data. Each instruction has two parts: one is the task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand**. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

5.31 Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the opcode and the operand in the same byte. For example:

Task	Opcode	Operand*	Binary Code	Hex Code
Copy the contents of the accumulator in register C.	MOV	C,A	0100 1111	<u>4FH</u>
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	<u>80H</u>
Invert (complement) each bit in the accumulator.	CMA		0010 1111	<u>2FH</u>

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction,

*In the operand, the destination register C is shown first, followed by the source register A.

the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

TWO-BYTE INSTRUCTIONS

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A,Data	0011 1110	3E	First Byte
			DATA	Data	Second Byte

Assume the data byte is 32H. The assembly language instruction is written as

Mnemonics	Hex Code
MVI A,32H	3E 32H

This instruction would require two memory locations to store in memory.

THREE-BYTE INSTRUCTIONS

In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011	C3*	First Byte
			1100 0011	85	Second Byte
			0010 0000	20	Third Byte

This instruction would require three memory locations to store in memory.

These commands are in many ways similar to our everyday conversation. For example, while eating in a restaurant, we may make the following requests and orders:

1. Pass (the) butter.
2. Pass (the) bowl.

HOW TO WRITE, ASSEMBLE, AND EXECUTE A SIMPLE PROGRAM

A program is a sequence of instructions written to tell a computer to perform a specific function. The instructions are selected from the instruction set of the microprocessor. To write a program, divide a given problem in small steps in terms of the operations the 8085 can perform, then translate these steps into instructions. Writing a simple program of adding two numbers in the 8085 language is illustrated below.

5.41 Illustrative Program: Adding Two Hexadecimal Numbers

PROBLEM STATEMENT

Write instructions to load the two hexadecimal numbers 32H and 48H in registers A and B, respectively. Add the numbers, and display the sum at the LED output port PORT1.

PROBLEM ANALYSIS

Even though this is a simple problem, it is necessary to divide the problem into small steps to examine the process of writing programs. The wording of the problem provides sufficient clues for the necessary steps. They are as follows:

1. Load the numbers in the registers.
2. Add the numbers.
3. Display the sum at the output port PORT1.

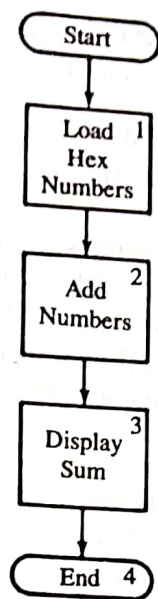
FLOWCHART

The steps listed in the problem analysis and the sequence can be represented in a block diagram, called a flowchart. Figure 5.2 shows such a flowchart representing the above steps. This is a simple flowchart, and the steps are self-explanatory. We will discuss flowcharting in the next chapter.

ASSEMBLY LANGUAGE PROGRAM

To write an assembly language program, we need to translate the blocks shown in the flowchart into 8085 operations and then, subsequently, into mnemonics. By examining the blocks, we can classify them into three types of operations: Blocks 1 and 3 are copy operations; Block 2 is an arithmetic operation; and Block 4 is a machine-control operation. To translate these steps into assembly and machine languages, you should review the instruction set. The translation of each block into mnemonics with comments is shown as follows:

FIGURE 5.2
Flowchart: Adding Two Numbers



44629

Block 1:	MVI A,32H	Load register A with 32H
	MVI B,48H	Load register B with 48H
Block 2:	ADD B	Add two bytes and save the sum in A
Block 3:	OUT 01H	Display accumulator contents at port 01H
Block 4:	HALT	End

FROM ASSEMBLY LANGUAGE TO HEX CODE

To convert the mnemonics into Hex code, we need to look up the code in the 8085 instruction set; this is called either manual or hand assembly.

Mnemonics	Hex Code	
MVI A,32H	3E	2-byte instruction
	32	
MVI B,48H	06	2-byte instruction
	48	
ADD B	80	1-byte instruction
OUT 01H	D3	2-byte instruction
	01	
HLT	76	1-byte instruction

STORING IN MEMORY AND CONVERTING FROM HEX CODE TO BINARY CODE

To store the program in R/W memory of a single-board microcomputer and display the output, we need to know the memory addresses and the output port address. Let us as-

sume that R/W memory ranges from 2000H to 20FFH, and the system has an LED output port with the address 01H. Now, to enter the program:

1. Reset the system by pushing the RESET key.
2. Enter the first memory address using Hex keys where the program should be stored. Let us assume it is 2000H.
3. Enter each machine code by pushing Hex keys. For example, to enter the first machine code, push the 3, E, and STORE keys. (The STORE key may be labeled differently in different systems.) When you push the STORE key, the program will store the machine code in memory location 2000H and upgrade the memory address to 2001H.
4. Repeat Step 3 until the last machine code, 76H.
5. Reset the system.

Now the question is: How does the Hex code get converted into binary code? The answer lies with the Monitor program stored in Read-Only memory (or EPROM) of the microcomputer system. An important function of the Monitor program is to check the keys and convert Hex code into binary code. The entire process of manual assembly is shown in Figure 5.3.

In this illustrative example, the program will be stored in memory as follows:

Mnemonics	Hex Code	Memory Contents	Memory Address
MVI A,32H	3E	0 0 1 1 1 1 1 0	2000
	32	0 0 1 1 0 0 1 0	2001
MVI B,48H	06	0 0 0 0 0 1 1 0	2002
	48	0 1 0 0 1 0 0 0	2003
ADD B	80	1 0 0 0 0 0 0 0	2004
OUT 01H	D3	1 1 0 1 0 0 1 1	2005
	01	0 0 0 0 0 0 0 1	2006
HLT	76	0 1 1 1 1 1 1 0	2007

This program has eight machine codes and will require eight memory locations to store the program. The critical concept that needs to be emphasized here is that the microprocessor can understand and execute only the binary instructions (or data); everything else (mnemonics, Hex code, comments) is for the convenience of human beings.

EXECUTING THE PROGRAM

To execute the program, we need to tell the microprocessor where the program begins by entering the memory address 2000H. Now, we can push the Execute key (or the key with a similar label) to begin the execution. As soon as the Execute function key is pushed, the microprocessor loads 2000H in the program counter, and the program control is transferred from the Monitor program to our program.

The microprocessor begins to read one machine code at a time, and when it fetches the complete instruction, it executes that instruction. For example, it will fetch the ma-

DATA TRANSFER (COPY) OPERATIONS

One of the primary functions of the microprocessor is copying data, from a register (or I/O or memory) called the source, to another register (or I/O or memory) called the destination. In technical literature, the copying function is frequently labeled as the **data transfer function**, which is somewhat misleading. In fact, the contents of the source are not transferred, but are copied into the destination register without modifying the contents of the source.

Several instructions are used to copy data (as listed in Chapter 5). This section is concerned with the following operations.

MOV : Move	Copy a data byte.
MVI : Move Immediate	Load a data byte directly.
OUT : Output to Port	Send a data byte to an output device.
IN : Input from Port	Read a data byte from an input device.

The term *copy* is equally valid for input/output functions because the contents of the source are not altered. However, the term *data transfer* is used so commonly to indicate the data copy function that, in this book, these terms are used interchangeably when the meaning is not ambiguous.

In addition to data copy instructions, it is necessary to introduce two machine-control operations to execute programs.

HLT: Halt	Stop processing and wait.
NOP: No Operation	Do not perform any operation.

These operations (opcodes) are explained and illustrated below with examples.

INSTRUCTIONS

The data transfer instructions copy data from a source into a destination without modifying the contents of the source. The previous contents of the destination are replaced by the contents of the source.



Important Note: In the 8085 processor, data transfer instructions do not affect the flags.

Opcode	Operand	Description
MOV	Rd, Rs*	<p>Move</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Copies data from source register Rs to destination register Rd
MVI	R, 8-bit	<p>Move Immediate</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Loads the 8 bits of the second byte into the register specified
OUT	8-bit port address	<p>Output to Port</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Sends (copies) the contents of the accumulator (A) to the output port specified in the second byte
IN	8-bit port address	<p>Input from Port</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Accepts (reads) data from the input port specified in the second byte, and loads into the accumulator
HLT		<p>Halt</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> The processor stops executing and enters wait state <input type="checkbox"/> The address bus and data bus are placed in high impedance state. No register contents are affected
NOP		<p>No Operation</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> No operation is performed <input type="checkbox"/> Generally used to increase processing time or substitute in place of an instruction. When an error occurs in a program and an instruction needs to be eliminated, it is more convenient to substitute NOP than to reassemble the whole program

*The symbols Rd and Rs are generic terms; they represent any of the 8085 registers: A, B, C, D, E, H, and L.

Load the accumulator A with the data byte 82H (the letter H indicates hexadecimal number), and save the data in register B.

Instructions MVI A, 82H,
 MOV B,A

The first instruction is a 2-byte instruction that loads the accumulator with the data byte 82H, and the second instruction MOV B,A copies the contents of the accumulator in register B without changing the contents of the accumulator.

output ports with addresses from 00H to FFH can be connected to the system.

The answer to the second question depends on the logic circuit (called interfacing) used to connect and identify a port by the system designer (see Chapter 4).

6.11 Addressing Modes *(Various ways of specifying the data)*

The above instructions are commands to the microprocessor to copy 8-bit data from a source into a destination. In these instructions, the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are, in fact, operands. The various formats of specifying the operands are called the addressing modes. The 8085 instruction set has the following addressing modes. (Each mode is followed by an example and by the corresponding piece of restaurant conversation from the analogy discussed in Chapter 5.)

1. Immediate Addressing—MVI R,Data (Pass the butter)
2. Register Addressing—MOV Rd,Rs (Pass the bowl)
3. Direct Addressing—IN/OUT Port# (Combination number 17 on the menu)
4. Indirect Addressing—Illustrated in the next chapter (I will have what Susie has)

MOV R, m	LDAX B, D	MVI M, 8bit	ADD m	IN RAM
MOV m, R	STAX B, D	CMP m, 8bit	SUB m	DCR m

6.12 Illustrative Program: Data Transfer—From Register to Output Port

PROBLEM STATEMENT

Load the hexadecimal number 37H in register B, and display the number at the output port labeled PORT1.

PROBLEM ANALYSIS

This problem is similar to the illustrative program discussed in Section 5.41. Even though this is a very simple problem it is necessary to break the problem into small steps and to outline the thinking process in terms of the tasks described in Section 6.1.

STEPS

Step 1: Load register B with a number.

Step 2: Send the number to the output port.

QUESTIONS TO BE ASKED

- ☐ Is there an instruction to load the register B? YES—MVI B.
- ☐ Is there an instruction to send the data from register B to the output port? NO. Review the instruction OUT. This instruction sends data from the accumulator to an output port.
- ☐ The solution appears to be as follows: Copy the number from register B into accumulator A.
- ☐ Is there an instruction to copy data from one register to another register? YES—MOV Rd,Rs.

program in the next section.

ASSEMBLY LANGUAGE PROGRAM

Tasks

1. Load register B with 37H.
2. Copy the number from B to A.
3. Send the number to the output—port 01H.
4. End of the program.

8085 Mnemonics

MVI B,37H*
MOV A,B
OUT PORT1
HLT

TRANSLATION FROM ASSEMBLY LANGUAGE TO MACHINE LANGUAGE

Now, to translate the assembly language program into machine language, look up the hexadecimal machine codes for each instruction in the 8085 instruction set and write each machine code in the sequence, as follows:

8085 Mnemonics	Hex Machine Code
1. MVI B,37H	06 37
2. MOV A,B	78
3. OUT PORT1	D3 01
4. HLT	76

This program has six machine codes and will require six bytes of memory to enter the program into your system. If your single-board microcomputer has R/W memory starting at the address 2000H, this program can be entered in the memory locations 2000H to 2005H. The format generally used to write an assembly language program is shown below.

PROGRAM FORMAT

Memory Address (Hex)	Machine Code (Hex)	Instruction		Comments
		Opcode	Operand	
XX00 [†]	06	MVI	B,37H	:Load register B with data 37H
XX01	37			

*A number followed by the letter H represents a hexadecimal number.

[†]Enter high-order address (page number) of your R/W memory in place of XX.

INTRODUCTION TO 8085 INSTRUCTIONS

XX02	78	MOV	A,B	;Copy (B) into (A)
XX03	D3	OUT	PORT1	;Display accumulator contents
XX04	PORT1*			; (37H) at Port1
XX05	76	HLT		;End of the program

This program has five columns: Memory Address, Machine Code, Opcode, Operand, and Comments. Each is described in the context of a single-board microcomputer.

ARITHMETIC INST:-

ADD : Add	Add the contents of a register.*
ADI : Add Immediate	Add 8-bit data.
SUB : Subtract	Subtract the contents of a register.
SUI : Subtract Immediate	Subtract 8-bit data.
INR : Increment	Increase the contents of a register by 1.
DCR : Decrement	Decrease the contents of a register by 1.

The arithmetic operations Add and Subtract are performed in relation to the contents of the accumulator. However, the Increment or the Decrement operations can be performed in any register. The instructions for these operations are explained below.

INSTRUCTIONS

These arithmetic instructions (except INR and DCR)

1. assume implicitly that the accumulator is one of the operands.
2. modify all the flags according to the data conditions of the result.
3. place the result in the accumulator.
4. do not affect the contents of the operand register.

The instructions INR and DCR

1. affect the contents of the specified register.
2. affect all flags except the CY flag.

The descriptions of the instructions (including INR and DCR) are as follows:

Opcode	Operand	Description
ADD	R [†]	Add <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Adds the contents of register R to the contents of the accumulator
ADI	8-bit	Add Immediate <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Adds the second byte to the contents of the accumulator
SUB	R [†]	Subtract <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Subtracts the contents of register R from the contents of the accumulator
SUI	8-bit	Subtract Immediate <input type="checkbox"/> This is a 2-byte instruction

*Memory-related arithmetic operations are excluded here; they are discussed in Chapter 7.

INR

R*

☐ Subtracts the second byte from the contents of the accumulator

Increment

☐ This is a 1-byte instruction

☐ Increases the contents of register R by 1

Caution: All flags except the CY are affected

DCR

R*

Decrement

☐ This is a 1-byte instruction

☐ Decreases the contents of register R by 1

Caution: All flags except the CY are affected

6.21 Addition

The 8085 performs addition with 8-bit binary numbers and stores the sum in the accumulator. If the sum is larger than eight bits (FFH), it sets the Carry flag. Addition can be performed either by adding the contents of a source register (B, C, D, E, H, L, or memory) to the contents of the accumulator (ADD) or by adding the second byte directly to the contents of the accumulator (ADI).

Add the number 35H directly to the sum in the previous example when the CY flag is set.

Instruction ADI 35H

$$\begin{array}{rcl} & & \text{CY} \\ (A) : & 4AH = & \boxed{1} 0 1 0 0 1 0 1 0 \\ & + & \\ (\text{Data}) : & 35H = & 0 0 1 1 0 1 0 1 \\ (A) : & 7FH = & \boxed{0} 0 1 1 1 1 1 1 1 \end{array}$$

Flag Status: S = 0, Z = 0, CY = 0

The addition of 4AH and 35H does not generate a carry and will reset the previous Carry flag. Therefore, in adding numbers, it is necessary to count how many times the CY flag is set by using some other programming techniques (see Section 7.32).

Assume the accumulator holds the data byte FFH. Illustrate the differences in the flags set by adding 01H and by incrementing the accumulator contents.

Instruction ADI 01H

$$\begin{array}{rcl} & & \text{CY} \\ (A) : & FFH = & 1 1 1 1 1 1 1 1 \\ & + & \\ (\text{Data}) : & 01H = & 0 0 0 0 0 0 0 1 \\ & & 1 1 1 1 1 1 1 1 \text{ Carry} \\ & & \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \\ (A) : & \boxed{1} 00H = & \boxed{1} 0 0 0 0 0 0 0 0 \\ & & \text{CY} \end{array}$$

Flag Status: S = 0, Z = 1, CY = 1

After adding 01H to FFH, the sum in the accumulator is 0 with a carry. Therefore, the CY and Z flags are set. The Sign flag is reset because D₇ is 0.

Instruction INR A

The accumulator contents will be 00H, the same as before. However, the instruction INR will not affect the Carry flag; it will remain in its previous status.

Flag Status: S = 0, Z = 1, CY = NA

FLAG CONCEPTS AND CAUTIONS

As described in the previous chapter, the flags are flip-flops that are set or reset after the execution of arithmetic and logic operations, with some exceptions. In many ways, the flags are like signs on an interstate highway that help drivers find their destinations.

6.22 Illustrative Program: Arithmetic Operations—Addition and Increment

PROBLEM STATEMENT

Write a program to perform the following functions, and verify the output.

1. Load the number 8BH in register D.
2. Load the number 6FH in register C.
3. Increment the contents of register C by one.
4. Add the contents of registers C and D and display the sum at the output PORT1.

PROGRAM

The illustrative program for arithmetic operations using addition and increment is presented as Figure 6.5 to show the register contents during some of the steps.

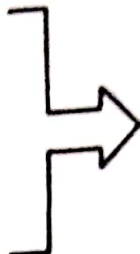
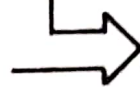
Memory Address (H)	Machine Code	Instruction Opcode	Operand	Comments and Register Contents																								
HI-LO XX00				The first four machine codes load the registers as																								
01	16	MVI	D,8BH	 <table><tr><td>A</td><td></td><td>S</td><td>Z</td><td>CY</td><td>F</td></tr><tr><td>B</td><td></td><td>X</td><td>X</td><td>X</td><td>C</td></tr><tr><td>D</td><td>8B</td><td></td><td></td><td></td><td>E</td></tr><tr><td>H</td><td></td><td></td><td></td><td></td><td>L</td></tr></table>	A		S	Z	CY	F	B		X	X	X	C	D	8B				E	H					L
A		S	Z		CY	F																						
B		X	X		X	C																						
D	8B					E																						
H					L																							
02	8B																											
03	0E	MVI	C,6FH																									
04	6F																											
05	0C	INR	C	Add 01 to (C): $6F + 01 = 70H$																								
06	79	MOV	A,C	 <table><tr><td>A</td><td>70</td><td>S</td><td>Z</td><td>CY</td><td>F</td></tr><tr><td>B</td><td></td><td>0</td><td>0</td><td>X</td><td>C</td></tr><tr><td>D</td><td>8B</td><td></td><td></td><td></td><td>E</td></tr></table>	A	70	S	Z	CY	F	B		0	0	X	C	D	8B				E						
A	70	S	Z		CY	F																						
B		0	0		X	C																						
D	8B				E																							
07	D3	OUT	PORT1																									
08	PORT #	PORT1																										
09	76	HLT		End of the program																								

FIGURE 6.5

Illustrative Program for Arithmetic Operations—Using Addition and Increment

4. Instruction ADD D adds (D) to (A), stores the sum in A, and sets the Sign flag as shown below:

$$\begin{array}{rcl}
 \text{(A)} : 70\text{H} & = & 01110000 \\
 + & & \\
 \text{(D)} : 8\text{BH} & = & 10001011 \\
 \hline
 \text{(A)} : \text{FBH} & = & \boxed{0}11111011 \text{ (see Figure 6.5)} \\
 & & \text{CY}
 \end{array}$$

Flag Status: S = 1, Z = 0, CY = 0

5. The sum FBH is displayed by the OUT instruction

6.24 Illustrative Program: Subtraction of Two Unsigned Numbers

PROBLEM STATEMENT

Write a program to do the following:

1. Load the number 30H in register B and 39H in register C.
2. Subtract 39H from 30H.
3. Display the answer at PORT1.

PROGRAM

The illustrative program for subtraction of two unsigned numbers is presented as Figure 6.6 to show the register contents during some of the steps.

PROGRAM DESCRIPTION

1. Registers B and C are loaded with 30H and 39H, respectively. The instruction MOV A,B copies 30H into the accumulator (shown as register contents). This is an essential step because the contents of a register can be subtracted only from the contents of the accumulator and not from any other register.
2. To execute the instruction SUB C the microprocessor performs the following steps internally:

Step 1:

$$\begin{array}{r} 39H = 00111001 \\ 1's \text{ complement of } 39H = 11000110 \\ + \end{array}$$

Step 2:

$$\begin{array}{r} \text{Add } 01 = 00000001 \\ 2's \text{ complement of } 39H = 11000111 \\ + \end{array}$$

Step 3:

$$\begin{array}{r} \text{Add } 30H \text{ to } 2's \text{ complement of } 39H = 00110000 \\ \text{CY } \boxed{0} \quad 11110111 \end{array}$$

Step 4: Complement carry

Flag Status: S = 1, Z = 0, CY = 1

$\boxed{11110111} = F7H$

- The number F7H is a 2's complement of the magnitude (39H - 30H) = 09H.
- The instruction OUT displays F7H at PORT1.

PROGRAM OUTPUT

This program will display F7H as the output. In this program, the unsigned numbers were used to perform the subtraction. Now, the question is: How do you recognize that the answer F7H is really a 2's complement of 09H and not a straight binary F7H?

The answer lies with the Carry flag. If the Carry flag (also known as the Borrow flag in subtraction) is set, the answer is in 2's complement. The Carry flag raises a second question: Why isn't it a positive sum with a carry? The answer is implied by the instruction SUB (it is a subtraction).

There is no way to differentiate between a straight binary number and 2's complement by examining the answer at the output port. The flags are internal and not easily displayed. However, a programmer can test the Carry flag by using the instruction Jump On Carry (JC) and can find a way to indicate that the answer is in 2's complement. (This is discussed in Branch instructions.)

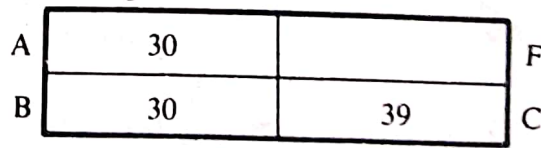
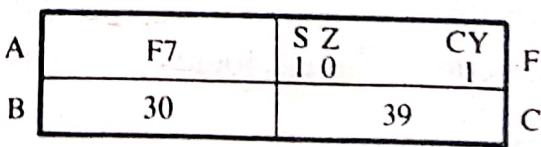
Memory Address (H)	Machine Code	Instruction Opcode	Operand	Comments and Register Contents
XX00	06	MVI	B,30H	Load the minuend in register B Load the subtrahend in register C The register contents: 
01	30			
02	0E	MVI	C,39H	
03	39			
04	78	MOV	A,B	
05	91	SUB	C	
06	D3	OUT	PORT1	
07	PORT#			
08	76	HLT		

FIGURE 6.6

Illustrative Program for Subtraction of Two Unsigned Numbers

6.25 Review of Important Concepts

LOGIC OPERATIONS

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hard-wired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, Ex OR, and NOT (complement). The opcodes of these operations are as follows:*

ANA:	AND	Logically AND the contents of a register.
ANI :	AND Immediate	Logically AND 8-bit data.
ORA:	OR	Logically OR the contents of a register.
ORI :	OR Immediate	Logically OR 8-bit data.
XRA:	X-OR	Exclusive-OR the contents of a register.
XRI :	X-OR Immediate	Exclusive-OR 8-bit data.

All logic operations are performed in relation to the contents of the accumulator. The instructions of these logic operations are described below.

INSTRUCTIONS

The logic instructions

1. implicitly assume that the accumulator is one of the operands.
2. reset (clear) the CY flag. The instruction CMA is an exception; it does not affect any flags.
3. modify the Z, P, and S flags according to the data conditions of the result.
4. place the result in the accumulator.
5. do not affect the contents of the operand register.

INTRODUCTION TO 8085 INSTRUCTIONS

Opcode	Operand	Description
ANA	R	<p>Logical AND with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Logically ANDs the contents of the register R with the contents of the accumulator <input type="checkbox"/> 8085: CY is reset and AC is set
ANI	8-bit	<p>AND Immediate with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Logically ANDs the second byte with the contents of the accumulator <input type="checkbox"/> 8085: CY is reset and AC is set
ORA	R	<p>Logically OR with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Logically ORs the contents of the register R with the contents of the accumulator
ORI	8-bit	<p>OR Immediate with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Logically ORs the second byte with the contents of the accumulator
XRA	R	<p>Logically Exclusive-OR with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction <input type="checkbox"/> Exclusive-ORs the contents of register R with the contents of the accumulator
XRI	8-bit	<p>Exclusive-OR Immediate with Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 2-byte instruction <input type="checkbox"/> Exclusive-ORs the second byte with the contents of the accumulator
CMA		<p>Complement Accumulator</p> <ul style="list-style-type: none"> <input type="checkbox"/> This is a 1-byte instruction that complements the contents of the accumulator <input type="checkbox"/> No flags are affected