Unit 5

Introduction to Microprocessors

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.

2.3 INSTRUCTION, DATA FORMAT, AND STORAGE

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.

2.31 Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size or byte size.

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.

1. 1-byte instructions
2. 2-byte instructions
3. 3-byte instructions

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	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (complement) each bit in the accumulator.	CMA		0010 1111	2FH

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, 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.

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

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,32H	0011 1110	3E	First Byte
			0011 0010	32	Second Byte
Load an 8-bit data byte in register B.	MVI	B,F2H	0000 0110	06	First Byte
			1111 0010	F2	Second Byte

These instructions would require two memory locations each to store the binary codes. The data bytes 32H and F2H are selected arbitrarily as examples.

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*	
Load contents of memory 2050H into A.	LDA	2050H	0011 1010	3A	First Byte
			0101 0000	50	Second Byte
			0010 0000	20	Third Byte
Transfer the program sequence to memory location 2085H.	JMP	2085H	1100 0011	C3	First Byte
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

These instructions would require three memory locations each to store the binary codes.

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.
3. (Let us) eat.

---

*The 16-bit addresses are stored in memory locations in reversed order, the low-order byte first, followed by the high-order byte.

4. I will have combination 17 (on the menu).
5. I will have what Susie ordered.

The first request specifies the exact item; it is similar to the instruction for loading a specific data byte in a register. The second request mentions the bowl rather than the contents, even though one is interested in the contents of the bowl. It is similar to the instruction MOV C,A where registers (bowls) are specified rather than data. The third suggestion (let us eat) assumes that one knows what to eat. It is similar to the instruction Complement, which implicitly assumes that the operand is the accumulator. In the fourth sentence, the location of the item on the menu is specified and not the actual item. It is similar to the instruction: Transfer the data byte from the location 2050H. The last order (what Susie ordered) is specified indirectly. It is similar to an instruction that specifies a memory location through the contents of a register pair. (Examples of the last two types of instruction are illustrated in later chapters.)

These various ways of specifying data are called the **addressing modes.** Although microprocessor instructions require one or more words to specify the operands, the notations and conventions used in specifying the operands have very little to do with the operation of the microprocessor. The mnemonic letters used to specify a command are chosen (somewhat arbitrarily) by the manufacturer. When an instruction is stored in memory, it is stored in binary code, the only code the microprocessor is capable of reading and understanding. The conventions used in specifying the instructions are valuable in terms of keeping uniformity in different programs and in writing assemblers. The important point to remember is that the microprocessor neither reads nor understands mnemonics or hexadecimal numbers.

2.3.2 Opcode Format

To understand operation codes, we need to examine how an instruction is designed into the microprocessor. This information will be useful in reading a user's manual, in which operation codes are specified in binary format and 8-bits are divided in various groups. However, this information is not necessary to understand assembly language programming.

In the design of the 8085 microprocessor chip, all operations, registers, and status flags are identified with a specific code. For example, all internal registers are identified as follows:

Code	Registers
000	B
001	C
010	D
011	E
100	H
101	L
111	A
110	Reserved for Memory-Related operation

Code	Register Pairs
00	BC
01	DE
10	HL
11	AF OR SP

Some of the operation codes are identified as follows:

Function	Operation Code
1. Rotate each bit of the accumulator to the left by one position.	00000111 = 07H (8-bit opcode)
2. Add the contents of a register to the accumulator.	10000 SSS (5-bit opcode—3 bits are reserved for a register)

This instruction is completed by adding the code of the register. For example,

Add	:	10000
Register B	:	000
to A	:	Implicit
Binary Instruction:		10000 000 = 80H
		Add Reg.B

In assembly language, this is expressed as

Opcode	Operand	Hex Code
ADD	B	80H

3. MOVE (Copy) the content of register Rs (source) to register Rd (destination)	01 2-bit Opcode for MOVE	DDD Reg. Rd	SSS Reg. Rs

This instruction is completed by adding the codes of two registers. For example,

Move (copy) the content:		0 1
To register C	:	0 0 1 (DDD)
From register A	:	1 1 1 (SSS)
Binary Instruction	:	0 1 0 0 1 1 1 → 4FH
		Opcode Operand

In assembly language, this is expressed as

Opcode	Operand	Hex Code
MOV	C,A	4F

Please note that the first register is the destination and the second register is the source—from A to C—which appears reversed for a general pattern from left to right. Typically, in the 8085 user's manual the data transfer (copy) instruction is shown as follows:

MOV rl, r2*

0	1	D	D	D	S	S	S

2.3.3 Data Format

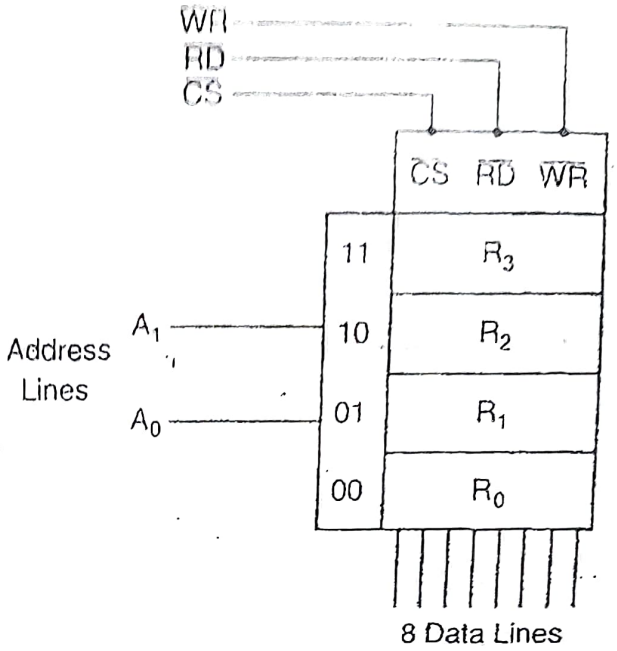
The 8085 is an 8-bit microprocessor, and it processes (copy, add, subtract, etc.) only binary numbers. However, the real world operates in decimal numbers and languages of alphabets and characters. Therefore, we need to code binary numbers into different media. Let us examine coding. What is the letter "A"? It is a symbol representing a certain sound in a visual medium that eyes can recognize. Similarly, we can represent or code groups of bits into different media. In 8-bit processor systems, commonly used codes and data formats are ASCII, BCD, signed integers, and unsigned integers. They are explained as follows.

- **ASCII Code**—This is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and nonprintable characters such as carriage return. Extended ASCII is an 8-bit code. The additional numbers (beyond 7-bit ASCII code) represent graphical characters. This code was discussed in Chapter 1 (Section 1.24).
- **BCD Code**—The term *BCD* stands for binary-coded decimal; it is used for decimal numbers. The decimal numbering system has ten digits, 0 to 9. Therefore, we need only four bits to represent ten digits from 0000 to 1001. The remaining numbers, 1010 (A) to 1111 (F), are considered invalid. An 8-bit register in the 8085 can accommodate two BCD numbers.
- **Signed Integer**—A signed integer is either a positive number or a negative number. In an 8-bit processor, the most significant digit, D_7, is used for the sign; 0 represents the positive sign and 1 represents the negative sign. The remaining seven bits, D_6–D_0, represent the magnitude of an integer. Therefore, the largest positive integer that can be processed by the 8085 at one time is 0111 1111 (7FH); the remaining Hex numbers, 80H to FFH, are considered negative numbers. However, all negative numbers in this microprocessor are represented in 2's complement format (see Appendix A.2 for additional explanation).
- **Unsigned Integers**—An integer without a sign can be represented by all the 8 bits in a microprocessor register. Therefore, the largest number that can be processed at one time is FFH. However, this does not imply that the 8085 microprocessor is limited to handling only 8-bit numbers. Numbers larger than 8 bits (such as 16-bit or 24-bit numbers) are processed by dividing them in groups of 8 bits.

Now let us examine how the microprocessor interprets any number. Let us assume that after performing some operations the result in the accumulator is 0100 0001 (41H). This number can have many interpretations: (1) It is an unsigned number equivalent to 65

*In this text, rl is specified as Rd and r2 is specified as Rs to indicate destination and source.

FIGURE 2.2
Simplified Memory Model



in decimal; (2) it is a BCD number representing 41 decimal; (3) it is the ASCII capital-letter "A"; or (4) it is a group of 8 bits where bits D_6 and D_0 turn on and the remaining bits turn off output devices. The processor processes binary bits; it is up to the user to interpret the result. In our example, the number 41H can be displayed on a screen as an ASCII "A" or 41 BCD.

2.3.4 Instruction and Data Storage: Memory

Now the next question is: How do we provide this information to the processor? It is provided by another electronic storage chip called memory. In some ways, the term *memory* is a misnomer; it is a storage of binary bits. Memory chips used in most systems are nothing but 8-bit registers stacked one above the other as shown in our memory model in Figure 2.2. It includes only four registers, and each register can store 8 bits. This chip can be referred to as a 4-byte or 32 (4×8) bits memory chip. It has two address lines, A_0 and A_1, to identify four registers, 8 data lines to store 8 bits, and three timing or control signals: Read ($\overline{RD}$), Write ($\overline{WR}$), and Chip Select ($\overline{CS}$); all control signals are designed to be active low, indicated by bars over the symbols. The processor can select this chip and identify its register, and store (Write) or access (Read) 8 bits at a time. Figure 2.2 shows only four registers to simplify the explanation; in reality, the size of a memory chip is in kilo- or megabytes. The memory addresses assigned to these registers are determined by the interfacing logic used in the system. In Chapters 3 and 4, we will discuss memory addressing and interfacing in more detail.

2.4 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.

2.4.1 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 2.3 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 op-

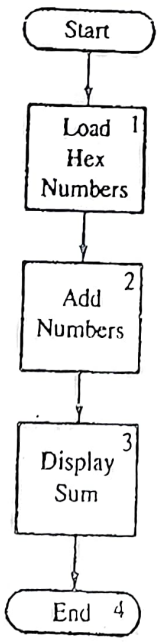


FIGURE 2.3
Flowchart: Adding Two Numbers

erations; 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:

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 assume 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 micro-

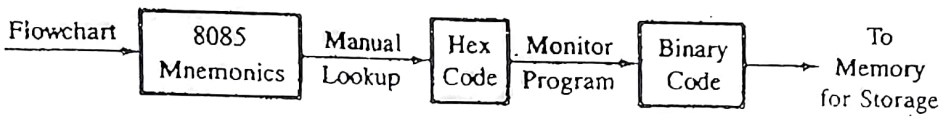


FIGURE 2.4
Manual Assembly Process

computer 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 2.4.

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 machine codes stored in memory locations 2000H and 2001H and execute the instruction MVI A,32H; thus it will load 32H in register A. The ADD instruction will add the two numbers, and the OUT instruction will display the answer 7AH (32H + 48H = 7AH) at the LED port. It continues to execute instructions until it fetches the HLT instruction.

RECOGNIZING THE NUMBER OF BYTES IN AN INSTRUCTION

Students who are introduced to an assembly language for the first time should hand assemble at least a few small programs. Such exercises can clarify the relationship among instruction codes, data, memory registers, and memory addressing. One of the stumbling blocks in hand

assembly is in recognizing the number of bytes in a given instruction. The following clues can be used to recognize the number of bytes in an instruction of the 8085 microprocessor.

1. One-byte instruction—A mnemonic followed by a letter (or two letters) representing the registers (such as A, B, C, D, E, H, L, M, and SP) is a one-byte instruction. Instructions in which registers are implicit are also one-byte instructions.
   Examples: (a) MOV A, B; (b) DCX SP; (c) RRC
2. Two-byte instruction—A mnemonic followed by 8-bit (byte) is a two-byte instruction.
   Examples: (a) MVI A, 8-bit; (b) ADI 8-bit
3. Three-byte instruction—A mnemonic followed by 16-bit (also terms such as adr or dble) is a three-byte instruction.
   Examples: (a) LXI B, 16-bit (dble); (b) JNZ 16-bit (adr); (c) CALL 16-bit (adr)

In writing assembly language programs, we can assign memory addresses in a sequence once we know the number of bytes in a given instruction. For example, a three-byte instruction has three Hex codes and requires three memory locations in a sequence. In hand assembly, omitting a byte inadvertently can have a disastrous effect on program execution, as explained in the next section.

2.4.2 How Does a Microprocessor Differentiate Between Data and Instruction Code?

The microprocessor is a sequential machine. As soon as a microprocessor-based system is turned on, it begins the execution of the code in memory. The execution continues in a sequence, one code after another (one memory location after another) at the speed of its clock until the system is turned off (or the clock stops). If an unconditional loop is set up in a program, the execution will continue until the system is either reset or turned off.

Now a puzzling question is: How does the microprocessor differentiate between a code and data when both are binary numbers? The answer lies in the fact that the microprocessor interprets the first byte it fetches as an opcode. When the 8085 is reset, its program counter is cleared to 0000H and it fetches the first code from the location 0000H. In the example of the previous section, we tell the processor that our program begins at location 2000H. The first code it fetches is 3EH. When it decodes that code, it knows that it is a two-byte instruction. Therefore, it assumes that the second code, 32H, is a data byte. If we forget to enter 32H and enter the next code, 06H, instead, the 8085 will load 06H in the accumulator, interpret the next code, 48H, as an opcode, and continue the execution in sequence. As a consequence, we may encounter a totally unexpected result.

OVERVIEW OF THE 8085 INSTRUCTION SET

The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. The set includes all the 8080A instructions plus two additional instructions (SIM and RIM, related to serial I/O). It is an overwhelming experience for a beginner to

study these instructions. You are strongly advised not to attempt to read all these instructions at one time. However, you should be able to grasp an overview of the set by examining the frequently used instructions listed below.

The following notations are used in the description of the instructions.

R = 8085 8-bit register	(A, B, C, D, E, H, L)
M = Memory register (location)	
Rs = Register source	
Rd = Register destination	(A, B, C, D, E, H, L)
Rp = Register pair	(BC, DE, HL, SP)
( ) = Contents of	

1. Data Transfer (Copy) Instructions. These instructions perform the following six operations.

- Load an 8-bit number in a register
- Copy from register to register
- Copy between I/O and accumulator
- Load 16-bit number in a register pair
- Copy between register and memory
- Copy between registers and stack memory

	Mnemonics	Examples	Operation
1.1	MVI R,** 8-bit	MVI B, 4FH	Load 8-bit data (byte) in a register
1.2	MOV Rd, Rs**	MOV B, A MOV C, B	Copy data from source register Rs into destination register Rd
1.3	LXI Rp,** 16-bit	LXI B, 2050H	Load 16-bit number in a register pair
1.4	OUT 8-bit (port address)	OUT 01H	Send (write) data byte from the accumulator to an output device
1.5	IN 8-bit (port address)	IN 07H	Accept (read) data byte from an input device and place it in the accumulator
1.6	LDA 16-bit	LDA 2050H	Copy the data byte into A from the memory specified by 16-bit address
1.7	STA 16-bit	STA 2070H	Copy the data byte from A into the memory specified by 16-bit address
1.8	LDAX Rp	LDAX B	Copy the data byte into A from the memory specified by the address in the register pair
1.9	STAX Rp	STAX D	Copy the data byte from A into the memory specified by the address in the register pair

*These instructions are explained and illustrated in Chapters 6 and 7. The complete instruction set is explained alphabetically in Appendix F for easy reference; the appendix also includes three lists of instruction summaries arranged according to the functions, hexadecimal sequence of machine codes, and alphabetical order.

**The letters R, Rd, Rs, Rp represent generic registers. In the 8085 instructions, these are replaced by registers such as A, B, C, D, E, H, and L or register pairs.

Address of M is in H-L pair.

1.10	MOV R, M	MOV B, M	Copy the data byte into register from the memory specified by the address in HL register
1.11	MOV M, R	MOV M, C	Copy the data byte from the register into memory specified by the address in HL register

2. Arithmetic Instructions. The frequently used arithmetic operations are:

- Add
- Subtract
- Increment (Add 1)
- Decrement (Subtract 1)

Mnemonics		Examples	Operation
2.1	ADD R	ADD B	Add the contents of a register to the register to the contents of A
2.2	ADI 8-bit	ADI 37H	Add 8-bit data to the contents of A
2.3	ADD M	ADD M	Add the contents of memory to A; the address of memory is in HL register
2.4	SUB R	SUB C	Subtract the contents of a register from the contents of A
2.5	SUI 8-bit	SUI 7FH	Subtract 8-bit data from the contents of A
2.6	SUB M	SUB M	Subtract the contents of memory from A; the address of memory is in HL register
2.7	INR R	INR D	Increment the contents of a register
2.8	INR M	INR M	Increment the contents of memory, the address of which is in HL
2.9	DCR R	DCR E	Decrement the contents of a register
2.10	DCR M	DCR M	Decrement the contents of a memory, the address of which is in HL
2.11	INX Rp	INX H	Increment the contents of a register pair
2.12	DCX Rp	DCX B	Decrement the contents of a register pair

3. Logic and Bit Manipulation Instructions. These instructions include the following operations:

- AND
- OR
- X-OR (Exclusive OR)
- Compare
- Rotate Bits

Mnemonics		Examples	Operation
3.1	ANA R	ANA B	Logically AND the contents of a register with the contents of A

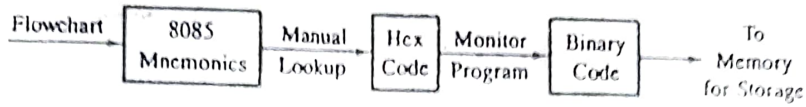


FIGURE 2.4
Manual Assembly Process

computer 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 2.4.

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 0 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 machine codes stored in memory locations 2000H and 2001H and execute the instruction MVI A,32H; thus it will load 32H in register A. The ADD instruction will add the two numbers, and the OUT instruction will display the answer 7AH (32H + 48H = 7AH) at the LED port. It continues to execute instructions until it fetches the HLT instruction.

RECOGNIZING THE NUMBER OF BYTES IN AN INSTRUCTION

Students who are introduced to an assembly language for the first time should hand assemble at least a few small programs. Such exercises can clarify the relationship among instruction codes, data, memory registers, and memory addressing. One of the stumbling blocks in hand

assembly is in recognizing the number of bytes in a given instruction. The following clues can be used to recognize the number of bytes in an instruction of the 8085 microprocessor.

1. One-byte instruction—A mnemonic followed by a letter (or two letters) representing the registers (such as A, B, C, D, E, H, L, M, and SP) is a one-byte instruction. Instructions in which registers are implicit are also one-byte instructions.
   Examples: (a) MOV A, B; (b) DCX SP; (c) RRC
2. Two-byte instruction—A mnemonic followed by 8-bit (byte) is a two-byte instruction.
   Examples: (a) MVI A, 8-bit; (b) ADI 8-bit
3. Three-byte instruction—A mnemonic followed by 16-bit (also terms such as adr or dble) is a three-byte instruction.
   Examples: (a) LXI B, 16-bit (dble); (b) JNZ 16-bit (adr); (c) CALL 16-bit (adr)

In writing assembly language programs, we can assign memory addresses in a sequence once we know the number of bytes in a given instruction. For example, a three-byte instruction has three Hex codes and requires three memory locations in a sequence. In hand assembly, omitting a byte inadvertently can have a disastrous effect on program execution, as explained in the next section.

2.4.2 How Does a Microprocessor Differentiate Between Data and Instruction Code?

The microprocessor is a sequential machine. As soon as a microprocessor-based system is turned on, it begins the execution of the code in memory. The execution continues in a sequence, one code after another (one memory location after another) at the speed of its clock until the system is turned off (or the clock stops). If an unconditional loop is set up in a program, the execution will continue until the system is either reset or turned off.

Now a puzzling question is: How does the microprocessor differentiate between a code and data when both are binary numbers? The answer lies in the fact that the microprocessor interprets the first byte it fetches as an opcode. When the 8085 is reset, its program counter is cleared to 0000H and it fetches the first code from the location 0000H. In the example of the previous section, we tell the processor that our program begins at location 2000H. The first code it fetches is 3EH. When it decodes that code, it knows that it is a two-byte instruction. Therefore, it assumes that the second code, 32H, is a data byte. If we forget to enter 32H and enter the next code, 06H, instead, the 8085 will load 06H in the accumulator, interpret the next code, 48H, as an opcode, and continue the execution in sequence. As a consequence, we may encounter a totally unexpected result.

OVERVIEW OF THE 8085 INSTRUCTION SET

The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. The set includes all the 8080A instructions plus two additional instructions (SIM and RIM, related to serial I/O). It is an overwhelming experience for a beginner to

3.2	ANI 8-bit	ANI 2FH	Logically AND 8-bit data with the contents of A
3.3	ANA M	ANA M	Logically AND the contents of memory with the contents of A; the address of memory is in HL register
3.4	ORA R	ORA E	Logically OR the contents of a register with the contents of A
3.5	ORI 8-bit	ORI 3FH	Logically OR 8-bit data with the contents of A
3.6	ORA M	ORA M	Logically OR the contents of memory with the contents of A; the address of memory is in HL register
3.7	XRA R	XRA B	Exclusive-OR the contents of a register with the contents of A
3.8	XRI 8-bit	XRI 6AH	Exclusive-OR 8-bit data with the contents of A
3.9	XRA M	XRA M	Exclusive-OR the contents of memory with the contents of A; the address of memory is in HL register
3.10	CMP R	CMP B	Compare the contents of register with the contents of A for less than, equal to, or greater than
3.11	CPI 8-bit	CPI 4FH	Compare 8-bit data with the contents of A for less than, equal to, or greater than

4. Branch Instructions. The following instructions change the program sequence.

4.1	JMP 16-bit address	JMP 2050H	Change the program sequence to the specified 16-bit address
4.2	JZ 16-bit address	JZ 2080H	Change the program sequence to the specified 16-bit address if the Zero flag is set
4.3	JNZ 16-bit address	JNZ 2070H	Change the program sequence to the specified 16-bit address if the Zero flag is reset
4.4	JC 16-bit address	JC 2025H	Change the program sequence to the specified 16-bit address if the Carry flag is set
4.5	JNC 16-bit address	JNC 2030H	Change the program sequence to the specified 16-bit address if the Carry flag is reset
4.6	CALL 16-bit address	CALL 2075H	Change the program sequence to the location of a subroutine
4.7	RET	RET	Return to the calling program after completing the subroutine sequence

5. **Machine Control Instructions.** These instructions affect the operation of the processor.

5.1	HLT	HLT	Stop processing and wait
5.2	NOP	NOP	Do not perform any operation

This set of instructions is a representative sample; it does not include various instructions related to 16-bit data operations, additional jump instructions, and conditional Call and Return instructions.

WRITING AND HAND ASSEMBLING A PROGRAM

In previous sections, we discussed the 8085 instructions, recognized the number of bytes per instruction, looked at the relationship between the number of bytes of an instruction and memory registers needed for storage, and examined the processor's computing capability in the overview of the instruction set. Now let us pull together all these concepts in a simple illustrative program.

2.6.1 Illustrative Program: Subtracting Two Hexadecimal Numbers and Storing the Result in Memory

PROBLEM STATEMENT

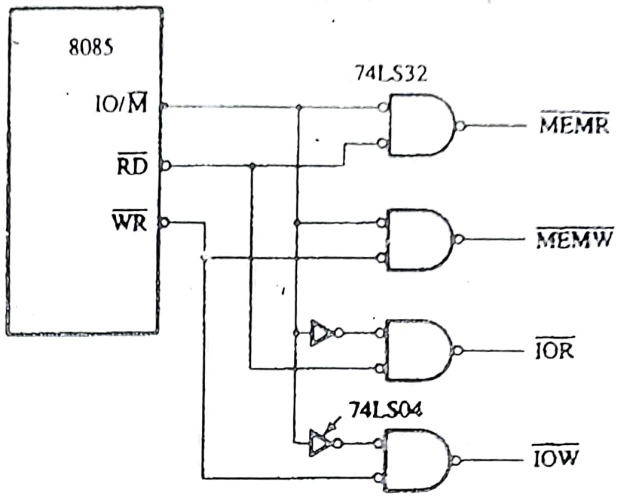
Write instructions to subtract two bytes already stored in memory registers (also referred to as memory locations or memory addresses) 2051H and 2052H. Location 2051H holds the byte 49H and location 2051H holds the byte 9FH. Subtract the first byte, 49H, from the second byte, 9FH, and store the answer in memory location 2053H. Write instructions beginning at memory location 2030H.

PROBLEM ANALYSIS

This is a problem similar to the problem in Section 2.4. However, we need to note some specific points in this problem.

1. The data bytes to be subtracted are already stored in memory registers 2051H and 2052H. We do not need to write instructions to store these bytes. You should store these bytes by using a keyboard of your trainer, or if you are using a simulator, you should observe the numbers in those memory locations when you store them.
2. The program should be written starting at memory location 2030H. This memory location is selected arbitrarily to emphasize that you can write a program beginning at any available memory location.

FIGURE 4.5
Schematic to Generate Read/Write Control Signals for Memory and I/O



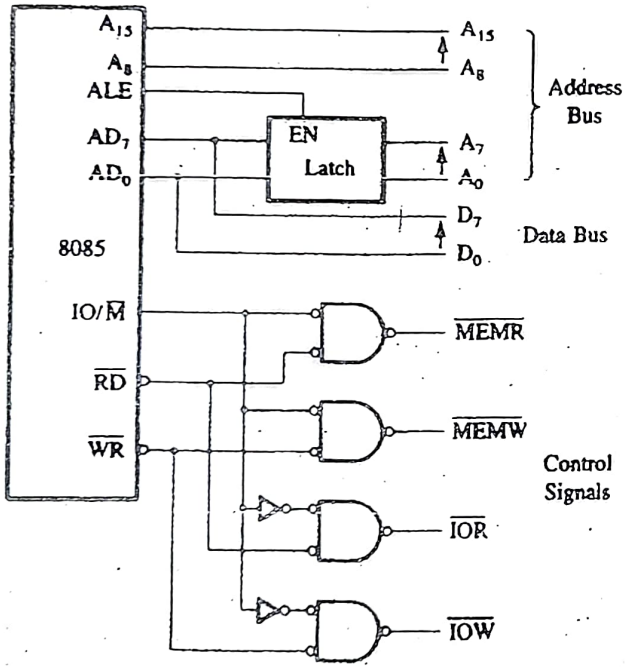


FIGURE 4.6
8085 Demultiplexed Address and Data Bus with Control Signals

To demultiplex the bus and to generate the necessary control signals, the 8085 microprocessor requires a latch and logic gates to build the MPU, as shown in Figure 4.6. This MPU can be interfaced with any memory or I/O.

4.1.5 A Detailed Look at the 8085 MPU and Its Architecture

Figure 4.7 shows the internal architecture of the 8085 beyond the programmable registers we discussed previously. It includes the ALU (Arithmetic/Logic Unit), Timing and

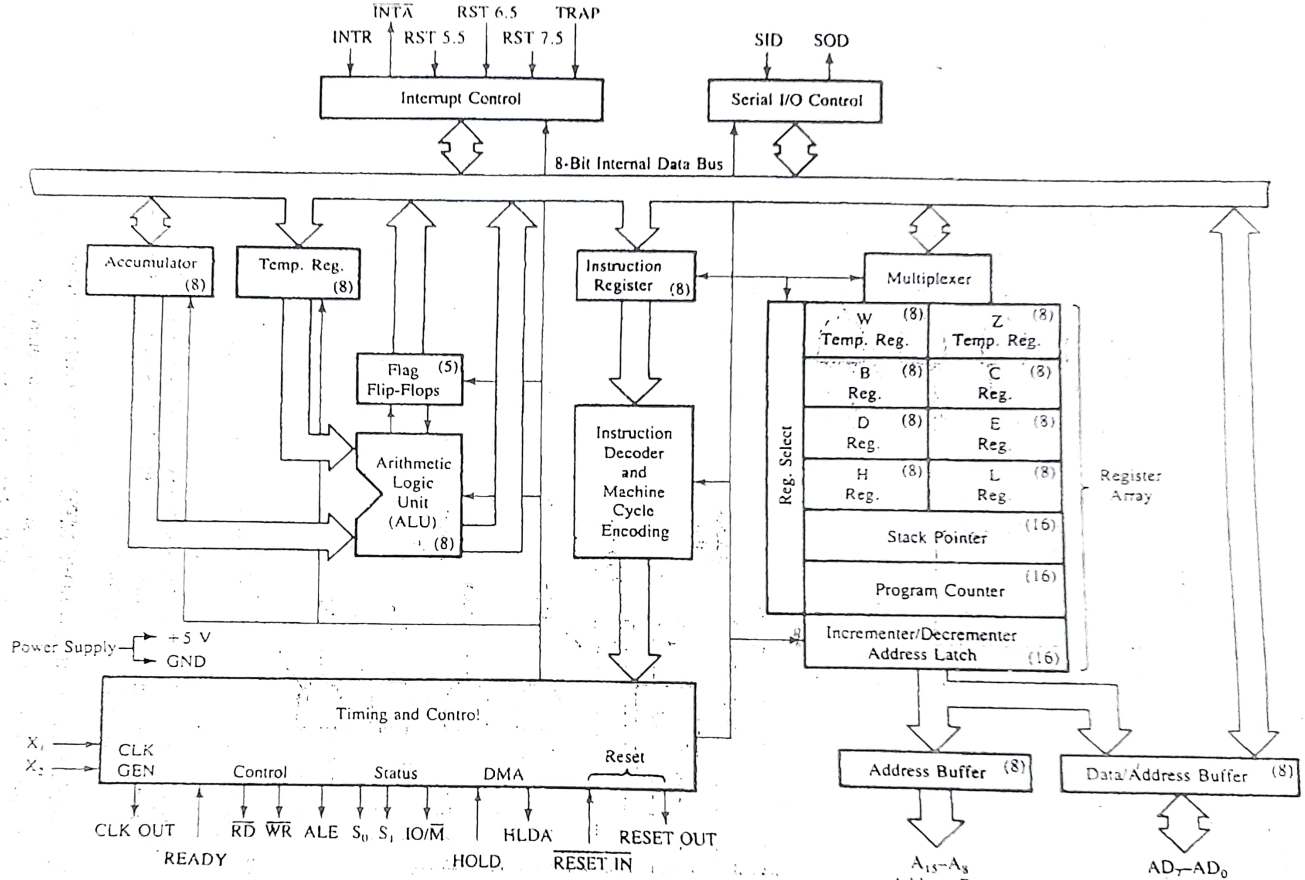


FIGURE 4.7
The 8085A Microprocessor: Functional Block Diagram

NOTE: The 8085A microprocessor is commonly known as the 8085.

SOURCE: Intel Corporation, *Embedded Microprocessors* (Santa Clara, Calif.: Author, 1994), pp. 1-11.

Control Unit, Instruction Register and Decoder, Register Array, Interrupt Control, and Serial I/O Control. We will discuss the first four units below; the last two will be discussed later in the book.

THE ALU

The arithmetic/logic unit performs the computing functions; it includes the accumulator, the temporary register, the arithmetic and logic circuits, and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator, and the flags (flip-flops) are set or reset according to the result of the operation.

The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator—with some exceptions. The descriptions and conditions of the flags are as follows:

- **S—Sign flag:** After the execution of an arithmetic or logic operation, if bit D_7 of the result (usually in the accumulator) is 1, the Sign flag is set. This flag is used with signed numbers. In a given byte, if D_7 is 1, the number will be viewed as a negative number; if it is 0, the number will be considered positive. In arithmetic operations with signed numbers, bit D_7 is reserved for indicating the sign, and the remaining seven bits are used to represent the magnitude of a number. However, this flag is irrelevant for the operations of unsigned numbers. Therefore, for unsigned numbers, even if bit D_7 of a result is 1 and the flag is set, it does not mean the result is negative. (See Appendix A2 for a discussion of signed numbers.)
- **Z—Zero flag:** The Zero flag is set if the ALU operation results in 0, and the flag is reset if the result is not 0. This flag is modified by the results in the accumulator as well as in the other registers.
- **AC—Auxiliary Carry flag:** In an arithmetic operation, when a carry is generated by digit D_3 and passed on to digit D_4, the AC flag is set. The flag is used only internally for BCD (binary-coded decimal) operations and is not available for the programmer to change the sequence of a program with a jump instruction.
- **P—Parity flag:** After an arithmetic or logical operation, if the result has an even number of 1s, the flag is set. If it has an odd number of 1s, the flag is reset. (For example, the data byte 0000 0011 has even parity even if the magnitude of the number is odd.)
- **CY—Carry flag:** If an arithmetic operation results in a carry, the Carry flag is set; otherwise it is reset. The Carry flag also serves as a borrow flag for subtraction.

The bit positions reserved for these flags in the flag register are as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Among the five flags, the AC flag is used internally for BCD arithmetic, the instruction set does not include any conditional jump instructions based on the AC flag. Of the remaining four flags, the Z and CY flags are those most commonly used.

TIMING AND CONTROL UNIT

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals are similar to a sync pulse in an oscilloscope. The $\overline{RD}$ and $\overline{WR}$ signals are sync pulses indicating the availability of data on the data bus.

INSTRUCTION REGISTER AND DECODER

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instruction.

REGISTER ARRAY

The programmable registers were discussed in the last chapter. Two additional registers, called temporary registers W and Z, are included in the register array. These registers are used to hold 8-bit data during the execution of some instructions. However, because they are used internally, they are not available to the programmer.

4.1.6 Decoding and Executing an Instruction

Decoding and executing an instruction after it has been fetched can be illustrated with the example from Section 4.12.

---

ple

Assume that the accumulator contains data byte 82H, and the instruction MOV C,A (4FH) is fetched. List the steps in decoding and executing the instruction.

on

This example is similar to the example in Section 4.12, except that the contents of the accumulator are specified. To decode and execute the instruction, the following steps are performed.

The microprocessor:

1. Places the contents of the data bus (4FH) in the instruction register (Figure 4.8) and decodes the instruction.
2. Transfers the contents of the accumulator (82H) to the temporary register in the ALU.
3. Transfers the contents of the temporary register to register C.

4.1.7 Review of Important Concepts

1. The 8085 microprocessor has a multiplexed bus AD_7–AD_0 used as the lower-order address bus and the data bus.
2. The bus AD_7–AD_0 can be demultiplexed by using a latch and the ALE signal.
3. The 8085 has a status signal IO/$\overline{M}$ and two control signals $\overline{RD}$ and $\overline{WR}$. By ANDing these signals, four control signals can be generated: $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, and $\overline{IOW}$.

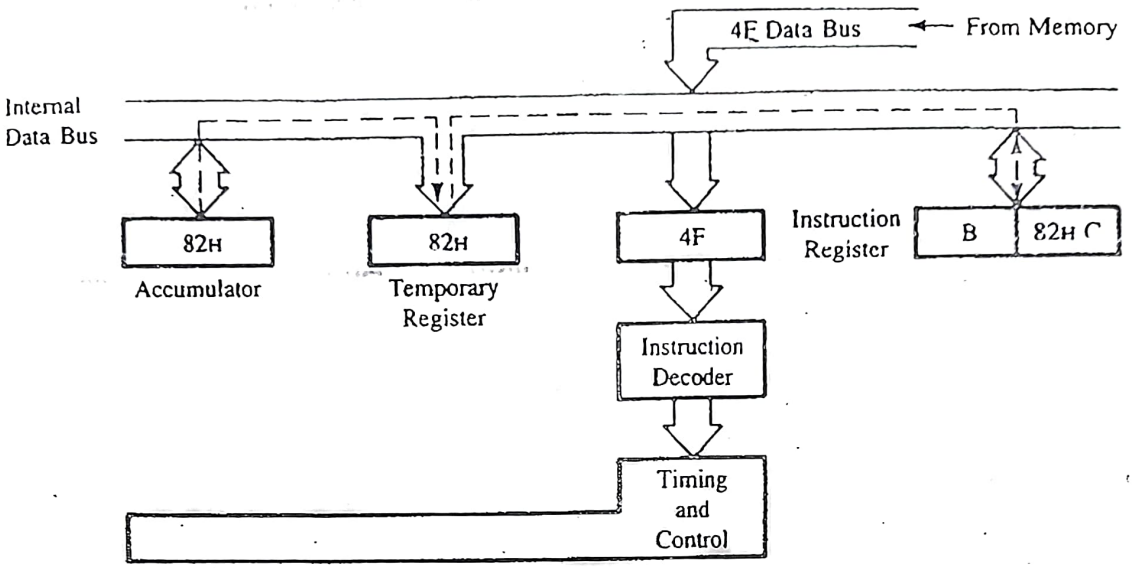


FIGURE 4.8
Instruction Decoding and Execution

4.The 8085 MPU

- ☐ transfers data from memory locations to the microprocessor by using the control signal Memory Read ($\overline{\text{MEMR}}$—active low). This is also called **reading from memory.** The term *data* refers to any byte that is placed on the data bus; the byte can be an instruction code, data, or an address.
- ☐ transfers data from the microprocessor to memory by using the control signal Memory Write ($\overline{\text{MEMW}}$—active low). This is also called **writing into memory.**
- ☐ accepts data from input devices by using the control signal I/O Read ($\overline{\text{IOR}}$—active low). This is also known as **reading from an input port.**
- ☐ sends data to output devices by using the control signal I/O Write ($\overline{\text{IOW}}$—active low). This is also known as **writing to an output port.**

5. To execute an instruction, the MPU

- ☐ places the memory address of the instruction on the address bus.
- ☐ indicates the operation status on the status lines.
- ☐ sends the $\overline{\text{MEMR}}$ control signal to enable the memory, fetches the instruction byte, and places it in the instruction decoder.
- ☐ executes the instruction.

4.2 EXAMPLE OF AN 8085-BASED MICROCOMPUTER

A general microcomputer system was illustrated in Figure 3.15, in the last chapter. After our discussion of the 8085 microprocessor and the interfacing devices, we can expand the system to include more details, as shown in Figure 4.9. The system includes interfacing devices such as buffers, decoders, and latches.