

Fundamentals Of Microprocessors

LEARNING OBJECTIVES

After studying this chapter you should be able to

-
-
-

13.1

INTRODUCTION

The most important technological invention of recent times is the *microprocessor*. The developments in the technology of the integrated circuits made it possible for the engineers of the Intel Corporation of America to develop a microprogrammable computer on a chip in 1971. This device (Intel 4004) consists of about 2300 transistors on a chip, which was fabricated using silicon-gate *p*-channel MOS technology. It was later named as microprocessor.

Since the introduction of Intel 4004, a 4-bit microprocessor, in November 1971, a large number of other microprocessors have been developed by various companies which have found applications in a large variety of products, such as laboratory instruments, consumer products, pocket calculators, process-control systems, aircraft flight control systems, toys, games, computers, etc. In fact, it has brought in a technological revolution.

Table 13.1 gives some of the important landmarks in the evolution of microprocessors. Some of the popular 8-bit and 16-bit microprocessors, with their features, are given in Table 13.2.

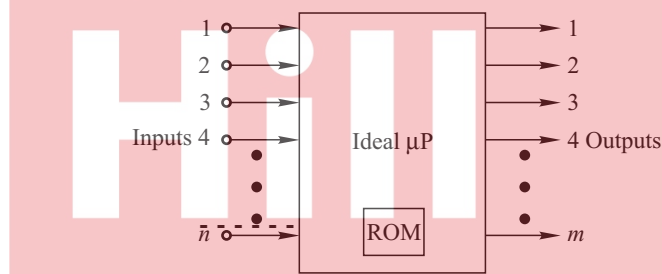
The developments in the field of microprocessors have resulted in tremendous growth in the power of microprocessors. The data-bus width and the memory size, which can be used with some of the popular Intel family of microprocessors, is given in Table 13.3. The clock-frequency has increased from about 3 MHz for 8085A μ P to 3.06 GHz for the latest Pentium 4 Intel microprocessor announced in November, 2002. More and more powerful and faster microprocessors are expected to be available in the near future.

Table 13.1 Evolution of Microprocessors

Microprocessor name	Manufacturer	Distinction
4004	INTEL	The first microprocessor (1971)
8008	INTEL	First 8-bit microprocessor (1972)
8080A	INTEL	First n -channel, second generation microprocessor (1974)
6800	MOTOROLA	First +5 V-only microprocessor (1974)
PACE	NATIONAL SEMI-CONDUCTOR	First 16-bit microprocessor (1974)
1802	RCA	First CMOS microprocessor (1974)
8048	INTEL	First 8-bit single-chip microcomputer (1976)
8088	INTEL	First 8-bit processor with 16-bit internal architecture (1979)
2920	INTEL	First analog-signal processor (1979)
80386	INTEL	First 32-bit microprocessor (1982)
PENTIUM	INTEL	First 64-bit microprocessor (1993)

13.2 AN IDEAL MICROPROCESSOR

Although there is nothing like an ideal microprocessor (μP), we shall still introduce this hypothetical device to explain the function of a microprocessor. Figure 13.1 shows an ideal μP with n inputs and m outputs. It is assumed that a ROM is included in this μP which stores the sequence of operations, referred to as the *program*, in the form of binary codes.

**Fig. 13.1** An Ideal Microprocessor

Input signals in the binary form, applied at the input terminals, are processed according to the dictates of the program and the outputs in binary form are available at the output terminals. The inputs are applied from an input device, which may be:

1. Switches,
2. Sensors,
3. A/D converters,
4. Key-boards,
5. Tape readers, etc.

Table 13.2 Popular 8- and 16-bit Microprocessors with their Features

Features/ μ P	8080A	8085	MC6800	Z80	8748	8086	MC68000	Z8000
Year	1974	1976	1974	1976	1977	1978	1979	1979
Process technology	NMOS	NMOS	NMOS	NMOS	NMOS	HMOS (<i>n</i> -channel)	NMOS	NMOS
Data bus width (bits)	8	8	8	8	8	16	16	16
Address bus width (bits)	16	16	16	16	12	20	24	16/23
Memory space (bytes)	64K	64K	64K	64K	4K	1M	16M	64K/8M
Power supply voltages	+12 V, +5 V, -5 V	+5 V	+5 V	+5 V	+5 V	+5 V	+5 V	+5 V
Packaging	40-pin DIP	40-pin DIP	40-pin DIP	40-pin DIP	40-pin DIP covered with transparent quartz lid	40-pin DIP	64-pin DIP	40/48-pin DIP
Number of basic instructions	78	80	72	158	96	97	61	110 +
Instruction time (μ s)	2	1	1	1	2.5	0.5	0.5	0.75
Resident program memory (bytes)	None	None	None	None	1K ROM 64 RAM	None	None	None

Table 13.3 Popular Intel Microprocessors

Microprocessor	Data bus width	Memory size
8085A	8	64K
8086	16	1M
8088	8	1M
80186	16	1M
80188	8	1M
80286	16	16M
80386 SX	16	16M
80386 SL	16	32M
80386 DX	32	4G
80486 SX	32	4G
80486 DX	32	4G
Pentium	64	4G

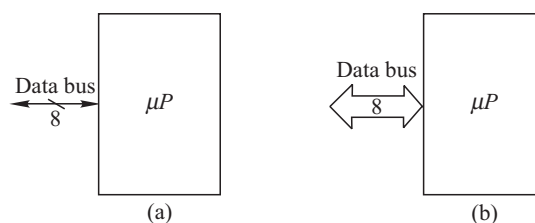
The outputs are fed to an output device, which may be:

1. Actuators,
2. Lamps,
3. Alarms,
4. Indicators,
5. D/A converters,
6. CRT displays, etc.

The basic operation of a μP , explained above, can be realized by using an ALU (arithmetic logic unit) such as 74181, with some control circuitry and ROM. As discussed earlier, the 74181 can perform various arithmetic and logical operations on two 4-bit numbers. The operation to be performed depends on the signals applied at the mode control and the 4-bit function select inputs. By using proper control circuitry and program stored in ROM, it is possible to perform a sequence of operations according to the program.

13.3 THE DATA BUS

An ideal μP , as defined in Sec. 13.2, needs a large number of pins for input and output, whereas a real μP chip cannot afford to have a large number of input and output pins because of the limitations on the number of pins available on any practical IC package. Therefore, the values of m and n are limited and usually $m = n$. Also, the number of pins get reduced if it is possible to use the same pins for inputs as well as outputs. This is made possible by using bidirectional pins. The number of such pins is referred to as the *data path width*.

**Fig. 13.2** Representation of Bidirectional Bus

Collectively, these pins are referred to as *data bus*. The word *bus* is derived from the Latin *omnibus*, meaning *for all*, and is commonly used in digital systems for a group of lines. The data bus in μP is bidirectional and is represented as shown in Fig. 13.2. The number of lines forming the bus may be written by its side.

The number of lines in the data bus shown in Fig. 13.2 is 8. This number is also known as *bus width*. The organisation of bidirectional bus is shown in Fig. 13.3, which is self-explanatory. Here, the direction (input/output) is specified with respect to the microprocessor.

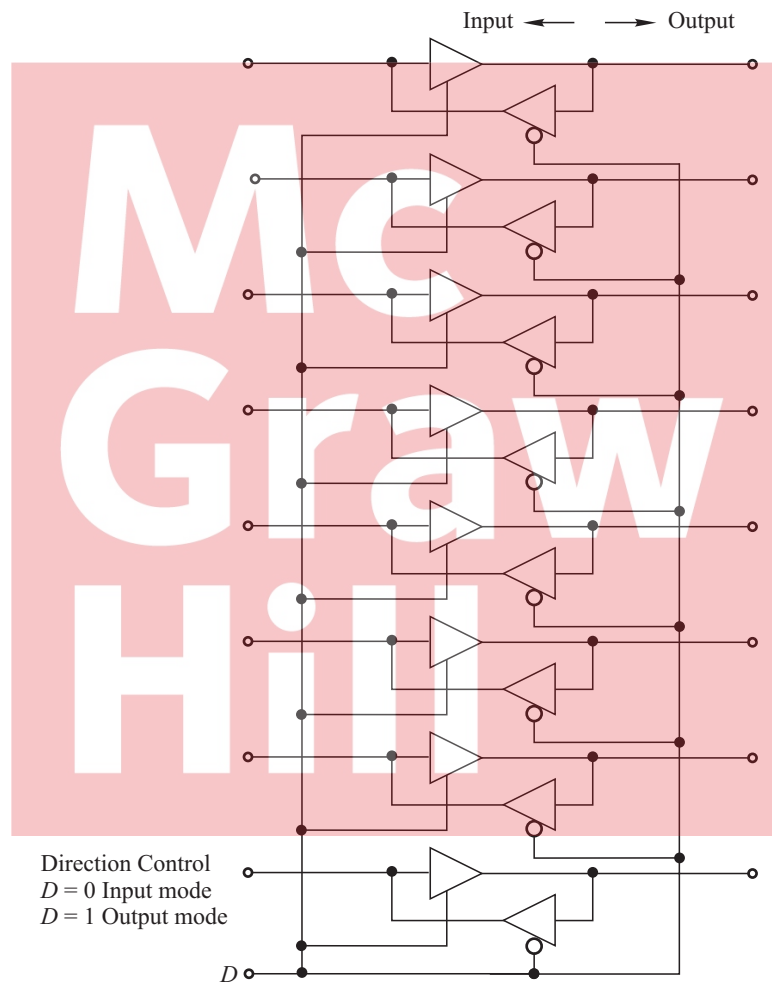


Fig. 13.3 Organization of an 8-bit Bidirectional Bus

A microprocessor processes data in the form of binary digits (bits). The number of bits that it can process at a time is referred to as the *word length* or the *word size* of the microprocessor which is same as the data-bus width. Word size is an important factor in measuring the power of a microprocessor, and a microprocessor is referred to as 8-bit, 16-bit, or 32-bit microprocessor according to its word size.

13.4 THE ADDRESS BUS

In the ideal μP , we have assumed an internal memory (ROM) in which the program is stored. In real μPs , either the internal memory is not present or it is very small, that is, only a small program can be stored in the internal memory. Therefore, we have to use an external memory, which can be RAM and/or ROM. Instructions and data are stored in memory at a set of memory locations. Each memory location has a unique address. The μP must be able to store information in this memory and fetch information from it. For this purpose, both the μP and the memory must have a set of lines known as the *address bus*. The size of the memory which can be addressed with P lines in the address bus is 2^P , referred to as *address space* or *memory space*. Therefore, for connecting a large memory to the μP , the address bus width has to be large. In many real μPs , the address bus width is 16, which can address up to $2^{16} = 65,536$ ($= 64\text{K}$, where $2^{10} = \text{K}$) memory locations. Usually, the memories are arranged to store information in the form of bytes, that is, at every memory location one byte is stored. The address can be specified in the normal binary form or in a more compact form using the hexadecimal notation. Figure 13.4 depicts the interfacing of a memory with μP .

Before reading from or writing into a memory location, the μP must first select the desired memory location by sending signals on the address bus. The address must be present as long as the reading or writing operation is not complete. In many real situations, latches are present at the address input lines of the memory, which are used to latch the address as soon as it arrives. Once the address is latched, the address bus is no longer required for the rest of the operation of the *memory read* or *memory write* cycle and can be used for any other purpose. Earlier microprocessors have the *dedicated* address bus, which means the address bus is used exclusively for transmitting addresses only. Some of the latest microprocessors, such as Intel 8085A and 8086, use the same bus for transmitting addresses as well as for transferring data. This means that a bus is used for dual purpose: it is used as an address bus when an address is to be sent and as a data bus when data transfer is to take place. For example, in the Intel 8085A μP , the lower byte of the address is sent over the address/data (AD) bus with pins marked as $AD_0 - AD_7$, and the higher byte of the address is sent over a dedicated 8-bit address bus with pins marked as $A_8 - A_{15}$. This type of operation, where a single bus is used for two different functions, is known as *multiplexing*, which results in a saving of pins on the IC chip.

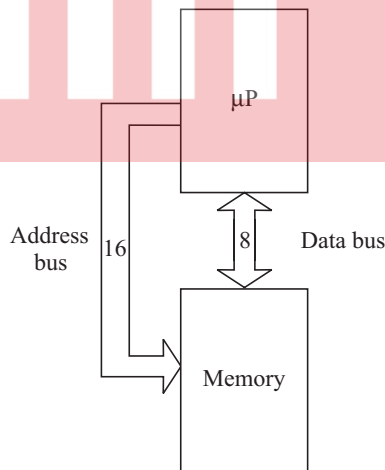


Fig. 13.4 Interfacing of Memory with μP

A microprocessor is accessed by the outside world through the input and the output devices. These devices are referred to as *I/O devices* and are interfaced with the μP through *I/O ports*. A number of such devices may be connected to a μP , the number depending on the address bus width, and are addressed by the same address bus to have access to a particular *I/O device*. Figure 13.5 illustrates the interfacing of *I/O devices* with a μP . The address bus width may not be same for memory and *I/O* devices. For example, in the 8085A μP , the address bus width for memory addressing is 16 bits, whereas for *I/O* addressing it is only 8 bits. This means that the number of *I/O* devices (*I/O* address space) which can be addressed is $2^8 = 256$. In fact, the same 8-bit address is available on $AD_7 - AD_0$ and $A_{15} - A_8$ pins. A μP with separate address space for memory and *I/O* devices must indicate whether the address at a particular instant is meant for a memory location or an *I/O* device. In contrast, the 6800 μP has an address space of 64K bytes which is shared by both the memory and the *I/O* devices. A particular address will be either for a memory location or an *I/O* device.

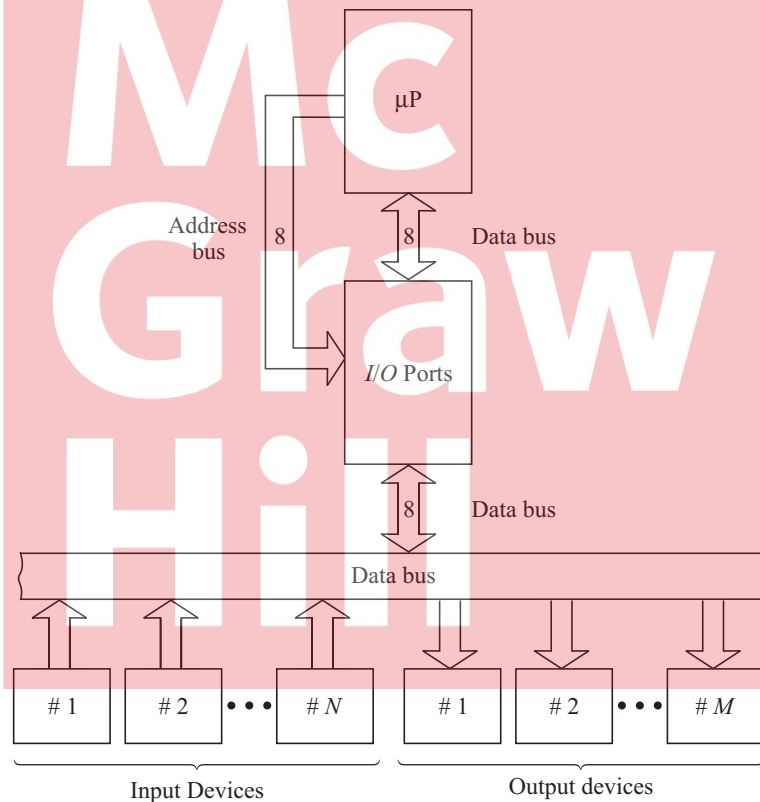


Fig. 13.5 Interfacing of I/O Devices with μP

13.5 THE CONTROL BUS

For proper operation, a microprocessor must have a set of control lines, both input and output control lines. This set of lines is referred to as the *control bus*, which is used to synchronize the operation of the μP with the operation of the external circuitry. For example, a control line is required to inform whether the address

on the address bus at a particular time is meant for a memory location or on *I/O* device, since the address bus is common for both. In 8085A μP , there is a line $IO/\overline{M}$. If the signal on this line is 1, the address is meant for *I/O*, whereas a 0 on the line means that the address is for the memory. Another example of control signals is the facility of *interrupting* the execution of the normal sequence of the program. For this purpose, there is a control input line, say *INTR*. When an *I/O* device wants to interrupt the μP , it sends out 1 over this line. The device could be an analog-to-digital (A/D) converter, which informs the μP of the fact that the conversion process is over and the data is available, or could be an indication of exceeding the safe limit of a process parameter such as temperature or pressure. The μP , after completing the current instruction, sends out a 0 on the *INTA* (interrupt acknowledgement) line, acknowledging the request of the interrupting device and then services the interrupt. Here, *INTR* is an input control line whereas *INTA* is an output control line. This communication protocol of making a request and then waiting for a response from the μP before proceeding is very common in μP -based systems, and is referred to as *handshaking*.

Another very useful operation is *direct memory access* (DMA). Normally, data transfer between the *I/O* devices and memory is through the μP , according to the instructions stored in the memory. This process of data transfer is very slow and is highly inconvenient if a large number of bytes are to be transferred at a time. For this purpose a μP is provided with some special arrangement by which a direct link is established between the *I/O* device and the memory. A *HOLD* input line informs the μP of the intention of an *I/O* device for having direct access to the memory. This is acknowledged by the μP by giving a logical 1 signal on *HLDA* (hold acknowledgement) line. The μP releases the address and data buses and the DMA operation starts.

13.6 MICROPROCESSOR BASED SYSTEM—BASIC OPERATION

Figure 13.6 shows the block diagram of a simple μP -based system. Before getting into a detailed discussion of a μP -based system, the steps involved in performing a simple task will be described. This will help the reader understand clearly how the μP , memory, and *I/O* ports function as a system.

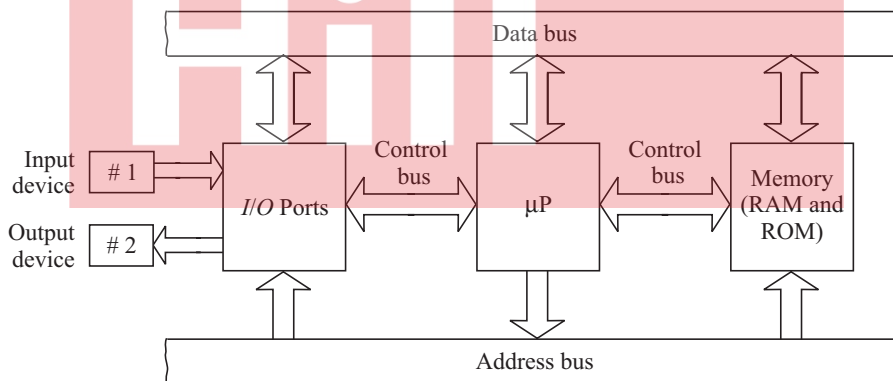


Fig. 13.6 Block Diagram of a Simple μP -based System

Consider the following simple task:

1. Input a number from input port # 01.
2. Add to this the contents of memory location 0A2F (hex).
3. Output the result to output port # 02.

Since a microprocessor is a digital device consisting of various digital circuits, it performs various operations on digital signals. The operations, which a microprocessor is capable of performing, are also required to be given in the form of digital signals. Therefore, every operation which a μP can perform is to be given in the form of a binary code which is specified by the manufacturer of the μP . For example, if one byte is used for specifying codes of operations in a μP , there are $2^8 = 256$ different codes possible for the μP . These are known as *operation codes* (Op codes). Similarly, addresses of various memory locations and *I/O* devices are also specified using binary numbers.

The task to be performed is required to be specified in terms of Op codes, addresses of memory and *I/O* devices, and data, in a proper sequence, known as *program* and this process is known as *programming*.

Let us now examine the above task step by step.

1. “Input a number from input port # 01.” It has two pieces of information. The first one indicates an ‘input’ operation from an input port and the second one specifies the address of the port (01H). This information is specified using two bytes: the first byte for the Op code and the second byte for the port number. This two-byte sequence is known as *instruction*.
2. “Add the contents of the memory location 0A2FH.” This again has two pieces of information: ‘addition’ operation and location of the data to be added, the other data is available in one of the registers (accumulator) in the μP . Here, it is assumed that the address of the memory location is stored in another special register in the μP , for which no separate information is required to be provided. This means that only the Op code for the operation ‘addition from the memory location’ is required. Therefore, this instruction has just one byte.
3. “Output the result to output port # 02.” It has a two-byte instruction: the first one is for the operation ‘output’ and the second, for the port address.

Using the above procedure, the program is written and then stored in contiguous memory locations from the initial memory location 0000 (hex.). Table 13.4 gives the contents of the memory locations for this sample program.

Table 13.4 Contents of Memory for Sample Program

Memory address (hex)	Memory contents (hex)	Remarks
0000	DB	Input Op code
0001	01	Input port number
0002	86	Add memory
0003	D3	Output Op code
0004	02	Output port number
0005		
0A2F	03	Data

The sequence of operations performed for the execution of this program is illustrated in Fig. 13.7. The execution of the program starts with the μP sending out the address of the first memory location 0000H (H stands for hexadecimal) to fetch the first instruction (step 1A). In response to this, the memory returns the contents of the memory location 0000H to the μP (step 1B). The μP decodes this word (11011011) as an *INPUT* instruction and generates control signals for the next memory-fetch operation, and

sends out the address of the next memory location 0001H to get the number of the desired input port (step 2A). In response to this, the memory returns the contents of the memory location 0001H to the μP (step 2B). This is recognized by the μP as the port number 1. Now, the first instruction is completely available in the μP . The μP will now execute this instruction by sending out the address of the input port number 1 (step 2C). In response to this, the binary number available at the input port 1 is received by the μP (step 2D) and stored in an internal register (accumulator). With this, the first instruction is complete and the μP sends out the address of the next memory location 0002H to fetch the next instruction (step 3A). In response to this, the contents of the memory location 0002H are received by the μP (step 3B). This is recognized by the μP as: *ADD* the contents of the memory location pointed to by the register in the μP meant for this purpose. This instruction is executed by the μP by sending out the address of the memory location (0A2FH), the contents of which are to be added to the contents of the accumulator (step 3C). The memory returns the contents of this location to the μP (step 3D) which are then added to the contents of the accumulator. The sum is stored in the accumulator. This completes the execution of the second instruction.

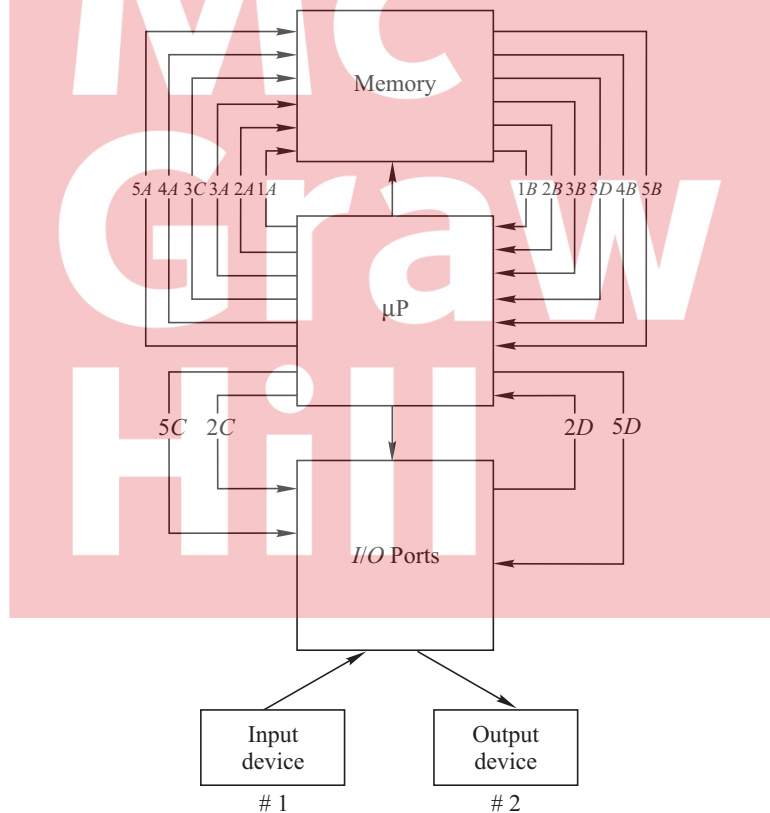


Fig. 13.7 Execution Sequence of the Sample Program

Next, the address of the next memory location 0003H is sent out by the μP (step 4A), and in response to this, the memory returns the contents of the memory location 0003H to the μP (step 4B). This is recognized by the μP as an *OUTPUT* instruction. It will then send out the address of the next memory location to fetch the number of the output port (step 5A) and receives this number (step 5B). Now, the third instruction is

completely available in the μP and its execution starts with the μP sending out the address of the port (step 5C), and after selecting the port, the contents of the accumulator are sent out to port number 2 (step 5D). This completes the execution of the program.

The above steps can be summarized as follows:

Step	Operation
1A	The μP sends out the address (0000H) of the first instruction.
1B	The contents of 0000H memory location are received by the μP .
2A	The μP sends out the next address (0001H).
2B	The contents of 0001H memory location are received by the μP .
2C	The μP sends out the address (01) of the input port.
2D	The μP receives the data from the input port (01).
3A	The μP sends out the next memory address (0002H).
3B	The contents of 0002H memory location are received by the μP .
3C	The address of the required memory location is sent out by the μP .
3D	The data from memory location 0A2FH is received by the μP .
4A	The μP sends out the next memory address (0003H).
4B	The contents of 0003H memory location are received by the μP .
5A	The μP sends out the next memory address (0004H).
5B	The contents of 0004H memory location are received by the μP .
5C	The μP sends out the address (02) of the output port.
5D	The μP sends out the data to the output port (02).

The above procedure, for the execution of such a simple program, may seem to be tedious and time consuming. However, a μP can run through all these steps in a few microseconds. While a μP is very fast, it is not capable of thinking and it does whatever it is asked to do by a program. Therefore, writing a correct program is most essential.

13.7

MICROPROCESSOR OPERATION

From the above discussion, it is amply clear that basically a microprocessor performs the following two operations:

1. FETCH an instruction from memory, and
2. EXECUTE the instruction.

The various steps involved in performing these operations are as follows:

FETCH

1. The μP places the address of the first byte of instruction on the address bus, along with a control signal, to read from the addressed memory location.
2. The μP gets this byte on the data bus. This byte is referred to as the *operation code (Op code)* and the operation of getting this byte from the memory as the *Op-code fetch*.
3. The Op code is decoded and the necessary control signals are generated.
4. If the instruction is a multi-byte instruction, the second and the subsequent bytes are read from the memory one by one by following steps similar to 1 and 2.

The operation of getting instruction bytes from the memory is known as *instruction fetch*.

EXECUTE

After the μP gets the complete instruction (all the bytes of the instruction), it performs the operation specified by the instruction. This is referred to as *execution*.

Thus, a microprocessor can be defined as a digital device on a chip which can fetch instructions from a memory, decode and execute them, i.e. perform certain arithmetic and logical operations, accept data from input devices, and send results to output devices.

It should be clearly understood that a μP alone cannot do anything. It can be used meaningfully only when it is interfaced with memory and *I/O* devices.

13.8

MICROPROCESSOR ARCHITECTURE

For performing the various operations discussed above, a microprocessor must consist of the following components:

1. Data bus,
2. Address bus,
3. Control bus,
4. Arithmetic logic unit (ALU),
5. Registers,
6. Program counter,
7. Flags, and
8. Timing and control section.

13.8.1 System Bus

The various sub-systems of the microprocessor system are interconnected by system bus, which includes the data bus, address bus, and control bus. Their functions have been discussed earlier. The data bus is used to transfer data from one part of a μP to its another part, between μP and memory, and between μP and *I/O* devices. The address bus is used to carry the address of memory locations or *I/O* devices. The address bus is unidirectional. A number of lines are required for the control operation, such as reading from memory, writing into memory, interrupt, DMA, etc.

13.8.2 Arithmetic Logic Unit (ALU)

The arithmetic logic unit is the heart of a microprocessor. It is used to perform certain arithmetic operations, such as addition, subtraction, etc. and logical operations such as AND, OR, EX-OR, etc.

13.8.3 Registers

Registers are used for storage of small data in the microprocessor. Some of the registers are accessible to the user through instructions, whereas others are not. Out of the accessible registers, are general purpose registers, referred to as *scratch-pad registers*; accumulator, used for arithmetic, logical and many other operations; and a special register known as the *stack pointer*, used to keep track of a portion of the RAM, which is used as a stack. The operation of stack will be discussed later in detail.

The registers which are not accessible to the user include the instruction register, which gets the instruction from the memory, and some temporary registers.

13.8.4 Program Counter (PC)

A program counter is required to keep track of the address of the next instruction to be fetched from the memory for execution. It always holds the address of either the first byte of the next instruction to be fetched for execution or the address of the next byte of a multi-byte instruction which has not yet been completely fetched. In either case, it gets automatically incremented one by one as the instruction bytes get fetched.

One of the control inputs to the microprocessor is the reset input. When the microprocessor is reset, the program counter sets to zero. This is the memory address from which the first instruction is to be fetched.

13.8.5 Flags

Flags are single-bit registers used to store certain conditions which arise as a result of execution of certain instructions. Some of the commonly used flags are:

Carry

If a carry is generated from MSB as a result of certain operation, the carry flag is set (1), otherwise it is reset (0).

Zero

If the result of an operation is zero, the zero flag is set, otherwise it is reset.

Sign

If the result of an operation produces 1 as MSB in the accumulator, the sign flag is set, otherwise it is reset.

Parity

If the result of an operation makes the parity of the bits in the accumulator even, the parity flag is set, otherwise it is reset.

Auxiliary Carry (half-carry)

If an operation produces carry out from the lower order four bits, the auxiliary carry flag is set, otherwise it is reset. It is used for BCD arithmetic.

Overflow

Subtraction is performed using 2's complement representation of numbers. If the result of an operation produces overflow then the overflow flag is set, otherwise it is reset.

13.8.6 Timing and Control Unit

It is used to generate proper timing and control signals which control and synchronize all the operations performed by the various sections of the microprocessor and other devices in the system.

The way in which all the above units are organized to construct a microprocessor is referred to as its architecture.

13.9 INSTRUCTION SET

Basically, the microprocessor must receive data in binary form, from the outside world (from an input device); process the data in accordance with the program stored in the memory (in binary form); and send the results in binary form, back to the outside world (to an output device). The microprocessor itself is capable of performing certain specified actions only in response to instructions in binary form. The sequence of instructions that causes the microprocessor to perform a complete task is referred to as a *program*. The collection of instructions that the microprocessor recognizes is known as its *instruction set* which is specified by the manufacturers. A user must fully understand the instruction set of the microprocessor, he/she wishes to use.

Each instruction must contain sufficient amount of information to perform the desired operation. It must contain the following information implicitly or explicitly:

- 1. The operation to be performed. It is known as Op code. For convenience of the user, it is specified by suitable names, known as *mnemonics*. For example, ADD for addition, ANA for logical AND operation, etc.
- 2. The source(s) of the operands to be used. Some instructions require only one operand, which may be in the memory, or the register, or in the instruction itself. For example, the instruction MOV A, B transfers the contents of the register B to the accumulator. Here, the operand is in the register B, which is the source of data.

Some instructions require two operands, one of which is normally in the accumulator and the source of the other operand is to be specified in the instruction. For example, ADD C, adds the contents of the register C to the contents of the accumulator. Here, one of the source operands is in the register C and the other operand is implicitly contained in the accumulator.

- 3. The destination of the result. After the operation has been performed, the result goes to the destination, specified in the instruction. For example, MOV A, M instruction uses the accumulator as the destination. The destination may be a register, memory location or an output device.

In some instructions, the destination is not required to be specified and it is implicitly taken as the accumulator. For example, in the instruction ADD C, the result of addition goes to the accumulator, that is, the result replaces one of the operands.

OP CODE	ADDRESS/DATA
Field 1	Field 2

Fig. 13.8 General Instruction Format

In general, an instruction has one or two fields, as shown in Fig. 13.8.

Usually, the fields are in the form of bytes. The Op code field is a single byte field, whereas the address/data field may be absent in some cases, for example, ADD B, else it may have an integral number of bytes. For example, in the 8085A microprocessor, an instruction may be one, two, or three bytes long. The three possible arrangements of the bytes are shown in Fig. 13.9. The instruction bytes are stored in contiguous memory locations starting from the first byte.

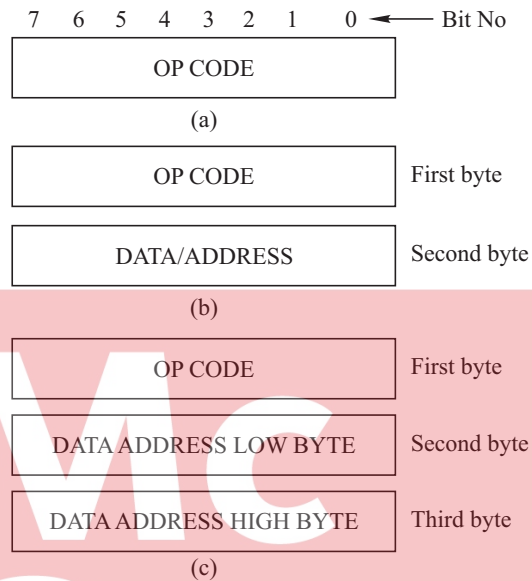


Fig. 13.9 Arrangement of Instruction Bytes in 8085A Microprocessor: (a) Single-byte Instruction (b) Two-byte Instruction (c) Three-byte Instruction

Example 13.1

It is required to add the contents of memory locations $2C4A$ (hex) and $3DA6$ (hex) and store the sum in memory location $7AFF$ (hex).

Solution

The required program will have the following sequence of instructions for 8085A microprocessor.

```
LDA    2C4AH
LXI    H,3DA6H
ADD    M
STA    7AFFH
```

The number of bytes in these instructions are 3, 3, 1, and 3, respectively. The program is to be stored in the memory before it can be executed by the microprocessor. Let us assume that the program is stored as shown in Table 13.5.

Table 13.5 Sequence of bytes for the Program of Ex. 13.1

Memory address (hex)	Contents (binary)	Remarks
0000	00111010	Op code of LDA
0001	01001010	Lower eight bits of address (4A)
0002	00101100	Higher eight bits of address (2C)
0003	00100001	Op code of LXI H

(Continued)

Table 13.5 (Continued)

Memory address (hex)	Contents (binary)	Remarks
0004	10100110	Lower byte of address (46)
0005	00111101	Higher byte of address (3D)
0006	10000110	Op code of ADD <i>M</i>
0007	00110010	Op code of STA
0008	11111111	Lower byte of address (FF)
0009	01111010	Higher byte of address (7A)

It should be clearly understood that the design of microprocessor-based systems require a knowledge of both hardware (circuitry) and software (program).

13.10 THE 8085A MICROPROCESSOR

The basic concepts involved in the architecture and operation of a microprocessor have resulted in a large number of different microprocessor chips with varying flexibilities. Although the basic concepts are more or less same, they differ in architecture, operation, and instruction set. If one tries to learn about all the microprocessors simultaneously, he/she will simply end up in confusion. On the other hand, if one can master one type of microprocessor he/she can easily understand the other microprocessors. Therefore, we have chosen the most popular microprocessor, Intel 8085A, for discussion.

13.10.1 Organisation and Operation

The Data and Address Buses

The Intel 8085A is an 8-bit microprocessor. Its block diagram is shown in Fig. 13.10. It has an 8-bit bidirectional data bus which is used for data input as well as data output in the form of 8-bit ($D_0 - D_7$) words.

The address bus is 16-bit wide, which means that memory address space is 64K bytes. The address bus is divided into two sections. The lower-order eight bits of the address ($A_0 - A_7$) are transferred on the data bus itself, and hence, the bus is referred to as *AD* ($AD_0 - AD_7$) bus. The higher-order eight bits of the address are transferred on $A_8 - A_{15}$ lines. The time-multiplexed bus system is used to save the number of pins in the IC package. The address and data information is present on the bus in different time slots.

Arithmetic Logic Unit

It has an 8-bit ALU. The arithmetic and logical operations are performed in the ALU on 8-bit words.

Accumulator Register

There is an 8-bit register, known as the accumulator. For arithmetic operations, such as addition, subtraction, etc. and logical operations, such as AND, OR, EX-OR, etc. one of the operands is contained in the accumulator. The result of these operations is also stored in the accumulator. This register is labelled as A.

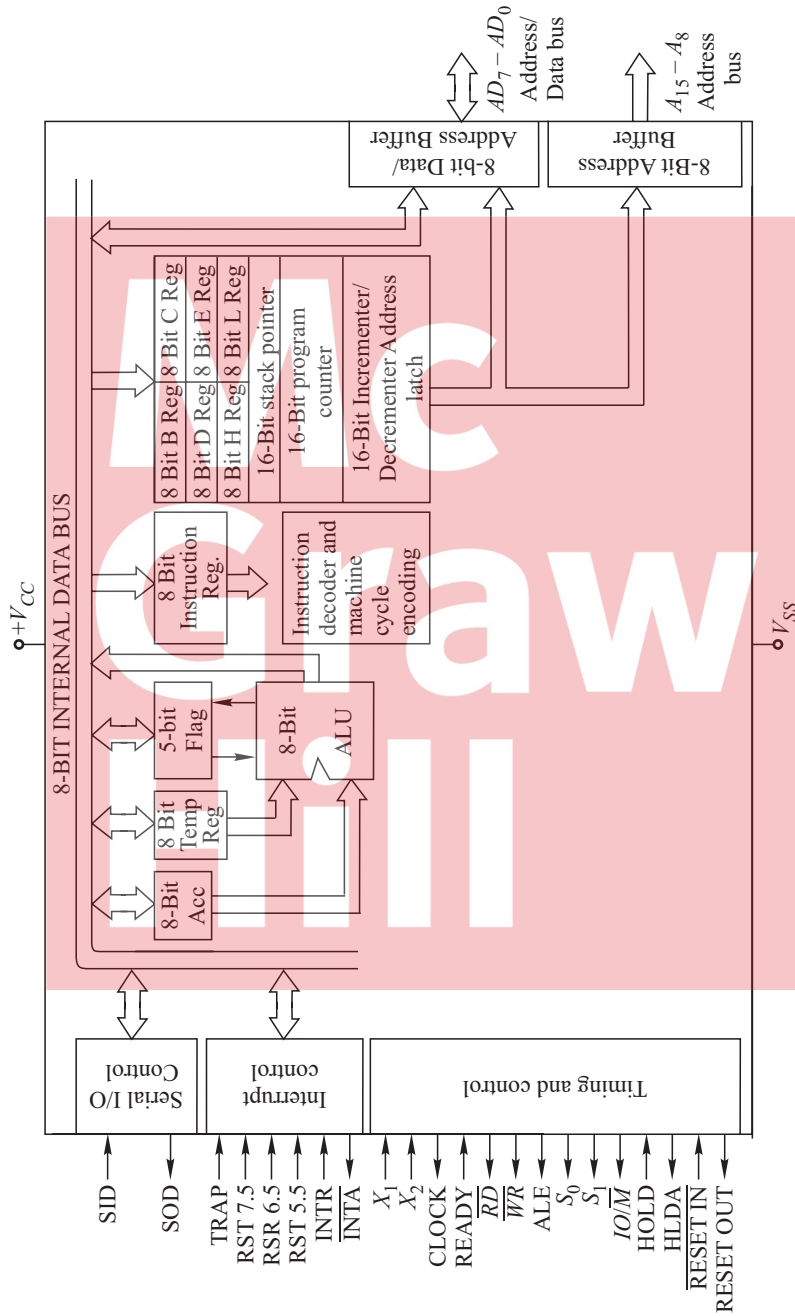


Fig. 13.10 Functional Block Diagram of 8085A Microprocessor

General-purpose Registers

There are six 8-bit general-purpose registers, labelled as B, C, D, E, H, and L. These registers can be used as single 8-bit registers or in pairs, as 16-bits registers. When used in pairs, the allowed pairs are B and C; D and E; and H and L. The higher-order eight bits (or higher byte) are stored in the first register of the pair (B, D, or H) and the lower-order eight bits (or lower byte) are stored in the second register of the pair (C, E, or L). For example, if 1AC2 (hex) is to be stored in H – L pair, 1AH will be stored in the H register and C2H in the L register. A multiplexer is used to route the data on the internal data bus to either the higher-order registers column or the lower-order registers column.

Stack Pointer

The stack pointer (SP) is a 16-bit register used to store the address of the stack top. The use of stack and stack pointer are discussed later in detail.

All these registers are accessible to the user through instructions. They are not accessible physically. The names and binary codes of the 8-bit registers are given in Table 13.6.

Table 13.6 Names and Binary Codes of 8-bit Registers of 8085 A

Register name	Binary code
A	111
B	000
C	001
D	010
E	011
H	100
L	101

The names and binary codes for register pairs are given in Table 13.7.

Table 13.7 Names and Binary Codes of Register Pairs of 8085A

Register pair	Name	Binary code
B–C	B	00
D–E	D	01
H–L	H	10
Stack pointer (16-bit register)	SP	11

The 16-bit H–L register pair is used to provide the implied memory address for most memory-reference instructions, for example,

MOV A, M

ADD M

ANA M

A limited number of memory-reference instructions use the B–C and D–E register pairs, for example,

LDAX B

LDAX *D*
 STAX *B*
 STAX *D*

Flags

There are five FLIP-FLOPs (one-bit registers), which serve to indicate certain conditions that arise during arithmetic and logical operations and are referred to as *flags*. These are:

1. Zero (Z)
2. Sign (S)
3. Parity (P)
4. Carry (CY)
5. Auxiliary carry (AC)

A flag is *set* by forcing the bit to 1 and *reset* by forcing the bit to 0. When an instruction affects a flag, it affects it in the manner discussed in Sec. 13.8. These flags may be accessed by some instructions along with the accumulator, as a single processor status word (PSW). PSW bits are assigned as shown in Fig. 13.11.

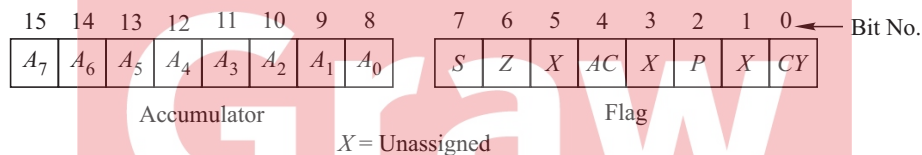


Fig. 13.11 Organization of PSW

There are two instructions STC (set carry) and CMC (complement carry) which directly address the carry flag.

The flags are examined in conditional instructions to determine if the specified condition is true or false. The names and the binary codes of the conditions are given in Table 13.8.

Table 13.8 Names and Binary Codes for Conditions

Condition name	Flag status	Binary code
NZ — Not zero	Z = 0	000
Z — Zero	Z = 1	001
NC — No carry	CY = 0	010
C — Carry	CY = 1	011
PO — Parity odd	P = 0	100
PE — Parity even	P = 1	101
P — Plus	S = 0	110
M — Minus	S = 1	111

Program Counter

The instructions to be executed are stored in contiguous memory locations. The address of the next instruction to be fetched from memory is contained in the 16-bit program counter (PC). For example, consider an

instruction LDA 02C4H. This instruction means load the accumulator with the contents of the memory location 02C4H. Let us assume that the instruction code (LDA) of this instruction is stored in the memory location 0100H. The next two memory locations 0101H and 0102H will contain C4H and 02H, respectively, and the next instruction will be stored starting from the memory location 0103H.

When the microprocessor has completed the execution of the instruction prior to LDA 02C4H, the PC will contain 0100H. It goes on incrementing automatically as it fetches the instruction bytes from the memory. At the end of the execution of this instruction, the contents of the PC will be 0103H.

In addition to the above registers, which can be accessed through instructions, there are some registers which cannot be accessed. The instruction register gets the operation code of the instruction in Op code fetch machine cycle and passes it on to the decoder. The function of the decoder is to interpret the instruction and enable the control section to produce proper signals to carry out the tasks required to be performed in the instruction.

Several other registers, such as temporary register, are involved only in internal operations and are not accessible to the user.

Clock

A crystal or RC network connected between pins X_1 and X_2 determines the frequency of the internal clock generator which synchronizes the operation of the 8085A. The clock frequency generated is one half of the crystal frequency. This is also available at the clock output pin which can be used for synchronizing external devices.

Power Supply

The 8085A operates on a single +5 V power supply, connected between V_{CC} and V_{SS} pins.

Interrupt System

In many applications, it may be necessary to postpone the routine job that the microprocessor is doing to attend to some urgent task. For example, a microprocessor-based system may be used for monitoring the temperature and pressure of a process. If the temperature and/or pressure goes beyond a prespecified safe limit, the microprocessor should be able to initiate proper actions to avoid any damage to life and equipment.

For this purpose, a signal indicating such an emergency must be able to interrupt the microprocessor. In fact, the interrupting system of the microprocessor allows signals of higher priority to usurp and steer the processor into a different program.

The interrupt system of the 8085A microprocessor consists of five different input signals, which have a fixed priority unless changed by SIM instruction (discussed later). These are given below according to their priorities.

TRAP	Highest priority
RST 7.5	
RST 6.5	
RST 5.5	
INTR	Lowest priority

In the case of the first four interrupts, the control is transferred to the memory location as given below, whereas for INTR, a CALL address instruction is supplied to the microprocessor by an external device, known as *interrupt controller*.

<i>Interrupt</i>		<i>Branching address</i>
TRAP		$4.5 \times 8 = 24\text{H}$
RST	5.5	$5.5 \times 8 = 2\text{CH}$
RST	6.5	$6.5 \times 8 = 34\text{H}$
RST	7.5	$7.5 \times 8 = 3\text{CH}$

13.10.2 Programming

The instruction set of the 8085A microprocessor have been grouped into the following five groups:

1. Data transfer group,
2. Arithmetic group,
3. Logical group,
4. Branch group, and
5. Stack, I/O and machine control group.

Data Transfer Group Instructions

This includes instructions to move data between registers, between memory and registers, and the data contained in the instruction itself to memory or registers.

Move Instructions These are one-byte instructions for the transfer of a byte of data from one 8-bit register to another 8-bit register, or from a memory location to an 8-bit register, or from an 8-bit register to a memory location. The mnemonic for these instructions is MOV. The destination and the source follow the mnemonic and are separated by comma. The destination is always listed before the comma and the source, after. For example, the instruction *MOV A, B* moves the contents of the register *B* to the register *A*. In fact, the contents of register *B* are copied in register *A*. These instructions do not change the contents of the source (register or memory location).

Whenever memory location is involved as the source or destination of data, the address of the memory location is taken as the contents of the H-L pair. Therefore, the desired address must first be loaded into the H-L register pair. For example, the *MOV A, M* instruction, copies the contents of the memory location, whose address is stored in the H-L register pair into the accumulator. The H-L register pair is used as a pointer to a memory location, as shown in Fig. 13.12.

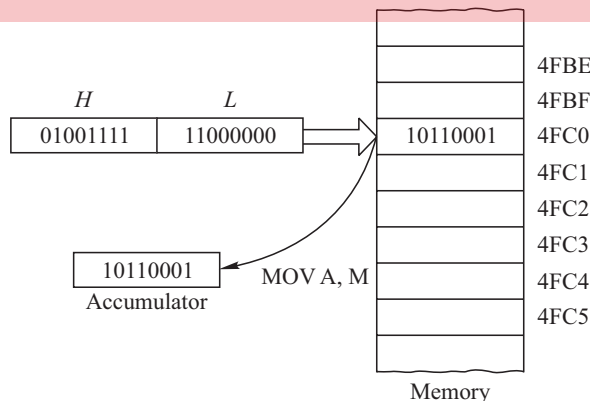


Fig. 13.12 Use of H-L Register Pair as Memory Pointer

Similarly, the instruction `MOV M, B` copies the contents of register *B* into the memory location pointed to by the H–L register pair.

Move Immediate Instructions The move immediate instructions (MVI) transfer the byte of data specified within the instruction itself to the register or the memory location (whose address is stored in the H–L register pair). For example, `MVI C, 02H`, transfers the data byte (02H) to the register *C*, whereas, `MVI M, 3AH`, transfers the data byte (3AH) to the memory location pointed to by the H–L register pair. These instructions are two bytes long, requiring two memory locations to store them. The first byte is for the Op code which is stored in the first memory location, followed by the byte corresponding to the data contained in the instruction.

Load Extended Immediate This group of instructions is used to transfer a 16-bit number, specified in the instruction in the form of two bytes, to the specified register pair. For example, `LXI H, 23A7H` loads 00100011 into the H register and 10100111, into the L register. These instructions are three bytes long, one for the Op code, and two for the data. The instruction bytes are stored in three consecutive memory locations, as shown in Fig. 13.13.

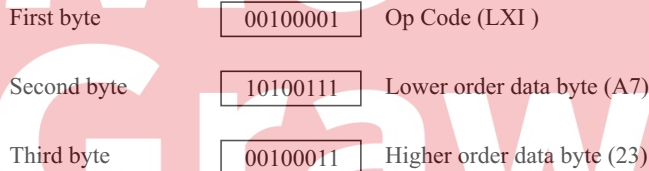


Fig. 13.13 Instruction bytes for `LXI H, 23A7H`

Example 13.2

Write a program to transfer one byte of data from the memory location 0010H to 1000H.

Solution

Instruction	Comments
LXI H, 0010H	Load H–L pair with 0010H.
MOV A, M	Transfer the contents of memory location pointed to by the H–L pair to the A register (accumulator).
LXI H, 1000H	Load the H–L pair with 1000H.
MOV M, A	Transfer the contents of the register A to the memory location addressed to by the H–L pair.

We note that there is no instruction to directly transfer the contents of one memory location to another. The transfer is possible only through one of the registers of the microprocessor.

Load/Store Accumulator Direct Load accumulator direct (LDA) loads the accumulator with the byte stored at the memory location whose address is specified in the instruction. Similarly, store accumulator direct (STA) stores the contents of the accumulator in the memory location whose address is specified in the instruction. These are three-byte instructions. The first byte is for the Op code, second, for the lower-order 8 bits of the address and the third, for the higher-order 8 bits of the address.

Considering Ex. 13.2, the same program can be written as:

```
LDA    0010H    Load the accumulator with the contents of memory location 0010H.
STA    1000H    Store the contents of the accumulator in the memory location 1000H.
```

Load/Store H and L Direct The instruction LHLD or load H and L direct, loads the contents of the memory location, whose address is specified in the second and third bytes of the instruction, into the L register and the byte in the next memory location is loaded into the H register. The instruction SHLD, or store H and L direct, is the reverse operation. It transfers the contents of the L register in the memory location specified by the second and third bytes of the instruction, and the contents of the H register is transferred into the next memory location. For example, the LHLD 0A22H instruction transfers the contents of the memory location 0A22H to the L register and 0A23H, to the H register. This is illustrated in Fig. 13.14.

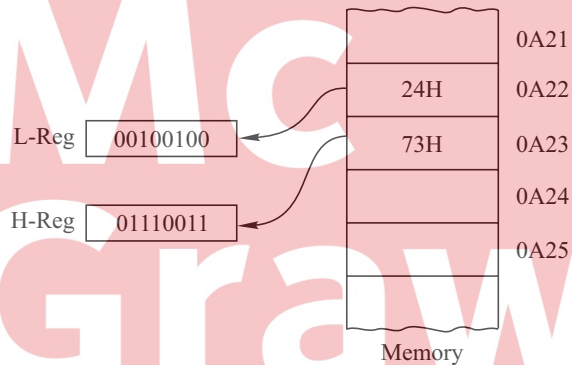


Fig. 13.14 Operation Performed by LHLD Instruction

Similarly, the operation performed by the SHLD instruction is shown in Fig. 13.15.

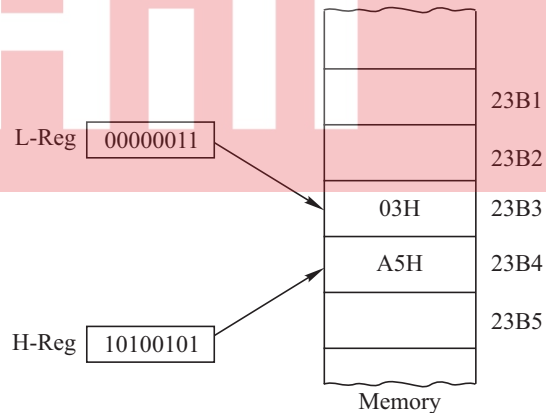


Fig. 13.15 Operation Performed by the Instruction SHLD 23B3 H

Load/Store Accumulator Indirect The instruction LDAX B loads the accumulator with the contents of the memory location whose address is in the B–C register pair. Similarly, LDAX D uses the contents of the register pair D–E as the address of the memory location.

The instructions STAX B and STAX D store the contents of the accumulator at the address pointed to by register pairs B–C or D–E, respectively. All four instructions are single-byte instructions.

Example 13.3

Explain the program given below:

```
LXI    B, 2475H
LXI    D, 3794H
LDAX   B
MOV    L, A
LDAX   D
STAX   B
MOV    A, L
STAX   D
```

Solution

The operation performed by each instruction is given below.

Instruction	Operation
LXI B, 2475H	Loads the B–C pair with 2475H.
LXI D, 3794H	Loads the D–E pair with 3794H.
LDAX B	Loads A with the contents of memory location pointed to by the B–C pair (2475H).
MOV L, A	Moves the contents of A to L register
LDAX D	Loads A with the contents of memory location pointed to by the D–E pair (3794H).
STAX B	Stores A at the memory location whose address is in the B–C pair (2475H).
MOV A, L	Moves the contents of L to A.
STAX D	Store the contents of A at the memory location whose address is in the D–E pair.

In short, this program stores the addresses 2475H and 3794H of two memory locations in register pairs B–C and D–E, respectively, and exchanges the contents of these memory locations.

The same job can be performed by writing a program in a different way. Try and make sure that there is no unique way of writing a program.

Exchange Instruction The instruction XCHG, a single byte instruction, exchanges the contents of H–L and D–E register pairs.

None of the five flags are affected by the instructions of the data-transfer group.

All the instructions of the data-transfer group, in terms of their mnemonics, and operands along with their Op code (hex), are given in Table 13.9.

Table 13.9 Data Transfer Group Instructions

Instruction		OP	Instruction		Op	Instruction		Op	Instruction		Op		
Mnemonic	Operands	Code (hex)	Mnemonic	Operands	Code (hex)	Mnemonic	Operands	Code (hex)	Mnemonic	Operands	Code (hex)		
MOV	A, A	7F	MOV	D, A	57	MOV	L, A	6F	LXI	B, 16-bit	01		
	A, B	78		D, B	50		L, B	68		D, 16-bit	11		
	A, C	79		D, C	51		L, C	69		H, 16-bit	21		
	A, D	7A		D, D	52		L, D	6A		SP, 16-bit	31		
	A, E	7B		D, E	53		L, E	6B					
	A, H	7C		D, H	54		L, H	6C					
	A, L	7D		D, L	55		L, L	6D					
A, M	7E	D, M	56	L, M	6E			STA	16-bit addr	32			
MOV	B, A	47	MOV	E, A	5F	MOV	M, A	77	SHLD	16-bit addr	2A		
	B, B	40		E, B	58		M, B	70					
	B, C	41		E, C	59		M, C	71					
	B, D	42		E, D	5A		M, D	72					
	B, E	43		E, E	5B		M, E	73					
	B, H	44		E, H	5C		M, H	74					
	B, L	45		E, L	5D		M, L	75					
B, M	46	E, M	5E					LDAX	B	0A			
									D	1A			
MOV	C, A	4F	MOV	H, A	67	MVI	A, byte	3E	STAX	B	02		
	C, B	48		H, B	60		B, byte	06		D	12		
	C, C	49		H, C	61		C, byte	0E					
	C, D	4A		H, D	62		D, byte	16					
	C, E	4B		H, E	63		E, byte	1E					
	C, H	4C		H, H	64		H, byte	26		XCHG			EB
	C, L	4D		H, L	65		L, byte	2E					
C, M	4E	H, M	66	M, byte	36								

Arithmetic Group Instructions

Add Instructions Addition instructions are used to add the contents of the specified register, memory location, or the data-byte in the instruction itself, to the contents of the accumulator. The result is stored in the accumulator. If the addition produces any carry, the carry flag is set ($CY = 1$), otherwise it is reset. All the other flags are also affected, depending upon the results. In the case of an operand being in the memory, the memory address is the contents of the H–L register pair. The addition instructions are:

ADD r	Add the contents of the register r.
ADD M	Add the contents of the memory location (address in the H–L pair).
ADI data	Add the data byte.

There is another group of ADD instructions in which the carry bit (CY) is also added in addition to the specified register, memory location or the data byte. These are:

ADC r	Add the register with the carry.
ADC M	Add the memory with the carry.
ACI data	Add the data-byte with the carry.

Example 13.4

Assume the contents of the accumulator and register C as 2EH and 6CH, respectively. The instruction ADD C performs the addition as follows:

$$\begin{array}{r}
 2EH = 00101110 \\
 6CH = 01101100 \\
 \hline
 9AH = 10011010
 \end{array}$$

The contents of registers A, C and flags, following execution of the ADD C instruction, are given in the Table 13.10.

Table 13.10 Table for Ex. 13.4

Register or Flag	Contents		Remarks
	Before ADD C	After ADD C	
A	00101110	10011010	Result of addition
C	01101100	01101100	No change
CY	Don't Care	0	No carry
S	"	1	MSB of result
Z	"	0	Result not zero
P	"	1	Even parity
AC	"	1	Carry from bit 3

Example 13.5

Assume that the accumulator contains 14 H and that the carry bit is set ($CY = 1$). The instruction ACI 42H will have the following effect:

Register A	=	14H	=	00010100
Immediate data	=	42H	=	01000010
Carry (CY)	=	1	=	1
				01010111 = 57H

The contents of the accumulator following ACI 42H instruction will be 57H and the carry flag will be reset ($CY = 0$). All other flags will also be affected.

Subtract Instructions Subtraction instructions are used to subtract the contents of the specified register, memory location or data-byte in the instruction itself, from the contents of the accumulator. The result of a subtraction is stored in the accumulator. If the result is negative, the carry flag will be set ($CY = 1$). The setting of the carry flag indicates that the result in the accumulator is in 2's complement form. The subtraction is actually performed by adding 2's complement of the subtrahend to the minuend. All the flags are affected by these instructions. The subtraction instructions are given below:

SUB r	Subtract the contents of the register r from the accumulator.
SUB M	Subtract the contents of the memory location (pointed to by H–L register pair) from the accumulator.
SUI data	Subtract the data-byte from the accumulator.

Example 13.6

Consider the instruction SUB M. Let the contents of the relevant memory location, registers and flags be as given below:

Accumulator	=	3EH
Register H	=	00H
Register L	=	7CH
Memory location (007CH)	=	9FH
CY	=	0
S	=	0
P	=	1
Z	=	0
AC	=	1

The subtraction operation performed, following the execution of SUB M instruction, is as follows:

Accumulator	=	3EH = 00111110
Memory location	=	9FH = 01100001 (2's complement)
contents	=	9FH = 10011111

The contents of the memory location, registers H, and L are not affected by this instruction. The contents of the accumulator will be 10011111 and the condition flags are set as follows:

CY = 1	Indicates negative result.
S = 1	Indicates MSB of result is 1.
Z = 0	Indicates result is non-zero.
P = 1	Indicates even parity.

$AC = 0$ Indicates absence of auxiliary carry.

Since the carry flag is set indicating negative result, the contents of the accumulator is in 2's complement form and, therefore, the actual result is $-61H$.

Note carefully the setting of the carry flag in the subtraction operation. In fact, it is named as *borrow flag* during the subtraction operation. If the addition of 2's complement of the subtrahend does not produce a final carry output then the flag is set ($CY = 1$), whereas if a carry is produced then the carry flag is reset ($CY = 0$).

Another group of subtraction instructions subtracts the contents of memory location, register or immediate data-byte and the borrow flag (CY) from the contents of the accumulator, and the result is stored in the accumulator. These are:

SBB r	Subtract the register r with the borrow.
SBB M	Subtract the memory with the borrow.
SBI data	Subtract the data-byte with the borrow.

All the flags are affected in the same manner as discussed above, for normal subtraction operation.

Example 13.7

In Ex. 13.6, if $CY = 1$ and SBB M instruction is executed, the subtraction operation will be performed as shown below:

Accumulator = $3EH =$	0011 1110
Subtract CY	1
	0011 1101
Memory location = $9FH$	0110 0001 (2's complement)
contents	1001 1110

Since the addition process does not produce a carry, the borrow flag is set ($CY = 1$), following the execution of this instruction, indicating a negative result. You can find the settings of other condition flags.

Increment/Decrement Instructions The increment instructions increment the contents of the specified register or memory location by one and the decrement instructions decrement the contents by one. The results of increment/decrement are left in the same register or memory location.

These instructions affect all the flags except the carry flag. Since the carry flag is preserved by these instructions, these can be used within multi-byte routines for incrementing/decrementing counts and similar purposes. The increment/decrement instructions are given below:

INR r	Increment the contents of the register r by one.
INR M	Increment the contents of the memory location (pointed to by H-L register pair) by one.
DCR r	Decrement the contents of the register r by one.
DCR M	Decrement the contents of the memory location by one.

Increment/Decrement Register Pair The instruction INX rp adds one to the contents of specified register pair (rp) and DCX rp decrements the contents of the specified register pair by one. Any one of the four (B, D, H) register pairs or SP may be incremented or decremented by these instructions.

Since these instructions do not affect the condition flags, they can be used to modify the address in any instruction sequence.

Double Precision Addition The instruction DAD rp adds the contents of the specified register pair (rp) to the H–L register pair. The 16-bit result is stored in the H–L register pair and the carry flag is set if there is a carry-out of the 16-bit addition. No other flags are affected.

Example 13.8

Assume the following contents of various registers before the DAD B instruction is executed. Find the contents after the execution of this instruction.

(PC) = 0F78H (H) = 02H
(B) = 3FH (L) = 46H
(C) = BAH (PSW)_L = 17H

Solution

The contents after the execution of the instruction DAD B are given below:

(PC) = 0F79H
(B) = 3FH
(C) = BAH
(H) = 42H
(L) = 00H
(PSW)_L = 16H

Decimal Adjust Accumulator Instruction The instruction DAA is used when adding decimal numbers. When decimal numbers are added, the 8-bit value in the accumulator may not be a valid BCD. This instruction operates as follows:

1. If the least-significant four bits of the accumulator is not a valid BCD digit (greater than 9), or if the auxiliary carry flag is set ($AC = 1$), then 0110 is added to the accumulator.
2. If the most-significant four bits of the accumulator is not a valid BCD digit, or if the carry flag is set ($CY = 1$), then 0110 0000 is added to the accumulator.

This is the only instruction which requires the use of the auxiliary carry (AC) flag. All the flags are affected by this instruction.

All the instructions of arithmetic group, in terms of their mnemonics and operands along with their Op code (hex), are given in Table 13.11.

Logic Group Instructions

Logic Instructions There are instructions for performing AND, OR, EX-OR and comparison operations. The bits of the specified register, memory location or data-byte in the instruction and the corresponding bits of the accumulator are involved in the operation on a bit-by-bit basis. The result of AND, OR, and EX-OR operation is placed in the accumulator. All the flags are affected, but the carry (CY) and the auxiliary carry (AC) flags setting depend upon the operation being performed rather than on the results of operation. These are given in Table 13.12.

In the case of comparison instruction, the contents of the specified register, the memory location, or the immediate byte are subtracted from the contents of the accumulator. The accumulator contents do not change, but the comparison is indicated by the zero and carry flags. The zero flag is set ($Z = 1$) if both the numbers

Table 13.11 Arithmetic Group Instructions

Instruction			Instruction			Instruction		
Mnemonic	Operands	Op code (hex)	Mnemonic	Operands	Op code (hex)	Mnemonic	Operands	Op code (hex)
ADD	A	87	SBB	A	9F	INX	B	03
	B	80		B	98		D	13
	C	81		C	99		H	23
	D	82		D	9A		SP	33
	E	83		E	9B	DCX	B	0B
	H	84		H	9C		D	1B
	L	85		L	9D		H	2B
	M	86		M	9E		SP	3B
ADC	A	8F	INR	A	3C	DAD	B	09
	B	88		B	04		D	19
	C	89		C	0C		H	29
	D	8A		D	14		SP	39
	E	8B		E	1C	ADI ACI SUI SBI	byte	C6
	H	8C		H	24		byte	CE
	L	8D		L	2C		byte	D6
	M	8E		M	34		byte	DE
SUB	A	97	DCR	A	3D	DAA		27
	B	90		B	05			
	C	91		C	0D			
	D	92		D	15			
	E	93		E	1D			
	H	94		H	25			
	L	95		L	2D			
	M	96		M	35			

Table 13.12 Flag Setting for Logic Instructions

Operation	Instructions	Flags settings
AND	ANA r	CY = 0, AC = 1
	ANA M	
	ANI byte	
EX-OR	XRA r	CY = 0, AC = 0
	XRA M	
	XRI byte	
OR	ORA r	CY = 0, AC = 0
	ORA M	
	ORI byte	
Compare	CMP r	Z = 1 if (A) = (byte) CY = 1 if (A) < (byte)
	CMP M	
	CPI byte	

are equal and the carry flag is set ($CY = 1$) if the contents of the accumulator is less than the byte being compared. The other flags are affected as in the usual subtraction process.

Example 13.9

Suppose the accumulator contains *AAH* and the register *B* contains *0FH*. Assume the following flag settings:

$$Z = 1, \quad CY = 1, \quad S = 1, \quad P = 1, \quad AC = 0$$

Find the contents of the accumulator, register *B*, and flag settings, following execution of each of the following instructions:

ANA B
XRA B
ORA B
CMP B

Solution

These are given in Table 13.13.

Table 13.13 Table for Ex. 13.9

Instruction	Contents of registers		Flags				
	A	B	CY	AC	Z	S	P
ANA B	00001010	00001111	0	1	0	0	1
XRA B	10100101	00001111	0	0	0	1	1
ORA B	10101111	00001111	0	0	0	1	1
CMP B	10101010	00001111	0	0	0	1	0

Note that the contents of the accumulator do not change, following the execution of the CMP instruction. Also, $CY = 0$ and $Z = 0$ and the contents of A are greater than those of B.

Rotate Instructions The rotate or shift instructions have two variations which differ in the way the carry bit is used. Operands are not permitted with these instructions, and only the carry flag is affected following their execution. The instructions are explained below.

Rotate Accumulator Left (RLC) RLC sets the carry flag equal to the most-significant bit (MSB) of the accumulator and then rotates the contents of the accumulator one bit position to the left with the MSB transferring to the LSB. This is shown in Fig. 13.16a.

Rotate Accumulator Right (RRC) RRC sets the carry flag equal to the LSB of the accumulator and then rotates the contents of the accumulator one bit position to the right with the LSB transferring to the MSB. This operation is shown in Fig. 13.16b.

Rotate Left through Carry (RAL) RAL rotates the contents of the accumulator and the carry flag one bit position to the left. The carry flag is transferred to LSB and is set by the MSB. This operation is shown in Fig. 13.16c.

Rotate Right through Carry (RAR) RAR rotates the contents of the accumulator and the carry flag one bit position to the right. The carry flag is transferred to the MSB and is set by the LSB. This operation is shown in Fig. 13.16d.

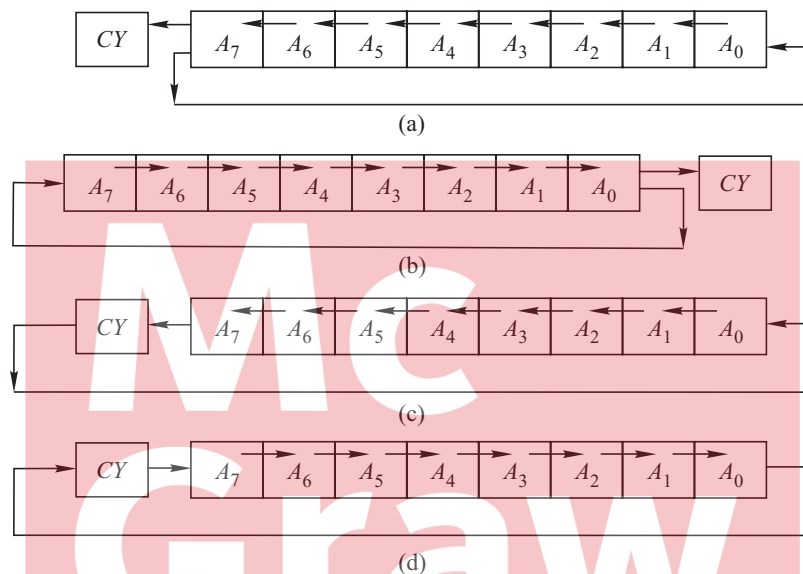


Fig. 13.16 Rotate Instructions

Special Instructions The instruction CMA complements or inverts each bit in the accumulator and the flags are not affected. STC sets the carry flag to one ($CY = 1$) and CMC complements the carry flag. These instructions do not affect the other flags.

All the instructions of logic group in terms of their mnemonics and operands along with their Op code (hex) are given in Table 13.14.

Table 13.14 Logical Group Instructions

Instruction			Instruction		
Mnemonic	Operand	Op code (hex)	Mnemonic	Operand	Op code (hex)
ANA	A	A7	CMP	A	BF
	B	A0		B	B8
	C	A1		C	B9
	D	A2		D	BA
	E	A3		E	BB
	H	A4		H	BC
	L	A5		L	BD
	M	A6		M	BE
	A	AF	ANI	byte	E6

(Continued)

Table 13.14 (Continued)

Instruction		Op code (hex)	Instruction		Op code (hex)
Mnemonic	Operand		Mnemonic	Operand	
XRA	B	A8	XRI	byte	EE
	C	A9	ORI	byte	F6
	D	AA	CPI	byte	FE
	E	AB	RLC		07
	H	AC	RRC		0F
	L	AD	RAL		17
	M	AE	RAR		1F
ORA	A	B7	CMA		2F
	B	B0	STC		37
	C	B1	CMC		3F
	D	B2			
	E	B3			
	H	B4			
	L	B5			
	M	B6			

Branch-Group Instructions

The instructions in this group change the normal sequence of program execution. As discussed earlier, the microprocessor fetches instructions from contiguous memory locations. The address of the next byte to be fetched from the memory is in the program counter (PC), which automatically gets incremented by one when the byte is fetched from the memory. The sequence of fetching bytes from the memory can be altered by changing the contents of the PC. For this purpose, the various instructions are: jump, call, return, restart, and move H–L to the PC.

The jump, call, and return operations may be unconditional or conditional. For conditional instructions, the operation is carried out if the specified condition is true. The conditions that may be specified are as follows:

NZ – Not zero	(Z = 0)
Z – Zero	(Z = 1)
NC – No carry	(CY = 0)
C – Carry	(CY = 1)
PO – Parity odd	(P = 0)
PE – Parity even	(P = 1)
P – Plus	(S = 0)
M – Minus	(S = 1)

The condition flags are not affected by the branch-group instructions.

Jump Instructions The jump instructions are 3-byte instructions with the first byte containing the Op code and the second and third bytes containing a 16-bit address. The lower byte of the address is the second byte and the higher byte is the third byte.

When an unconditional jump instruction (JMP addr) is executed, the address in the instruction is loaded into the program counter. The CPU will therefore fetch its next instruction from the memory at this new address. The microprocessor then continues to execute the instructions sequentially from this address. This instruction may be used to keep a program running over and over again in a continuous loop. For this, the last instruction in the program is a JMP. The address bytes of the instruction are loaded with the starting address of the program. Each time the microprocessor reaches this instruction, it sets the PC back to the starting address. The program then runs again from the starting point.

Conditional jump instructions, such as JC (jump if carry), will cause the PC to be loaded with the address bytes, specified in the instruction only if the specified condition is true ($CY = 1$), otherwise, the execution of instructions will continue in its normal sequence. The conditional jump instructions can be used to keep running around a program loop, until the specified condition is satisfied.

The various conditional jump instructions are:

JNZ	Jump if zero flag is not set	($Z = 0$)
JZ	Jump if zero flag is set	($Z = 1$)
JNC	Jump if carry flag is not set	($CY = 0$)
JC	Jump if carry flag is set	($CY = 1$)
JPO	Jump if the parity of the byte is odd	($P = 0$)
JPE	Jump if the parity of the byte is even	($P = 1$)
JP	Jump if sign flag is not set	($S = 0$)
JM	Jump if sign flag is set	($S = 1$)

Example 13.10

Write a program to transfer ten bytes of data from one area of the memory to another. Assume that the bytes are in contiguous memory locations starting from 0100 (hex) address, and are to be transferred to starting from A200 (hex) address.

Solution

The program is given below:

	LXI D, 0100H	Load the register pair D–E with 0100H.
	LXI H, A200H	Load the register pair H–L with A200H.
	MVI C, 0AH	Load the register C with decimal 10.
LOOP:	LDAX D	Load the accumulator.
	STAX H	Store the accumulator.
	INX D	Increment the D–E register pair.
	INX H	Increment the H–L register pair.
	DCR C	Decrement the register C.
	JNZ LOOP	Jump not zero to LOOP.

In the above program, register C has been used as a counter. Let LOOP be the label of the address at which the instruction LDAX D has been stored. The conditional jump instruction JNZ will test the zero flag and, as long as it is not set, the control will be transferred to the address LOOP.

Call Instructions A call instruction is used to jump to the starting address of a subprogram or subroutine in the memory. A subroutine is a short program section which does a specific task, for example, multiplying two numbers. If, in a program, it is required to multiply two numbers a number of times, the sequence of

instructions for performing the multiplication can be stored once in memory and then just called each time it is needed. This approach uses the memory space only once rather than storing this sequence of instructions in the main program again and again whenever it is needed.

A call instruction contains the starting address of the subroutine in the second byte (lower-order address) and third byte (higher-order address) of the instruction. Upon execution of the call instruction, the program counter is loaded with the address specified in the instruction.

The execution returns to the main program at the end of the subroutine by using a return (RET) instruction. This is made possible by saving the contents of the PC in a special section of RAM, the stack, during the execution of the call instruction.

Example 13.11

Let the instruction CALL *A2F7H* be stored starting at the memory location *147AH* and the contents of the stack pointer (SP) be *FFFFH*. Find the contents of the PC and relevant memory locations following the execution of the call instruction.

Solution

These are given in Table 13.15.

Table 13.15 Table for Ex. 13.11

Register/Memory location	Contents	
	Before CALL	After CALL
PC	147A H	A2F7 H
SP	FFFF H	FFFD H
147AH	CALL (op code)	CALL
147BH	F7H (lower byte)	F7H
147CH	A2H (higher byte)	A2H
FFFFH	Any byte	no change
FFFEH		14 (higher byte of next instruction)
FFFDH		7D (lower byte of next instruction)

Note that the stack grows downward in the memory.

A call can be conditional. In this case, the execution jumps to the address specified in the instruction only if the condition specified is true. The various call instructions with their meanings are given below:

CNZ	Call if	Z = 0
CZ	Call if	Z = 1
CNC	Call if	CY = 0
CC	Call if	CY = 1
CPO	Call if	P = 0
CPE	Call if	P = 1
CP	Call if	S = 0
CM	Call if	S = 1

Return Instruction The return instruction (RET) at the end of a subroutine transfers the program execution back to the next instruction after the call instruction in the main program. The return instruction does not contain the address and the program counter will be loaded with the top two bytes of the stack following the execution of this instruction. Now, the execution of the main program sequence will continue.

Return also can be unconditional or conditional. The various conditional return instructions with their meanings are given below:

RNZ	Return if	$Z = 0$
RZ	Return if	$Z = 1$
RNC	Return if	$CY = 0$
RC	Return if	$CY = 1$
RPO	Return if	$P = 0$
RPE	Return if	$P = 1$
RP	Return if	$S = 0$
RM	Return if	$S = 1$

The subroutine can be simple or nested. The program flow in a simple subroutine is shown in Fig. 13.17a and that in a nested subroutine, in Fig. 13.17b. It is clear from Fig. 13.17b that a call instruction in a subroutine can call another sub-routine.

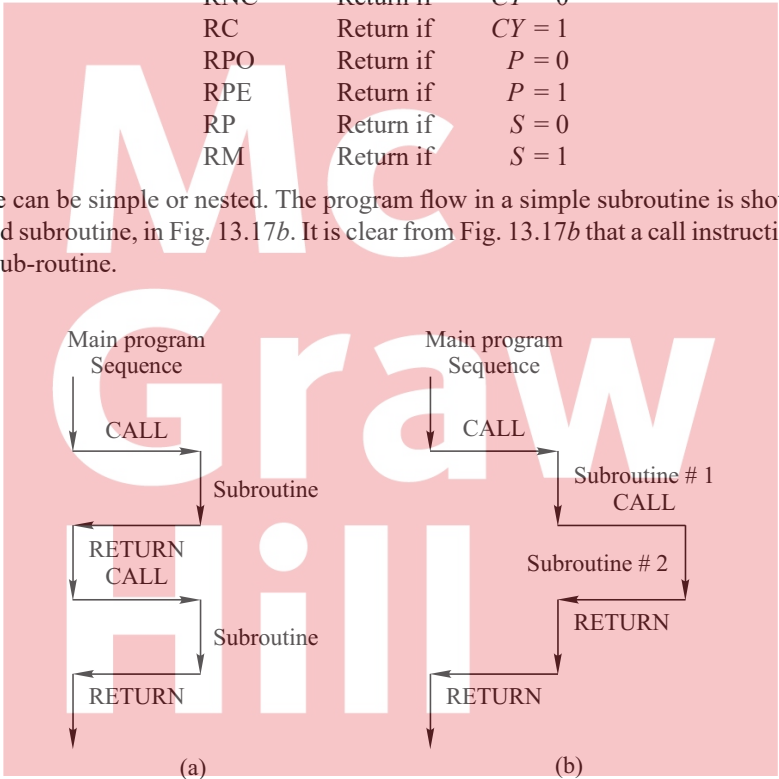


Fig. 13.17 Program Flow with Call Instructions: (a) Simple Subroutines (b) Nested Subroutines

The microprocessor executes the program until it encounters a call instruction, then goes to subroutine # 1. It executes subroutine # 1 until it encounters another call instruction, then it goes to subroutine # 2. When subroutine # 2 is completed, the return instruction at the end of this subroutine transfers the control to subroutine # 1 to the instruction, next to the call instruction. When subroutine # 1 is completed, the return instruction at the end of this subroutine transfers the control to the instruction next to the call instruction in the main program.

Restart Instruction Restart instruction (RST n) is a special purpose CALL instruction, designed primarily for use with interrupts. It pushes the contents of the program counter onto the stack to provide a return address and then jumps to one of the eight addresses, determined by the value of n (a three-bit binary

code). It is a single-byte instruction and seldom appears as an instruction in the program. More often, the peripheral devices seeking interrupt service pass this one-byte instruction to the processor. The eight restart instructions, and the corresponding addresses to which the control is transferred following their execution are given below:

Instruction		Address to which control is transferred
RST	0	0000H
RST	1	0008H
RST	2	0010H
RST	3	0018H
RST	4	0020H
RST	5	0028H
RST	6	0030H
RST	7	0038H

PCHL Instruction This instruction moves the contents of the H–L pair to the program counter. Following the execution of this instruction, the program execution jumps to the address which was stored in the H–L register pair.

Condition flags are not affected by any instruction in this group. All the instructions of branch group, in terms of their mnemonics and operands along with their Op code (hex), are given in Table 13.16.

Table 13.16 Branch group instructions

Instruction		Op code (hex)	Instruction		Op code (hex)
Mnemonic	Operand		Mnemonic	Operand	
JMP	addr	C3	RET		C9
JNZ	addr	C2	RNZ		C0
JZ	addr	CA	RZ		C8
JNC	addr	D2	RNC		D0
JC	addr	DA	RC		D8
JPO	addr	E2	RPO		E0
JPE	addr	EA	RPE		E8
JP	addr	F2	RP		F0
JM	addr	FA	RM		F8
PCHL		E9			
CALL	addr	CD		0	C7
CNZ	addr	C4		1	CF
CZ	addr	CC		2	D7
CNC	addr	D4	RST	3	DF
CC	addr	DC		4	E7
CPO	addr	E4		5	EF
CPE	addr	EC		6	F7
CP	addr	F4		7	FF
CM	addr	FC			

Stack, I/O and Machine Control Group Instructions

This group of instructions manipulates the stack, performs I/O operations, and alters internal control flags.

Stack Operations A small portion of the RAM is set aside to store return addresses for subroutine calls and restart instructions. It can also be used to store the contents of the register pairs (B–C, D–E, or H–L) or the program status word (PSW), so that they will not be changed or destroyed by a subroutine. For example, data stored in the register B in the main program will be altered if the register B is also used in the subroutine. The PUSH instruction is used to push the contents of B–C, D–E, H–L, or PSW to the stack.

Data has to be stored in the stack on a last-in-first-out (LIFO) basis. The stack-pointer register (SP) holds the address of the stack top, which is the last occupied location of the stack. The stack grows downward in memory from the starting address, which means that the SP decrements after each byte is pushed onto the stack. It increments after each byte is popped off the stack, following the execution of POP instruction. The popped-off bytes can be stored in the B–C, D–E, H–L register pairs or in the accumulator and flags (PSW). The organisation of a PSW is shown in Fig. 13.11.

Example 13.12

Assume that the stack begins at the memory location FFFFH and the bottom-most two locations are full, as shown in Fig. 13.18a. Determine the contents of the stack when (a) Push H instruction is executed and then (b) pop h instruction is executed.

Solution

- (a) Now, if the instruction PUSH H is executed, the contents of the register pair H–L will be pushed in the stack as shown in Fig. 13.18b.

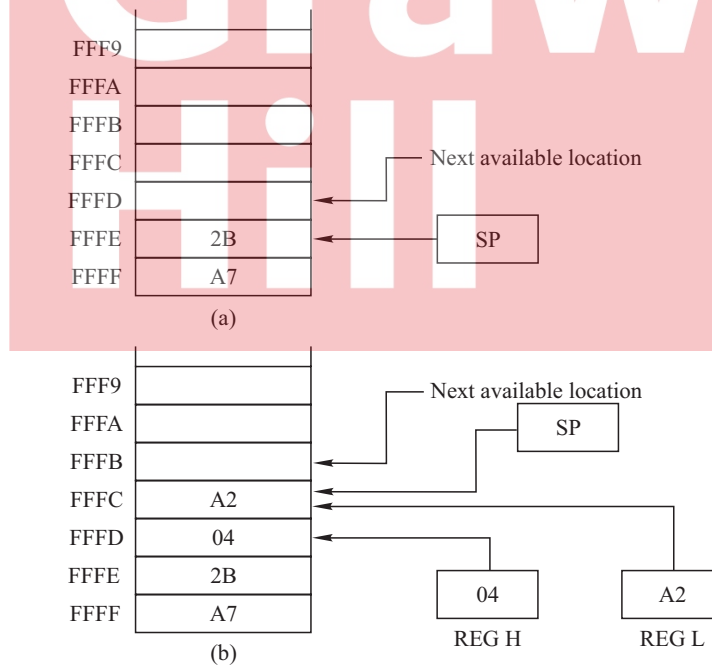


Fig. 13.18 Stack (a) before, and (b) after the PUSH Instruction

- (b) Now, if POP H instruction is executed, the stack will be as shown in Fig. 13.18a. The byte from the memory location *FFFC* will be transferred to the lower-order register (L) and the byte from the next location *FFFD* will be transferred to the higher-order register (H). The SP gets incremented by two and it now contains *FFFE*.

There are two more instructions involving the stack. These are XTHL and SPHL. XTHL exchanges two bytes of data from the top of the stack with the two bytes in the H and L registers. The contents of the L register is exchanged with the contents of the memory location whose address is stored in SP, and the contents of the H register is exchanged with the contents of the memory location whose address is one more than the contents of the SP.

The instruction SPHL moves the contents of the registers H and L into the stack-pointer register.

All instructions, except POP PSW, involving stack operation do not affect the condition flags.

I/O Instructions The IN instruction inputs a byte of data from an input port, specified in the instruction, to the accumulator. The OUT instruction outputs a byte of data from the accumulator to the port specified in the instruction. These are two-byte instructions, the second byte being the port address. Since the port address is 8-bit, the microprocessor can address upto 256 input and output devices. The condition flags are not affected by these instructions.

Control Instructions

These are single-byte instructions without any operand. The various control instructions are explained below.

Interrupt Enable (EI) The interrupt system is enabled following the execution of the next instruction after EI.

Disable Interrupt (DI) The interrupt system is disabled immediately, following the execution of the DI instruction.

Halt (HLT) The halt instruction stops the processor. The registers and flags are unaffected.

There are two ways in which the microprocessor comes out of the halt state. These are: hold and interrupt. If the signal on the hold line is a logical 1 then the processor leaves the halt state as long as the hold is 1. As soon as the hold line goes LOW, it again enters the halt state.

The processor permanently comes out of the halt state, if interrupted, provided the interrupt system is enabled before the HLT instruction.

No Operation (NOP) During the execution of this instruction, the processor does nothing. The registers and flags are unaffected. The NOP instructions are used in a program for the processor to wait for a certain time. Also, they are put in to be substituted by other instructions when some instructions are missed due to some mistake in programming, to avoid shifting of the entire program in the memory.

Set Interrupt Mask (SIM) The SIM instruction is used to perform the following functions:

1. Set the interrupt mask (disable) for the hardware interrupts RST 5.5, RST 6.5, and RST 7.5.
2. Reset RST 7.5 interrupt.
3. Output bit 7 of the accumulator to the serial-output data latch.

The desired bit-configuration is to be loaded into the accumulator before executing the SIM instruction. Figure 13.19 shows the interpretation of each bit of the accumulator by the SIM instruction. Bit 3 is used for enabling the interrupt mask and resetting the operation of RST 7.5 FLIP-FLOP. Unless it is a 1, bits 0, 1, 2, and 4 will have no effect.

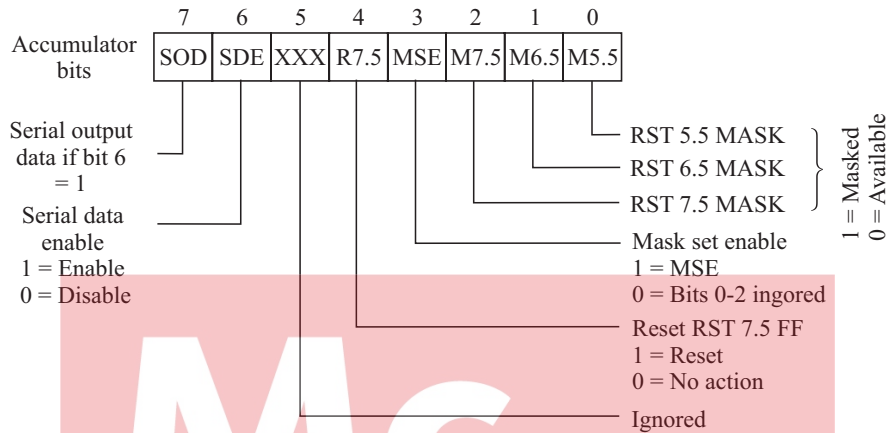


Fig. 13.19 Bit format of SIM instruction

If it is desired to send a serial output bit, without affecting the interrupt mask, the bit 3 is made 0. If bit 3 is a 1 and, for example, bit 1 is a 1, then the RST 6.5 interrupt input will not respond to an interrupt signal. This means that RST 6.5 interrupt is masked.

The DI instruction overrides the SIM instruction. This means, whether masked or not, RST 5.5, RST 6.5, and RST 7.5 interrupts are all disabled when the DI instruction is in effect.

The RST 7.5 is sensed via a processor FLIP-FLOP, that is, set when a peripheral device issues a pulse with a rising edge. This FLIP-FLOP is not affected by the setting of the interrupt mask or the DI instruction and, therefore, can be set at any time. However, the interrupt cannot be serviced when RST 7.5 is masked or the DI instruction is in effect.

If bit 6 is set to 1, the serial-output data function is enabled. The processor latches the accumulator bit 7 into the SOD output where it can be accessed by a peripheral device. If bit 6 is 0, then bit 7 is ignored. A hardware reset will set all RST masks, reset/disable all interrupts, and sets the SOD latch to zero.

Example 13.13

Assume that the accumulator contains 00011100 when SIM instruction is executed. The SIM instruction resets the RST 7.5 interrupt. If an RST 7.5 interrupt is pending when this SIM instruction is executed, it is overridden without being serviced. Also, any subsequent RST 7.5 interrupt is masked and cannot be serviced until the interrupt mask is serviced.

Example 13.14

Assume that the accumulator contains 11001111. The SIM instruction masks out the RST 5.5, RST 6.5, and RST 7.5 level interrupts and loads the accumulator bit 7 into the SOD latch.

Read Interrupt Mask (RIM) The RIM instruction loads the accumulator indicating the following:

1. current setting of the interrupt mask,
2. setting of the interrupt flag,
3. pending interrupts, and
4. one bit of serial input data, if any.

Figure 13.20 shows the interpretation of each bit of the accumulator following the execution of the RIM instruction.

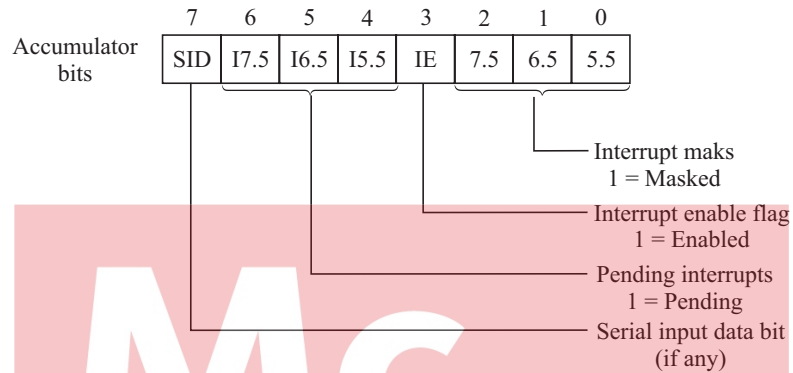


Fig. 13.20 Bit Format of RIM Instruction

The mask and pending flags refer to the RST 5.5, RST 6.5, and RST 7.5 interrupts. The IE flags refer to the entire interrupt system. Bit 7 contains data brought into the accumulator from the SID input.

All the instructions of the stack, I/O, and machine control group in terms of their mnemonics and operands along with their Op codes (hex) are given in Table 13.17.

Table 13.17 Stack, I/O, and machine control group instructions

Instruction		Op code	Instruction		Op code
Mnemonic	Operand	(hex)	Mnemonic	Operand	(hex)
PUSH	B	C5	OUT	byte	D3
	D	D5	IN	byte	DB
	H	E5	DI		F3
	PSW	F5	EI		FB
POP	B	C1	NOP		00
	D	D1	HLT		76
	H	E1	RIM		20
	PSW	F1	SIM		30
XTHL		E3			
SPHL		F9			

13.11 THE 8086 MICROPROCESSOR

The 16-bit 8086 microprocessor was introduced by Intel in 1978. It was fabricated using silicon-gate H-MOS process, containing 29,000 transistors on a chip of about 225 mils square. The high performance MOS technology resulted in a 5 MHz clock rate, making it faster than any μ P chip available at that time.

The 8086 μ P has the attributes of both 8- and 16-bit μ Ps. It has capabilities not available in 8-bit microprocessors, such as, 16-bit arithmetic, signed 8- and 16-bit arithmetic, including multiply and divide, efficient interruptible byte-string operations, and improved bit manipulation. It also includes operations

such as reentrant code, position-independent code, and dynamically relocatable programs. The 8086 μ P can directly address upto 1 megabyte (MB) of memory and can support multiple-processor configurations.

13.11.1 The 8086 Architecture

Figure 13.21 shows internal block diagram of the 8086 microprocessor. It is divided into two units, the *bus interface unit* (BIU), and the *execution unit* (EU). Since the operation in this microprocessor is divided into two units, its speed of operation increases.

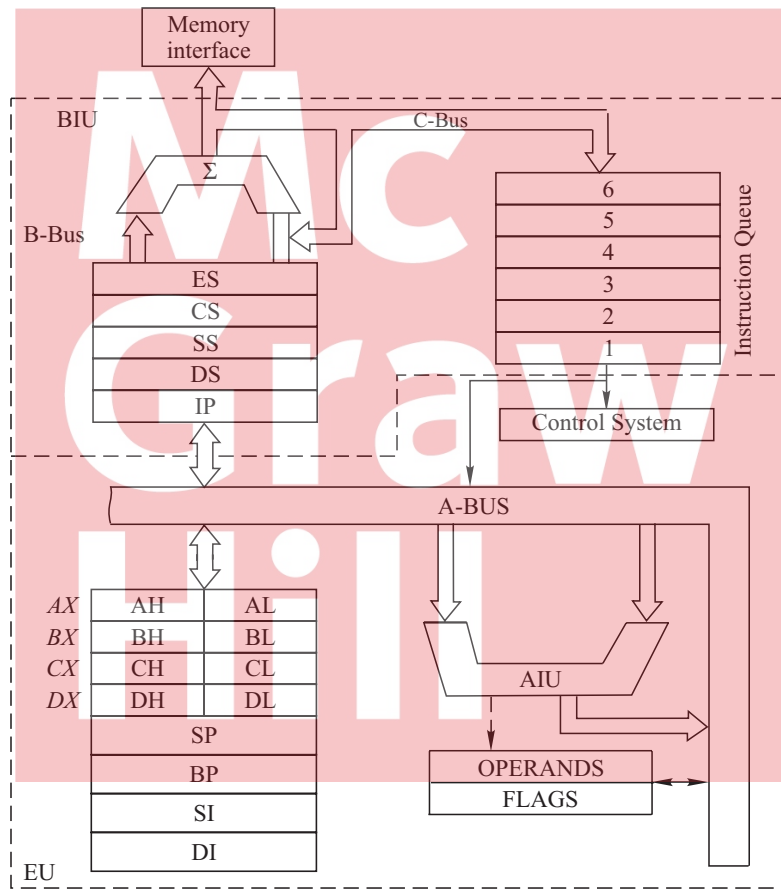


Fig. 13.21 Internal Block Diagram of Intel 8086 Microprocessor

The Bus Interface Unit The bus interface unit handles all transfers of data and addresses on the buses for fetching instructions from the memory, read and write operations for the memory and the I/O devices. It has a queue of six bytes for holding prefetched instruction bytes operating in the FIFO configuration. Due to this facility, the BIU can fetch instruction bytes while the EU is decoding or executing an instruction, which does not require use of the buses.

The 16-bit data bus is time multiplexed with the sixteen bits of address for both memory and *I/O* ($AD_{15}-AD_0$). The width of the address bus for memory referencing is 20-bit, consisting of $A_{19}-A_{16}$ and $AD_{15}-AD_0$, which can address $2^{20} = 1,048,576 = 1$ Megabyte (MB) of memory. The memory is organised in the form of 64K byte (KB) segments. The method of addressing memory is given below.

The BIU contains four 16-bit segment registers: The code segment (CS) register, the stack segment (SS) register, the extra segment (ES) register, and the data segment (DS) register. These segment registers are loaded with the most-significant 16 bits of the starting addresses of the corresponding memory segments which the 8086 is working with at a particular time. The BIU inserts zeros for the lowest four bits of the 20-bit starting address for a segment. For example, if the code segment register contains *73AEH*, the code segment will start at *73AE0H*.

If the first byte of the word is at an even address, 8086 reads the entire word in one operation. On the other hand, if the first byte of the word is at an odd address, 8086 will read the first byte in one operation and the second byte, in another operation. For an even starting address of the word, the data byte with even address is transferred on the D_7-D_0 bus (lower byte), while odd-addressed data byte is transferred on the $D_{15}-D_8$ bus lines (higher byte). When the 8086 μP is reset, it always begins execution at the location *FFFF0H*, where the jump instruction must be stored in memory.

Instruction-pointer (IP) register is a 16-bit register which holds the address of the next byte of the instruction code within the code segment of the memory. The value contained in the IP register is added to the 20-bit base address formed using CS register and it generates 20-bit physical address of the code byte. For example, if the CS contains *124AH* and IP contains *0728H*, the 20-bit physical address of the code byte generated will be:

$$\begin{array}{r} 1\ 2\ 4\ A\ 0H \\ +\ 0\ 7\ 2\ 8H \\ \hline 1\ 2\ B\ C\ 8H \end{array}$$

The Execution Unit The execution unit (EU) decodes the instructions fetched from the memory and executes the instructions. It tells the BIU about the locations from where to fetch instructions or data at any time. It contains control circuitry, 16-bit ALU, general purpose and index registers, flags, etc.

It contains eight 8-bit general purpose registers, which can also be used in pairs to form four 16-bit registers. The 16-bit registers are labelled as AX (AH–AL), BX (BH–BL), CX (CH–CL), and DX (DH–DL) registers. The AX register is called the accumulator.

The stack-segment register (SS) contains the 16-bit (upper) part of the starting address of the stack and the stack pointer register (SP) contains the 16-bit offset value. The physical address in the stack segment is obtained by shifting the contents of the SS register towards left by four bit positions, adding four zeros in the least-significant four bit positions and adding the 16-bit contents of the SP in this 20-bit number.

The base pointer (BP), source index (SI) and destination index (DI) registers are used to hold the 16-bit offset of a data word in data-segment (DS) register or extra-segment (ES) register. Various addressing modes use these registers in different ways.

The 8086 μP contains nine flags. Six of them are conditional flags which indicate some condition produced by the execution of an instruction and the remaining three are used to control certain operations of the processor. The conditional flags are:

1. Carry flag: *CF*
2. Parity flag: *PF*
3. Auxiliary carry flag: *AF*
4. Zero flag: *ZF*

5. Sign flag: SF
6. Overflow flag: OF

The control flags are:

1. Trap flag (TF): Used for single stepping through a program.
2. Interrupt flag (IF): Used to allow/prohibit the interruption of a program.
3. Direction flag (DF): Used with string instructions.

13.12 PROGRAMMING LANGUAGES

As is evident from the way microprocessors operate and information is stored and transferred between various subsystems, it understands only the binary signals (0 and 1). Therefore, the program is to be written in binary form which is referred to as the *machine language*. Machine language alone is directly executable by a microprocessor. The microprocessor decides whether the binary number corresponds to an instruction, part of an address, or simply a piece of data, according to what it expects at a particular point in the instruction cycle. Therefore, the binary codes corresponding to the sequence of instructions in a program must be stored in proper sequence, otherwise the information will be misinterpreted by the microprocessor, resulting in serious errors.

For programming in machine language, the programmer needs a table of binary instruction codes. Also, he/she has to keep track of the contents of memory locations, registers, and flags. Of course, he/she should be thoroughly familiar with the details of the microprocessor.

Consider the program of Ex. 13.3 written in machine language:

```
00000001
01110101
00100100
00010001
10010100
00110111
00001010
01101111
00011010
00000010
01111101
00010010
```

In this, it is difficult to determine whether a particular byte corresponds to an instruction, a data, or an address. Also, working with binary numbers is not convenient and we are liable to commit mistakes.

The machine-language program listing can be simplified somewhat by converting the binary codes to octal or hexadecimal format. Although such a shorthand notation alleviates some of the drudgery of writing the codes, it does not improve the understandability of the program. Also, the program has to be converted back to the machine language by some means before it is loaded into memory.

A better alternative to machine language is *assembly language*. In this, names are assigned to instructions, registers, data, and memory locations. These names are usually the abbreviations of the names or descriptions of the instructions (mnemonic), addresses, or data and are used to aid the designer's memory.

The assembly-language program is much easier to write, modify, understand and interpret than the corresponding machine-language program.

The program written in assembly language has to be translated into machine language for loading into the memory. This translation process is referred to as *assembly*. While short programs can be hand-assembled (manual translation) without much difficulty, for long programs, another program, referred to as *assembler*, is used to perform automatic translation.

Example 13.15

Consider the following program written in assembly language:

```
LXI H, BUFFER
MVI C, 0AH
LOOP: IN DATA
      MOV M,A
      INX H
      DCR C
      JNZ LOOP
```

Here, BUFFER has been used as the symbolic name for 16-bit data. LOOP is the symbolic address of the memory location at which the Op code of IN is stored and DATA is the symbolic address of the input port.

The ease of programming is further enhanced by using high-level languages such as, PASCAL, FORTRAN, BASIC, etc. A single instruction of high-level language is equivalent to several machine-language instructions. High-level language programs must be translated to machine language for microprocessor execution. This is done by using a *compiler* program.

High-level languages have some drawbacks which make them unsuitable for some applications. The main drawbacks are:

1. They require more memory than machine- or assembly-language program.
2. The execution time required for high-level language programs is usually more than that required for machine- or assembly-language program.
3. They require a compiler program in the memory which is fairly expensive and need large memory space.

SUMMARY

The vast topic of microprocessors has been introduced in such a way so as to enable a novice to understand clearly the basic concepts involved. Since the introduction of the first microprocessor in 1971, a variety of microprocessors have been evolved which are commercially available. Although their principle of operation is more or less similar, they have wide differences as far as their architecture and instruction sets are concerned. It is not advisable for a novice to try to understand the details of different microprocessors simultaneously. The best course is to understand one microprocessor completely and then, without much difficulty, the other microprocessors can be learnt.

With this in view, the most popular 8-bit microprocessor, Intel's 8085A, has been selected for detailed discussion. Brief discussion of Intel's 16-bit microprocessor has also been included. The reader should write a number of programs to fully understand the options and the flexibilities available. It is not possible to cover all aspects of hardware and software of microprocessor-based systems in just one chapter, nevertheless, the basic concepts have been clearly discussed which will enable a beginner to initiate the study of microprocessor without any difficulty.

GLOSSARY

Absolute address A memory address whose exact value is specified, and is not required to be computed or inferred from any other information.

Accumulator A register in the CPU of a computer or in a microprocessor that is used for most of the arithmetic and logic operations.

Addressing modes The methods used for specifying the address of hardware registers and/or memory to be used in an instruction.

Address space The number of memory locations and or I/O ports that a μP is capable of addressing.

Assembler A computer program that converts assembly language program into executable binary codes, i.e. machine language.

Assembler directive An instruction to the assembler. It is not translated into machine code.

Assembling The process of translating an assembly language program into machine language.

Assembly language It is a low-level programming language in which mnemonic instruction codes, labels, and names for data and addresses are used to refer directly to their binary equivalents.

Borrow flag A status bit indicating whether a borrow bit is produced or not as a result of a subtraction operation.

Branching instruction A μP instruction which causes transfer to a nonsuccessive memory address for the next instruction.

Bus, time-multiplexed A bus used for different tasks at different timings in a system. For example, a bus may be used to transmit address in one time slot and data in another time slot.

Call instruction An instruction that transfers control from one program segment to another program segment.

Carry flag A status bit indicating whether a carry bit is produced or not as a result of certain operation.

Compiler A computer program that translates commands in high-level language into machine code.

Computer architecture The specific interconnections of registers and logic blocks that define the design of a CPU or μP .

Conditional branching instruction A μP (or computer) instruction which causes transfer to a nonsuccessive memory address for the next instruction only when a specific condition is satisfied.

CPU (Central processing unit) The subsystem of a computer which is responsible for performing various arithmetic and logic operations and controls the overall operation of the computer. The microprocessor performs all these operations and hence it acts as CPU of a computer.

Data transfer instruction An instruction that causes data to be moved between internal registers of μP , or between μP and memory or μP and I/O devices.

Flag A single bit register in the μP that indicates certain conditions which may be used for taking decisions.

General purpose register A register that can be used for temporary data storage. It is also known as scratch pad register.

Hand assembly The process of translating an assembly language program into a machine language program manually rather than by an assembler.

Hardware interrupt An interrupt caused by an external device connected to one of the interrupt pins of a μP .

High-level language A programming language in which the statements represent procedures rather than single instructions, for example, PASCAL, BASIC, C++ etc.

Input port A circuit that is used to interface an input device with the μP .

Instruction A group of bits that tells the μP what to do. It is a part of the instruction set of the μP that is specified by its manufacturers.

Instruction code Same as the op code.

Instruction cycle The process of fetching, decoding, and executing an instruction.

Instruction cycle time The time required to fetch, decode, and execute an instruction.

Instruction decoder A logic circuit used to interpret the instruction in μP .

Instruction execution Carrying out the tasks specified in the instruction after it has been fetched from the memory.

Instruction fetch Getting the instruction from the memory to the μP .

Instruction mnemonic A short and convenient name used to represent the operation in an instruction.

Instruction set The instructions which are allowed in a μP . It is specified by its manufacturer.

Interrupt A μP input that causes the temporary suspension of the normal sequence of operations and transfers control to a special routine meant to service the interrupt.

I/O address space The number of I/O devices which a μP is capable of addressing as standard I/O.

I/O device The device used to establish link between the μP and the outside world.

I/O mapped I/O I/O devices addressed as I/O rather than as memory locations. It is same as the standard I/O.

I/O port The circuit used to interface I/O devices with the μP . The data from μP or I/O device comes into and goes out to I/O device or μP respectively through the I/O port.

Jump instruction An instruction that causes the control to be transferred to a new location, whose value gets loaded in the program counter.

Logical shift A shift operation in which zeros are placed in empty bit locations.

Machine language The programming language that a computer or μP can directly understand. It is in the form of 1s and 0s.

Memory address space The number of memory locations which a μP is capable of addressing.

Memory mapped I/O I/O devices connected as memory locations.

Microcomputer A computer which uses a μP as its CPU.

Microprocessor architecture Same as the computer architecture.

Mnemonic Symbolic names or abbreviations for instructions, data, registers, addresses etc.

Op code (Operation code) A binary number that serves as a code for a particular operation in a μP .

Operand The quantity that is operated on.

Output port A circuit that is used to interface an output device with the μP .

Peripheral Same as the I/O device.

Program A sequence of instructions that direct a computer or a μP based system to perform a specific task.

Program counter (PC) A register in the μP that contains the address of the next instruction byte to be fetched from the memory.

Programming (of computer or μP) To program a computer or a μP based system.

Scratch-pad memory Memory locations or registers that are used to store temporary or intermediate results.

Stack A set of contiguous memory locations used in LIFO configuration.

Stack pointer (SP) A register in μP whose contents point to the top of the stack.

Standard I/O Same as I/O mapped I/O.

Unidirectional bus A bus that can transmit signals only in one direction.

Word size (or length) The number of bits that a μP or a computer is capable of processing at a time. It is usually same as the data bus width.

REVIEW QUESTIONS

- 13.1 The word size of 8085 mP is _____.
- 13.2 The 8085 mP has 8-bit address bus and 8-bit AD bus, it can address _____ memory.
- 13.3 A mP with an address bus width of _____ can address 1M bytes of memory.
- 13.4 I/O devices are interfaced with mP using _____.
- 13.5 Stack resides in _____.
- 13.6 The stack pointer gets _____ after PUSH instruction.
- 13.7 There are _____ number of flags in 8085 mP.
- 13.8 The I/O address space of 8085 mP is _____.
- 13.9 NOP instruction of 8085 mP performs _____ operation.
- 13.10 The number of bytes of memory locations available for handling RST interrupts in 8085 mP based system is _____.
- 13.11 For an 8085 mP operating at 3.125 MHz will have a crystal of frequency _____.
- 13.12 The contents of 8085 mP's PC after resetting the mP will be _____.
- 13.13 The 8086 mP can address _____ of memory.
- 13.14 In an 8086 mP based system if $\text{SS} = 1234\text{H}$ and $\text{SP} = 0100\text{H}$, the current address of the stack is _____.
- 13.15 In an 8086 mP based system if $\text{IP} = 1000\text{H}$ and $\text{CS} = 3000\text{H}$, the physical address of the next instruction will be _____.

PROBLEMS

13.1 What is the size of the memory address space for the following microprocessors?

Microprocessor	Data-bus width	Address-bus width
8080A	8-bit	16-bit
6800	8-bit	16-bit
8086	16-bit	20-bit
9900	16-bit	16-bit
Z8000	16-bit	23-bit

- 13.2 How many different instructions are possible in an 8-bit microprocessor? Compare this number with the number of instructions of an 8085A.
- 13.3 For a particular 8085A microprocessor-based system, it is required to use 2K bytes of EPROM and 4K bytes of RAM. Assume 2142 RAMs and 2716 EPROMs are available.
- Find the number of RAM and EPROM chips required.
 - Connect the memory chips to the microprocessor and show all the connections clearly. You can make suitable assumptions wherever required.
 - Find the addresses of EPROMs and RAMs and express them in hex notation.
- 13.4 It is desired to clear the accumulator of 8085A. Explain the possible instructions for this purpose.
- 13.5 Write a program for an 8085A to complement the bytes stored in locations 0F00H through 0FFFH and store them in locations starting from 1F00H.
- 13.6 Write a program for an 8085A to find the larger of the two numbers stored in locations 0A00H and 0A01H and store it in memory location 0A02H.
- 13.7 Write a program to clear the memory location 01A0H in an 8085A microprocessor system.
- 13.8 Write a program to find the smallest element in a block of 8-bit unsigned binary data, whose number is stored in memory location A001H, and the data are stored in memory locations beginning from A002H. Store the smallest number in the memory location FF00H. The mP used is 8085A.
- 13.9 It is desired to mask (set to 0) the most-significant four bits of a byte in the accumulator. Suggest an instruction to accomplish this in an 8085A mP system.
- 13.10 An 8085A program is given below. Find the task performed by this program.

```

START:      LXI H, BUFR
            MVI C, 0BH

LOOP:      DCR C
            JZ FINIS
            IN DATA
  
```

```

MOV M, A
INX H
JMP LOOP
FINIS:  MOV B, A

```

- 13.11** Find the memory requirement of the program of Ex. 13.3.
- 13.12** Assume that the accumulator contains 08 and a register B contains 93 (both in BCD format). Find the contents of the accumulator after the ADD B instruction is executed. Is the result a valid BCD number? If not, what should be done to obtain the result in BCD format? Explain clearly.
- 13.13** Suggest a suitable system to display a BCD number entered through a keyboard to the mP system on a 7-segment LED. Using an 8085A microprocessor, write the program. Make the necessary assumptions.
- 13.14** The first four instructions of a subroutine are:
- ```

PUSH B
PUSH D
PUSH H
PUSH PSW

```
- What will be the last six instructions of the subroutine? Explain clearly.
- 13.15** Write an 8085A program to convert a decimal digit in memory location 00F0 to an ASCII character and store it in memory location 00F1H. If memory location 00F0H does not contain a decimal digit, an ASCII question mark should be placed in memory location 00F1H. See Table 2.10 for decimal to ASCII conversion.
- 13.16** What is the address-bus width of the following microprocessors?
- ```

8086
80186
80286
80386 SL
80386 DX
80486 DX
Pentium

```
- 13.17** How many 16-bit general-purpose registers are available in the 8086 mP?
- 13.18** 16-bit Segment registers are used to address a 64k byte block of memory in an 8086 mP, whereas the address-bus width is 20 bits. Is it possible? Explain.
- 13.19** If IP = 1A00H and CS = 2000H, find the address of the next instruction to be fetched.
- 13.20** If SS = 2400 H and SP = A000 H, find the current address of the stack.