A list of instructions that performs a task is called a ***program***. Usually the program is stored in the memory. The processor then fetches the instructions that make up the program from the memory, one after another, and performs the desired operations. The computer is completely controlled by the ***stored program***, except for possible external interruption by an operator or by I/O devices connected to the machine.

***Data*** are numbers and encoded characters that are used as operands by the instructions. The term data, however, is often used to mean any digital information. Within this definition of data, an entire program (that is, a list of instructions) may be considered as data if it is to be processed by another program. An example of this is the task of *compiling* a high-level language *source program* into a list of machine instructions constituting a machine language program, called the *object program*. The source program is the input data to the *compiler* program which translates the source program into a machine language program.

Information handled by a computer must be encoded in a suitable format. Most present-day hardware employs digital circuits that have only two stable states, ON and OFF (see Appendix A). Each number, character, or instruction is encoded as a string of binary digits called *bits*, each having one of two possible values, 0 or 1. Numbers are usually represented in positional binary notation, as discussed in detail in Chapters 2 and 6. Occasionally, the ***binary-coded decimal*** (BCD) format is employed, in which each decimal digit is encoded by four bits.

Alphanumeric characters are also expressed in terms of binary codes. Several coding schemes have been developed. Two of the most widely used schemes are ASCII (American Standard Code for Information Interchange), in which each character is represented as a 7-bit code, and EBCDIC (Extended Binary-Coded Decimal Interchange Code), in which eight bits are used to denote a character. A more detailed description of binary notation and coding schemes is given in Appendix E.

### 1.2.1 INPUT UNIT

Computers accept coded information through input units, which read the data. The most well-known input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Many other kinds of input devices are available, including joysticks, trackballs, and mouses. These are often used as graphic input devices in conjunction with displays. Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing. Detailed discussion of input devices and their operation is found in Chapter 10.

### 1.2.2 MEMORY UNIT

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

*Primary storage* is a fast memory that operates at electronic speeds. Programs must be stored in the memory while they are being executed. The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written as individual cells but instead are processed in groups of fixed size called *words*. The memory is organized so that the contents of one word, containing *n* bits, can be stored or retrieved in one basic operation.

To provide easy access to any word in the memory, a distinct *address* is associated with each word location. Addresses are numbers that identify successive locations. A given word is accessed by specifying its address and issuing a control command that starts the storage or retrieval process.

The number of bits in each word is often referred to as the *word length* of the computer. Typical word lengths range from 16 to 64 bits. The capacity of the memory is one factor that characterizes the size of a computer. Small machines typically have only a few tens of millions of words, whereas medium and large machines normally have many tens or hundreds of millions of words. Data are usually processed within a machine in units of words, multiples of words, or parts of words. When the memory is accessed, usually only one word of data is read or written.

Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of the processor. It is essential to be able to access any word location in the memory as quickly as possible. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called *random-access memory* (RAM). The time required to access one word is called the *memory access time*. This time is fixed, independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for modern RAM units. The memory of a computer is normally implemented as a *memory hierarchy* of three or four levels of semiconductor RAM units with different speeds and sizes. The small, fast, RAM units are called *caches*. They are tightly coupled with the processor and are often contained on the same integrated circuit chip to achieve high performance. The largest and slowest unit is referred to as the *main memory*. We will give a brief description of how information is accessed in the memory hierarchy later in the chapter. Chapter 5 discusses the operational and performance aspects of the computer memory in detail.

Although primary storage is essential, it tends to be expensive. Thus additional, cheaper, *secondary storage* is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. A wide selection of secondary storage devices is available, including *magnetic disks* and *tapes* and *optical disks* (CD-ROMs). These devices are also described in Chapter 5.

### 1.2.3 Arithmetic and Logic Unit

Most computer operations are executed in the *arithmetic and logic unit* (ALU) of the processor. Consider a typical example: Suppose two numbers located in the memory are to be added. They are brought into the processor, and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

Any other arithmetic or logic operation, for example, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. When operands are brought into the processor, they are stored in high-speed storage elements called *registers*. Each register can store one word of data. Access times to registers are somewhat faster than access times to the fastest cache unit in the memory hierarchy.

The control and the arithmetic and logic units are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as keyboards, displays, magnetic and optical disks, sensors, and mechanical controllers.

### 1.2.4 OUTPUT UNIT

The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world. The most familiar example of such a device is a *printer*. Printers employ mechanical impact heads, ink jet streams, or photocopying techniques, as in laser printers, to perform the printing. It is possible to produce printers capable of printing as many as 10,000 lines per minute. This is a tremendous speed for a mechanical device but is still very slow compared to the electronic speed of a processor unit.

Some units, such as graphic displays, provide both an output function and an input function. The dual role of such units is the reason for using the single name I/O unit in many cases.

### 1.2.5 CONTROL UNIT

The memory, arithmetic and logic, and input and output units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the task of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states.

I/O transfers, consisting of input and output operations, are controlled by the instructions of I/O programs that identify the devices involved and the information to be transferred. However, the actual *timing signals* that govern the transfers are generated by the control circuits. Timing signals are signals that determine when a given action is to take place. Data transfers between the processor and the memory are also controlled by the control unit through timing signals. It is reasonable to think of a control unit as a well-defined, physically separate unit that interacts with other parts of the machine. In practice, however, this is seldom the case. Much of the control circuitry is physically distributed throughout the machine. A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

The operation of a computer can be summarized as follows:

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.

- Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities inside the machine are directed by the control unit.

## 1.3 BASIC OPERATIONAL CONCEPTS

In Section 1.2, we stated that the activity in a computer is governed by instructions. To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands are also stored in the memory. A typical instruction may be

```
Add   LOCA,R0
```

This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum into register R0. The original contents of location LOCA are preserved, whereas those of R0 are overwritten. This instruction requires the performance of several steps. First, the instruction is fetched from the memory into the processor. Next, the operand at LOCA is fetched and added to the contents of R0. Finally, the resulting sum is stored in register R0.

The preceding Add instruction combines a memory access operation with an ALU operation. In many modern computers, these two types of operations are performed by separate instructions for performance reasons that are explained in Chapter 8. The effect of the above instruction can be realized by the two-instruction sequence

```
Load  LOCA,R1
Add   R1,R0
```

The first of these instructions transfers the contents of memory location LOCA into processor register R1, and the second instruction adds the contents of registers R1 and R0 and places the sum into R0. Note that this destroys the former contents of register R1 as well as those of R0, whereas the original contents of memory location LOCA are preserved.

Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.

Figure 1.2 shows how the memory and the processor can be connected. It also shows a few essential operational details of the processor that have not been discussed yet. The interconnection pattern for these components is not shown explicitly since here we discuss only their functional characteristics. Chapter 7 describes the details of the interconnection as part of processor design.

In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes. The *instruction register* (IR) holds the instruction that is currently being executed. Its output is available to the control circuits,

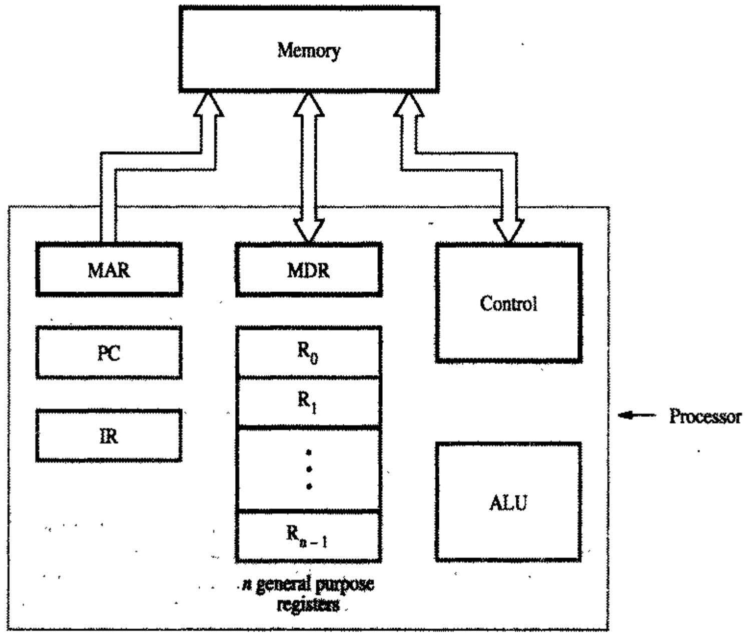


**Figure 1.2** Connections between the processor and the memory.

which generate the timing signals that control the various processing elements involved in executing the instruction. The *program counter* (PC) is another specialized register. It keeps track of the execution of a program. It contains the memory address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. It is customary to say that the PC *points* to the next instruction that is to be fetched from the memory. Besides the IR and PC, Figure 1.2 shows *n general-purpose registers*, $R_0$ through $R_{n-1}$. Their roles are explained in Chapter 2.

Finally, two registers facilitate communication with the memory. These are the *memory address register* (MAR) and the *memory data register* (MDR). The MAR holds the address of the location to be accessed. The MDR contains the data to be written into or read out of the addressed location.

Let us now consider some typical operating steps. Programs reside in the memory and usually get there through the input unit. Execution of the program starts when the PC is set to point to the first instruction of the program. The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory. After the time required to access the memory elapses, the addressed word (in this case, the first instruction of the program) is read out of the memory and loaded into the MDR. Next, the contents of the MDR are transferred to the IR. At this point, the instruction is ready to be decoded and executed.

If the instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands. If an operand resides in the memory (it could also be in a general-purpose register in the processor), it has to be fetched by sending its address to the MAR and initiating a Read cycle. When the operand has been read from the memory into the MDR, it is transferred from the MDR to the ALU. After one or more operands are fetched in this way, the ALU can perform the desired operation. If the result of this operation is to be stored in the memory, then the result is sent to the MDR. The address of the location where the result is to be stored is sent to the MAR, and a Write cycle is initiated. At some point during the execution of the current instruction, the contents of the PC are incremented so that the PC points to the next instruction to be executed. Thus, as soon as the execution of the current instruction is completed, a new instruction fetch may be started.

In addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to output devices. Thus, some machine instructions with the ability to handle I/O transfers are provided.

Normal execution of programs may be preempted if some device requires urgent servicing. For example, a monitoring device in a computer-controlled industrial process may detect a dangerous condition. In order to deal with the situation immediately, the normal execution of the current program must be interrupted. To do this, the device raises an ***interrupt*** signal. An interrupt is a request from an I/O device for service by the processor. The processor provides the requested service by executing an appropriate ***interrupt-service routine***. Because such diversions may alter the internal state of the processor, its state must be saved in memory locations before servicing the interrupt. Normally, the contents of the PC, the general registers, and some control information are stored in memory. When the interrupt-service routine is completed, the state of the processor is restored so that the interrupted program may continue.

The processor unit shown in Figure 1.2 is usually implemented on a single Very Large Scale Integrated (VLSI) chip, with at least one of the cache units of the memory hierarchy contained on the same chip.

## 1.4 Bus Structures

So far, we have discussed the functions of individual parts of a computer. To form an operational system, these parts must be connected in some organized way. There are many ways of doing this. We consider the simplest and most common of these here.

To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time. When a word of data is transferred between units, all its bits are transferred in parallel, that is, the bits are transferred simultaneously over many wires, or lines, one bit per line. A group of lines that serves as a connecting path for several devices is called a ***bus***. In addition to the lines that carry the data, the bus must have lines for address and control purposes.

The simplest way to interconnect functional units is to use a ***single bus***, as shown in Figure 1.3. All units are connected to this bus. Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus

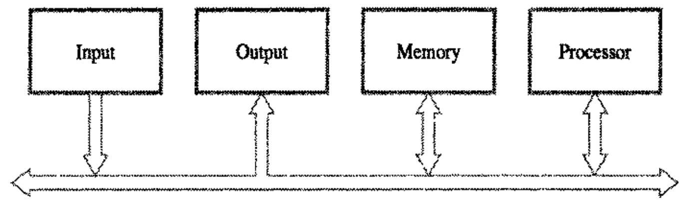


**Figure 1.3** Single-bus structure.

control lines are used to arbitrate multiple requests for use of the bus. The main virtue of the single-bus structure is its low cost and its flexibility for attaching peripheral devices. Systems that contain multiple buses achieve more concurrency in operations by allowing two or more transfers to be carried out at the same time. This leads to better performance but at an increased cost.

The devices connected to a bus vary widely in their speed of operation. Some electromechanical devices, such as keyboards and printers, are relatively slow. Others, like magnetic or optical disks, are considerably faster. Memory and processor units operate at electronic speeds, making them the fastest parts of a computer. Because all these devices must communicate with each other over a bus, an efficient transfer mechanism that is not constrained by the slow devices and that can be used to smooth out the differences in timing among processors, memories, and external devices is necessary.

A common approach is to include *buffer registers* with the devices to hold the information during transfers. To illustrate this technique, consider the transfer of an encoded character from a processor to a character printer. The processor sends the character over the bus to the printer buffer. Since the buffer is an electronic register, this transfer requires relatively little time. Once the buffer is loaded, the printer can start printing without further intervention by the processor. The bus and the processor are no longer needed and can be released for other activity. The printer continues printing the character in its buffer and is not available for further transfers until this process is completed. Thus, buffer registers smooth out timing differences among processors, memories, and I/O devices. They prevent a high-speed processor from being locked to a slow I/O device during a sequence of data transfers. This allows the processor to switch rapidly from one device to another, interweaving its processing activity with data transfers involving several I/O devices.

## 1.5 SOFTWARE

In order for a user to enter and run an application program, the computer must already contain some system software in its memory. *System software* is a collection of programs that are executed as needed to perform functions such as

- Receiving and interpreting user commands
- Entering and editing application programs and storing them as files in secondary storage devices