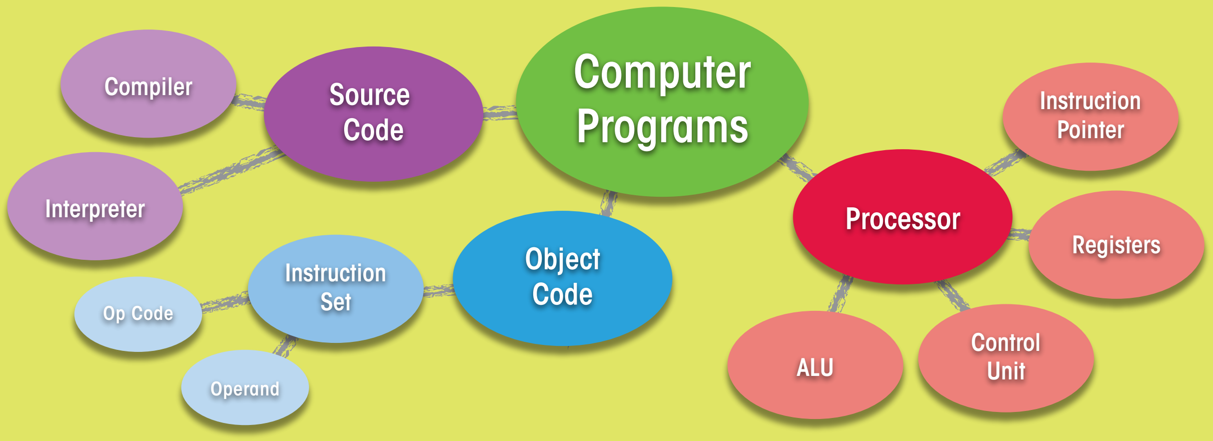


Digital Processing

SECTION **D**

COMPUTERS AND OTHER DIGITAL DEVICES process data, but how do they know what to do with it? The instructions you issue aren't 0s and 1s that a digital device can work with. So what goes on inside the box? Section D explains the programs that make digital devices tick. You'll discover that although digital devices appear to perform very complex tasks, under the hood they are really performing some very simple operations, but doing them at lightning speed.

PROGRAMS AND INSTRUCTION SETS

How do digital devices process data? Computers and dedicated handheld devices all work with digital data under the control of a computer program. Let's take a closer look at programs to see how they are created and how digital devices work with them.

Who creates programs? Computer programmers create programs that control digital devices. These programs are usually written in a high-level **programming language**, such as C, BASIC, COBOL, or Java.

Programming languages use a limited set of command words such as *Print*, *If*, *Write*, *Display*, and *Get* to form sentence-like statements designed as step-by-step directives for the processor chip. An important characteristic of most programming languages is that they can be written with simple tools, such as a word processor, and they can be understood by programmers. A simple program to select a song on your iPod might contain the statements shown in Figure 1-33.

TRY IT!

Was the first computer programmer a man or a woman? There is controversy surrounding the answer to this question. Check online to find the name of the person who is often cited as the first computer programmer.

- ❍ Bill Gates
- ❍ Ada Lovelace
- ❍ Grace Hopper
- ❍ Charles Fortran

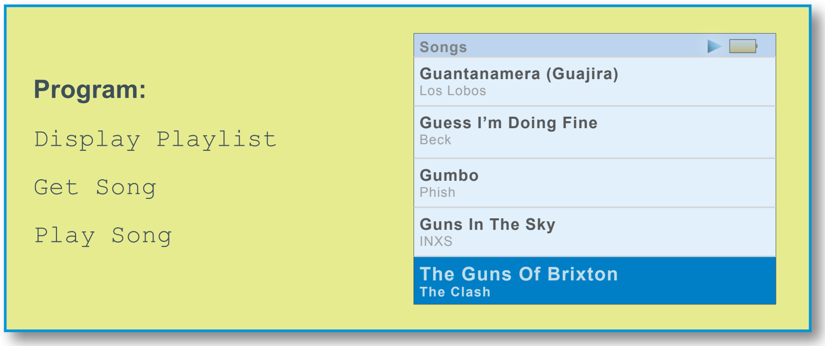

Program:
```
Display Playlist
Get Song
Play Song
```
Songs
Guantanamera (Guajira)
Los Lobos
Guess I'm Doing Fine
Beck
Gumbo
Phish
Guns In The Sky
INXS
The Guns Of Brixton
The Clash

FIGURE 1-33

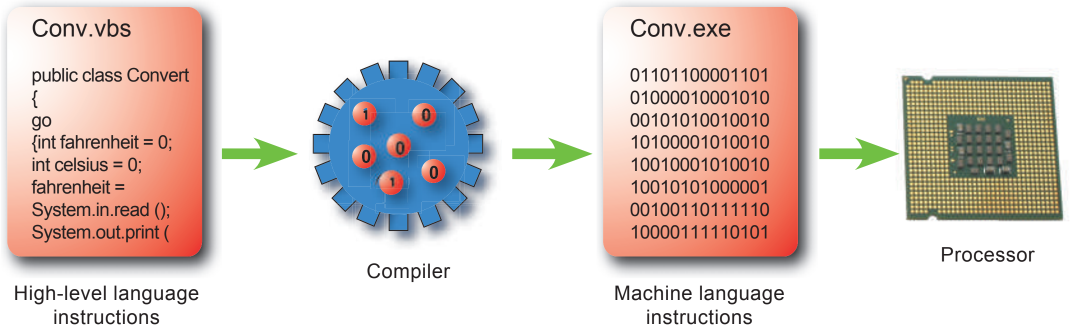
The program for an iPod displays a list of songs that the user can choose to play. A program works behind the scenes to display the list, get your selection, process it, and play the song.

The human-readable version of a program, like the one above, created in a high-level language by a programmer is called **source code**. However, just as a digital device can't work directly with text, sounds, or images until they have been digitized, source code has to be converted into a digital format before the processor can use it.

◗ **How does source code get converted?** The procedure for translating source code into 0s and 1s can be accomplished by a compiler or an interpreter. A **compiler** converts all the statements in a program in a single batch, and the resulting collection of instructions, called **object code**, is placed in a new file (Figure 1-34). Most of the program files distributed as software contain object code that is ready for the processor to execute.

FIGURE 1-34

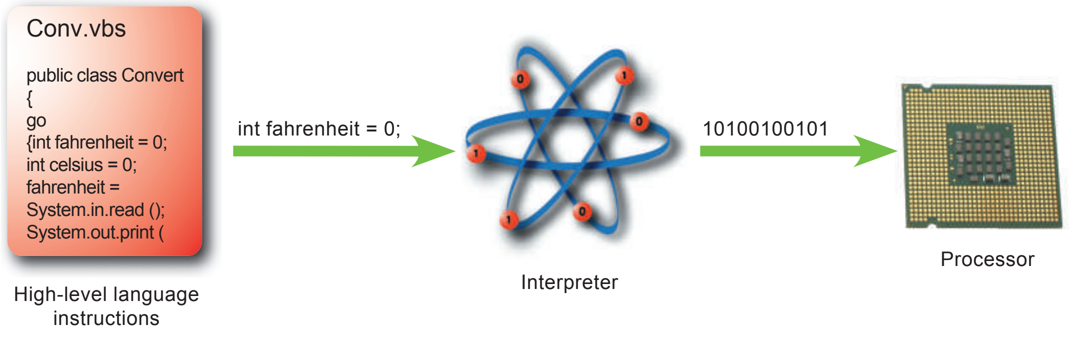
A compiler converts statements written in a high-level programming language into object code that the processor can execute.
▶ Watch a compiler in action.



As an alternative to a compiler, an **interpreter** converts and executes one statement at a time while the program is running. After a statement is executed, the interpreter converts and executes the next statement, and so on (Figure 1-35).

FIGURE 1-35

An interpreter converts high-level statements one at a time as the program is running.
▶ Watch an interpreter in action.



Compilers and interpreters don't simply convert the characters from source code into 0s and 1s. For example, in the first line of the iPod program, Display Playlist, a compiler would not simply convert the *D* into its ASCII equivalent. No, computers are a little trickier than that.

◗ **What does the conversion process produce?** A microprocessor is hard-wired to perform a limited set of activities, such as addition, subtraction, counting, and comparisons. This collection of preprogrammed activities is called an **instruction set**. Instruction sets are not designed to carry out any specific task, such as word processing or playing music. Instead, an instruction set is designed to be general purpose so that programmers can use it in creative ways for the wide variety of tasks performed by all kinds of digital devices.

Each instruction has a corresponding sequence of 0s and 1s. For example, 00000100 might correspond to *Add*. The list of codes for a microprocessor's instruction set, called **machine language**, can be directly executed by the processor's circuitry. A set of machine language instructions for a program is called **machine code**.

TRY IT!

Suppose a programmer wants to distribute the object code for an iPhone app. After creating the program as source code, the programmer needs to use a(n):

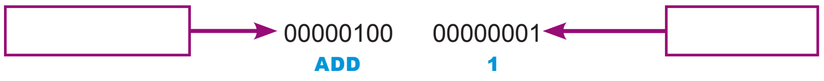
❍ interpreter

❍ compiler

❍ COBOL

❍ ASCII code

A machine language instruction has two parts: the op code and the operands. An **op code**, which is short for *operation code*, is a command word for an operation such as add, compare, or jump. The **operand** for an instruction specifies the data, or the address of the data, for the operation. In the following instruction, the op code means add and the operand is 1, so the instruction means add 1.



TRY IT!

Label the operand and the op code.

A single high-level instruction very often converts into multiple machine language instructions. Figure 1-36 illustrates the number of machine language instructions that correspond to a simple high-level program.

```
#include <stdio.h>
int main ()
{
int i;

for (i=1; i<=100; i=i+1)
  printf("%d\t",i);
return(0);
}
```

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000001111000000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000010000000000011101100
00100100100001000000001000011000
```

FIGURE 1-36

The source code program in the left column prints numbers from 1 to 100. This source code is converted to machine language instructions shown in the right column that the computer can directly process.

To summarize what you should now know about programs and instruction sets, a programmer creates human-readable source code using a programming language. A compiler or an interpreter converts source code into machine code. Machine code instructions are a series of 0s and 1s that correspond to a processor's instruction set.

PROCESSOR LOGIC

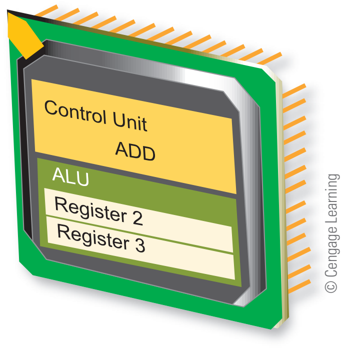
▶ What happens inside a computer chip? A microprocessor contains miles of microscopic circuitry and millions of miniature components divided into different kinds of operational units, such as the ALU and the control unit.

The **ALU** (arithmetic logic unit) is the part of the microprocessor that performs arithmetic operations, such as addition and subtraction. It also performs logical operations, such as comparing two numbers to see if they are the same. The ALU uses **registers** to hold data that is being processed, just as you use a mixing bowl to hold the ingredients for a batch of cookies.

The microprocessor's **control unit** fetches each instruction, just as you get each ingredient out of a cupboard or the refrigerator. Data is loaded into the ALU's registers, just as you add all the ingredients to the mixing bowl. Finally, the control unit gives the ALU the green light to begin processing, just as you flip the switch on your electric mixer to begin blending the cookie ingredients. Figure 1-37 illustrates a microprocessor control unit and an ALU preparing to add 2 + 3.

FIGURE 1-37

The control unit fetches the ADD instruction, then loads data into the ALU's registers where it is processed.


Control Unit
ADD
ALU
Register 2
Register 3
© Cengage Learning