CHAPTER 4 — The Processor

Chapter Topics:

- Machine operations and machine language
- Example of machine language
- Different types of processor chips
- High level programming languages
- Language translators (compilers)
- Language interpreters

From previous chapters you know that the major hardware components of a computer system are:

- Processor
- Main memory
- Secondary memory devices
- Input/output devices

This chapter examines the "brain" of the computer system — the processor. Then it shows how the programs you write are translated into instructions for the processor.

QUESTION 1:

What component of a computer system is directly responsible for running a program?

Answer:

The processor.

Electronic Operations of a Processor

When a program is running on a computer the processor is constantly performing very many tiny electronic operations. For example, one such operation reads one byte of data from main memory into the processor. Another operation tests if one of the bits in a byte is a 1 bit or a 0 bit. Most processors are able to perform several thousand types of tiny operations like these, and can perform billions of them per second.

Those are the only things that a processor can do. It has a set of tiny electronic operations that it can to perform, and that is all. These tiny electronic operations are performed one at a time. But billions of them are performed per second, and billions of small operations can add up to a large and useful action.

Everything that a processor does is built out of these tiny operations! Luckily, you don't need to know the details of these operations to write programs in Java. The purpose of a *high-level language* like Java is to organize the tiny electronic operations into large, useful units represented by program statements.

QUESTION 2:

(Thought question:) When you click on a hyperlink on a Web page, your Web browser (a computer program) finds and displays a new page. About how many tiny electronic operations does the processor perform in doing this?

- 1
- 10
- 100
- 100,000

Answer:

I would guess about 100,000 operations. Certainly not as few as 100. All these tiny operations add up to a useful big operation: displaying a new Web page.

Machine Instructions

Users and programmers of computers usually don't think about the billions of tiny electronic operations that go on each second. The situation is (very roughly) similar to when you are driving your car. You think about the "big operations" it can perform, such as "accelerate", "turn left", "brake", and so on. You don't think about tiny operations, such as the valves in your engine opening and closing 24,000 times per minute or the crankshaft spinning at 3000 revolutions per minute.

Each tiny electronic operation that a processor core can perform is called a**machine operation**. A processor (a "machine") performs these one at a time, but billions of them in a second.

A **machine instruction** consists of several bytes in memory that tell the processor to perform one machine operation. The processor looks at machine instructions in main memory one after another, and performs one machine operation for each machine instruction. The collection of machine instructions in main memory is called a **machine language program** or (more commonly) an **executable program**.

Don't panic if the above seems incomprehensible. It takes some getting used to. (And to *really* understand it all takes several courses.)

QUESTION 3:

When they are running, are machine langage programs located in main memory along with data?

Answer:

Yes. Programs and data are both in main memory when they are active.

Example of Machine Language

Say that a light bulb is controlled by a processor running a program in main memory. The controller can turn the light bulb fully on and fully off, can brighten or dim the bulb (but not beyond fully on or off.) The machine instructions are one byte long, and correspond to the following machine operations:

Machine Instruction	Machine Operation
00000000	Stop Program
00000001	Turn bulb fully on
00000010	Turn bulb fully off
00000100	Dim bulb by 10%
00001000	Brighten bulb by 10%
00010000	If bulb is fully on, skip over next instruction
00100000	If bulb is fully off, skip over next instruction
01000000	Go to start of program (address 0)

The bulb is wired so that when the switch is first turned on the instruction at address zero is performed. Then, instructions are performed one at a time, in order, until the "Halt" instruction is encountered or the controller crashes.

The controller crashes if it encounters an instruction not in the table or tries to get an instruction from an address that does not exist.

The A "Skip" instruction checks the brightness of the bulb and possibly skips over the instruction after it.

The "Go back" instruction causes the processor to start again with the instruction at address 0.

Here (let us pretend) is the main memory for the controller:

Address	Machine Instruction
0	00000001
1	00000010
2	00000001
3	00000010
4	00000001
5	00000100
6	00000100
7	00000100

8	00000100
9	00000000
10	00000000
11	00000000
12	00000000

Click on the light bulb to start or stop the program.

QUESTION 4:

Change the program currently in memory. Fill in instructions starting at address 0. Copy and paste from the list of machine instructions or directly edit the bit patterns.

Try to alter the program so that the light bulb blinks on and off without stopping. Or, write a program so that the bulb gradually brightens and dims repeatedly.

Answer:

Hopefully your machine language program ran successfully and did not crash the system.

If it did crash, check that every instruction is one of the eight bit patterns in the table.

Machine Language Program

This is, of course, a silly example. Actual processors have many more machine instructions and the instructions are much more detailed. A typical processor has a thousand or more different machine instructions. Those instructions do not directly affect any output device (such as our light bulb.)

Even the simplest controller has much more memory. But the essential ideas of the example are these:

- A machine language program is a sequence of machine language instructions in main memory.
- A machine instruction consists of one or more bytes (in this example, only one).
- The processor runs a program one machine instruction at a time.
- All the little machine operations add up to something useful.

QUESTION 5:

Is it obvious what each machine instruction in our example means? That is, is it obvious what operation "00100000" calls for?

Answer:

No. Bit patterns have been arbitrarily assigned to operations, much like with real processors. Usually with real processors, bit patterns have been systematically assigned to machine operations, but there is no inherent meaning to the patterns.

Executing Instructions

The word "execute" is often used to mean "perform the machine operation that an instruction asks for." So you can say that "execute the instruction 00000000 to stop the processor," or "billions of instructions execute per second." "Execute" is also used for an entire program or part of a program: "to execute the program, turn the switch to on."

Most programs have groups of instructions that are executed again and again. The light bulb program does this with an instruction that causes the processor to return to the beginning of the program and so to repeat what it just did.

A group of machine instructions that executes repeatedly is called a **loop**.

A modern processor executes billions of instructions per second. A program without loops would execute for only a few seconds even if it had billions of instructions.

A typical processor is made up of millions of transistors on a small wafer of silicon called an **integrated circuit** (also called a **chip**.) The light bulb's processor could probably be built with just a few hundred transistors. Integrated circuits are used for other electronic parts of a computer. For example main memory is implemented with memory chips.

Most modern processor chips have two or more processors on them. Each of the processors on the chip is called a "processor core" or often just called a **core**. Each core has the same set of tiny electronic operations and runs programs independently of the other cores. When a program is running, it runs on a single core, which performs the tiny electronic operations of the program one at a time.

QUESTION 6:

Say that a computer has four billion bytes of memory and that memory is entirely filled with a single program. Each instruction is one byte long, and the processor can execute one billion instructions per second.

How long will the program run if execution starts with the first byte and there are no loops in the program?

Answer:

Four billion / one billion = 4 seconds

In fact, most instructions are longer than one byte, most desktop computers execute more than one billion instructions in a second, and most programs are much shorter than four billion bytes. So without loops, most programs would run for a fraction of a second.

Different Processors

There are many types of processors used in computer systems. You probably know something about the processors used in most desktop computers, Intel Corporation's Pentium processors. But there are other types of processors, such as the processors used in cell phones and game machines. A computer system is designed around its processor. The electronics of a computer system are designed for a particular type of processor.

Different types of processors have different machine operations and different machine languages. A machine language program for a typical desktop system (with a Pentium processor) would make no sense to a computer built around a different processor type.

However, the machine operations available with all types of processors can be used to build the same things. All processor types have enough power in their little machine operations to create the same large applications. Anything one processor can do with its machine language program another processor can do with a program written in its machine language. For example, cell phones are built around a variety of processor types, but all cell phones can do the same things.

The **architecture** of a processor is the choices that have been made for its machine operations, how they have been organized and implemented, and how they interact with main memory and other components. Architecture is concerned with the general plan and

functions of a processor; it is not much concerned with electronic details. A course in computer architecture is part of most computer science departments.

QUESTION 7:

(Thought question:) Say that you are looking at two different processor chips. The first processor has twice as many types of machine operations as the second chip. Is the first processor necessarily the best one?

Answer:

No—the smaller chip might have a well chosen set of operations that work together better and faster than the poorly chosen operations of the other processor.

High Level Programming Languages

It is rare for programmers to write programs in machine language like we did for the light bulb. The executable files (the directly runnable machine language programs) for most applications contain hundreds of thousands (or even millions) of machine language instructions. It would be very hard to create something like that from scratch.

As an experiment, look through your hard disk with the file listing utility (the "Explorer" on Microsoft systems.) Look at the size of the *something*.EXE files. There are about four bytes per machine instruction on Intel processors, so divide by four to get the number of instructions.

Most programs are created using a **high level programming language** such as Java, C, C++, or BASIC. With a high level language, a programmer creates a program using powerful, "big" operations which will later be converted into many little machine operations.

For example, here is a line from a program in the language C:

```
int sum = 0;
```

This declares and initializes a variable to zero (a big operation).

The machine operations that correspond to this big operation set up a part of main memory to hold a number, store the number zero there, and arrange things so other parts of the program can use it. It might take a hundred machine operations to do all this. Obviously, it is easier for a human programmer to ask for all these operations using C.

QUESTION 8:

Say that a corporation pays programmers $50 an hour. Will the corporation want programmers to program in machine language or in a high level language?

Answer:

A high level language. Programmers can create programs in much less time (costing much less money) by using a high level language.

Source Programs

Programmers create programs by writing commands in a high level language. A high level language program consists of lines of text that have been created with a text editor and are kept in a file on the hard disk. For example, here is a complete program in C (Java will be discussed later):

```
#include <stdio.h>
main()
{
  int sum = 0;
  sum = 2 + 2;
  printf( "%d\n", sum );
}
```

This program could be saved on the hard disk in a file called `addup.c`. Like all files, it consists of a sequence of bytes. Since it is a text file, these bytes represent character data. You can edit the file with a text editor and print the file on a printer. It *does not* contain machine instructions. If the bytes are copied into main memory, they cannot run as a program without some extra work being done.

A **source program** is a text file that contains instructions written in a high level language. It can not be executed (made to run) by a processor without some additional steps.

A source program is also called a *source file*, *source code*, or sometimes, just *source*.

Usually a source program is **translated** into a machine language program. An application program called a **translator** takes a source program as input and produces a machine language program as output.

A machine language program is also called an *executable program*, *executable file*, or sometimes, just *executable*.

For example, the C program `addup.c` could be translated into an executable program. The executable program might be called `addup.exe`and can be saved on the hard disk. Now the executable version of the program can be copied into main memory and executed.

The word **compile** means the same thing as **translate**. So one can say that a source program is compiled into an executable program.

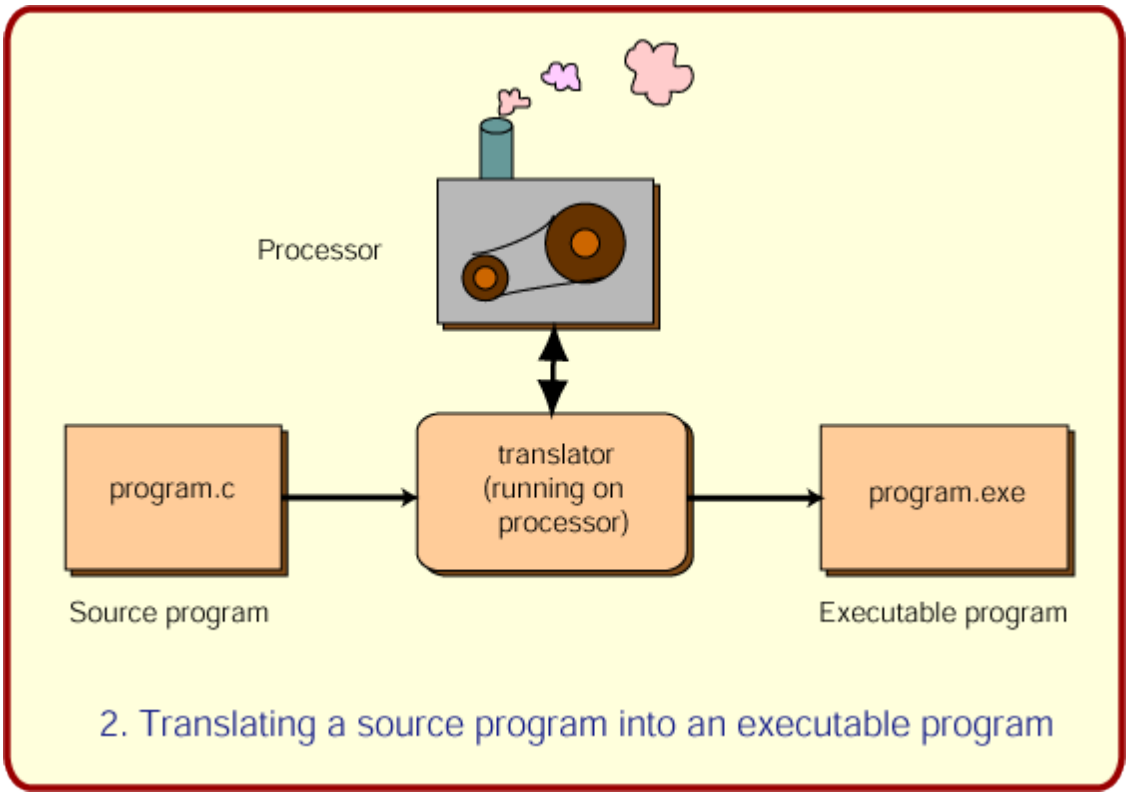
QUESTION 9:

Say that a source program has been translated into an executable program. The executable program has been run a few times, and the programmer decides to make a change to the program. Where is the change made? To the source program or to the executable program?

Answer:

To the source program, where the change is easy to make with a text editor. The changed source program is then translated into a new executable program.

Program Translation



2. Translating a source program into an executable program

The picture shows what usually happens with programs written in C (Java is different; it will be discussed in the next chapter.) Here is an overview:

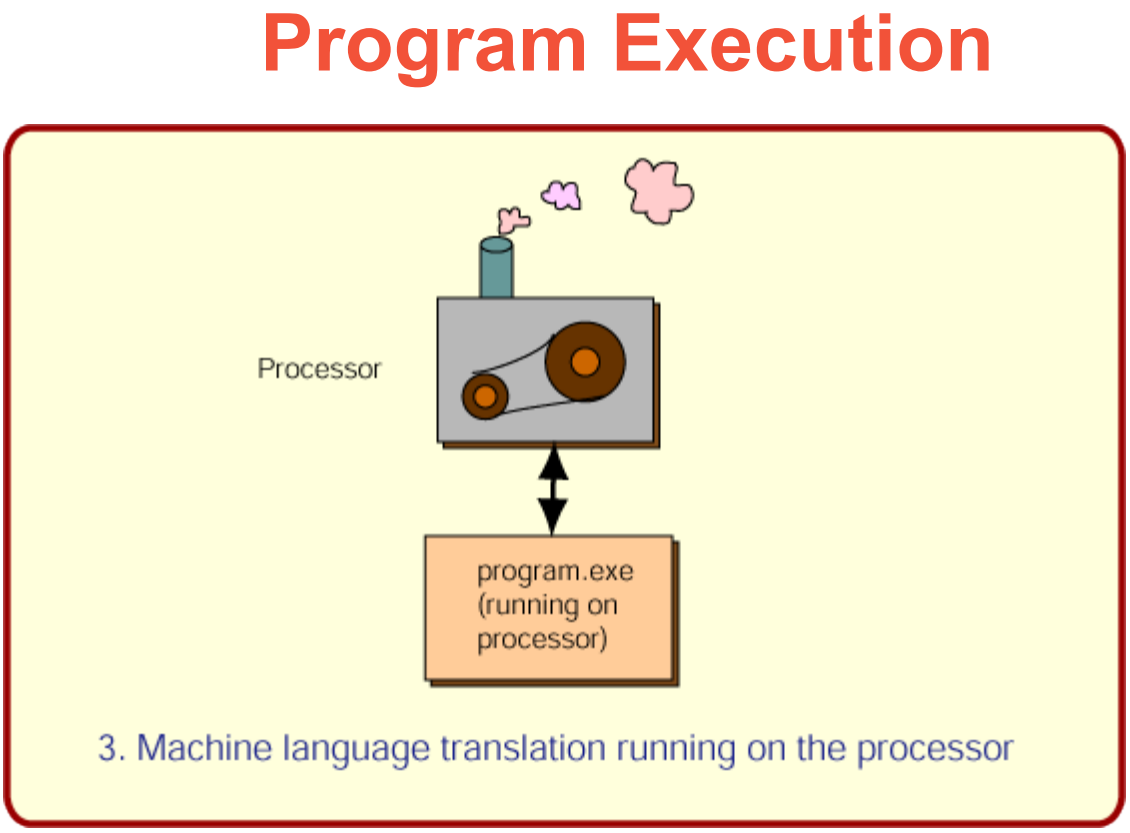
1. The source program is created using a text editor.
    - It contains instructions in a high level language.
    - It contains bytes that represent characters.
    - The source program is kept on the hard disk.
    - The source program can not be run by the processor.
2. A translator (compiler) program translates the source program into an executable program.
    - The source program remains unchanged; a new executable program is created.
    - The executable program contains machine instructions.
    - A translator translates from a specific high level language (like C) into machine instructions for a specific processor type (like Pentium).
    - The executable program is also kept on hard disk.
3. The program is run by copying machine language from the executable program on disk into main memory. The processor directly executes these machine language instructions.

QUESTION 10:

Can the processor directly execute the machine language that the program has been translated into?

Answer:

For languages like C, the answer is "yes".



Once the source program has been translated into machine code (the executable program), the machine code can be directly executed by the processor. This is like the light bulb's processor directly executing its machine code. The source program is only a means to create the machine code.

For commercial software like games and word processors, the machine code is the product that is sold to the user. The user does not get a copy of the source program.

A student learning programming, or a programmer developing an application, creates source programs and translates them (with a compiler) into executable programs.

The above is what goes on with languages like Ada, Pascal, C, C++, FORTRAN and others. Java adds a few more steps, which will be discussed in the next chapter.

QUESTION 11:

Say that you have a source program written in C. You copy the program onto the hard disk of a Pentium-based computer and also copy it onto the hard disk of an Motorola-based computer. What must you do so both computers can execute the program?

Answer:

Each copy of the source program must be translated into an executable program that is correct for each machine:

- **On the Pentium:** A C translator that outputs machine language for Pentium processors is used.
- **On the Motorola-based computer:** A C translator that outputs machine language for Motorola processors is used.

Portability

Ideally, only one program needs to be written in the high level language. That source file can then be translated into several executable files, each containing the correct machine instructions for its intended processor. This is how the same game can be made for desktop computers and game machines.

The idea of using one source file for executable programs that run on different processors is called **software portability**. You would like to write a program just once (in a high level

language) and then to run it on any computer system by translating it into that system's machine language.

Usually, unfortunately, things do not work out that nicely. There are enough little problems so that it takes a substantial amount of human effort to get a source program running on a several different systems. Sometimes it is months before a game program that has been released for PCs is released for game consoles.

One of the big advantages of Java is that it is automatically portable between computer systems that have Java support. No human effort is involved at all.

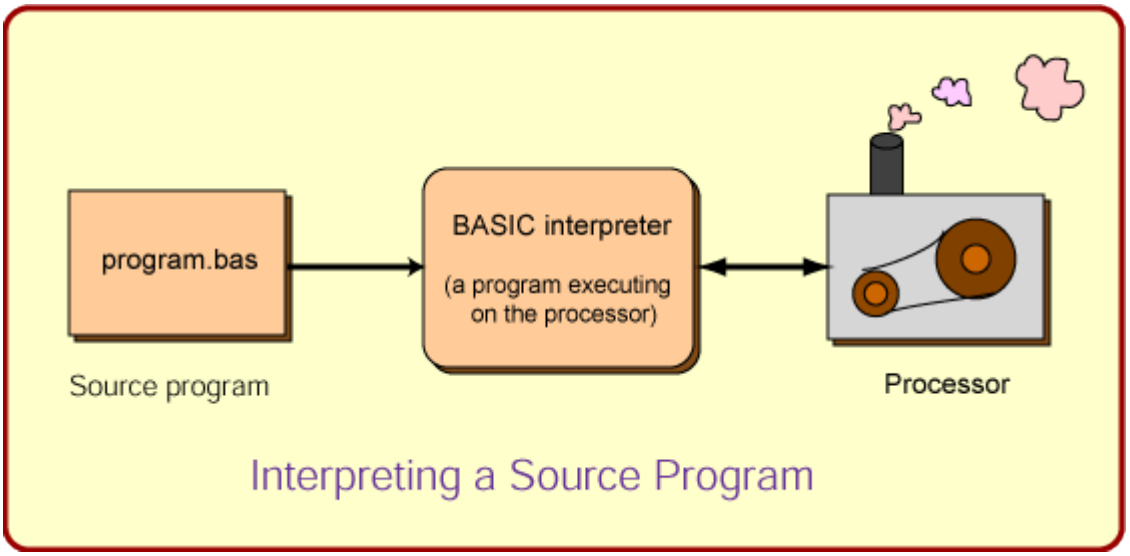
QUESTION 12:

Say that a corporation pays programmers $50 an hour to write application programs that will run on both Apple and Windows computers. Will the corporation want programmers to program in Java or in some other high level language?

Answer:

Java. The time (and expense) of getting the program to run on other computers is avoided. (Actually, sometimes things do not work out as nicely as this.)

Interpreter



Programs written in a high level language are never directly executed by the processor. You have already seen one way to execute such a program: use a translator to create a machine language program that can be executed directly.

Another way to is to use an **interpreter** for the language. An interpreter is an executable program that runs directly on the processor. An interpreter reads through a source program written in a high level language and performs the actions that the source program asks for.

This is a fairly complicated thought. The figure might help.

In this figure, the source program `program.bas` has been written in BASIC (a programming language) by a programmer. It is being interpreted by the BASIC interpreter, which is an executable program running on the processor.

The BASIC interpreter works by reading in commands of the BASIC source program one by one. Each time it reads in a command, the interpreter does what the command asks. A BASIC command might ask to add two numbers together. That is fine. The BASIC interpreter is a program, and programs can easily add together two numbers. The BASIC program might then ask to write the sum to the monitor. Still fine. The BASIC interpreter can easily do that.

The BASIC interpreter uses the fundamental machine operations of the processor to perform the actions requested in the BASIC source program. But the source program itself is not translated into machine language.

This is like a cook who can perform basic cooking operations but can't read recipies. The cook hires an assistant who can read recipies. The assistant reads the instructions of a recipe one by one to the cook, who performs them one by one as they are read.

You have probably done this many times without realizing it. It is exactly accurate (although an unusual use of the words) to say that the computer game Free Cell Solitaire (or any computer game) is an interpreter for the commands that you enter using the mouse and keyboard. The commands for the game Free Cell are not the usual commands used in a general-purpose programming language, but none the less, they are commands expressed in a formal language. You enter commands, and the interpreter (the game) executes them one by one. The set of commands and the rules for how to use them correctly form a programming language for the Free Cell interpreter.

In playing a computer game, you are writing a program that solves a problem. (Usually the problem is to get to the end of the game.) If you recorded every move you made, you would have a program that the game, acting as an interpreter, could run to solve the problem.

(Of course, each time you play solitaire, the problem is a randomly selected initial arrangement of cards, so you need a different program for each problem.)

QUESTION 13:

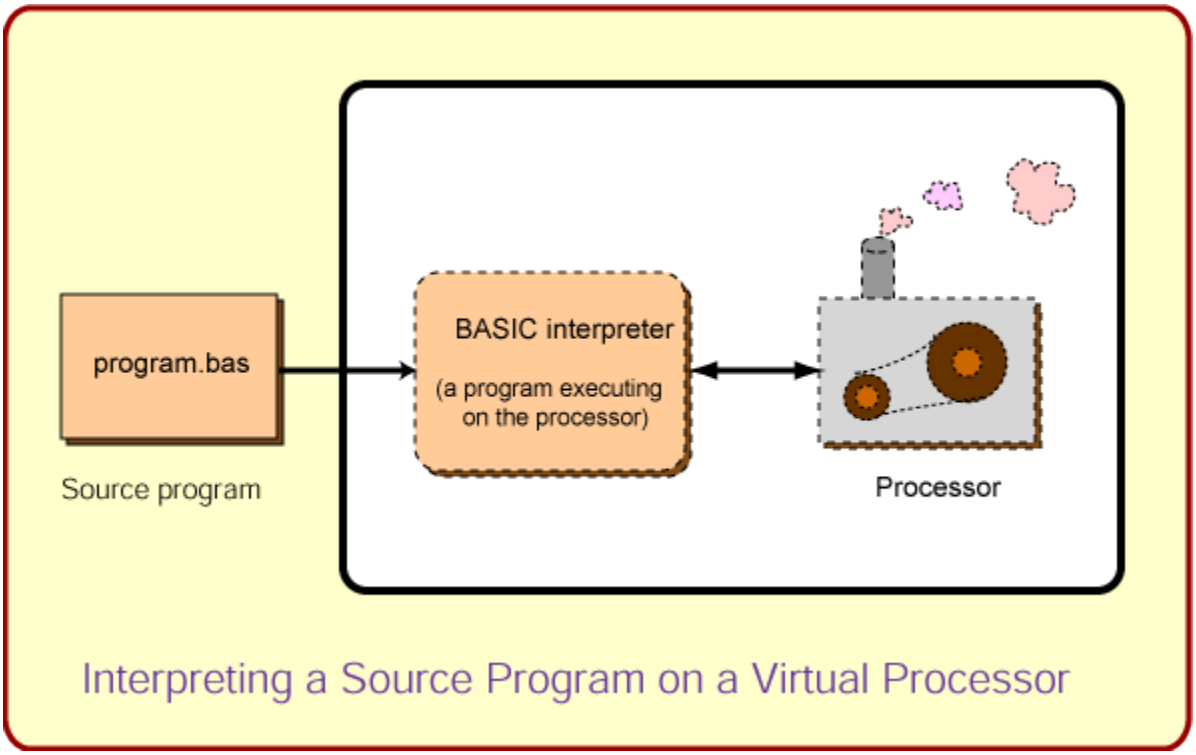
An interpreter runs directly on the processor. What type of instructions must an interpreter consist of?

Answer:

An interpreter is an executable program, which consists of machine instructions, like all programs that execute directly on the processor.

Of course, a programmer creates an interpreter by writing it in a high level language and compiling it into the executable program.

Virtual Machine



Interpreting a Source Program on a Virtual Processor

When an interpreter is running a BASIC source program, both the interpreter and the source program are in main memory. The interpreter consists of machine instructions that the hardware can execute directly. The BASIC source program consists of commands that the interpreter can perform.

From the perspective of the BASIC program, *it looks like the commands in BASIC are being directly executed by some sort of machine.* The figure has been modified to show this.

This is really the same as the previous figure, but now a box has been drawn around the actual (hardware) processor and the interpreter that it is executing. The combination looks like a machine that can directly execute BASIC commands. It is as if BASIC commands are the machine language for combination of processor and interpreter.

The word **virtual** is used in situations where software has been used to make something look like the real thing. In this case it looks like we have a machine that can directly execute BASIC, so we can say that we have a BASIC virtual machine.

QUESTION 14:

Do you think that a BASIC source program could be translated into an executable file of machine instructions?

Answer:

Yes. There are both translators and interpreters for most high level language. Which you use depends on how you use the high level program.

Speed

The situation with computer languages is somewhat like that with human languages:

- **Translator:** takes a complete document in one language and translates it into a complete document in a second language, which can then be used by a reader of the second language.
- **Interpreter:** acts as an intermediate between a speaker of one language and a speaker of another language. Usually an interpreter works one sentence at a time. Immediately after a sentence is spoken in the first language, the translator converts it into the second language. You and your French translator (say) could in combination be regarded as a "virtual French speaker".

Using a human interpreter as an intermediate is slower than conversing directly in a particular language. The same is true with computer language interpreters. The interpreter has to do quite a bit of work to deal with the language it is interpreting. The extra work makes it look like the virtual processor is much slower than the real one.

QUESTION 15:

Is it always important for a program to run as fast as possible? (Hint: take some time to answer this question.)

Answer:

No. Programs should get their work done in a timely fashion, but often moderately slow is fast enough.

For example, your Web browser mostly does nothing as it waits for your commands. It is an interpreter for the commands you enter using the mouse and keyboard. It would be only a slight improvement to make it run much faster.

End of the Chapter!

You may wish to review the following. Click on a subject that interests you to go to where it was discussed.

- Machine operations and machine instructions.
- Executing a program.
- Differences in types of processor chips.
- High level programing language.
- Source program.
- Program translation.
- Program interpretation.
- Portability.

The next chapter will discuss how the language Java fits into the concepts this chapter has discussed.

You have reached the end of the chapter.