Microprocessor programming

The "vocabulary" of instructions which any particular microprocessor chip possesses is specific to that model of chip. An Intel 80386, for example, uses a completely different set of binary codes than a Motorola 68020, for designating equivalent functions. Unfortunately, there are no standards in place for microprocessor instructions. This makes programming at the very lowest level very confusing and specialized.

When a human programmer develops a set of instructions to directly tell a microprocessor how to do something (like automatically control the fuel injection rate to an engine), they're programming in the CPU's own "language." This language, which consists of the very same binary codes which the Control Unit inside the CPU chip decodes to perform tasks, is often referred to as *machine language*. While machine language software can be "worded" in binary notation, it is often written in hexadecimal form, because it is easier for human beings to work with. For example, I'll present just a few of the common instruction codes for the Intel 8080 micro-processor chip:

Hexadecimal	Binary	Instruction description
7B	01111011	Move contents of register A to register E
87	10000111	Add contents of register A to register D
1C	00011100	Increment the contents of register E by 1
D3	11010011	Output byte of data to data bus

Even with hexadecimal notation, these instructions can be easily confused and forgotten. For this purpose, another aid for programmers exists called *assembly language*. With assembly language, two to four letter mnemonic words are used in place of the actual hex or binary code for describing program steps. For example, the instruction`7B` for the Intel 8080 would be "`MOV A,E`" in assembly language. The mnemonics, of course, are useless to the microprocessor, which can only understand binary codes, but it is an expedient way for programmers to manage the writing of their programs on paper or text editor (word processor). There are even programs written for computers called *assemblers* which understand these mnemonics, translating them to the appropriate binary codes for a specified target microprocessor, so that the programmer can write a program in the computer's native language without ever having to deal with strange hex or tedious binary code notation.

Once a program is developed by a person, it must be written into memory before a microprocessor can execute it. If the program is to be stored in ROM (which some are), this can be done with a special machine called a *ROM programmer*, or (if you're masochistic), by plugging the ROM chip into a breadboard, powering it up with the appropriate voltages, and writing data by making the right wire connections to the address and data lines, one at a time, for each instruction. If the program is to be stored in volatile memory, such as the operating computer's RAM memory, there may be a way to type it in by hand through that computer's keyboard (some computers have a mini-program stored in ROM which tells the microprocessor how to accept keystrokes from a keyboard and store them as commands in RAM), even if it is too dumb to do anything else. Many "hobby" computer kits work like this. If the computer to be programmed is a fully-functional personal computer with an operating system, disk drives, and the whole works, you can simply command the assembler to store your finished program onto a disk for later retrieval. To "run" your program, you would simply type your program's filename at the prompt, press the Enter key, and the microprocessor's Program Counter register would be set to point to the location ("address") on the disk where the first instruction is stored, and your program would run from there.

Although programming in machine language or assembly language makes for fast and highly efficient programs, it takes a lot of time and skill to do so for anything but the simplest tasks, because each machine language instruction is so crude. The answer to this is to develop ways for programmers to write in "high level" languages, which can more efficiently express human thought. Instead of typing in dozens of cryptic assembly language codes, a programmer writing in a high-level language would be able to write something like this . . .

```
Print "Hello, world!"
```

. . . and expect the computer to print "Hello, world!" with no further instruction on how to do so. This is a great idea, but how does a microprocessor understand such "human" thinking when its vocabulary is so limited?

The answer comes in two different forms: *interpretation*, or *compilation*. Just like two people speaking different languages, there has to be some way to transcend the language barrier in order for them to converse. A translator is needed to translate each person's words to the other person's language, one way at a time. For the microprocessor, this means another program, written by another programmer in machine language, which recognizes the ASCII character patterns of high-level commands such as Print (P-r-i-n-t) and can translate them into the necessary bite-size steps that the microprocessor can directly understand. If this translation is done during program execution, just like a translator intervening between two people in a live conversation, it is called "interpretation." On the other hand, if the entire program is translated to machine language in one fell swoop, like a translator recording a monologue on paper and then translating all the words at one sitting into a written document in the other language, the process is called "compilation."

Interpretation is simple, but makes for a slow-running program because the microprocessor has to continually translate the program between steps, and that takes time. Compilation takes time initially to translate the whole program into machine code, but the resulting machine code needs no translation after that and runs faster as a consequence. Programming languages such as BASIC and FORTH are interpreted. Languages such as C, C++, FORTRAN, and PASCAL are compiled. Compiled languages are generally considered to be the languages of choice for professional programmers, because of the efficiency of the final product.

Naturally, because machine language vocabularies vary widely from microprocessor to microprocessor, and since high-level languages are designed to be as universal as possible, the interpreting and compiling programs necessary for language translation must be microprocessor-specific. Development of these interpreters and compilers is a most impressive feat: the people who make these programs most definitely earn their keep, especially when you consider the work they must do to keep their software product current with the rapidly-changing microprocessor models appearing on the market!

To mitigate this difficulty, the trend-setting manufacturers of microprocessor chips (most notably, Intel and Motorola) try to design their new products to be *backwardly compatible* with their older products. For example, the entire instruction set for the Intel 80386 chip is contained within the latest Pentium IV chips, although the Pentium chips have additional instructions that the 80386 chips lack. What this means is that machine-language programs (compilers, too) written for 80386 computers will run on the latest and greatest Intel Pentium IV CPU, but machine-language programs written specifically to take advantage of the Pentium's larger instruction set will not run on an 80386, because the older CPU simply doesn't have some of those instructions in its vocabulary: the Control Unit inside the 80386 cannot decode them.

Building on this theme, most compilers have settings that allow the programmer to select which CPU type he or she wants to compile machine-language code for. If they select the 80386 setting, the compiler will perform the translation using only instructions known to the 80386 chip; if they select the Pentium setting, the compiler is free to make use of all instructions known to Pentiums. This is analogous to telling a translator what minimum reading level their audience will be: a document translated for a child will be understandable to an adult, but a document translated for an adult may very well be gibberish to a child.

Source: http://www.allaboutcircuits.com/vol_4/chpt_16/5.html