and slow but could often be upgraded (at a huge cost) with hardware that would permit software development for more than one manufacturer's parts. If you can find one of those old development systems today, it probably will be in use as a doorstop or boat anchor.

It is unarguable that the standardization of the business world around the IBM PC and its derivatives has been a real advantage to the embedded systems developer. Most manufacturers of microprocessor ICs now provide development software instead of systems for their parts. These *cross-compilers* run on a PC to compile or assemble code for the manufacturer's microprocessor. (Technically, a cross-assembler is a special case of a cross-compiler, and in this book the term *cross-compiler* will refer to both types of software.) Some manufacturers even give away some sort of development tools (usually an assembler) to potential customers on the premise that they are in the IC business, not the software business. It is unknown how many microprocessor selections have been made based on the availability of these free tools, but the number must be large.

Many IC manufacturers still provide complete development systems for their parts. These are usually PCs with the manufacturer's software included, and they constitute a complete development environment. But buy carefully—these PCs can be a bad deal from a cost perspective.

Some will argue that the PROM programmer no longer is an essential development tool, and they are right. If the project is to be developed in RAM or on an embedded PC or on a flash-based, downloadable processor, a PROM programmer is not needed. As more and more microcontrollers move to a flash-based architecture, the need for PROM programmers in the engineering lab will decline further. However, some projects still are developed in an environment in which parts have to be erased and programmed every time the code changes; for those developers, a PROM programmer is needed.

When the development system consists of a cross-compiler and a PROM programmer and little else, debugging is simple, although often tedious. The code is run, the operation of the system is observed, and the code is examined to see why things do not work. This process is repeated until all bugs are found. For some systems, especially at small companies, this stare-at-the-code method is still used and works well. This method becomes less and less attractive as system complexity grows and development schedules shrink.

The next level of debug is a *monitor program*, sometimes called a *debugger*. This simple program resides in the embedded system and provides commands to examine and alter memory, download code, and insert breakpoints into the code. A *breakpoint* is an unconditional branch instruction that takes the code back to the monitor program, where the registers and memory may be examined. Monitor programs require some kind of terminal, and the monitor program itself takes up considerable memory, so they typically are not used with very simple microprocessors.

As microprocessors become more complex, debugging the completed system becomes more difficult. Many designers, especially at large companies, use an *emulator* for system debugging. The emulator has a probe that replaces the microprocessor IC in the system (the *target*) and is supposed to run exactly the same as the target part. However, the emulator allows the engineer to insert breakpoints into the code so that the microprocessor's operation can be stopped. While stopped, the memory, internal registers, and other information about the microprocessor can be examined, the same as with a monitor program. In a simple emulator, the breakpoint is typically a specific address—for example at the instruction in the pool timer that turns on the motor relay. More sophisticated emulators have additional hardware that allows breakpoints when specific values are written or read to or from memory, when a specific sequence of instructions is executed, or for many other causes.

One drawback to emulators is their cost. Ranging from a few hundred dollars for a simple microprocessor (such as the Intel 8031 family) to several thousand dollars for a more complicated IC, the cost is often prohibitive. As I mentioned earlier, many companies base several products on a single microprocessor type due to the cost of buying new emulator equipment.

As microprocessors have grown even faster, their speed has outpaced the emulator industry's ability to keep up. In addition, the use of more powerful processors for applications often means some CPU horsepower is left over after the application is developed. Many developers have moved away from emulators and back to the monitor or debugger programs. These now more-sophisticated programs take advantage of leftover CPU capacity to provide event tracing, throughput measurement, code histograms that show how much time the CPU spends in each section of code, and other powerful debugging information. In addition, many processors now include debugging resources on-chip. We'll examine this in more detail in a later chapter.

Development Costs

In most companies, someone must produce an estimate of the development costs for a major product. As for any project, these costs include labor and materials. Estimating these costs is a matter of experience, which is why it usually is left to the more senior engineers. However, some additional costs must not be forgotten when developing embedded microcontrollers:

- Development systems and development software
- PROM or other device programmers
- ROM mask charges and other NREs
- RTOS licensing fees