seems that we must go to a much faster, more expensive processor to keep up with the motor, thus raising the cost of the system.

There is another solution, however. Many microprocessors have PWM outputs or timers that can be configured to operate as PWM outputs. Typical examples are the Microchip PIC 16C/17C family, the Atmel AT90S family, and the Intel 80C196 series. Using the internal PWM controller relieves the microprocessor of the burden of generating every motor current change. Instead, the processor just sends *changes* in the duty cycle (or frequency) to the PWM controller.

This is just one example of how picking the right processor can solve a real-time problem. Other examples include selecting a processor with a built-in, high-speed serial port for interprocessor communications; selecting a processor with an on-chip direct memory access (DMA) controller (more about that in a later chapter); or selecting a processor with special memory manipulation registers that will speed things up. Sometimes you can find a microcontroller that has exactly the right interface for your application, such as an on-chip LCD controller.

Development Environment

The development environment often is a key consideration. New development tools require a learning curve, and with a tight development schedule there often is no time to research, acquire, and become proficient with a new set of tools. If the company has several tens of thousands of dollars (a not unrealistic figure) invested in emulators for a specific processor, and if all the software engineers are comfortable with those tools, someone usually objects to changing processors just so an enthusiastic engineer can tinker with the latest chip. That is not much fun for the frustrated engineer, but it is an economic fact of life. This is why some companies (or subsidiaries within very large companies) expend a great deal of effort to pick a processor family they can live with for a long time.

Even if a design starts with a blank slate, however, the development tools can be a major consideration. For example, selecting a widely used processor, such as the 8031, allows you to select from a wide array of tools from a number of vendors. The capability of these tools (such as emulators) can be matched to whatever budget you have. On the other hand, the tools for some specialized processors are available only from the manufacturer, and the cost can be prohibitive.

Tools can be a major factor. If the processor choice gets down to just two, researching the cost of tools may make the decision obvious. In any event, be sure you know the cost of these tools, especially emulators from the IC manufacturer, before you make the final selection.

If you're planning to use an RTOS (real-time operating system), the choice of which one to use also may drive your processor selection. RTOSs come in various flavors, with some charging a one-time fee and others charging a license fee or

royalty for every unit you build. Some have a flat royalty; some charge a little for every module you include. I worked on a system once where one engineer wanted to embed an RTOS in four of the processors. We'd have spent around $800 per system just in RTOS license fees. Make sure you choose a processor for which a suitable RTOS is available and that the RTOS costs are compatible with your product cost.

Processing Speed Required

This is another area that is easier to get right after you have some experience with it, but a few guidelines can help:

- Add up the interrupt latencies. The processor must be fast enough that a worst-case stackup of interrupts (it *will* happen) can be handled without anything bad occurring. We'll return to this in the chapter on interrupts.
- The length of the polling loop (more about this in a later chapter) must be short enough to never miss a byte of serial data or a byte from any other interface. In an interrupt-driven system, the same considerations apply to the length of any polling loop plus the worst-case interrupt latencies.
- Note that in some cases, going to higher speeds gains nothing if wait states must be inserted to meet the memory access time requirements. We'll look at wait states in Chapter 2.

Common pitfalls about processor speed are as follows:

Confusing clock rate with processor speed. A standard 8031, for example, can accommodate an input clock of 12 MHz. So it's a 12 MHz processor, right? Wrong. The clock circuitry divides the clock by 12 because the internal logic needs several phases, or clock edges, per instruction. This yields a processor rate of 1 MHz. Many processors, such as the 80186/80188, divide the external clock by 2. The PIC-family processors divide the clock by 4, whereas the Atmel AT90S series parts do not. So, at least in raw execution speed, an 8 MHz AT90S part (8 MHz clock, 8 MHz execution rate) is faster than a 20 MHz PIC part (20 MHz clock, 5 MHz execution rate). None of these characteristics is bad unless you do not know them or do not take them into account.

Not evaluating the instruction set. The Atmel AT90S and Microchip PIC 16C/17C series parts have a fairly high execution speed. However, the reduced instruction set computer (RISC) architecture can be a real trap. For example, these parts lack sophisticated indirect (lookup table) branch capability. An indirect branch function can be constructed, but that takes some instructions. Similarly, the parts only have one branch instruction (GOTO). Conditional branches require two or more instructions to construct. Consequently, the potential execution rate is reduced by the extra code involved in manipulating