- The codebase becomes lopsided. Simple hardware devices such as the IDE interface are simple translations of the specification documents, and correspond to two classes: `IDEChannel` and `PIIX3IDEInterface`. The processor, a more complicated device, comparatively has a huge amount of code related to it. In total, it is represented by eight distinct packages and over 50 classes.
- As developers, we find that we need to become as schizophrenic as the codebase is. Crudely speaking, when you are working on hardware emulation outside the memory or processor systems, you are aiming for ultimate code clarity and modular design. When working within the processor or memory system, you are aiming for ultimate performance.

The hard hat that must be worn while working in the sensitive parts of the architecture is one of pessimistic inventiveness. In order to gain as much performance as possible, we continually have to be prepared to experiment. But even small changes to the codebase must be viewed with suspicion until they have been proven to have, at a minimum, no detrimental effect on performance.

The requirement for maximum performance at the bottlenecks of the emulation is what makes JPC an interesting project to work on and with. The remainder of this chapter concentrates on how we achieved what we believe is a maintainable and logical design, without compromising the performance of the system.

## Java Performance Tips

**The First Rule of Optimization: Don't do it.**

**The Second Rule of Optimization (for experts only): Don't do it yet.**

*—Michael A. Jackson*

Like all performance tips, the following are guidelines and not rules. Code that is well designed and cleanly coded is almost always infinitely preferable to "optimized" code. Invoke these guidelines only when either a positive effect will be seen on the design or that last drop of performance is really necessary.

*Tip #1: Object creation is bad*
Excessive object instantiation (especially of short-lived objects) will cause poor performance. This is because object churn causes frequent young generation garbage collections, and young generation garbage-collection algorithms are mostly of the "stop-the-world" type.

*Tip #2: Static is good*
If a method can be made static, then make it so. Static methods are not virtual, and so are not dispatched dynamically. Advanced VMs can inline such methods much more easily and readily than instance methods.