running other emulated instances. This technique avoids possible issues with even the most carefully thought through features available in modern hardware to support virtualization.[‡]

In the rest of this chapter, we describe the process used to prototype and develop JPC, and we show how a number of incremental improvements to design resulted in the JPC available today. We then outline more applications and implications of the unique combination of technologies that JPC represents.

Proof of Concept

The x86 PC has been around for over 30 years and has evolved through many different generations of hardware. At each stage, backward-compatibility has been maintained so that even today, an original 8086 program is likely to run on a new PC. Although this has had undoubted benefits and has contributed to the unparalleled success of the platform, it does mean the architecture is packed with extra complexity as new technologies are incorporated, in order to avoid breaking existing code. If a PC were built today from scratch, many aspects of the hardware would be substantially different, and almost certainly a lot simpler.

Nevertheless, this x86 platform is ubiquitous, with over 1 billion in the world today and over 200 million more being manufactured each year. Consequently, the most widely useful emulator will be one that targets the x86 PC architecture.

However, this is not an easy task. Just some of the hardware components that must be emulated in software include the x86 processor, hard disk (and its controller), keyboard and mouse drivers, VGA graphics card, DMA controller, PCI bus, PCI host bridge, interval timer, real-time clock, interrupt controller, and PCI ISA bridge. Each device has its own specification sheet, which must be read and translated into software. The x86 processor manual runs to 1,500 pages, and in all there are approximately 2,000 pages of technical manuals. The x86 instruction set is large, with up to 65,000 possible instructions that could be issued in a program's code, four different protection levels to set on memory pages, four different processor "modes," and in each mode the instructions can (and do) have different effects.

So before embarking on this mammoth task, it is important to assess whether the outcome will be worth it. Both the Bochs and QEMU projects have achieved this feat, and some reassurance can be gained by examining how they approached the problems. However, these projects are C/C++ programs and therefore give only limited assistance, as JPC has to stay within the pure Java environment with the extra design restrictions and performance considerations that implies.

Some simplification is possible by selecting a simple set of hardware to emulate. As the processor emulation will be the major bottleneck in the system, there is little point in emulating

[‡] For example, the hardware-supported x86 CPU virtualization (Intel VT, AMD Pacifica) has security vulnerabilities due to the shared L1/L2 cache of multicore chips.