computers are bad

You are receiving this facsimile because you signed up for fax delivery of this newsletter. To stop delivery, contact Computers Are Bad by email or fax.

https://computer.rip - me@computer.rip - fax: +1 (505) 926-5492

2022-12-17 the keyboard controller

One of the more common arguments you hear in contemporary computer security circles is about hardware trust and embedded controllers. Technologies like Intel ME, the ARM Secure Element, TPMs, etc, remind us that the architecture of the modern computer is complex and not entirely under our control. A typical modern computer contains a multitude of different independent processors, some running relatively complete software stacks. They come from different vendors, serve different purposes, and are often unified only by opacity: for several different reasons, the vendors of these controllers don't like to discuss the details of the implementation.

It's tragic how the modern PC has put us into this situation, where we no longer have control or even visibility into the working of core, privileged components of our computers---components running software that could potentially be malicious. By the modern PC I do, of course, mean the IBM PC of 1981.

I don't want to belabor this post with much background, but if you are quite new to the world of computer history I will briefly state one of the field's best-known facts: for reasons that are ultimately more chance than logic, the original IBM PC established many *de facto* standards that are still used in computers today. "PC compatibles," in the 1980s meaning computers that could run software targeted originally at the IBM PC, had to duplicate its architecture rather exactly. The majority of modern computers, with Apple products as a partial exception, are directly descended from these PC compatibles and are thus strongly influenced by them.

Let's talk a little bit about the architecture of the IBM PC, although I'm going to pull a slight trick and switch gears to the 1984 PC/AT, which has more in common with modern computers. By architecture here I don't mean the ISA or even really anything about the processor, but rather the architecture of the mainboard: the arrangement of the data and address busses, and the peripheral controllers attached to them. The 80286 CPU at the core of the IBM PC had an 16-bit data bus and a 24-bit address bus. Together, these were called the system bus.

The system bus connected, on the mainboard, to a variety of peripheral devices. The RAM and ROM sat on the bus, as well as the 8254 timer, the 8259 interrupt controller (actually two of them), the 8237 DMA controller (once again, two of them), and the 8042 keyboard controller.

We are going to talk about the keyboard controller. See, there's something sort of interesting about these peripheral controllers. Most of them are purpose-built ICs, like the 8237 which was designed bottom to top as a DMA controller (actually by AMD, and used under license by Intel). The 8042, though, is not really a keyboard controller. It's a general-purpose microcontroller from the same product line used as a CPU in some early video game consoles. The 8042 on the PC/AT mainboard was simply programmed with software that made it function as a keyboard controller, reading scancodes from the keyboard, interrupting the CPU (via the 8259), and reporting the scancode read on the system bus.

The actual software on the 8042 is poorly known, and seemed to vary in its details from one model to the next (this is one of the things that could create subtle compatibility issues between ostensibly PC-compatible computers). In fact, the OS/2 museum reports that the software of the 8042 on the PC/AT itself wasn't dumped and analyzed until 2009-2010. And IBM, of course, was not the only vendor of 8042-based keyboard controllers. For decades following, various manufacturers offered keyboard controllers intended to replicate the function of IBM's own software.

These keyboard controllers were known to have a number of undocumented commands, the details of which depended on the mainboard vendor. And most interestingly, since they *were* general-purpose microcontrollers and had spare capacity after handling the keyboard, the 8042 in practice served as more of a general-purpose embedded controller. The term "keyboard controller" is a misnomer in this way, and it would probably be more accurate to call it an "embedded controller," but that term wasn't yet in use in the mid '80s.

The sort of miscellaneous functions of the 8042 are well-illustrated by the A20 mask in the PC/AT. Explaining this requires brief IBM PC backstory: the original PC used the 8086 processor, which had a 20-bit address bus. The 80286 in the PC/AT offered a 24-bit address bus, allowing it to address appreciably more memory (16MB as compared to 1MB). The problem is that the 8086 had a mostly accidental behavior in which memory addresses beyond the 20-bit limit (the details of this relate to the unusual addressing mode used by the 8086) would wrap back to zero and instead address early sections of memory. Some software written in the 8086 period actively exploited this behavior as an optimization, and some of that software was quite popular. Suddenly, on the PC AT, these just-past-1MB addresses actually referred to the new upper section of memory. Software that assumed it could address early memory by wrapping past the 1MB limit was in for a surprise---typically one that halted execution.

The 80386 would introduced "virtual" mode that addressed this problem by emulating the 8086 in a bug-compatible way, but the 80286 lacked this feature. To make sure that existing IBM PC software would be usable on the PC/AT, IBM introduced a trick to the architecture: the A20 gate. An extra logic gate on the motherboard would force the A20 line (the 21st bit of the address bus, due to zero-indexing) to zero, restoring the original "wraparound" behavior. For older software to run, the A20 gate should be "closed" for compatibility. To use memory beyond 1MB, though, the A20 gate needed to be "open" so that the second MB of memory could be addressed.

Anyone who has studied the boot process of modern computers knows that it is littered with the detritus of many generations of awkward additions. The PC/AT was sort of the genesis of this situation, as the 80286 introduced the concept of "real" vs. "protected" mode and the need for the operating system or bootloader to switch between them. Along with this came a less widely known need, to also manage the A20 gate. On boot, the A20 gate was closed. Software (such as an operating system) that wanted to use memory past 1MB had to open it. Given that the A20 gate was just a random bit of extra logic in the mainboard, though, how would software interact with it?

The answer, of course, is the keyboard controller. The 8042 could be sent a command that would it cause it to open the A20 gate. In fact, this wasn't the only role the 8042 served in the basic management of the processor. The 80286 couldn't switch from protected mode back to real mode without a reset, so for the computer to switch between real mode and protected mode dynamically (such as for multitasking with legacy software) something had to "poke" the reset line on demand. Once again, the PC/AT placed the 8042 keyboard controller in this role.

Over time, the 8042 tended to accumulate more roles. It handled the mouse as well in later computers with PS/2-type mice. It drove the PC speaker on some motherboards. Some early laptops used the 8042 for power management or display functions. None of these things were ever really standardized and often become a headache today.

Over time, the number of discrete components on motherboards has plummeted. The 8042 fell victim to this process fairly early on. On most PC-compatible computers, the 8042 gave way to the "Super I/O controller." This larger genera-purpose microcontroller combined even more functions, combining the serial, parallel, floppy, and keyboard/mouse controllers into one device that also handled auxiliary functions like fan speed control, MIDI/game port (different functions but for practical reasons usually packaged together), power management, and yes, the A20 gate and processor reset. The Super I/O is still more or less present in computers today, but it's usually referred to as the Embedded Controller or EC in modern architectures. The EC is often connected to modern processors over the LPC or "low pin count" bus, which amusingly is a somewhat modernized version of the original ISA expansion architecture on the PC/AT... long ago replaced by AGP, PCI, PCIe, etc. for most purposes.

That said, all of these terms are becoming somewhat irrelevant today. Modern processor architectures are pushing perpetually further into unified chipset designs in which progressively larger controllers of various names integrate more and more functions into one device. I learned about computer architecture in the days in which the "northbridge" and "southbridge" were the major active components of the system bus... but few computers today have discrete north and south bridges (or front-side and back-side buses for that matter), the functions of the two having been integrated into various combinations of the processor die and a large, monolithic motherboard chipset like the platform controller hub (PCH), with the functions of essentially all motherboard components put into one IC.

The 8042 thing all feels like sort of a hack, and indeed it was, but the PC and PC/AT (and PS/2 for that matter) architectures were full of such hacks. The IBM PC had a device we've mentioned, the PIC or programmable interrupt controller, that served to multiplex 7 different interrupt lines onto the processor's single IRQ pin. By the release of the PC/XT, all seven of those interrupts were in use on many machines, so IBM decided to get some more... by daisy chaining a second PIC onto the interrupt 2 line of the first one. In other words, interrupts 8-15 on a PC/AT (and subsequent PC compatibles) are in fact interrupts 0-7 on the second PIC. When the second PIC receives an interrupt, it forwards it to the first PIC by triggering interrupt 2, which then interrupts the processor.

Conventionally the keyboard used interrupt 1, so the 8042 was wired up to the interrupt 1 line of the PIC. The mouse, though, assuming a PS/2 mouse, used interrupt 12... reflecting the fact that the interrupt-capable mouse interface introduced on the IBM PS/2 was copied to the PC/AT, at the same time that the second PIC was added. In other words, the mouse is one of the things that they had run out of interrupts for. Of course today, we use keyboards and mice primarily via a polling strategy over USB.

This kind of hardware-level hackjob is not at all a thing of the '80s, and most modern systems still have plenty of it going on. Two of the most common cases are the need to reset the processor for various reasons and low-power modes, where something needs to wake up the processor... ideally something that consumes less power than keeping the processor awake. I am dealing with a microcontroller device right now, for example, where the embedded capacitative touch controller does double-duty to wake up the processor from deep sleep state on certain events. This allows the processor to shut off its own power, a huge savings considering the tiny draw of keeping the capacitative touch controller powered. It's oddly like the original 8042, taking some input controller that's a little more powerful than it needs to be and giving it some miscellaneous extra tasks.