What is the developer set-up for developing Windows for multiple processor architectures?

devblogs.microsoft.com/oldnewthing/20250513-00/?p=111176

May 13, 2025

The operating systems in the Windows NT lineage have supported multiple processor architectures for nearly all of its existence. There was a brief period of darkness when Compaq ceased production of Alpha AXP systems and before Intel released its Itanium processors. But even during that period, the Alpha AXP systems were still being used, just not in ways visible to the public: They were being used to develop 64-bit Windows. So there was never a period of time where Windows *developers* weren't developing for multiple processor architectures. How did this work? Did every developer have a machine of every architecture in their office to validate their work on every processor?

No, it didn't work that way.

Most developers had an Intel-class system as their primary machine, and another Intel-class system as their test machine. Some developers had a second test machine which was one of the other architectures. And some developers were all-in and had a non-Intel-class system as their primary machine.

I believe Jon Vert, the original author of the Windows NT blue screen, had a MIPS system as his primary system, and then when support for MIPS was dropped, he switched to an Alpha AXP as his main machine. That's dedication to the non-Intel architectures.

For a long time, Windows NT did not have cross-compilers. If you wanted to compile for MIPS, you needed to have a MIPS system, for example. Every team knew who their representatives were for the non-Intel architectures, and if you needed to test your changes on one of those systems, you would ask them to build a binary for you, and you could go to their office to run your changes on a test system.

Most of the time, this step wasn't necessary. Since Windows NT was designed as a portable operating system, the architectural differences rarely showed up in how you wrote your code, so if it worked on one 32-bit architecture, it probably worked on the others. We used the same principle when porting Windows to 64-bit: The vast majority of the work was in the initial port from 32-bit processors to 64-bit Alpha AXP. The port from 64-bit Alpha AXP to 64-bit Intel Itanium was handled almost entirely by the kernel team, since it is deep inside the kernel that the architectural differences become significant.

There are still places where architectural differences are visible in high-level code, however. The two biggest ones were multithreading and misaligned data. The Intel processors have rather strict memory models, so you didn't observe race conditions that would be problems on systems with weaker memory models. And the Intel processors are quite forgiving of misaligned data, so code that used misaligned pointers would run just fine on an Intel system and then crash horribly on a processor that enforces alignment more strictly. To catch those types of issues, we had to exercise vigilance during code reviews and supplement it with a lot of testing to exercise the new code paths.

The Windows division still works this way. Windows supports Intel x86-32, Intel x86-64, ARM AArch32, and ARM AArch64. Most developers have an Intel x86-64 system as their main system and run Intel x86-32 and Intel x86-64 virtual machines for testing. Developers who are working with ARM-specific features will usually have an Intel x86-64 system as their main system and an ARM test system. Fortunately, we now have cross-compiling, so you no longer need to have a build environment on your ARM system. You can build an ARM binary on your Intel system.

Bonus reading: Anybody who writes `#pragma pack(1)` may as well just wear a sign on their forehead that says "I hate RISC."