

On how different Windows ABIs choose how to pass 32-bit values in 64-bit registers

 devblogs.microsoft.com/oldnewthing/20250324-00

Ben Voigt

March 24, 2025



Raymond Chen

Different 64-bit processor ABIs have different policies on how 32-bit bit values are passed in 64-bit registers. Let's see if we can find a pattern among the Windows ABIs.



Processor	32-bit signed value	32-bit unsigned value
AArch64	Garbage	Garbage
Alpha AXP	Sign extend	Sign extend
ia64	Garbage	Garbage
MIPS64	Sign extend	Sign extend
POWER3	Sign extend	Zero extend
RISC-V	Sign extend	Sign extend
x86-64	Garbage	Garbage

Contents of upper 32 bits of
64-bit register holding a 32-bit value

There are basically three groups.

- Always sign extend (Alpha AXP, MIPS64, RISC-V)
- Extend based on signedness of 32-bit type (POWER3)
- Garbage (AArch64, ia64, x86-64)

Sign-extending unsigned 32-bit types sure feels weird. I wonder why that is.

Let's regroup the processors by putting those with similar policies together. A pattern emerges when you add another column: Does this processor support comparison instructions (such as conditional branch instructions) that operate only on the lower 32 bits of a register?

Processor	Policy	Can compare 32-bit values?
Alpha AXP	Always sign extend	No
MIPS64	Always sign extend	No
RISC-V	Always sign extend	No
POWER3	Use signedness of 32-bit type	Yes
AArch64	Garbage	Yes
ia64	Garbage	Yes
x86-64	Garbage	Yes

Contents of upper 32 bits of
64-bit register holding a 32-bit value

The architectures whose ABIs require that 32-bit values be sign-extended (even for unsigned types) to 64-bit values are precisely those which do not have the ability to compare 32-bit values.

If your processor can only compare full 64-bit values, then sign extending everything (even unsigned types) is the way to go because that allows you to use the 64-bit comparison instruction for 32-bit comparison, too!

For signed comparisons, sign extending 32-bit values to 64-bit values preserves the mathematical values, so the 64-bit comparison produces the same result as the hypothetical 32-bit comparison.

For unsigned comparisons, sign extension changes the mathematical value of values greater than or equal to 2^{31} , but in a consistent manner: They are all increased by `0xFFFFFFFF`00000000`. The relative order of the values doesn't change, so the results of comparisons did not change. The numbers are still in the same relative order.

Zero-extending 32-bit values to 64-bit values would result in negative 32-bit values comparing greater than positive 32-bit values when compared as 64-bit signed values, so that's not going to work.

POWER3's policy of extending the value according to the signed-ness of the underlying type would also work, and it also avoids the phenomenon of $-1 > 0U$, which is something that catches out beginners. Unfortunately, the C and C++ languages actually require that $-1 > 0U$ due to the signed-to-unsigned conversion rules, so this benefit goes wasted in those languages.

But at least the original mystery is solved. If your processor doesn't support 32-bit comparisons, then sign-extending all 32-bit values (even the unsigned ones) is the natural choice.

Bonus chatter: Of course, processor designers are aware of these issues when they design their instruction set. Nowadays, we don't have people trying to retrofit an ABI onto a newly-released processor. Rather, processor designers realize, "If we recommend an ABI that requires 32-bit parameters to be sign-extended to 64-bit values, then we can remove all the 32-bit comparison instructions from our instruction set!"

Bonus reading: [What are the dire consequences of having 32-bit values in non-canonical form on Alpha AXP.](#)

Author

[Raymond Chen](#)

Raymond has been involved in the evolution of Windows for more than 30 years. In 2003, he began a Web site known as The Old New Thing which has grown in popularity far beyond his wildest imagination, a development which still gives him the heebie-jeebies. The Web site spawned a book, coincidentally also titled The Old New Thing (Addison Wesley 2007). He occasionally appears on the Windows Dev Docs Twitter account to tell stories which convey no useful information.



11 comments

Discussion is closed. [Login to edit/delete existing comments.](#)

Newest



March 26, 2025

I've always thought this was "zero fill" (or "zero pad"), because it's not extending from an existing zero.



Rosyna Keller March 25, 2025

"Nowadays, we don't have people trying to retrofit an ABI onto a newly-released processor."

Is this a jab at ARM64EC?




Aaron Giles

One additional consideration (at least when we were discussing the ABI for AArch64) is security.

Sure, the ABI can specify that parameters are to be sign-extended, but any callee that trusts the caller to sign-extend its parameters is leaving open a potential attack surface. It's safer for the callee to assume nothing about the upper bits. And once you bake in that assumption, you've effectively pushed any sign/zero-extension logic onto the callee, so might as well leave those upper bits undefined.



Raymond Chen  Author

There's no point trying to defend against an in-process attacker. They already are inside the house. Sanitizing the upper 32 bits is the job of the marshaler.



Raymond Chen  Author 2 weeks ago

There is some component that receives the inbound request. It receives some data blob that originated from outside the process (by reading from a network socket, or receiving a chunk of shared memory), and it then has to parse that data blob and turn it into a function call. The outsider doesn't pass registers; it passes blobs. I guess if your data blob is "here's a bunch of values to shove into registers", then you need to validate those values before actually putting them into registers.



Aaron Giles March 25, 2025

Within a process, true, but if you export your APIs to the outside world then the issue is something to consider.

For example, if you are porting an API surface that numbers in the many thousands of exposed calls from one platform to AArch64, it's a security win if the compiler doesn't make assumptions about the cleanliness of incoming data.



Joshua Hudson March 25, 2025

I'm not convinced Aaron Giles isn't talking about the system call gate and its corresponding calling convention; which typically looks almost exactly like the function calling convention.



David Taylor

“sign extension changes the mathematical value of values greater than or equal to 2^{32} ”

Should presumably say $\geq 2^{31}$.



Brian Boorman

Isn't it actually $\geq 2^{32} - 1$?



Brian Boorman March 26, 2025

Raymond said:

“ $2^{31} - 1$ is 0x7FFFFFFF, and sign extension does not change that value.”

Sure, but my comment (correct or not) said “ $2^{32} - 1$ ” which is the value 0xFFFF_FFFF



Raymond Chen  Author March 26, 2025

” $\geq 2^{31}$ ” is correct (thanks). $2^{31} - 1$ is 0x7FFFFFFF, and sign extension does not change that value.

Stay informed

Get notified when new posts are published.