When is x/2 different from x>>1?

devblogs.microsoft.com/oldnewthing/20050527-25

May 27, 2005

Raymond Chen

Everyone “knows” that the following pairs of expressions are equivalent:

`x*2 ≡ x<<1`

`x/2 ≡ x>>1`

Too bad they aren’t. In the C language standard, there is no requirement that the internal representation of signed integers be two’s complement. All the permissible representations agree for positive numbers, but negative numbers can have different representations. If `x` is negative, then `x*2` and `x<<1` are quite different on a sign/magnitude system. However, Win32 requires a two’s complement machine, in which case the first equivalence `x*2 ≡ x<<1` is indeed always true. Of course, the compiler is free to recognize this and rewrite your multiplication or shift operation. In fact, it is very likely to do this, because `x+x` is more easily pairable than a multiplication or shift. Your shift or multiply-by-two is probably going to be rewritten as something closer to an `add eax, eax` instruction. As for the second so-called equivalence, the C language specification originally did not specify whether division of a negative number by a positive number rounds towards or away from zero, but in 1999, the specification was revised to require rounding towards zero. Furthermore, the result of a right-shift of a negative value is unspecified, so the expression `x>>1` has an unspecified result if `x` is negative.

Even if you assume that the shift fills with the sign bit, The result of the shift and the divide are different if `x` is negative.

`(-1) / 2 ≡ 0`

`(-1) >> 1 ≡ -1`

The moral of the story is to write what you mean. If you want to divide by two, then write “ `/2` “, not “ `>>1` “.

Raymond Chen

Follow