Integer signum in SSE

devblogs.microsoft.com/oldnewthing/20141229-00

December 29, 2014

Raymond Chen

The signum function is defined as follows:

signum(x) =	−1	if $x < 0$
signum(x) =	0	if $x = 0$
signum(x) =	+1	if $x > 0$

There are a couple of ways of calculating this in SSE integers.

One way is to convert the C idiom

```
int signum(int x) { return (x > 0) - (x < 0); }
```

The SSE translation of this is mostly straightforward. The quirk is that the SSE comparison functions return −1 to indicate `true`, whereas C uses +1 to represent `true`. But this is easy to take into account:

$x > 0$	⇔	− pcmpgt(x, 0)
$x < 0$	⇔	− pcmpgt(0, x)

Substituting this into the original `signum` function, we get

signum(x) =	($x > 0$)	−	($x < 0$)
=	− pcmpgt(x, 0)	−	− pcmpgt(0, x)
=	− pcmpgt(x, 0)	+	pcmpgt(0, x)
=	pcmpgt(0, x)	−	pcmpgt(x, 0)

In assembly:

```
        ; assume x is in xmm0
        pxor    xmm1, xmm1
        pxor    xmm2, xmm2
        pcmpgtw xmm1, xmm0 ; xmm1 = pcmpgt(0, x)
        pcmpgtw xmm0, xmm2 ; xmm0 = pcmpgt(x, 0)
        psubw   xmm0, xmm1 ; xmm0 = signum
        ; answer is in xmm0
```

With intrinsics:

```
__m128i signum16(__m128i x)
{
    return _mm_sub_epi16(_mm_cmpgt_epi16(_mm_setzero_si128(), x),
                         _mm_cmpgt_epi16(x, _mm_setzero_si128()));
}
```

This pattern extends *mutatus mutandis* to `signum8`, `signum32`, and `signum64`.

Another solution is to use the signed minimum and maximum opcodes, using the formula

$$\text{signum}(x) = \min(\max(x, -1), +1)$$

In assembly:

```
        ; assume x is in xmm0
        pcmpgtw xmm1, xmm1 ; xmm1 = -1 in all lanes
        pmaxsw  xmm0, xmm1
        psrlw   xmm1, 15   ; xmm1 = +1 in all lanes
        pminsw  xmm0, xmm1
        ; answer is in xmm0
```

With intrinsics:

```
__m128i signum16(__m128i x)
{
    // alternatively: minusones = _mm_set1_epi16(-1);
    __m128i minusones = _mm_cmpeq_epi16(_mm_setzero_si128(),
                                        _mm_setzero_si128());
    x = _mm_max_epi16(x, minusones);
    // alternatively: ones = _mm_set1_epi16(1);
    __m128i ones = _mm_srl_epi16(minusones, 15);
    x = _mm_min_epi16(x, ones);
    return x;
}
```

The catch here is that SSE2 supports only 16-bit signed minimum and maximum; to get other bit sizes, you need to bump up to SSE4. But if you're going to do that, you may as well use the `psign` instruction. In assembly:

```
        ; assume x is in xmm0
        pcmpgtw xmm1, xmm1 ; xmm1 = -1 in all lanes
        psrlw   xmm1, 15   ; xmm1 = +1 in all lanes
        psignw  xmm1, xmm0 ; apply sign of x to xmm1
        ; answer is in xmm1
```

With intrinsics:

```
__m128i signum16(__m128i x)
{
    // alternatively: ones = _mm_set1_epi16(1);
    __m128i minusones = _mm_cmpeq_epi16(_mm_setzero_si128(),
                                        _mm_setzero_si128());
    __m128i ones = _mm_srl_epi16(minusones, 15);
    return _mm_sign_epi16(ones, x);
}
```

The `psign` instruction applies the sign of its second argument to its first argument. We load up the first argument with the value `+1` in all lanes, then apply the sign of *x*, which negates the value if the corresponding lane of *x* is negative; sets the value to zero if the lane is zero, and leaves it alone if the corresponding lane is positive.

Raymond Chen

Follow