Notes on calculating constants in SSE registers

devblogs.microsoft.com/oldnewthing/20141215-00

December 15, 2014

Raymond Chen

There are a few ways to load constants into SSE registers.

- Load them from memory.
- Load them from general purpose registers via `movd`.
- Insert selected bits from general purpose registers via `pinsr[b|w|d|q]`.
- Try to calculate them in clever ways.

Loading constants from memory incurs memory access penalties. Loading or inserting them from general purpose registers incurs cross-domain penalties. So let's see what we can do with clever calculations.

The most obvious clever calculations are the ones for setting a register to all zeroes or all ones.

```
    pxor    xmm0, xmm0 ; set all bits to zero
    pcmpeqd xmm0, xmm0 ; set all bits to one
```

These two idioms are special-cased in the processor and execute faster than normal pxor and pcmpeqd instructions because the results are not dependent on the previous value in `xmm0`.

There's not much more you can do to construct other values from zero, but a register with all bits set does create additional opportunities.

If you need a value loaded into all lanes whose bit pattern is either a bunch of 0's followed by a bunch of 1's, or a bunch of 1's followed by a bunch of 0's, then you can shift in zeroes. For example, assuming you've set all bits in `xmm0` to 1, here's how you can load some other constants:

```
    pcmpeqd xmm0, xmm0 ; set all bits to one
-then-
    pslld  xmm0, 30    ; all 32-bit lanes contain 0xC0000000
-or-
    psrld  xmm0, 29    ; all 32-bit lanes contain 0x00000007
-or-
    psrld  xmm0, 31    ; all 32-bit lanes contain 0x00000001
```

Intel suggests loading 1 into all lanes with the sequence

```
    pxor    xmm0, xmm0 ; xmm0 = { 0, 0, 0, 0 }
    pcmpeqd xmm1, xmm1 ; xmm1 = { -1, -1, -1, -1 }
    psubd   xmm0, xmm1 ; xmm0 = { 1, 1, 1, 1 }
```

but that not only takes more instructions but also consumes two registers, and registers are at a premium since there are only eight of them. The only thing I can think of is that `psubd` might be faster than `psrld`.

In general, to load `2^n−1` into all lanes, you do

```
    pcmpeqd xmm0, xmm0 ; set all bits to one
-then-
    psrlw  xmm0, 16-n  ; clear top 16-n bits of all 16-bit lanes
-or-
    psrld  xmm0, 32-n  ; clear top 32-n bits of all 32-bit lanes
-or-
    psrlq  xmm0, 64-n  ; clear top 64-n bits of all 64-bit lanes
```

Conversely, if you want to load `~(2^n−1) = -2^n` into all lanes, you shift the other way.

```
    pcmpeqd xmm0, xmm0 ; set all bits to one
-then-
    psllw  xmm0, n     ; clear bottom n bits of all 16-bit lanes = 2^16 - 2^n
-or-
    pslld  xmm0, n     ; clear bottom n bits of all 32-bit lanes = 2^32 - 2^n
-or-
    psllq  xmm0, n     ; clear bottom n bits of all 64-bit lanes = 2^64 - 2^n
```

And if the value you want has all its set bits in the middle, you can combine two shifts (and stick something in between the two shifts to ameliorate the stall):

```
    pcmpeqd xmm0, xmm0 ; set all bits to one
-then-
    psrlw  xmm0, 13    ; all lanes = 0x0007
    psllw  xmm0, 4     ; all lanes = 0x0070
-or-
    psrld  xmm0, 31    ; all lanes = 0x00000001
    pslld  xmm0, 3     ; all lanes = 0x00000008
```

If you want to set high or low lanes to zero, you can use `pslldq` and `psrldq`.

```
    pcmpeqd xmm0, xmm0 ; set all bits to one
-then-
    pslldq xmm0, 2     ; clear bottom word, xmm0 = { -1, -1, -1, -1, -1, -1, -1, 0 }
-or-
    pslldq xmm0, 4     ; clear bottom dword, xmm0 = { -1, -1, -1, 0 }
-or-
    pslldq xmm0, 8     ; clear bottom qword, xmm0 = { -1, 0 }
-or-
    psrldq xmm0, 2     ; clear top word, xmm0 = { 0, -1, -1, -1, -1, -1, -1, -1 }
-or-
    psrldq xmm0, 4     ; clear top dword, xmm0 = { 0, -1, -1, -1 }
-or-
    psrldq xmm0, 8     ; clear top qword, xmm0 = { 0, -1 }
```

No actual program today. Just some notes from my days writing SSE assembly language.

Bonus chatter: There is an intrinsic for `pxor xmmReg, xmmReg`: `_mm_setzero_si128`. However, there is no corresponding intrinsic for `pcmpeqd xmmReg, xmmReg`, which would presumably be called `_mm_setones_si128` or `_mm_setmone_epiNN`. In order to get all-ones, you need to get a throwaway register and compare it against itself. The cheapest throwaway register is one that is set to zero, since that is special-cased inside the processor.

```
__m128i zero = _mm_setzero_si128();
__m128i ones = _mm_cmpeq_epi32(zero, zero);
```

Raymond Chen

Follow