More notes on calculating constants in SSE registers

devblogs.microsoft.com/oldnewthing/20150105-00

January 5, 2015

Raymond Chen

A few weeks ago I noted some tricks for creating special bit patterns in all lanes, but I forgot to cover the case where you treat the 128-bit register as one giant lane: Setting all of the least significant *N* bits or all of the most significant *N* bits.

This is a variation of the trick for setting a bit pattern in all lanes, but the catch is that the `pslldq` instruction shifts by bytes, not bits.

We'll assume that *N* is not a multiple of eight, because if it were a multiple of eight, then the `pslldq` or `psrldq` instruction does the trick (after using `pcmpeqd` to fill the register with ones).

One case is if *N* ≤ 64. This is relatively easy because we can build the value by first building the desired value in both 64-bit lanes, and then finishing with a big `pslldq` or `psrldq` to clear the lane we don't like.

; set the bottom *N* bits, where *N* ≤ 64									
`pcmpeqd xmm0, xmm0`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift right 64 − *N* bits				unsigned shift right 64 − *N* bits			
`psrlq   xmm0, 64 - N`	;	0000	0000	0FFF	FFFF	0000	0000	0FFF	FFFF
		unsigned shift right 64 bits							
`psrldq  xmm0, 8`	;	0000	0000	0000	0000	0000	0000	0FFF	FFFF

; set the top *N* bits, where *N* ≤ 64

`pcmpeqd xmm0, xmm0`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift left 64 − *N* bits				unsigned shift left 64 − *N* bits			
`psllq   xmm0, 64 - N`	;	FFFF	FFF0	0000	0000	FFFF	FFF0	0000	0000
		unsigned shift left 64 bits							
`pslldq  xmm0, 8`	;	FFFF	FFF0	0000	0000	0000	0000	0000	0000

If *N* ≥ 80, then we shift in zeroes into the top and bottom half, but then use a shuffle to patch up the half that needs to stay all-ones.

; set the bottom *N* bits, where *N* ≥ 80									
`pcmpeqd xmm0, xmm0`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift right 128 − *N* bits				unsigned shift right 128 − *N* bits			
`psrlq   xmm0, 128 - N`	;	0000	0000	0FFF	FFFF	0000	0000	0FFF	FFFF
		copy				shuffle			↓
		↓	↓	↓	↓				↓
`pshuflw xmm0, _MM_SHUFFLE(0, 0, 0, 0)`	;	0000	0000	0FFF	FFFF	FFFF	FFFF	FFFF	FFFF

; set the top *N* bits, where N ≥ 80									
`pcmpeqd xmm0, xmm0`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift left 128 − *N* bits				unsigned shift left 128 − *N* bits			
`psllq   xmm0, 128 - N`	;	FFFF	FFF0	0000	0000	FFFF	FFF0	0000	0000

		↓	shuffle			copy			
		↓				↓	↓	↓	↓
`pshufhw xmm0, _MM_SHUFFLE(3, 3, 3, 3)`	`;`	FFFF	FFFF	FFFF	FFFF	FFFF	FFF0	0000	0000

We have $N \geq 80$, which means that $128 - N \leq 48$, which means that there are at least 16 bits of ones left in low-order bits after we shift right. We then use a 4×16-bit shuffle to copy those known-all-ones 16 bits into the other lanes of the lower half. (A similar argument applies to setting the top bits.)

This leaves $64 < N < 80$. That uses a different trick:

`;` set the bottom *N* bits, where *N* ≤ 120

`pcmpeqd xmm0, xmm0`	`;`	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift right 8 bits							
`psrldq  xmm0, 1`	`;`	00FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		signed shift right 120 − *N* bits				signed shift right 120 − *N* bits			
`psrad   xmm0, 120 - N`	`;`	0000	00FF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF

The sneaky trick here is that we use a *signed* shift in order to preserve the bottom half. Unfortunately, there is no corresponding left shift that shifts in ones, so the best I can come up with is four instructions:

`;` set the top *N* bits, where 64 ≤ *N* ≤ 96

`pcmpeqd xmm0, xmm0`	`;`	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
		unsigned shift left 96 − *N* bits				unsigned shift left 96 − *N* bits			
`psllq   xmm0, 96 - N`	`;`	FFFF	FFFF	FFF0	0000	FFFF	FFFF	FFF0	0000

		shuffle							
		↓				↓		↓	
`pshufd xmm0, _MM_SHUFFLE(3, 3, 1, 0)`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFF0	0000
		unsigned shift left 32 bits							
`pslldq xmm0, 4`	;	FFFF	FFFF	FFFF	FFFF	FFFF	FF00	0000	0000

We view the 128-bit register as four 32-bit lanes. split the shift into two steps. First, we fill Lane 0 with the value we ultimately want in Lane 1, then we patch up the damage we did to Lane 2, then we do a shift the 128-bit value left 32 places to slide the value into position and zero-fill Lane 0.

Note that a lot of the ranges of *N* overlap, so you often have a choice of solutions. There are other three-instruction solutions I didn't bother presenting here. The only one I couldn't find a three-instruction solution for was setting the top *N* bits where 64 < *N* < 80.

If you find a three-instruction solution for this last case, share it in the comments.

Raymond Chen

Follow