Setting, clearing, and testing a single bit in an SSE register

devblogs.microsoft.com/oldnewthing/20141222-00

December 22, 2014

Raymond Chen

Today I'm going to set, clear, and test a single bit in an SSE register.

Why?

On Mondays I don't have to explain why.

First, we use the trick from last time that lets us generate constants where all set bits are contiguous, and apply it to the case where we want only one bit.

```
    pcmpeqd xmm0, xmm0      ; set all bits to one
    psrlq   xmm0, 63        ; set both 64-bit lanes to 1
IF N LT 64
    psrldq  xmm0, 64 / 8    ; clear the upper lane
ELSE
    pslldq  xmm0, 64 / 8    ; clear the lower lane
ENDIF
IF N AND 63
    psllq   xmm0, N AND 63  ; shift the bit into position
ENDIF
```

We start by setting all bits in `xmm0`.

We then shift both 64-bit lanes right by 63 positions, putting 1 in each lane.

If the bit we want is in the upper half, then we shift the entire value left 8 bytes (64 bits). This clears the bottom 64 bits and leaves the upper 64 bits with all bits set. (Similarly, if the bit we want is in the lower half, shifting right instead of left.)

Finally, if we need a bit other than 0 or 64, we shift left by the desired amount within the 64-bit lane.

Now that we can generate a single bit value, we can use it to set and clear individual bits.

```
; Set bit N in xmm1 (using xmm0 as a helper)
        ‹set xmm0 = 2^N›
        por     xmm1, xmm0
; Clear bit N in xmm1 (putting result in xmm0)
        ‹set xmm0 = 2^N›
        pandn   xmm0, xmm1
```

To test a bit, we can use the `PMOVMSKB` instruction.

```
IF 7 - (N AND 7)
    psllq xmm0, 7 - (N AND 7)
ENDIF
    pmovmskb eax, xmm0
IF N LT 64
    test  al, 1 SHL (N / 8)
ELSE
    test  ah, 1 SHL (N / 8 - 8)
ENDIF
```

First, we move the bit we want to test into a position that is 7 mod 8, because those are the bits captured by the `PMOVMSKB` instruction. (If the bit is already there, then we don't need to do anything.) Then we use the `PMOVMSKB` instruction to extract the bits into a general purpose register and test the one that corresponds to the bit we want.

Alternatives: I tend to stick to SSE2 instructions because they are widely supported (and are indeed part of the minimum system requirements for Windows 8), but if you are willing to do CPU dispatching on SSE4, you can use `PTEST`, which might be faster, I haven't tested it.

You could use `movd` and `movq` to load up a constant, but you do incur domain crossing penalties. Another alternative is to put the constant in memory, but then you pay an even bigger cost for memory access if the value is not in cache.

Other remarks: Of course, you want to schedule the instructions better than the way I wrote them above. I wrote them in a logical order above to make the algorithm clearer, but you will want to reorder them to avoid stalls.

Using intrinsics:

```
__m128i Calc2ToTheN(int N)
{
 __m128i zero = _mm_setzero_si128();
 __m128i ones = _mm_cmpeq_epi32(zero, zero);
 __m128i onesLowHigh = _mm_slli_epi64(ones, 63);
 __m128i singleOne = N < 64 ? _mm_srli_si128(onesLowHigh, 64 / 8) :
                              _mm_slli_si128(onesLowHigh, 64 / 8);
 return _mm_slli_epi64(singleOne, N & 63);
}
__m128i SetBitN(__m128i value, int N)
{
  return _mm_or_si128(value, Calc2ToTheN(N));
}
__m128i ClearBitN(__m128i value, int N)
{
  return _mm_andnot_si128(value, Calc2ToTheN(N));
}
__m128i TestBitN(__m128i value, int N)
{
 __m128i positioned = _mm_slli_epi64(value, 7 - (N & 7));
 return (_mm_movemask_epi8(positioned) & (1 << (N / 8))) != 0;
}
```

Note that since these functions pass a non-constant value to intrinsics like `_mm_slli_epi64`, you incur additional runtime penalties because the compiler is going to use a `movd` to load up the value, incurring the exact domain crossing penalty we are trying to avoid. To avoid this, templatize the function to force the bit number to be determined at compile time.

```
template<int N>
__m128i Calc2ToTheN()
{
 __m128i zero = _mm_setzero_si128();
 __m128i ones = _mm_cmpeq_epi32(zero, zero);
 __m128i onesLowHigh = _mm_slli_epi64(ones, 63);
 __m128i singleOne = N < 64 ? _mm_srli_si128(onesLowHigh, 64 / 8) :
                              _mm_slli_si128(onesLowHigh, 64 / 8);
 return _mm_slli_epi64(singleOne, N & 63);
}
template<int N>
__m128i SetBitN(__m128i value)
{
  return _mm_or_si128(value, Calc2ToTheN<N>());
}
template<int N>
__m128i ClearBitN(__m128i value)
{
  return _mm_andnot_si128(value, Calc2ToTheN<N>());
}
template<int N>
__m128i TestBitN(__m128i value)
{
 __m128i positioned = _mm_slli_epi64(value, 7 - (N & 7));
 return (_mm_movemask_epi8(positioned) & (1 << (N / 8))) != 0;
}
```

Raymond Chen

Follow