

Re: bsr-algorithm?

Source: <http://coding.derkeiler.com/Archive/Assembler/comp.lang.asm.x86/2008-05/msg00109.html>

- *From:* "Jacek Wawrzaszek" <spamtrap@xxxxxxxxxx>
 - *Date:* Sat, 24 May 2008 02:15:14 +0200
-

I saw the discussion about the popcnt, and it reminded me of my own search of a solution to this function:

```
/* first-bit position */
static int __fastcall fbpos(short int e) {
    __asm {
        /* handle '0'-case */
        __asm xor eax, eax
        __asm dec ax
        __asm bsr ax, cx
    };
}
```

but in MMX/XMMX.

I didn't find any slight dicussion of 'emulating' lzcnt or bsr at all, and the solutions I could think of are very unsmart and probably slower than sequencially transfering the data into GPRs and do the bsr there.

I think on AltiVec helps the permutation-op, that we don't have on x86.

Anybody got an idea about a branch-free 'emulation' of bsr?

Here it is – the code below treats input_vector as four 16-bit words and performs BSR on them, leaving the results in mm0. For 0 values it returns 0xffff, just like your fbpos function.

J.

```
const __int64 _0x8000800080008000 = 0x8000800080008000;
const __int64 _0x80ff80ff80ff80ff = 0x80ff80ff80ff80ff;
const __int64 _0x000f000f000f000f = 0x000f000f000f000f;
const __int64 _0x0003000300030003 = 0x0003000300030003;
const __int64 _0x0008000800080008 = 0x0008000800080008;
const __int64 _0x0004000400040004 = 0x0004000400040004;
```

Re: bsr-algorithm?

```
const __int64 _0x0002000200020002 = 0x0002000200020002;  
const __int64 _0x0001000100010001 = 0x0001000100010001;
```

```
__asm {  
movq mm1, [ input_vector ]  
pxor mm0, mm0  
movq mm2, mm1  
pxor mm1, [ _0x8000800080008000 ]  
movq mm3, [ _0x0008000800080008 ]  
pcmpeqw mm0, mm2  
pcmpgtw mm1, [ _0x80ff80ff80ff80ff ]  
pand mm3, mm1  
pandn mm1, mm2  
psrlw mm2, 8  
paddw mm0, mm3  
por mm1, mm2  
movq mm3, [ _0x0004000400040004 ]  
movq mm2, mm1  
pcmpgtw mm1, [ _0x000f000f000f000f ]  
pand mm3, mm1  
pandn mm1, mm2  
psrlw mm2, 4  
paddw mm0, mm3  
por mm1, mm2  
movq mm3, [ _0x0002000200020002 ]  
movq mm2, mm1  
pcmpgtw mm1, [ _0x0003000300030003 ]  
pand mm3, mm1  
pandn mm1, mm2  
psrlw mm2, 2  
paddw mm0, mm3  
por mm1, mm2  
pcmpgtw mm1, [ _0x0001000100010001 ]  
psubw mm0, mm1  
}
```

.