

Re: Inline assembler reference

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2006-05/msg00714.ht>

- *From:* "Shawn B." <leabre@xxxxxxxx>
 - *Date:* Tue, 30 May 2006 14:07:24 -0700
-

Concerning the DOS club..... well, I would rather join Linux one if MSFT takes away the possibility of writing platform-specific code (according to Intel manuals, Pentium D seems to be quite interesting platform for system-level developers). In fact, I am not 100% sure MSFT will stick to its current position on this issue – there is a good chance that they will be pressed hard to remove the restrictions they want to introduce.

I really don't want to get too involved in this. I, too, am greatly dissappointed that there is no more inlining of 64-bit assembly instructions in the 64-bit c/c++ compilers. I had a project that was highly optimized that way and I would have to rewrite from scratch for the 64-bit compile and just isn't that important to me. The compiler intrinsics that we are supposed to use in its stead don't produce the same code I would by hand and yes, I do know how to beat the compiler with float/SSE* instructions. Using MASM64 and C++ to link the ml64 output into my build only cases an extra CALL instruction most of the time when inlining it could avoid those precious clock cycles in the critical loops.

Some would say that I'm clock cycle obsessed. I'm actually not. But I don't come of the opinion that just because all new machines have a Gig of ram I need to write an application that consumes half of it just to perform some menial action. I write my own code the way I want and for my own reasons, and that includes using assembly language here and there. Sometimes I'll write the entire Windows application in assembly, just because I can. That's my choice. However, MS has limited my choice by making it difficult to do what I want to do. Compiler intrinsics they provide really are also hard to type in and hurt on my hands.

Anyway, get used to it. Its not just Microsoft. The Intel and GCC 64-compilers don't support inlining of 64-bit code, either. In fact, no Intel compatible 64-compiler at the moment supports inlining of assembly instructions. The vendors can brainwash us to thinks its because there's not enough demand all they want, and the people who feel its an unimportant feature will agree and cheer. The fact is, people who use inlining of assembly langauge *usually* know what they're doing and have a good reason

Re: Inline assembler reference

for it and are annoyed that the feature has been removed and likely will never be added again. It is the compiler vendors that are just too lazy to add the feature, really. Fortunately, with GCC we can add the feature ourselves (and since the Intel compiler boasts compatibility with GCC perhaps they'd follow suit) but I have never been able to make sense of the GCC code and so it won't be me that makes the change. I don't use GCC anyway.

They days of being in control of your own code are over, we must now welcome the days of the compiler limiting us and doing our jobs for us. Not saying that's bad, really, it isn't. But to completely undermine my own abilities and make it as difficult as possible to perform optimizations that outperform the compiler's own output as a bit too extreme, IMO. The workarounds add overhead and the point is to remove overhead, and increase performance. I shouldn't have to sacrifice. But, using any Microsoft tool, I always have to sacrifice. I'm a fan and arduant supporter of MS, but lets face it, more and more since the last 7 years or so, increasingly we have to conform to MS rather than them conforming to the customer.

Thanks,
Shawn