What are these strange cmp [ecx], ecx instructions doing in my C# code?

devblogs.microsoft.com/oldnewthing/20070816-00

August 16, 2007

Raymond Chen

When you debug through some managed code at the assembly level, you'll find a whole lot of seemingly pointless instructions that perform a comparison but ignore the result. What's the point of comparing two values if you don't care what the result is? In C++, invoking an instance method on a `NULL` pointer results in undefined behavior. In other words, if you do it, the compiler is allowed to do anything it wants. And what most compilers do is, um, nothing. They don't take any special steps if the `this` pointer is `NULL`; they just generate code on the assumption that it isn't. In practice, this often means that everything seems to run just fine until you access a member variables or call a virtual functions, and then you crash. The C# language, by comparison, is quite explicit about what happens if you invoke an instance method on a `null` object reference:

> The value of `E` is checked to be valid. If the value of `E` is null, a `System.NullReferenceException` is thrown and no further steps are executed.

The null reference exception must be thrown before the method can be called. That's what the strange `cmp [ecx], ecx` comparison is for.[1] The compiler doesn't actually care what the result of the comparison is; it just wants to raise an exception if `ecx` is null. If `ecx` is null, the attempt to dereference it (in order to perform the comparison) will raise an access violation, which the runtime inspects and turns into a `NullReferenceException`. The test is usually against the `ecx` register since the CLR happens to use[2] the `fastcall` calling convention, which for instance methods passes the `this` pointer in the `ecx` register. The pointer the compiler wants to test is going to wind up in the `ecx` register sooner or later,[3] so it's not surprising that the test, when it happens, is made against the `ecx` register.

Nitpicker's Corner [1]Although this statement is written as if it were a fact, it is actually my interpretation based on observation and thinking about how language features are implemented. It is not an official position of the CLR team nor Microsoft Corporation, and that interpretation may ultimately prove incorrect. [2]"Happens to use" means that this is an implementation detail, not a contractual guarantee.[1]

[3]Unless the call is optimized. For example, the function might be inlined.

Raymond Chen

Follow