

Re: Arithmetic on function address

Source: <http://coding.derkeiler.com/Archive/C/ CPP/comp.lang.c/2004-04/2837.html>

From: Arthur J. O'Dwyer (*ajo_at_nospam.andrew.cmu.edu*)

Date: 04/23/04

Date: Fri, 23 Apr 2004 17:08:34 -0400 (EDT)

On Fri, 23 Apr 2004, Stephen Biggs wrote:

>
> *Eric Sosman* <*Eric.Sosman@sun.com*> wrote...
> > *Stephen Biggs* wrote:
> >>
> >> *Given this code:*
> >> `void f(void){}`
> >> `int main(void){return (int)f+5;}`
> >>
> >> *Is there anything wrong with this in terms of the standards?*
> >
> > *Yes and no. The Standard permits you to cast a pointer*
> > *value (even a function pointer value) to an integer, but it*
> > *does not guarantee that the result is useful or even usable.*
>
> *But, it then should allow you to add a value to that integer, as the*
> *code says, no? Is this what you mean by no guarantees of it being*
> *usable?*

[This explanation based on N869 6.3.2.3#6.]

Not necessarily. The implementation might for instance map the address of 'f' onto 'INT_MAX', thus producing signed-int overflow when you try to add 5 to it. *Then*, and only then, is your program allowed to defrost your refrigerator.

Alternatively, the implementation could map 'f' directly onto a trap representation in 'int'; then, *any* attempt to use the value of '(int)f' at all would trigger undefined behavior. (Note that '(int)f' itself is still a valid construct on such systems; you can take the 'sizeof ((int)f)' with impunity, but that's all you can do.)

Finally, according to the word of the C99 draft standard, the implementation is allowed to map the address of 'f' directly onto a number so large that 'int' can't hold it. Instant undefined behavior! (Except IMO in the case of 'sizeof', as above.)

> >> *Is this legal C code?*

> >
> > *Legal but useless.*
>
>
> *Ok, fine... I agree completely that it is useless, but shouldn't correct
> code be generated for it?*

If it's completely useless --- and in fact could legitimately do *anything at all* to your machine --- then what, pray tell, would be the "correct code" you'd expect to see generated? "Garbage in, garbage out" is the rule that applies here. Well, more precisely, "Something that might or might not produce garbage in, something that might or might not be garbage out."

> >> *One compiler I'm working with compiles this quietly, even with the
> >> most stringent and pedantic ANSI and warning levels, but generates
> >> code that only loads the address of "f" and fails to make the
> >> addition before returning a value from "main".*

You mean that on this compiler, the expressions

(int)f AND (int)f+5

compile to the same machine code? This is odd, but perfectly legitimate, behavior for a conforming optimizing C compiler as long as it documents its behavior in a conforming fashion. There's nothing wrong with your compiler (although I would say it's a weird one); there is something wrong with your test suite.

[BTW, if any experts could explain what 6.3.2.3 #6 means by "except as previously specified," I'd love to hear it. I don't recall any "previously specified" cases of the pointer-to-integer cast's being defined.]

-Arthur