WG14/N308
X3J11/93-055

Hexadecimal FP Constants

X3J11/???, WG14/???

Jim Thomas
Taligent, Inc.
10201 N. DeAnza Blvd.
Cupertino, CA 95014-2233
jim_thomas@taligent.com

In response to recent items posted to nceg, this note explains why "Floating-Point C Extensions" (FPCE) specifies hexadecimal floating constants, as opposed to a bit pattern notation. FPCE's hexadecimal floating constants are exemplified by `0x1.FFFFFEp127`, meaning 1.FFFFFE (hex) times 2 to the 127th power. A bit pattern notation might represent the same number as `0x.7F7FFFFF`, depending on the system specific format of the type of the constant.

Hexadecimal floating constants determine exact numeric values on all systems with sufficient width (range and precision) in the indicated format. A bit pattern notation would determine correct values only on systems whose numeric format matched that of the system the constant was written for.

If format width is insufficient, hexadecimal floating constants provide a *best possible* approximation, rounding or over/underflowing in the usual way. A bit pattern notation would yield meaningless values on systems whose number format deviated in any way from that of the system the constant was written for.

With either approach, on any particular binary implementation, all finite numeric values could be represented exactly.

The particular issue raised with the hexadecimal floating constants seems to be that they (like the standard C decimal floating constants) do not provide representations for infinities and NaNs, whereas a bit pattern notation could. To allow the programmer to create infinities and NaNs, FPCE provides `INFINITY` and `NAN` macros and the `nan` function. The `nan` function takes a `const char*` argument, giving the implementation a means to support the determination of special NaNs. This approach allows meanings for character sequences to be defined and supported across multiple hardware systems. Or, if a direct representation of a particular system's NaN bit patterns were desired, the character sequences could be used in this way. A bit pattern notation would have the advantage that infinities and NaNs would be guaranteed to be suitable for static and aggregate initialization. I think providing macro expansions for `INFINITY`, `NAN`, and `nan()` that are suitable for static and aggregate initialization would be a useful enhancement to FPCE, without the negative aspects of a bit pattern notation.

Also, hexadecimal floating constants are distinctly more expressive than bit patterns—constants are easier to read. (With a bit pattern approach, how would the type be indicted? An `f` suffix wouldn't work.)

December 1, 1993