C support for IEEE 754-2008

WG14 C floating point study group

Status

24 September 2011

IEEE 754-2008

- Major update to IEEE 754-1985
- 8 Year effort
- Participation by AMD, Apple, HP, IBM, Intel, Sun, academics, etc.
- Adopted as 2011 update to ISO/IEC 60559
- Many new features not supported in C1x nor decimal FP TR 24732
- Less implementation latitude, clearer, bigger than 754-1985
- Recommendations for language standards
- No language binding
- Mostly compatible with C99

Formats (1)

- 128 bit binary quad basic format (recommended)
  - Not required, may be long double
- Unlimited number of interchange formats, which may be arithmetic, including 16 bit binary (optional)
  - Types _Float*N*, _Decimal*N*, N specified in IEC 60559
  - Macros FLT*N*_IS_ARITH, DEC*N*_IS_ARITH defined if arithmetic
  - Names required for supported IEEE types, e.g. _Float32 for float
  - _Float16 required (optionally arithmetic)
  - _Float*N* complex, _Float*N* imaginary, if _Float*N* is arithmetic

Formats (2)

- Extended formats - extending single, double, or quad (recommended)
  - _Float*N*x, *N*= 32, 64, 128
  - _Decimal*N*x, *N* = 64, 128
  - Names required for supported qualifying types, e.g., _Float32x for double (if no narrower extended single)
  - Common 80 bit type might be _Float64x
- Extendable formats – user-specified precision and range (recommended)
  - Not planned

Type-related Nomenclature (1)

- Characteristics macros
  - FLT*N*_MAX, etc.
  - FLT*N*_IS_ARITH defined if _Float*N* is arithmetic
  - DEC*N*_MAX, etc.
  - DEC*N*_IS_ARITH defined if _Decimal*N* is aithmetic
  - FLT*N*X_MAX, etc., for binary extended types
  - DEC*N*X_MAX, etc., for decimal extended types
- Type classification
  - Non arithmetic interchange types are not floating types

Type Nomeclature (2)

- Constant suffixes
  - F*N* or f*N* for _Float*N*
  - D*N* or d*N* for _Decimal*N*
  - F*N*x or f*N*x for _Float*N*x
  - D*N*x or d*N*x for _Decimal*N*x
- Function suffixes
  - f*N* for _Float*N*
  - d*N* for _Decimal*N*
  - f*N*x for _Float*N*x
  - d*N*x for _Decimal*N*x

Conversions

- Required for non arithmetic types too
  - Conversions among all interchange types
  - Conversions between character sequences and all interchange types
- Usual arithmetic conversions
  - Convert to wider, wider means exponent range or precision is larger and the other is at least as large
  - Conversion of types not ordered by width is implementation defined

Character sequence conversions

- No new I/O width specifiers
- int strfromf*N*(char * restrict s, rsize_t n, const char * restrict format, _Float*N* fp);
- Length modifier inferred from function suffix, not contained in format
- strfromf64(s, n, format, fp) is equivalent snprintf (s, n, format, fp)
- strfromd*N*, strfromf*N*x, strfromd*N*x
- strtof*N*, strtod*N*, strtof*N*x, strtod*N*x added to strtod family

New operations (1)

- roundToIntegralTiesToEven
  - double roundeven(double)
- Rounding to integer value with fixed rounding direction must not raise “inexact”
  - Changed ceil, floor, trunc, and round to disallow raising “inexact”
- nextUp, nextDown
  - double nextup(double)
  - double nextdown(double)
- minNumMag, maxNumMag
  - double fminmag(double, double)
  - double fmaxmag(double, double)

New operations (2)

- Integer logb to return outside the range ±2 × (emax+p–1) for invalid input
  - long int llogb(double x), FPLLOGB0, FPLLOGBNAN
- formatOf (narrowing) add, sub, mul, div, sqrt, FMA - infinitely precise result rounded to format narrower than parameters
  - float fadd(double x, double y)
  - float faddl(long double x, long double y)
  - double daddl(long double x, long double y)
  - _Float*M* f*M*addf*N*(_Float*N* x,_Float*N* y) for all *M* < *N*
  - _Float*M* f*M*addf*N*x(_Float*N*x x _Floatf*N*x) for all *M* <= *N*

  etc.

New operations (3)

- convertToInteger functions, with and without inexact signal, for 5 rounding directions (for all integer types, for all floating types???)
  - intmax_t fromfp(double x, int round, unsigned int width), width = number of bits, round = one of FE_CEIL, FP_FLOOR, FP_TRUNC, FP_ROUND, FP_ROUNDEVEN, without “inexact”
  - fromfpx, with “inexact”
  - ufromfp, ufromfpx, for unsigned integers
  - An integral-valued rounding function followed by a cast will handle most needs, e.g., (uint64_t)ceilf128(x) rounds _Float128 upward to uint64_t

New operations (4)

- Decimal reencoding functions
  - Types for encoded bits dpdencodingd*N*_t, bidencodingd*N*_t
  - dpdencodingd*N*_t encodedpdd*N*(_Decimal*N*)
  - _Decimal*N* decodedpdd*N*(dpdencodingd*N*_t)
  - bidencodingd*N*_t encodebidd*N*(_Decimal*N*)
  - _Decimal*N* decodebidd*N*(bidencodingd*N*_t)

- compareSignalingEqual, compareSignalingNotEqual
  - iseqsig
  - isnesig

New operations (5)

- isSubnormal
  - ❑ issubnormal() generic macro
- isSignaling
  - ❑ issignaling() generic macro
- Ten-way class
  - ❑ Covered by inquiries fpclassify , signbit, and issignaling
- isCanonical
  - ❑ iscanonical() generic macro
- totalOrder, totalOrderMag
  - ❑ int totalorder(double, double)
  - ❑ int totalordermag(double, double)
- raiseFlags (in IEC 60559 “raise” means set bit)
  - ❑ int fesetexcept(int excepts)
- Conformance macros
  - ❑ TBD

Character string conversions

- Vs C1x Annex F, increase by at least 3 the number of decimal digits for correct rounding to and from binary floating types
  - Require correct rounding for at least DECIMAL_DIG + 3 decimal digits
  - Applies to strtod, scanf, printf families

NaNs

- Requirements similar to IEEE 754-1985
- Propagation recommendations clarified
- NaN payload defined as integral value represented in NaN significand
- Signaling NaNs considered for removal, but retained

❑ double canonicalize(double) returns canonical version of input, tiggers signaling NaN inputs (1 * x)

❑ int setpayload(double *res, double pl)

❑ int setpayloadsignaling(double *res, double pl)

❑ double getpayload(const double *)

❑ Signaling NaN macros SNANF, …, DEC_SNAN32, …, ok for static initialization

Static rounding attributes

- Set rounding direction for a static scope
- Affects all IEC 60559 operations, including sqrt, fma, strtod, printf

❑In progress

New math functions (recommended) (1)

- New functions (vs C1x):
  - double exp2m1(double) double exp2m1(double)
  - double exp10(double)
  - double exp10m1(double)
  - double log2p1(double)
  - double log10p1(double)
  - double rsqrt(double)
  - double compound(double, double)

New math funtions (recommended) (2)

- double rootn(double, long int)
- double pown(double, long int)
- double powr(double, double)
- double sinpi(double)
- double cospi(double)
- double tanpi(double)
- double tan2pi(double, double)

Correctly rounded math functions (recommended)

- Correct rounding for all new functions above plus current C1x functions: exp, expm1, exp2, log, log2, log10, logp1, hypot, pow, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh
  - Reserve names with “cr” prefix and all applicable suffixes, e.g., crexp, crlogf, crsinf128, crrsqrtd64, for correctly rounded functions

Preferred quantum exponents

- IEC 60559 specifies preferred quantum exponent for its required decimal operations

❑ Specification of preferred quantum exponent for new decimal math functions

Reduction functions (recommended)

- Operate on vectors
- Sum reductions: sum, dot, sumSquare, sumAbs
  - double reduc_sum(size_t n, const double p[static n])
  - double reduc_sumabs(size_t n, const double p[static n])
  - double reduc_sumsquare(size_t n, const double p [static n])
  - double reduc_sumprod(size_t n, const double p[static n], const double q[static n])

Scaled reduction functions (recommended)

- Scaled product reductions scaledProd, scaledProdSum, scaledProdDiff – return an in-range scaled product sp and scale factor sf such that the product is sp × 2^sf
  - double scaled_prod(size_t n, const double p[static n], long int * restrict sf)
  - double scaled_prodsum(size_t n, const double p[static n], const double q[static n], long int * restrict sf)
  - double scaled_proddiff(size_t n, const double p[static n], const double q[static n], long int * restrict sf)

Default modes

- defaultModes function to install default settings for all FP modes
  - Type femode_t
  - Macro FE_DFL_MODE
  - int fegetmode(femode_t * modep)
  - int fesetmode(const femode_t * modep)

Alternate exception handling (recommended)

- For sub-exceptions as well as exceptions, e.g. invalid from ∞ - ∞
- Resuming – raiseNoFlag, mayRaiseFlag, recordException, substitute, substitureXor, abruptUnderflow
- Immediate or delayed – break, throw, goto

❑Not done

Other (recommended)

- Attributes for evaluation methods
  - ❑Not done
- Attributes to allow/disallow optimizations
  - ❑Not done
- Attribute for reproducible results
  - ❑Not done
- Debugging support
  - ❑Not planned