long-double

Claude Heiland-Allen

2018–2019

Contents

1 arm/Numeric/LongDouble.hs

```
-------------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble
-- Copyright   :  (C) 2018 Claude Heiland-Allen
-- License     :  BSD3
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable
--
-- This module re-exports the default platform-specfic ABI for C's long double.
module Numeric.LongDouble ( module Numeric.LongDouble.ARM_64 ) where

import Numeric.LongDouble.ARM_64
  ( LongDouble()
  , truncate', round', ceiling', floor'
  , fromDouble, toDouble, fromInt, toInt
  )
```

2 ChangeLog.md

```
# Revision history for long-double

## 0.1  -- 2018-03-09

* First release.
```

3 c/longdouble.c

```
#include <math.h>
#include <HsFFI.h>

HsDouble ld_get_d(const long double *a) { return *a; }
HsInt    ld_get_i(const long double *a) { return *a; }
void     ld_set_d(long double *r, HsDouble b) { *r = b; }
void     ld_set_i(long double *r, HsInt b) { *r = b; }

void ld_add(long double *r, const long double *a, const long double *b) { *r = *a + *b; }
void ld_sub(long double *r, const long double *a, const long double *b) { *r = *a - *b; }
void ld_mul(long double *r, const long double *a, const long double *b) { *r = *a * *b; }
void ld_div(long double *r, const long double *a, const long double *b) { *r = *a / *b; }
void ld_pow(long double *r, const long double *a, const long double *b) { *r = powl(*a, *b); }
void ld_atan2(long double *r, const long double *a, const long double *b) { *r = atan2l(*a, *b); }
void ld_min(long double *r, const long double *a, const long double *b) { *r = fminl(*a, *b); }
void ld_max(long double *r, const long double *a, const long double *b) { *r = fmaxl(*a, *b); }

int ld_lt(const long double *a, const long double *b) { return *a <  *b; }
int ld_le(const long double *a, const long double *b) { return *a <= *b; }
int ld_eq(const long double *a, const long double *b) { return *a == *b; }
int ld_ne(const long double *a, const long double *b) { return *a != *b; }
int ld_ge(const long double *a, const long double *b) { return *a >= *b; }
int ld_gt(const long double *a, const long double *b) { return *a >  *b; }

void ld_abs(long double *r, const long double *a) { *r = fabsl(*a); }
void ld_sgn(long double *r, const long double *a) { *r = (*a > 0) - (0 > *a); }
void ld_neg(long double *r, const long double *a) { *r = - *a; }
void ld_recip(long double *r, const long double *a) { *r = 1.0L / *a; }
void ld_sqrt(long double *r, const long double *a) { *r = sqrtl(*a); }
void ld_exp(long double *r, const long double *a) { *r = expl(*a); }
void ld_log(long double *r, const long double *a) { *r = logl(*a); }
void ld_sin(long double *r, const long double *a) { *r = sinl(*a); }
void ld_cos(long double *r, const long double *a) { *r = cosl(*a); }
void ld_tan(long double *r, const long double *a) { *r = tanl(*a); }
void ld_sinh(long double *r, const long double *a) { *r = sinhl(*a); }
void ld_cosh(long double *r, const long double *a) { *r = coshl(*a); }
void ld_tanh(long double *r, const long double *a) { *r = tanhl(*a); }
void ld_asin(long double *r, const long double *a) { *r = asinl(*a); }
void ld_acos(long double *r, const long double *a) { *r = acosl(*a); }
void ld_atan(long double *r, const long double *a) { *r = atanl(*a); }
void ld_asinh(long double *r, const long double *a) { *r = asinhl(*a); }
void ld_acosh(long double *r, const long double *a) { *r = acoshl(*a); }
void ld_atanh(long double *r, const long double *a) { *r = atanhl(*a); }
void ld_floor(long double *r, const long double *a) { *r = floorl(*a); }
void ld_ceil(long double *r, const long double *a) { *r = ceill(*a); }
void ld_round(long double *r, const long double *a) { *r = roundl(*a); }
void ld_trunc(long double *r, const long double *a) { *r = truncl(*a); }
```

```
int ld_isnan(const long double *a) { return fpclassify(*a) == FP_NAN; }
int ld_isinf(const long double *a) { return fpclassify(*a) == FP_INFINITE; }
int ld_isdenorm(const long double *a) { return fpclassify(*a) == FP_SUBNORMAL; }
int ld_isnegzero(const long double *a) { return fpclassify(*a) == FP_ZERO && signbit(*a); }

void ld_ldexp(long double *r, const long double *a, int b) { *r = ldexpl(*a, b); }
void ld_frexp(long double *r, const long double *a, int *b) { *r = frexpl(*a, b); }

void ld_pi(long double *r) { *r = 3.14159265358979323846264338327950288 4L; }
```

4 .gitignore

```
.cabal-sandbox
cabal.sandbox.config
dist
dist-newstyle
```

5 hs/Numeric/LongDouble/ARM_64.hs

```
{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
-------------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble.ARM_64
-- Copyright   :  (C) 2018 Claude Heiland-Allen
-- License     :  BSD3
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable (assumes sizeof(long double) == 8)
--
-- This module contains a LongDouble type that is the same as Double, as on ARM.
--
-- Most code should import Numeric.LongDouble instead, unless a specific ABI
-- is needed.  This module is for sizeof(long double) == 8, with the type being
-- a simple alias for double.
module Numeric.LongDouble.ARM_64
  (
  -- * long double data type
    LongDouble(..)
  -- * RealFrac alternatives
  , truncate'
  , round'
  , ceiling'
  , floor'
  -- * Conversions
  , fromDouble
  , toDouble
  , fromInt
  , toInt
  ) where

import Data.Coerce (coerce)
```

```
import Foreign (Ptr, with, alloca)
import Foreign.C.Types (CDouble(..))
import Foreign.Storable (Storable(..))
import Numeric (showFloat, readFloat, readSigned)
import System.IO.Unsafe (unsafePerformIO)

-- | The long double type on ARM: 64bits of double in 64bits of space.
newtype LongDouble = LD Double
  deriving (Storable, Eq, Ord, Num, Real, Fractional, RealFrac, Floating, ↙
      ↳ RealFloat)

instance Read LongDouble where
  readsPrec p = coerce . (readsPrec p :: ReadS Double)

instance Show LongDouble where
  showsPrec p (LD x) = showsPrec p x

fromInt :: Int -> LongDouble
fromInt = fromIntegral

toInt :: LongDouble -> Int
toInt = truncate

fromDouble :: Double -> LongDouble
fromDouble = coerce

toDouble :: LongDouble -> Double
toDouble = coerce

-- | Alternate versions of RealFrac methods that
--   keep the value as a long double.
truncate', round', ceiling', floor' :: LongDouble -> LongDouble
truncate' = f1 d_trunc
round'    = f1 d_round
ceiling'  = f1 d_ceil
floor'    = f1 d_floor

f1 :: (CDouble -> IO CDouble) -> LongDouble -> LongDouble
f1 f a = unsafePerformIO $ do
  r <- f (coerce a)
  return (coerce r)

foreign import ccall unsafe "math.h floor" d_floor :: CDouble -> IO CDouble
foreign import ccall unsafe "math.h ceil"  d_ceil  :: CDouble -> IO CDouble
foreign import ccall unsafe "math.h round" d_round :: CDouble -> IO CDouble
foreign import ccall unsafe "math.h trunc" d_trunc :: CDouble -> IO CDouble
```

6 hs/Numeric/LongDouble/X87_128.hs

```
{-# LANGUAGE CPP #-}
{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE MagicHash #-}
-------------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble.X87_128
-- Copyright   :  (C) 2018 Claude Heiland-Allen
-- License     :  BSD3
```

```
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable (assumes x87 with sizeof(long double) == 16)
--
-- This module contains a LongDouble type that can be used if you need that
-- extra precision or range from the x87 FPU extended precision type.
-- It has 15bit signed exponent (compared to 11bit signed exponent for Double)
-- and 64bit mantissa (compared to 53bit mantissa for Double).
--
-- Performance is likely to be poor, as the instances are implemented using
-- FFI with Ptr LongDouble, copying to and from memory around each operation.
-- If you need to bind to functions taking/returning long double you need to
-- write wrapper functions expecting pointers to long double instead, as GHC
-- does not expose a CLDouble FFI type.
-- See <https://ghc.haskell.org/trac/ghc/ticket/3353>.
--
-- Most code should import Numeric.LongDouble instead, unless a specific ABI
-- is needed.  This module is for x87 with sizeof(long double) == 16.
module Numeric.LongDouble.X87_128
  (
  -- * long double data type
    LongDouble(..)
  -- * RealFrac alternatives
  , truncate'
  , round'
  , ceiling'
  , floor'
  -- * Conversions
  , fromDouble
  , toDouble
  , fromInt
  , toInt
  ) where

import Data.Bits (bit, testBit, (.&.), shiftL, shiftR)
import Data.Ratio ((%), numerator, denominator)
import Data.Word (Word64)
import Foreign (Ptr, castPtr, with, alloca)
import Foreign.C.Types (CIntMax(..), CInt(..), CDouble(..))
import Foreign.Storable (Storable(..))
import Numeric (showFloat, readFloat, readSigned)
import System.IO.Unsafe (unsafePerformIO)
import GHC.Exts (Int(..))
import GHC.Integer.Logarithms (integerLog2#)

-- | The long double type on x86_64: 80bits of x87 FPU data in 128bits of space.
data LongDouble = LD !Word64 !Word64

instance Storable LongDouble where
  sizeOf _ = 2 * sizeOf (0 :: Word64)
  alignment _ = alignment (0 :: Word64)
  peek p = do
    let q :: Ptr Word64
        q = castPtr p
    a <- peekElemOff q 0
    b <- peekElemOff q 1
    return $ LD a b
```

```
  poke p (LD a b) = do
    let q :: Ptr Word64
        q = castPtr p
    pokeElemOff q 0 a
    pokeElemOff q 1 b

#include "X87.hs"
```

7 hs/Numeric/LongDouble/X87_96.hs

```
{-# LANGUAGE CPP #-}
{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE MagicHash #-}
-------------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble.X87_128
-- Copyright   :  (C) 2018 Claude Heiland-Allen
-- License     :  BSD3
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable (assumes x87 with sizeof(long double) == 12)
--
-- This module contains a LongDouble type that can be used if you need that
-- extra precision or range from the x87 FPU extended precision type.
-- It has 15bit signed exponent (compared to 11bit signed exponent for Double)
-- and 64bit mantissa (compared to 53bit mantissa for Double).
--
-- Performance is likely to be poor, as the instances are implemented using
-- FFI with Ptr LongDouble, copying to and from memory around each operation.
-- If you need to bind to functions taking/returning long double you need to
-- write wrapper functions expecting pointers to long double instead, as GHC
-- does not expose a CLDouble FFI type.
-- See <https://ghc.haskell.org/trac/ghc/ticket/3353>.
--
-- Most code should import Numeric.LongDouble instead, unless a specific ABI
-- is needed.  This module is for x87 with sizeof(long double) == 12.
module Numeric.LongDouble.X87_96
  (
  -- * long double data type
    LongDouble(..)
  -- * RealFrac alternatives
  , truncate'
  , round'
  , ceiling'
  , floor'
  -- * Conversions
  , fromDouble
  , toDouble
  , fromInt
  , toInt
  ) where

import Data.Bits (bit, testBit, (.&.), shiftL, shiftR)
import Data.Ratio ((%), numerator, denominator)
import Data.Word (Word64, Word32)
import Foreign (Ptr, castPtr, with, alloca)
import Foreign.C.Types (CIntMax(..), CInt(..), CDouble(..))
```

```
import Foreign.Storable (Storable(..))
import Numeric (showFloat, readFloat, readSigned)
import System.IO.Unsafe (unsafePerformIO)
import GHC.Exts (Int(..))
import GHC.Integer.Logarithms (integerLog2#)

-- | The long double type on i386: 80 bits of x87 FPU data in 96 bits of space.
data LongDouble = LD !Word64 !Word32

instance Storable LongDouble where
  sizeOf _ = sizeOf (0 :: Word64) + sizeOf (0 :: Word32)
  alignment _ = alignment (0 :: Word32)
  peek p = do
    let q :: Ptr Word64
        q = castPtr p
        r :: Ptr Word32
        r = castPtr p
    a <- peekElemOff q 0
    b <- peekElemOff r 2
    return $ LD a b
  poke p (LD a b) = do
    let q :: Ptr Word64
        q = castPtr p
        r :: Ptr Word32
        r = castPtr p
    pokeElemOff q 0 a
    pokeElemOff r 2 b

#include "X87.hs"
```

8 hs/Numeric/LongDouble/X87.hs

```
instance Eq LongDouble where
  (==) = cmp ld_eq
  (/=) = cmp ld_ne

instance Ord LongDouble where
  (<=) = cmp ld_le
  (< ) = cmp ld_lt
  (>=) = cmp ld_ge
  (> ) = cmp ld_gt
  min  = f2 ld_min
  max  = f2 ld_max

instance Num LongDouble where
  fromInteger z = encodeFloat z 0
  negate = f1 ld_neg
  (+) = f2 ld_add
  (-) = f2 ld_sub
  (*) = f2 ld_mul
  abs = f1 ld_abs
  signum = f1 ld_sgn

instance Real LongDouble where
  toRational l = case decodeFloat l of
    (m, e)
      | e >= 0 -> m `shiftL` e % 1
```

```
      | otherwise -> m % bit (negate e)

instance Fractional LongDouble where
  fromRational q = -- FIXME accuracy?
    let a = fromInteger (numerator q) / fromInteger (denominator q)
        r = q - toRational a
        b = fromInteger (numerator r) / fromInteger (denominator r)
    in  a + b
  (/) = f2 ld_div
  recip = f1 ld_recip

instance RealFrac LongDouble where
  properFraction l
    | l >= 0 = let n' = floor' l
                   f = l - n'
               in  (fromInteger . toInteger' $ n', f)

    | l <  0 = let n' = ceiling' l
                   f = l - n'
               in  (fromInteger . toInteger' $ n', f)
    | otherwise = (0, l) -- NaN
  truncate = fromInteger . toInteger' . truncate'
  round    = fromInteger . toInteger' . round'
  ceiling  = fromInteger . toInteger' . ceiling'
  floor    = fromInteger . toInteger' . floor'

toInteger' :: LongDouble -> Integer
toInteger' l = case decodeFloat l of
  (m, e)
    | e >= 0 -> m `shiftL` e
    | otherwise -> m `shiftR` negate e

-- | Alternate versions of RealFrac methods that
--   keep the value as a long double.
truncate', round', ceiling', floor' :: LongDouble -> LongDouble
truncate' = f1 ld_trunc
round'    = f1 ld_round
ceiling'  = f1 ld_ceil
floor'    = f1 ld_floor

instance Floating LongDouble where
  pi = unsafePerformIO $ do
    alloca $ \lp -> do
      ld_pi lp
      peek lp
  exp = f1 ld_exp
  log = f1 ld_log
  sqrt = f1 ld_sqrt
  (**) = f2 ld_pow
  -- logBase
  sin = f1 ld_sin
  cos = f1 ld_cos
  tan = f1 ld_tan
  sinh = f1 ld_sinh
  cosh = f1 ld_cosh
  tanh = f1 ld_tanh
  asin = f1 ld_asin
```

```
  acos = f1 ld_acos
  atan = f1 ld_atan
  asinh = f1 ld_asinh
  acosh = f1 ld_acosh
  atanh = f1 ld_atanh

instance RealFloat LongDouble where
  floatRadix _ = 2
  floatDigits _ = 64
  floatRange _ = (-16381,16384) -- FIXME verify?

  decodeFloat l@(LD a b)
    | isNaN l = (0, 0)
    | isInfinite l = (0, 0)
    | l == 0 = (0, 0)
    | isDenormalized l = case decodeFloat (scaleFloat 128 l) of
        (m, e) -> (m, e - 128)
    | otherwise =
        ( (if s then negate else id) (fromIntegral a)
        , fromIntegral e0 - 16383 - 63
        )
    where
      s = b `testBit` 15
      e0 = b .&. (bit 15 - 1)

  encodeFloat m e
    | m == 0 = LD 0 0
    | m <  0 = negate (encodeFloat (negate m) e)
    | b >= bit 15 - 1 = LD (bit 63) (bit 15 - 1) -- overflow to infinity
    | b <= 0 = scaleFloat (b - 128) (encodeFloat m (e - b + 128)) -- denormal
    | otherwise = LD a (fromIntegral b) -- normal
    where
      l = I# (integerLog2# m)
      t = l - 63
      a | t >= 0    = fromInteger (m `shiftR`        t)
        | otherwise = fromInteger (m `shiftL` negate t)
      b = e + t + 16383 + 63

  exponent l@(LD _ b)
    | isNaN l = 0
    | isInfinite l = 0
    | l == 0 = 0
    | isDenormalized l = snd (decodeFloat l) + 64
    | otherwise = fromIntegral e0 - 16383 - 63 + 64
    where
      e0 = b .&. (bit 15 - 1)

  significand l = unsafePerformIO $ do
    with l $ \lp -> alloca $ \ep -> do
      ld_frexp lp lp ep
      peek lp

  scaleFloat e l = unsafePerformIO $ do
    with l $ \lp -> do
      ld_ldexp lp lp (fromIntegral e)
      peek lp
```

```
    isNaN = tst ld_isnan
    isInfinite = tst ld_isinf
    isDenormalized = tst ld_isdenorm
    isNegativeZero = tst ld_isnegzero
    isIEEE _= True

    atan2 = f2 ld_atan2

instance Read LongDouble where
  readsPrec _ = readSigned readFloat

instance Show LongDouble where
  showsPrec p x = showParen (p >= 7 && take 1 s == "-") (s ++) -- FIXME: ↙
      ↳ precedence issues?
    where s = showFloat x ""

fromInt :: Int -> LongDouble
fromInt i = unsafePerformIO $ do
  alloca $ \lp -> do
    ld_set_i lp (fromIntegral i)
    peek lp

toInt :: LongDouble -> Int
toInt l = unsafePerformIO $ with l ld_get_i

fromDouble :: Double -> LongDouble
fromDouble i = unsafePerformIO $ do
  alloca $ \lp -> do
    ld_set_d lp i
    peek lp

toDouble :: LongDouble -> Double
toDouble l = unsafePerformIO $ with l ld_get_d

f2 :: F2 -> LongDouble -> LongDouble -> LongDouble
f2 f a b = unsafePerformIO $ do
  with a $ \ap -> with b $ \bp -> alloca $ \rp -> do
    f rp ap bp
    peek rp

type F2 = Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> IO ()

foreign import ccall unsafe "ld_add"   ld_add   :: F2
foreign import ccall unsafe "ld_sub"   ld_sub   :: F2
foreign import ccall unsafe "ld_mul"   ld_mul   :: F2
foreign import ccall unsafe "ld_div"   ld_div   :: F2
foreign import ccall unsafe "ld_pow"   ld_pow   :: F2
foreign import ccall unsafe "ld_min"   ld_min   :: F2
foreign import ccall unsafe "ld_max"   ld_max   :: F2
foreign import ccall unsafe "ld_atan2" ld_atan2 :: F2

f1 :: F1 -> LongDouble -> LongDouble
f1 f a = unsafePerformIO $ do
  with a $ \ap -> alloca $ \rp -> do
    f rp ap
    peek rp
```

```
type F1 = Ptr LongDouble -> Ptr LongDouble -> IO ()

foreign import ccall unsafe "ld_abs"   ld_abs   :: F1
foreign import ccall unsafe "ld_sgn"   ld_sgn   :: F1
foreign import ccall unsafe "ld_neg"   ld_neg   :: F1
foreign import ccall unsafe "ld_sqrt"  ld_sqrt  :: F1
foreign import ccall unsafe "ld_recip" ld_recip :: F1
foreign import ccall unsafe "ld_exp"   ld_exp   :: F1
foreign import ccall unsafe "ld_log"   ld_log   :: F1
foreign import ccall unsafe "ld_sin"   ld_sin   :: F1
foreign import ccall unsafe "ld_cos"   ld_cos   :: F1
foreign import ccall unsafe "ld_tan"   ld_tan   :: F1
foreign import ccall unsafe "ld_sinh"  ld_sinh  :: F1
foreign import ccall unsafe "ld_cosh"  ld_cosh  :: F1
foreign import ccall unsafe "ld_tanh"  ld_tanh  :: F1
foreign import ccall unsafe "ld_asin"  ld_asin  :: F1
foreign import ccall unsafe "ld_acos"  ld_acos  :: F1
foreign import ccall unsafe "ld_atan"  ld_atan  :: F1
foreign import ccall unsafe "ld_asinh" ld_asinh :: F1
foreign import ccall unsafe "ld_acosh" ld_acosh :: F1
foreign import ccall unsafe "ld_atanh" ld_atanh :: F1
foreign import ccall unsafe "ld_floor" ld_floor :: F1
foreign import ccall unsafe "ld_ceil"  ld_ceil  :: F1
foreign import ccall unsafe "ld_round" ld_round :: F1
foreign import ccall unsafe "ld_trunc" ld_trunc :: F1

type CMP = Ptr LongDouble -> Ptr LongDouble -> IO CInt

cmp :: CMP -> LongDouble -> LongDouble -> Bool
cmp f a b = unsafePerformIO $ do
  with a $ \ap -> with b $ \bp -> do
    r <- f ap bp
    return (r /= 0)

foreign import ccall unsafe "ld_eq" ld_eq :: CMP
foreign import ccall unsafe "ld_ne" ld_ne :: CMP
foreign import ccall unsafe "ld_lt" ld_lt :: CMP
foreign import ccall unsafe "ld_le" ld_le :: CMP
foreign import ccall unsafe "ld_gt" ld_gt :: CMP
foreign import ccall unsafe "ld_ge" ld_ge :: CMP

type TST = Ptr LongDouble -> IO CInt

tst :: TST -> LongDouble -> Bool
tst f a = unsafePerformIO $ do
  with a $ \ap -> do
    r <- f ap
    return (r /= 0)

foreign import ccall unsafe "ld_isnan" ld_isnan :: TST
foreign import ccall unsafe "ld_isinf" ld_isinf :: TST
foreign import ccall unsafe "ld_isdenorm" ld_isdenorm :: TST
foreign import ccall unsafe "ld_isnegzero" ld_isnegzero :: TST

foreign import ccall unsafe "ld_get_d" ld_get_d :: Ptr LongDouble -> IO Double
foreign import ccall unsafe "ld_get_i" ld_get_i :: Ptr LongDouble -> IO Int
```

```
foreign import ccall unsafe "ld_set_d" ld_set_d :: Ptr LongDouble -> Double -> ↙
    ↳ IO ()
foreign import ccall unsafe "ld_set_i" ld_set_i :: Ptr LongDouble -> Int -> IO ↙
    ↳ ()
255
foreign import ccall unsafe "ld_ldexp" ld_ldexp :: Ptr LongDouble -> Ptr ↙
    ↳ LongDouble -> CInt -> IO ()
foreign import ccall unsafe "ld_frexp" ld_frexp :: Ptr LongDouble -> Ptr ↙
    ↳ LongDouble -> Ptr CInt -> IO ()

foreign import ccall unsafe "ld_pi"    ld_pi    :: Ptr LongDouble -> IO ()
```

9 i386/Numeric/LongDouble.hs

```
------------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble
-- Copyright   :  (C) 2018 Claude Heiland-Allen
5 -- License     :  BSD3
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable
--
10 -- This module re-exports the default platform-specfic ABI for C's long double.
module Numeric.LongDouble ( module Numeric.LongDouble.X87_96 ) where

import Numeric.LongDouble.X87_96
  ( LongDouble()
15  , truncate', round', ceiling', floor'
  , fromDouble, toDouble, fromInt, toInt
  )
```

10 LICENSE

A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11 long-double.cabal

```
name:                long-double
version:             0.1
synopsis:            FFI bindings for C long double
description:
  This package provides a LongDouble type, being 80 bits of x87 data taking up
  96 bits on i386 and 128 bits on x86_64. On arm it is an alias for 64 bit double.
  It does not provide a CLDouble type usable for FFI without wrapping in Ptr,
  this needs to be done by the compiler.
  See <https://ghc.haskell.org/trac/ghc/ticket/3353>.

homepage:            https://code.mathr.co.uk/long-double
license:             BSD3
license-file:        LICENSE
author:              Claude Heiland-Allen
maintainer:          claude@mathr.co.uk
copyright:           (c) 2018 Claude Heiland-Allen
category:            Math
build-type:          Simple
extra-source-files:
  ChangeLog.md
  i386/Numeric/LongDouble.hs
  x86_64/Numeric/LongDouble.hs
  arm/Numeric/LongDouble.hs
  hs/Numeric/LongDouble/ARM_64.hs
  hs/Numeric/LongDouble/X87.hs
  hs/Numeric/LongDouble/X87_96.hs
  hs/Numeric/LongDouble/X87_128.hs

cabal-version:       >=1.10

source-repository head
  type: git
  location: https://code.mathr.co.uk/long-double.git

source-repository this
  type: git
  location: https://code.mathr.co.uk/long-double.git
  tag: long-double-0.1

library
  exposed-modules:
    Numeric.LongDouble
  build-depends:
    base >=4.6 && <4.14,
    integer-gmp >=0.5 && <1.1
  c-sources: c/longdouble.c
```

```
  hs-source-dirs: hs
  if arch(i386)
    hs-source-dirs: i386
    other-modules: Numeric.LongDouble.X87_96
  else
    if arch(x86_64)
      hs-source-dirs: x86_64
      other-modules: Numeric.LongDouble.X87_128
    else
      if arch(arm)
        hs-source-dirs: arm
        other-modules: Numeric.LongDouble.ARM_64
      else
        buildable: False
  default-language: Haskell2010
  other-extensions:
    CPP,
    ForeignFunctionInterface,
    GeneralizedNewtypeDeriving,
    MagicHash
```

12 Setup.hs

```
import Distribution.Simple
main = defaultMain
```

13 x86_64/Numeric/LongDouble.hs

```
-----------------------------------------------------------------------------
-- |
-- Module      :  Numeric.LongDouble
-- Copyright   :  (C) 2018 Claude Heiland-Allen
-- License     :  BSD3
-- Maintainer  :  Claude Heiland-Allen <claude@mathr.co.uk>
-- Stability   :  experimental
-- Portability :  non-portable
--
-- This module re-exports the default platform-specfic ABI for C's long double.
module Numeric.LongDouble ( module Numeric.LongDouble.X87_128 ) where

import Numeric.LongDouble.X87_128
  ( LongDouble()
  , truncate', round', ceiling', floor'
  , fromDouble, toDouble, fromInt, toInt
  )
```