AMD LIBM (AMD Math Library)

AMD LibM is a software library containing a collection of basic math functions optimized for x86-64 processor based machines. It provides many routines from the list of standard C99 math functions.

AMD LibM is a C library, which users can link in to their applications to replace compiler-provided math functions. Generally, programmers access basic math functions through their compiler. But those who want better accuracy or performance than their compiler's math functions can use this library to help improve their applications.

Users can also take advantage of the vector functions in this library. The vector variants can be used to speed up loops and perform math operations on multiple elements conveniently.

AMD LibM supports dynamic dispatch mechanism with no run-time overhead to automatically select the optimized functions. The library supports FMA3/FMA4 code path for the key math functions to make use of the new instruction set architecture (ISA), and boost the performance of the library.

The table below lists the set of functions supported by AMD LibM and indicates the availability of FMA3/FMA4 code paths.

Function Prototype	FMA4	FMA3	Description
double acos (double x)	N	N	Returns the arc cosine in radians and the value is mathematically defined to be between 0 and PI (Inclusive)
float acosf (float x)	N	N	
double acosh (double x)	N	N	Returns the hyperbolic arccosine.
float acoshf (float x)	N	N	
double asin (double x)	N	N	Returns the arc sine in radians and the value is mathematically defined to be between -PI/2 and PI/2
float asinf (float x)			
double asinh (double x)	N	N	Returns the hyperbolic arcsine.
float asinhf (float x)	N	N	
double atan (double x)	N	N	Returns the arc tangent in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).
float atanf (float x)	N	N	
double atan2 (double x, double y)	N	N	Calculates the arc tangent of the two variables. Returns the result in radians, which is between -PI and PI (inclusive).
float atan2f (float x, float y)	N	N	

double atanh (double x)	N	N	Returns the hyperbolic arctangent.
float atanhf (float x)	N	N	
double cbrt(double x)	Y	Y	Returns the (real) cube root of x.
float cbrtf (float x)	Y	Y	
double ceil (double x)	N	N	Rounds the argument up to the nearest integer
float ceilf (float x)	N	N	
double copysignf (float x, float y)	N	N	Return a value whose absolute value matches that of x, but whose sign matches that of y. If x is a NaN, then a NaN with the sign of y is returned.
float copysign (float x, float y)	N	N	
double cos (double x)	Y	Y	Returns the cosine of x
float cosf (float x)	Y	Y	
double cosh (double x)	N	N	Returns the hyperbolic cosine.
float coshf (float x)	N	N	
double cospi (double x)	N	N	Returns the cosine of pi * x.
float cospif (float x)	N	N	
double exp (double x)	Y	Y	Returns the value of e (the base of natural logarithms) raised to the power of x
float expf (float x)	Y	Y	
double exp10(double x)	Y	Y	Returns the value of 10 raised to the power of x
float exp10f (float x)	Y	Y	
double exp2(double x)	Y	Y	Returns the value of 2 raised to the power of x
float exp2f (float x)	Y	Y	
double expm1 (double x)	Y	Y	Returns a value equivalent to 'exp (x) - 1'
float expm1f (float x)	Y	Y	
double fabs (double x)	N	N	Returns the absolute value of floating-point number
float fabsf (float x)	N	N	
double fdim (double x,double y)	N	N	These functions return max(x-y,0). If x or y or both are NaN, Nan is returned
float fdimf (float x, float y)	N	N	

int finite(double x)	N	N	Returns 0 if x is infinite or NaN, otherwise returns 1
int finitef(float x)	N	N	
double floor (double x)	N	N	Rounds the argument down to the nearest integer.
float floorf (float x)	N	N	
double fma(double x, double y, double z)	Y	Y	Computes (x * y) + z, rounded as one ternary operation: they compute the value (as if) to infinite precision and round once to the result format, according to the current round
float fmaf(float x, float y, float z)	Y	Y	
double fmax (double x, double y)	N	N	Selects the greater of x and y.
float fmaxf (float x, float y)	N	N	
double fmin (double x, double y)	N	N	Selects the lesser of x and y
float fminf (float x, float y)	N	N	
double fmod (double x, double y)	N	N	Computes the remainder of dividing x by y. The return value is x - n * y, where n is the quotient of x / y, rounded towards zero to an integer
float fmodf (float x, float y)	N	N	
double frexp (double x, int * exp)	N	N	Splits the number x into a normalized fraction and an exponent which is stored in exp.
float frexpf (float x, int * exp)	N	N	
double hypot (double x, double y)	N	N	Returns sqrt(x*x+y*y). This is the length of the hypotenuse of a right-angle triangle with sides of length x and y.
float hypotf (float x, float y)	N	N	
int ilogb (double x)	N	N	Return the exponent part of their argument as a signed integer
int ilogbf (float x)	N	N	
double ldexp (double x, double exp)	N	N	Returns the result of multiplying the floating-point number x by 2 raised to the power exp
float ldexpf (float x, float exp)	N	N	
long long int llrint (double x)	N	N	Rounds x to the nearest integer
long long int llrintf (float x)	N	N	
long long int llround(double x)	N	N	Rounds x to the nearest integer

long long int llroundf (float x)	N	N	
double log (double)	Y	Y	Returns the natural logarithm of x
float logf (float x)	Y	Y	
double log10(double x)	Y	Y	Returns the base 10 logarithm of x
float log10f (float x)	Y	Y	
double log1p(double x)	Y	Y	Returns a value equivalent to 'log (1 + x)'. It is computed in a way that is accurate even if the value of x is near zero.
float log1pf(float x)	Y	Y	
double log2 (double x)	Y	Y	Returns the base 2 logarithm of x
float log2f (float x)	Y	Y	
double logb(double x)	N	N	Extracts the exponent of x and return it as a floating-point value
float logb (float x)	N	N	
long int lrint (double x)	N	N	Rounds x to the nearest integer
long int lrintf (float x)	N	N	
long int lround (double x)	N	N	Rounds x to the nearest integer
long int lroundf (float x)	N	N	
double modf (double x, double * iptr)	N	N	Breaks the argument x into an integral part and a fractional part, each of which has the same sign as x. The integral part is stored in iptr.
float modff (float x, float * iptr)	N	N	
double nan (int tagp)	N	N	Return a representation (determined by tagp) of a quiet NaN. If the implementation does not support quiet NaNs, these functions return zero.
float nanf (int tagp)	N	N	
double nearbyintf (double x)	N	N	Rounds the argument to an integer value in floating point format, using the current rounding direction
float nearbyintf (float x)	N	N	
double nextafter (double x, double y)	N	N	Returns the next representable neighbor of x in the direction towards y. The size of the step between x and the result depends on the type of the result. If x = y the function simply returns y. If either value is NaN,

float nextafter (float x, float y)	N	N	then NaN is returned. Otherwise a value corresponding to the value of the least significant bit in the mantissa is added or subtracted, depending on the direction.
double nexttoward (double x, long double y)	N	N	The nexttoward functions are equivalent to the nextafter functions except that the second parameter has type long double and the functions return y converted to the type of the function if x equals y.
float nexttowardf (float x, long double y)	N	N	
double pow (double x, double y)	N	N	Returns the value of x raised to the power of y
float powf (float x, float y)	N	N	
double remainder (double x,double y)	N	N	Computes the remainder of dividing x by y. The return value is x - n * y, where n is the value x / y, rounded to the nearest integer. If this quotient is 1/2 (mod 1), it is rounded to the nearest even number (independent of the current rounding mode). If the return value is 0, it has the sign of x.
float remainderf (float x, float y)	N	N	
double remquo(double x, double y, int * quo)	N	N	Computes the remainder and part of the quotient upon division of x by y. A few bits of the quotient are stored via the quo pointer. The remainder is returned.
float remquof (float x, float y, int * quo)	N	N	
double rint (double x)	N	N	round to integral value in floating-point format
float rintf (float x)	N	N	
double round (double x)	N	N	Rounds x to the nearest integer.
float roundf (float x)	N	N	
double scalbln (double x, long int n)	N	N	Multiplies their first argument x by FLT_RADIX (probably 2) to the power exp
float scalblnf (float x, long int n)	N	N	
double scalbn(double x, int n)	N	N	Multiplies their first argument x by FLT_RADIX (probably 2) to the power exp.

float scalbnf (float x, int n)	N	N	If FLT_RADIX equals 2, then scalbn() is equivalent to ldexp(). The value of FLT_RADIX is found in <float.h>
double sin (double x)	Y	Y	Returns the sine of x
float sinf (float x)	Y	Y	
void sincos (double x, double y, double z)	N	N	Returns the sine and cosine of x
void sincosf (float x, float * y, float * z)	N	N	
double sinh(double x)	N	N	Returns the hyperbolic sine of x
float sinhf(float x)	N	N	
double sinpi (double x)	N	N	Returns the sine of pi * x.
float sinpif(float x)	N	N	
double sqrt (double x)	N	N	Returns the square root of x
float sqrtf (float x)	N	N	
double tan (double x)	Y	Y	Returns the tangent of x.
float tanf (float x)	Y	Y	
double tanh (double x)	N	N	Returns the hyperbolic tangent of x.
float tanhf (float x)	N	N	
double tanpi (double x)	N	N	Returns the tangent of pi * x.
float tanpif (float x)	N	N	
double trunc (double x)	N	N	Rounds x to the nearest integer not larger in absolute value
float truncf (float x)	N	N	
__m128d vrd2_cbrt (__m128d x)	Y	Y	Computes the (real) cube root of two packed double precision numbers.
__m128 vrs4_cbrtf (__m128 x)	Y	Y	Computes the (real) cube root of four packed single precision numbers.
void vrda_cbrt (int len,double *src,double *dst)	Y	Y	Computes the (real) cube root of array of packed double precision numbers.
void vrsa_cbrtf (int len, float *src,float *dst)	Y	Y	Computes the (real) cube root of array of packed single precision numbers.

__m128d vrd2_cos (__m128d x)	Y	Y	Computes the (real) cosine of two packed double precision numbers.
__m128 vrs4_cosf (__m128 x)	Y	Y	Computes the (real) cosine of four packed single precision numbers.
void vrda_cos (int n,double *x, double *y)	Y	Y	Computes the (real) cosine of array of packed double precision numbers.
void vrsa_cosf (int len, float *src,float *dst)	Y	Y	Computes the (real) cosine of array of packed single precision numbers.
__m128d vrd2_cosh (__m128d x)	N	N	Computes hyperbolic cosine of two packed double precision numbers.
__m128d vrd2_exp (__m128d x)	Y	Y	Computes the value of e (the base of natural logarithms) raised to the power of two packed double precision numbers.
__m128 vrs4_expf (__m128 x)	Y	Y	Computes the value of 'exp (x) - 1' of four packed single precision numbers.
void vrda_exp (int len,double *src,double *dst)	Y	Y	Compute the value of e (the base of natural logarithms) raised to the power of array of packed double precision numbers.
void vrsa_expf (int len, float* x, float* y)	Y	Y	Computes the value of e (the base of natural logarithms) raised to the power of array of packed single precision numbers.
__m128d vrd2_exp10 (__m128d x)	Y	Y	Computes the value of 10 raised to the power of two packed double precision numbers.
__m128 vrs4_exp10f (__m128 x)	Y	Y	Computes the value of 10 raised to the power of four packed single precision numbers.
void vrda_exp10 (int len,double *src,double *dst)	Y	Y	Computes the value of 10 raised to the power of array of packed double precision numbers.
void vrsa_exp10f (int len, float *src,float *dst)	Y	Y	Computes the value of 10 raised to the power of array of packed single precision numbers.

__m128d vrd2_exp2 (__m128d x)	Y	Y	Computes the value of 2 raised to the power of two packed double precision numbers.
__m128 vrs4_exp2f (__m128 x)	Y	Y	Computes the value of 2 raised to the power of four packed single precision numbers.
void vrda_exp2 (int len,double *src,double *dst)	Y	Y	Computes the value of 2 raised to the power of array of packed double precision numbers.
void vrsa_exp2f (int len, float *src,float *dst)	Y	Y	Computes the value of 2 raised to the power of array of packed single precision numbers.
__m128d vrd2_expm1 (__m128d x)	Y	Y	Computes the value of 'exp (x) - 1' of two packed double precision numbers.
__m128 vrs4_expm1f (__m128 x)	Y	Y	Computes the value of 'exp (x) - 1' of four packed single precision numbers.
void vrda_expm1 (int len,double *src,double *dst)	Y	Y	Computes the value of 'exp (x) - 1' of array of packed double precision numbers.
void vrsa_expm1f (int len, float* x, float* y)	Y	Y	Computes the value of 'exp (x) - 1' of array of packed single precision numbers.
__m128d vrd2_log (__m128d x)	Y	Y	Computes the natural logarithm of x two packed double precision numbers.
__m128 vrs4_logf (__m128 x)	Y	Y	Computes the natural logarithm of four packed single precision numbers.
void vrda_log (int len,double *src,double *dst)	Y	Y	Computes the natural logarithm of array of packed double precision numbers.
void vrsa_logf (int len, float* x, float* y)	Y	Y	Computes the natural logarithm of array of packed single precision numbers.
__m128d vrd2_log10 (__m128d x)	Y	Y	Computes the base 10 logarithm of x two packed double precision numbers.
__m128 vrs4_log10f (__m128 x)	Y	Y	Computes the base 10 logarithm of four packed single precision numbers.
void vrda_log10 (int len,double *src,double *dst)	Y	Y	Computes the base 10 logarithm of array of packed double precision numbers.

void vrsa_log10f (int len, float* x, float* y)	Y	Y	Computes the base 10 logarithm of array of packed single precision numbers.
__m128d vrd2_log1p (__m128d x)	Y	Y	Computes the value of (Natural logrithm) 'log (1 + x)' of two packed double precision numbers. It is computed in a way that is accurate even if the value of x is near zero
__m128 vrs4_log1pf (__m128 x)	Y	Y	Computes the value of (Natural logrithm) 'log (1 + x)' of four packed single precision numbers. It is computed in a way that is accurate even if the value of x is near zero
void vrda_log1p (int len,double *src,double *dst)	Y	Y	Computes the value of (Natural logrithm) 'log (1 + x)' of array of packed double precision numbers. It is computed in a way that is accurate even if the value of x is near zero
void vrsa_log1pf (int len, float* x, float* y)	Y	Y	Computes the value of (Natural logrithm) 'log (1 + x)' of array of packed single precision numbers. It is computed in a way that is accurate even if the value of x is near zero
__m128d vrd2_log2 (__m128d x)	Y	Y	Computes the base 2 logarithm of two packed double precision numbers.
__m128 vrs4_log2f (__m128 x)	Y	Y	Computes the base 2 logarithm of four packed single precision numbers.
void vrda_log2 (int len,double *src,double *dst)	Y	Y	Computes the base 2 logarithm of array of packed double precision numbers.
void vrsa_log2f (int len, float* x, float* y)	Y	Y	Computes the base 2 logarithm of array of packed single precision numbers.
__m128d vrd2_pow (__m128d x, __m128d y)	N	N	Computes yth power of two packed double precision numbers (x).
__m128 vrs4_powf (___m128 x, __m128 y)	N	N	Computes yth power of four packed single precision numbers (x).
void vrsa_powf (int len, float *src1, float *src2, float* dst)	N	N	Computes yth power of array of packed single precision numbers (x).

__m128 vrs4_powxf (__m128 x, float y)	N	N	Computes yth power of four packed single precision numbers (x).
void vrsa_powxf (int len, float *src1, float src2, float* dst)	N	N	Computes yth power of array of packed single precision numbers (x).
__m128d x vrd2_sin (__m128d x)	Y	Y	Computes sine of two packed double precision numbers.
__m128 vrs4_sinf (__m128 x)	Y	Y	Computes sine of four packed single precision numbers.
void vrda_sin (int n, double *x, double *y)	Y	Y	Computes sine of array of packed double precision numbers.
void vrsa_sinf (int n, float *x, float *y)	Y	Y	Computes sine of array of packed single precision numbers.
void vrd2_sincos (__m128d x, __m128d y, __m128d z)	N	N	Computes sine and cosine of two packed double precision numbers.
void vrs4_sincosf (__m128 x, __m128 ys, __m128 yc)	N	N	Computes sine and cosine of four packed single precision numbers.
void vrda_sincos (int n, double *x, double *ys, double *yc)	N	N	Computes sine and cosine of array of packed double precision numbers.
void vrsa_sincosf (int n, float *x, float *ys, float *yc)	N	N	Computes sine and cosine of array of packed single precision numbers.
__m128d vrd2_tan (__m128d x)	Y	Y	Computes tangent of two packed double precision numbers.
__m128 vrs4_tanf (__m128 x)	Y	Y	Computes tangent of four packed single precision numbers.