AN660

Floating Point Math Functions

Author: Frank.J. Testa
FJT Consulting

INTRODUCTION

This application note presents implementations of the following math routines for the Microchip PICmicro™ microcontroller family:

$\mathrm{sqrt}(x)$	square root function, $\sqrt{x}$
$\exp(x)$	exponential function, e^{x}
$\mathrm{exp10}(x)$	base 10 exponential function, 10^{x}
$\log(x)$	natural log function, $\ln x$
$\mathrm{log10}(x)$	common log function, $\log 10x$
$\sin(x)$	trigonometric sine function
$\cos(x)$	trigonometric cosine function
$\sin\cos(x)$	trigonometric sine and cosine functions
$\mathrm{pow}(x, y)$	power function, x^{y}
$\mathrm{floor}(x)$	floor function, largest integer not greater than x, as float, $\lfloor x \rfloor$
$\mathrm{taxxb}(a, b)$	floating point logical comparison tests
$\mathrm{rand}(x)$	integer random number generator

Routines for the PIC16CXXX and PIC17CXXX families are provided in a modified IEEE 754 32-bit format together with versions in 24-bit reduced format.

The techniques and methods of approximation presented here attempt to balance the usually conflicting goals of execution speed verses memory consumption, while still achieving full machine precision estimates. Although 32-bit arithmetic routines are available and constitute extended precision for the 24-bit versions, no extended precision routines are currently supported for use in the 32-bit routines, thereby requiring more sophisticated error control algorithms for full or nearly full machine precision function estimation. Differences in algorithms used for the PIC16CXXX and PIC17CXXX families are a result of performance and memory considerations and reflect the significant platform dependence in algorithm design.

MATHEMATICAL FUNCTION EVALUATION

Evaluation of elementary and mathematical functions is an important part of scientific and engineering computing. Although straightforward Taylor series approximations for many functions of interest are well known, they are generally not optimal for high performance function evaluation. Many other approaches are available and the proper choice is based on the relative speeds of floating point and fixed point arithmetic operations and therefore is heavily implementation dependent.

Although the precision of fixed point arithmetic is usually discussed in terms of absolute error, floating point calculations are typically analyzed using relative error. For example, given a function f and approximation p, absolute error and relative error are defined by

$$\text{abs error} \equiv |p - f| \qquad \text{rel error} \equiv \left|\frac{p - f}{f}\right|$$

In binary arithmetic, an absolute error criterion reflects the number of correct bits to the right of the binary point, while a relative error standard determines the number of significant bits in a binary representation and is in the form of a percentage.

In the 24-bit reduced format case, the availability of extended precision arithmetic routines permits strict 0.5*ulp, or one-half **U**nit in the **L**ast **P**osition, accuracy, reflecting a relative error standard that is typical of most floating point operations. The 32-bit versions cannot meet this in all cases. The absence of extended precision arithmetic requires more time consuming pseudo extended precision techniques to only approach this standard. Although noticeably smaller in most cases, the worst case relative error is usually less than 1*ulp for the 32-bit format. Most of the approximations, presented here on the PIC16CXXX and PIC17CXXX processors, utilize minimax polynomial or minimax rational approximations together with range reduction and some segmentation of the interval on the transformed argument. Such segmentation is employed only when it occurs naturally from the range reduction, or when the gain in performance is worth the increased consumption of program memory.

RANGE REDUCTION

Since most functions of scientific interest have large domains, function identities are typically used to map the argument to a considerably smaller region where accurate approximations require a reasonable effort. In most cases range reduction must be performed carefully in order to prevent the introduction of cancellation error to the approximation. Although this process can be straightforward when extended precision routines are available, their unavailability requires more complex pseudo extended precision methods[3,4]. The resulting interval on the transformed argument sometimes naturally suggests a segmented representation where dedicated approximations are employed in each subinterval. In the case of the trigonometric functions $\sin(\boldsymbol{x})$ and $\cos(\mathrm{x})$, reduction of the infinite natural domain to a region small enough to effectively employ approximation cannot be performed accurately for an arbitrarily large $\boldsymbol{x}$ using finite precision arithmetic, resulting in a threshold in $|\boldsymbol{x}|$ beyond which a loss of precision occurs. The magnitude of this threshold is implementation dependent.

MINIMAX APPROXIMATION

Although series expansions for the elementary functions are well known, their convergence is frequently slow and they usually do not constitute the most computationally efficient method of approximation. For example, the exponential function has the Maclaurin series expansion given by

$$\boldsymbol{e}^{\boldsymbol{x}} = \sum_{\boldsymbol{j}=0}^{\infty} \frac{\boldsymbol{x}^{\boldsymbol{j}}}{\boldsymbol{j}!} = 1 + \boldsymbol{x} + \frac{\boldsymbol{x}^2}{2!} + \frac{\boldsymbol{x}^3}{3!} + \ldots$$

To estimate the function on the interval [0,1], truncation of the series to the first two terms yields the linear approximation,

$$\boldsymbol{e}^{\boldsymbol{x}} \approx 1 + \boldsymbol{x} \ ,$$

a straight line tangent to the graph of the exponential function at $\boldsymbol{x} = 0$. On the interval [0,1], this approximation has a minimum relative error of zero at $\boldsymbol{x} = 0$, and a maximum relative error of $|2\text{-}\boldsymbol{e}|/\boldsymbol{e} = 0.26424$ at $\boldsymbol{x} = 1$, underestimating the function throughout the interval. Recognizing that this undesirable situation is in part caused by using a tangent line approximation at one of the endpoints, an improvement could be made by using a tangent line approximation, for example, at the midpoint $\boldsymbol{x} = 0.5$, yielding the linear function,

$$\boldsymbol{e}^{\boldsymbol{x}} \approx \boldsymbol{e}^{1/2}(\boldsymbol{x} + 0.5) \ ,$$

with a minimum relative error of zero at $\boldsymbol{x} = 0.5$, a maximum relative error of 0.17564 at $\boldsymbol{x} = 0$, and relative error of 0.09020 at $\boldsymbol{x} = 1$, again underestimating the function throughout the interval. We could reduce the maximum error even further by adjusting the intercept of the above approximation, producing subintervals of both positive and negative error, together with possibly equalizing the values of maximum error at each occurrence by manipulating both the slope and intercept of the linear approximation. This is a simple example of a very powerful result in approximation theory known as minimax approximation, whereby a polynomial approximation of degree n to a continuous function can always be found such that the maximum error is a minimum, and that the maximum error must occur at least at n + 2 points with alternating sign within the interval of approximation. It is important to note that the resulting minimax approximation depends on the choice of a relative or absolute error criterion. The evaluation of the minimax coefficients is difficult, usually requiring an iterative procedure known as Remes' method, and historically accounting for the attention given to near-minimax approximations such as Chebyshev polynomials because of greater ease of computation. With the advances in computing power, Remes' method has become much more tractable, resulting in iterative procedures for minimax coefficient evaluation[3]. Remarkably, this theory can be generalized to rational functions, offering a richer set of approximation methods in cases where division is not too slow. In the above simple example, the minimax linear approximation on the interval [0,1] is given by

$$\boldsymbol{e}^{\boldsymbol{x}} \approx 1.71828\boldsymbol{x} + 0.89407$$

$$\text{max error} = 0.10593 \ ,$$

with a maximum relative error of 0.10593, occurring with alternating signs at the n + 2 = 3 points ($\boldsymbol{x} = 0$, $\boldsymbol{x} = 0.5413$, and $\boldsymbol{x} = 1$). Occasionally, constrained minimax approximation[2] can be useful in that some coefficients can be required to take on specific values because of other considerations, leading to effectively near-minimax approximations.

The great advantage in using minimax approximations lies in the fact that minimizing the maximum error leads to the fewest number of terms required to meet a given precision. The number of terms is also dramatically affected by the size of the interval of approximation[1], leading to the concept of segmented representations, where the interval of approximation is split into subintervals, each with a dedicated minimax approximation. For the above example, the interval [0,1] can be split into the subintervals [0,0.5] and [0.5,1], with the linear minimax approximations given by

$$\boldsymbol{e}^{\boldsymbol{x}} \approx \begin{cases} 1.29744\boldsymbol{x} + 0.97980, \ [0, 0.5], \ \text{max error} = 0.02020 \\ 2.13912\boldsymbol{x} + 0.54585, \ [0.5, 1], \ \text{max error} = 0.03331. \end{cases}$$

Since the subintervals were selected for convenience, the maximum relative error is different for the two subintervals but nevertheless represents a significant improvement over a single approximation on the interval [0,1], with the maximum error reduced by a factor greater than three. Although a better choice for the split, equalizing the maximum error over the subintervals, can be found, the overhead in finding the correct subinterval for a given argument would be much greater than that for the convenient choice used above. The minimax approximations used in the implementations for the PIC16CXXX and PIC17CXXX device families presented here, have been produced by applying Remes' method to the specific intervals in question[3].

USAGE

For the unary operations, input argument and result are in AARG, with the exception of the sincos routines where the cosine is returned in AARG and the sine in BARG. The power function requires input arguments in AARG and BARG, and produces the result in AARG. Although the logical test routines also require input arguments in AARG and BARG, the result is returned in the W register.

SQUARE ROOT FUNCTION

The natural domain of the square root function is all nonnegative numbers, leading to the effective domain [0,MAXNUM] for the given floating point representation. All routines begin with a domain test on the argument, returning a domain error if outside the above interval.

On the PIC17CXXX, the greater abundance of program memory together with improved floating point division, using the hardware multiply permits a standard Newton-Raphson iterative approach for square root evaluation[1]. Range reduction is produced naturally by the floating point representation,

$$x = f \cdot 2^{e}, \text{ where } 1 \le f < 2,$$

leading to the expression

$$\sqrt{x} = \begin{cases} \sqrt{f} \cdot 2^{e/2}, & e \ \text{even} \\ \sqrt{f} \cdot \sqrt{2} \cdot 2^{e/2}, & e \ \text{odd} \end{cases}$$

The approximation to $\sqrt{f}$ utilizes a table lookup of 16-bit estimates of the square root as a seed to a single Newton-Raphson iteration

$$y = \left(y_0 + \frac{f}{y_0}\right)/2,$$

where the precision of the result is guaranteed by the precision of the seed and the quadratic conversion of the method, whereby the number of significant bits is doubled upon each iteration. For the 24-bit case, the seed is generated by zeroth degree minimax approximations, while in the 32-bit case, linear interpolation between consecutive square root estimates is employed.

Because of limited memory on the PIC16CXXX as well as a slower divide routine, alternative methods must be used.

For the 24-bit format, the approximation to $\sqrt{f}$ is obtained from segmented fourth degree minimax polynomials on the intervals [1,1.5] and [1.5,2.0]. In the 32-bit case, the function $\sqrt{f} = \sqrt{1+z}$ on the interval [0,1] in z, is obtained from a minimax rational approximation of the form

$$\sqrt{1+z} \approx 1 + z\frac{p(z)}{q(z)}, \text{ where } z \equiv f - 1.$$

EXPONENTIAL FUNCTIONS

While the actual domain of the exponential function consists of all the real numbers, a limitation must be made to reflect the finite range of the given floating point representation. In our case, this leads to the effective domain for the exponential function [MINLOG,MAXLOG], where

$$\mathrm{MINLOG} \equiv \ln(2^{-126}) \qquad \mathrm{MAXLOG} \equiv \ln(2^{128}).$$

All routines begin with a domain test on the argument returning a domain error if outside the above interval.

For the 24-bit reduced format, given the availability of extended precision routines, the exponential function is evaluated using the identity

$$e^{x} = 2^{x/\ln 2} = 2^{n+z} = 2^{n} \cdot 2^{z},$$

where n is an integer and $0 \le z < 1$. Range reduction is performed by first finding the integer n and then computing z. The base two exponential function is then approximated by third degree minimax polynomials in a segmented representation on the subintervals [0,0.25], [0.25,0.5], [0.5,0.75] and [0.75,1.0], permitting 0.5*ulp accuracy throughout the domain [MINLOG,MAXLOG].

For the 32-bit modified IEEE format, the lack of extended precision routines requires a more complex algorithm to approach a 0.5*ulp standard in most cases, leading to a worst case error less than 1*ulp. The exponential function in this case is based on the expansion

$$e^{x} = e^{z + n\ln 2} = 2^{n} \cdot e^{z},$$

where n is an integer and $-0.5\ln 2 \le z < 0.5\ln 2$, with the exponential function evaluated on this interval using segmented fifth degree minimax approximations on the subintervals $[-0.5\ln 2, 0]$ and $[0, 0.5\ln 2]$.

During range reduction, the integer n is first evaluated and then the transformed argument z is obtained from the expression $z = x - n\ln 2$.

Because of the problem of serious cancellation error in this difference, pseudo extended precision methods have been developed[4], where $\ln 2$ is decomposed into a number close to $\ln 2$ but containing slightly more than half its lower significant bits zero, and a much smaller residual number. Specifically, the decomposition given by $\ln 2 = c_1 - c_2$,

where $c_1 \equiv 0.693359375$

and $c_2 \equiv 0.00021219444005469$,

produces the evaluation of z in the form

$$z = (x - n \cdot c_1) + n \cdot c_2,$$

where the term in parentheses is usually computed exactly, with only rounding errors present in the second term[3].

The base 10 exponential function routines for the reduced 24-bit and 32-bit formats are completely analogous to the standard exponential routines with the base e replaced by the base 10 in most places.

LOG FUNCTIONS

The effective domain for the natural log function is (0,MAXNUM], where MAXNUM is the largest number in the given floating point representation. All routines begin with a domain test on the argument, returning a domain error if outside the above interval.

For the 24-bit reduced format, given the availability of extended precision routines, the natural log function is evaluated using the identity[1]

$$\ln x = \ln 2 \cdot \log_2 x = \ln 2 \cdot (n + \log_2 f),$$

where n is an integer and $0.5 \le f < 1$. The final argument z is obtained through the additional transformation[3]

$$z \equiv \begin{cases} 2f - 1, n = n - 1, & f < 1/\sqrt{2} \\ f - 1, & otherwise \end{cases},$$

naturally leading to a segmented representation of $\log_2 f = \log_2(1 + z)$ on the subintervals $[1/\sqrt{2} - 1, 0]$ and $[0, \sqrt{2} - 1]$, utilizing minimax rational approximations in the form

$$\log_2(1 + z) \approx z\frac{p(z)}{q(z)},$$

where $p(\mathrm{x})$ is linear and $q(\mathrm{x})$ is quadratic in x.

For the 32-bit format, computation of the natural log is based on the alternative expansion[3]

$$\ln x = \ln f + \ln 2^n = \ln f + n \cdot \ln 2,$$

where n is an integer and $0.5 \le f < 1$. The final argument z is obtained through the additional transformation

$$z \equiv \begin{cases} 2f - 1, n = n - 1, & f < 1/\sqrt{2}, \\ f - 1, & otherwise \end{cases}$$

naturally leading to a segmented representation of $\ln f = \ln(1 + z)$ on the subintervals $[1/\sqrt{2} - 1, 0]$ and $[0, \sqrt{2} - 1]$, using the effectively constrained minimax form[4] given by

$$\log_2(1 + z) \approx z - 0.5 \cdot z^2 + z\left(z^2 \cdot \frac{p(z)}{q(z)}\right),$$

where $p(\mathrm{x})$ is linear and $q(\mathrm{x})$ is quadratic in x. The rationale for this form is that if the argument z is exact, the first term has no error and the second has only rounding error, thereby leading to more control over the propagation of rounding error than is possible in the simpler form used in the 24-bit case. The final step in the log evaluation is again performed in pseudo extended precision arithmetic in the form[3]

$$\ln f + n \cdot \ln 2 = (\ln f - n \cdot c_2) + n \cdot c_1$$

where the decomposition of $\ln 2$ is the same used in the exponential function.

The common logarithm routine for the reduced 24-bit format is completely analogous to the natural log routine with the base e replaced by the base 10 in most places. In the 32-bit case, the common log is obtained from the natural log through a standard conversion via fixed point multiplication by the common log of e in extended precision.

TRIGONOMETRIC FUNCTIONS

Evaluation of the sine and cosine functions, given their infinite natural domains, clearly requires careful range reduction techniques, especially in the absence of extended precision routines in the 32-bit format.

Susceptible to cancellation and roundoff errors, this process will always fail for arguments beyond some large threshold, leading to potentially serious loss of precision. The size of this threshold is heavily dependent on the range reduction algorithm and the available precision, leading to the value[3,4]

$$\mathrm{LOSSTHR} = \frac{\pi}{4} \cdot 2^{\frac{24}{2}} = 1024 \cdot \pi$$

for this implementation utilizing pseudo extended precision methods and the currently available fixed point and single precision floating point routines. A domain error is reported if this threshold is exceeded.

The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the alternative trigonometric argument z on $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, through the definition[3]

$$z = x \mod \frac{\pi}{4},$$

produced by first evaluating y and j through the relations

$$y = \left\lfloor \frac{x}{\pi/4} \right\rfloor, \quad j = y - 8 \cdot \left\lfloor \frac{y}{8} \right\rfloor,$$

where j equals the correct octant. For j odd, adding one to j and y eliminates the odd octants. Additional logic on j and the sign of the result, representing a reflection of angles greater than π through the origin, leads to appropriate use of the sine or cosine routine in each case. The calculation of z is then obtained through a pseudo extended precision method[3,4]

$$z = x \mod \frac{\pi}{4} = x - y \cdot \frac{\pi}{4}$$

$$= ((x - p_1 \cdot y) - p_2 \cdot y) - p_3 \cdot y$$

where

$$\frac{\pi}{4} = p_1 + p_2 + p_3, \quad p_1 \approx \frac{\pi}{4} \quad \text{and} \quad p_2 \approx \frac{\pi}{4} - p_1$$

with

$$p_1 = 0.78515625$$

$$p_2 = 2.4187564849853515624\text{x}10^{-4}$$

$$p_3 = 3.77489497744597636\text{x}10^{-4}.$$

The numbers p_1 and p_2 are chosen to have an exact machine representation with slightly more than the lower half of the mantissa bits zero, typically leading to no error in computing the terms in parenthesis. This calculation breaks down leading to a loss of precision for $|x|$ beyond the loss threshold or for $|x|$ close to an integer multiple of $\frac{\pi}{4}$. In the latter case, the loss in precision is proportional to the size of y and the number of guard bits available. In the 32-bit modified IEEE implementation, an additional stage of pseudo extended precision is added to control error in this case, where p_3 is chosen to have an exact machine representation with slightly more than the lower half of the mantissa bits zero and p_4 is the residual.

$$p_3 = 3.7747668102383613583\text{x}10^{-8}$$

$$p_4 = 1.28167207614641725\text{x}10^{-12}$$

Although some of the multiplications are performed in fixed point arithmetic, additions are all in floating point and therefore limited by the current single precision routines. It is useful to note that although only the sine and cosine are currently implemented, relatively simple modifications to this range reduction algorithm are necessary for evaluation of the remaining trigonometric functions.

Minimax polynomial expansions for the sine and cosine functions on the interval $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ are in the constrained forms[4]

$$\sin x \approx x + x \cdot x^2 \cdot p(x^2)$$

$$\cos x \approx 1 - 0.5 \cdot x^2 + x^4 \cdot q(x^2)$$

for the full 32-bit single precision format, where p is degree three and q is degree two. In the reduced 24-bit format, we use the simpler forms

$$\sin x \approx x \cdot p(x^2)$$

$$\cos x \approx 1 - x^2 \cdot q(x^2)$$

where p and q are degree two. Because of the patently odd and even nature, respectively, of the sine and cosine functions, the minimax polynomial approximations were generated on the interval $\left[0, \frac{\pi}{4}\right]$. In addition to both sine and cosine routines, a $\text{sincos}(x)$ routine, utilizing only one range reduction calculation, is provided for those frequent situations where both the sine and cosine functions are needed, returning $\cos(x)$ in AARG and $\sin(x)$ in BARG. Generally, in the 32-bit case, these routines meet the 1*ulp relative error performance criterion except in an extremely small number of cases as implied above. The reduced 24-bit format always meets the 0.5*ulp criterion.

POWER FUNCTION

The power function x^y, while defined for all y with $x>0$, is clearly only defined for negative x when y is an integer or an odd root. Unfortunately, odd fractions such as 1/3 for the cube root, cannot be represented exactly in a binary floating point representation, thereby posing problems in defining and recognizing such cases. Therefore, since an integer data type for y in this function is not currently supported, the domain of the power function will be restricted to the interval [0,MAXNUM] for x and [-MAXNUM,MAXNUM] for y, subject to the requirement that the range is also [0,MAXNUM]. In addition, the following special cases will be satisfied:

$$x^0 \equiv 1, \quad x \geq 0$$

$$0^y \equiv \text{MAXNUM}, \quad y < 0,$$

where MAXNUM will be returned through the floating point overflow and saturate if enabled. When extended precision routines are available, evaluation of the power function x^y is usually performed through direct calculation using the identity

$$x^y = \exp(y \cdot \ln x),$$

relying on the extended precision evaluation of the log and exponential functions for control of error propagation. The implementation for the 24-bit format utilizes the 32-bit log and exponential functions to successfully meet the 0.5*ulp relative error criterion.

The unavailability of extended precision routines for the 32-bit format requires considerably more effort with more sophisticated pseudo extended precision methods to control error propagation[3,4]. Because the relative error in the exponential function is proportional to the absolute error of its argument[4], great care must be taken in any algorithm based on an exponential identity. Such methods generally rely on extracting as much of the result as an integer power of two as possible, followed by computations requiring approximations over a relatively small interval. To that end, consider the representation of the argument x given by

$$x = f \cdot 2^e, \text{ where } 0.5 \le f < 1.$$

The power function can then be expressed in the form

$$x^y = 2^{y \cdot \log_2 x},$$

with the base 2 log of x represented as

$$\log_2 x = \log_2(f \cdot 2^e) = e + \log_2\left(\frac{a \cdot f}{a}\right)$$

$$= e + \log_2 a + \log_2\left(1 + \frac{f - a}{a}\right),$$

where a is chosen so that $(f - a)/a$ is small. Rather than a single value of a, we choose a set of values of the form

$$a_k = 2^{-k/16}, \quad k = 0, 1 \ldots, 16 ,$$

resulting in an effectively segmented representation[3,4]. For a given f, the value of a_k for even k, nearest to f is chosen, resulting in an argument $v = (f - a_k)/a_k$ to the function

$$\log_2(1 + v), \quad 2^{-1/16} - 1 < v < 2^{1/16} - 1.$$

Since the numbers a_k cannot be represented exactly in full precision, psuedo extended precision evaluation of v is performed through the expansion

$$= \frac{(f - a_k)}{a_k} = \left(\frac{f - A_k - B_k}{A_k + B_k}\right) = \frac{f - A_k - (f \cdot C_k}{A_k}$$

$$C_k \equiv \frac{B_k}{A_k},$$

where $a_k = A_k + B_k$. The number A_k is equal to a_k rounded to machine precision, and then B_k is the difference computed in higher precision. This method assures evaluation of v with a maximum relative error less than 1*ulp. A minimax approximation of the form

$$\log(1 + v) \approx v - \frac{v^2}{2} + v^3 \cdot \frac{p(v)}{q(v)},$$

with first degree polynomials p and q, is used to estimate $\log(1 + v)$, followed by conversion to the required function $\log_2(1 + v)$, leading to the result

$$\log_2 x = e - \frac{k}{16} + \log_2(1 + v).$$

The product $y \cdot \log_2 x$ is now carefully computed by reducing the number y into a sum of two parts with one less than 1/16 and first evaluating small products of similar magnitude and collecting terms. Each stage of this strategy is followed by a similar reduction operation where the large part is an integer plus a number of 16ths. The final form of the product is then expressed as an integer plus a number of 16ths plus a number on the interval [-0.0625,0], leading to a final result expressed in the form

$$x^y = 2^{y \cdot \log_2 x} = 2^i \cdot 2^{-n/16} \cdot 2^h,$$

where 2^h is evaluated by a minimax approximation of the form

$$2^h - 1 \approx h + h \cdot p(h),$$

with a second degree polynomial p. These elaborate measures for controlling error propagation are necessitated by attempting to obtain a full machine precision estimate without any extended precision routines. This is an especially difficult problem in the case of the power function since the relative error in the exponential function is proportional to the absolute error of its argument[4]. Notwithstanding these efforts, the absence of a sticky bit in the floating point implementation leads to a maximum relative error of approximately 2*ulp in a small number of cases. Currently, this function is only supported on the PIC17CXXX.

FLOOR FUNCTION

As a member of the standard C library of mathematical functions, $\text{floor}(\boldsymbol{x}) \equiv \lfloor \boldsymbol{x} \rfloor$, finds the largest integer not greater than $\boldsymbol{x}$, as a floating point number. The implementation used here finds the location of the binary point implied by the exponent, thereby determining the number of low ordered bits to be zeroed. The bits are cleared by byte while greater than or equal to eight, and the remaining bits are cleared by a table lookup for the appropriate mask. When $\boldsymbol{x}$ is negative, the result is rounded down by one in the units position followed by a check for carry out and possible overflow.

FLOOR24(**123.45**) =

FLOOR24(0x8576E6) = 0x857600 = 123.0

FLOOR24(-**123.45**) =

FLOOR24(0x85F6E6) = 0x857800 = -124.0

FLOATING POINT LOGICAL COMPARISON TESTS

Scientific computing frequently requires relational tests on floating point numbers with the operators < (less), <= (less or equal), > (greater), >= (greater or equal), == (equal), != (not equal). The necessary comparisons are made beginning with the exponent, followed if necessary by the mantissa bytes in the format in decreasing order of significance, all modulo the signs of the arguments. The arguments to be tested are placed in AARG and BARG, returning an integer result in the W register of one if the test is true and zero if false.

INTEGER RANDOM NUMBER GENERATOR

The utility function rand() in the standard C library generates random nonnegative integers initially seeded by the related function srand($\boldsymbol{x}$), where $\boldsymbol{x}$ is an integer. This implementation of an integer random number generator uses a standard linear congruential method, based on the relation[6]

$$\boldsymbol{x}_{\boldsymbol{i}+1} = (\boldsymbol{a} \cdot \boldsymbol{x}_{\boldsymbol{i}} + \boldsymbol{c}) \mod \boldsymbol{m},$$

with multiplier $\boldsymbol{a}$, increment $\boldsymbol{c}$, modulus $\boldsymbol{m}$ and initial seed $\boldsymbol{x}_0$. Considerable research has yielded spectral methods for carefully selecting these constants to insure a maximum period together with other important performance criteria. Since the best such performance is usually associated with the largest word size, $\boldsymbol{x}$ is chosen here as a 32-bit integer, together with the following constants useful for this implementation[6]

$$\boldsymbol{a} = 1664525,$$
$$\boldsymbol{c} = 1,$$
$$\boldsymbol{m} = 2^{32},$$

producing excellent results from standard spectral tests[6]. In this case, the value of m corresponds to the period of the generator, indicating that all possible 32-bit integers will be generated before any repetitions and leading to the corresponding definition

$$\mathrm{RAND\_MAX} = 2^{32} - 1 = 4294967295.$$

Actually, the non-zero value of c is arbitrary for a good choice of the multiplier a with the restriction that it has no common factor with m. Although the calculation must be performed exactly, performance can be improved by recognizing that the binary representation of the multiplier a uses only three bytes, thereby requiring only a 32- by 24-bit fixed point multiply in the algorithm with no possible carryout after the addition of c, chosen here as $c = 1$ for simplicity. It is important to note that the initial seed x_0 may be chosen arbitrarily and the full 32-bit current value of x must saved between calls to preserve the efficacy of the method. Additional RAM locations, $\mathrm{RANDB}x$, $x = 0,1,2,3$, have been added for this purpose and are not used by any other routine in the library.

Since the least significant bits of x are not very random, the best approach in constructing random integers over a given range is to view x/m as a random fraction between 0 and 1 with the binary point to the left of the MSb, and multiply by the desired integer range[6].

EXAMPLES

In evaluating any of the above functions, the appropriate PIC16CXXX or PIC17CXXX floating point values must be loaded into AARG for a unary operation, and AARG and BARG for a binary operation. For example, the argument $x = 27.465$ has the extended PICmicro™ microcontroller floating point representation $0x835BB851EB$, leading to the 32-bit, rounded to the nearest number, 0x835BB852. An extended precision calculation of this nearest machine number is given by 27.465000152587890625, illustrating the effect of truncation error in floating point representations of even apparently simple numbers. Evaluation of $\mathrm{sqrt}(x)$ is then implemented as follows:

```
MOVLW           0x83
MOVWF           AEXP
MOVLW           0x5B
MOVWF           AARGB0
MOVLW           0xB8
MOVWF           AARGB1
MOVLW           0x52
MOVWF           AARGB3

CALL            SQRT32
```

If rounding is enabled, the 32-bit result in AARG is $0x8127B3DD$. If rounding is disabled, an additional byte of guard bits is available contiguously and AARG = $0x8127B3DD00$. For any of the other unary operations, simply call the appropriate function in place of the square root. Using the values $x = 0x835BB852$ and $y = 0x8127B3DD$, calls to the above functions yield the results shown in Table 1.

It is important to note that the exact PIC16CXXX results were computed on an extended precision calculator and converted to Microchip format using the exact decimal values of the 32-bit numbers x and y. The relative errors are all less than 0.5*ulp except for the sine function, where the error is slightly greater than 0.5*ulp, resulting in a rounded to the nearest result with a 1*ulp error.

On the PIC17CXXX, the fractional part of AARG resides in p-registers, thereby permitting direct register to register moves using the `MOVFP` and `MOVPF` instructions during loading of AARG and BARG from other RAM locations.

TABLE 1: FUNCTION ROUTINES PERFORMANCE DATA

Routine	Unrounded PICmicro	Exact PICmicro	Decimal
SQRT32	0x8127B3DD00	0x8127B3DD39	5.24070607
EXP32	0xA64536D500	0xA64536D4DE	$8.47028477\text{x}10^{11}$
EXP1032	0xDA16D3D6E0	0xDA16D3D6AE	$2.91742804\text{x}10^{27}$
LOG32	0x805406C210	0x805406C208	3.31291247
LOG1032	0x7F3829EE22	0x7f3829EE1C	1.43877961
SIN32	0x7E394CC500	0x7E394CC459	$7.23827621\text{x}10^{-1}$
COS32	0x7EB0A29580	0x7EB0A295C5	$-6.89980851\text{x}10^{-1}$
POW32	0x9804563F38	0x9804563EC1	$3.46913232\text{x}10^{7}$

APPENDIX A: PERFORMANCE DATA

TABLE A-1: PIC17CXXX ELEMENTARY FUNCTION PERFORMANCE DATA

Routine	Max Cycles	Min Cycles	Program Memory	Data Memory
SQRT24	327	6	325	7
EXP24	999	645	339	5
EXP1024	1002	646	339	5
LOG24	1442	12	235	10
LOG1024	1457	12	236	10
SIN24	1625	834	317	11
COS24	1637	942	317	11
SINCOS24	2248	1516	339	15
POW24	4255	2852	43	4
FLOOR24	39	18	94	8
TALTB24	27	8	43	6
TALEB24	25	8	47	6
TAGTB24	27	8	47	6
TAGEB24	25	8	43	6
TAEQB24	11	4	10	6
TANEB24	11	4	10	6
RND3224	21	3	20	5
SQRT32	568	10	357	10
EXP32	2024	14	374	15
EXP1032	2084	14	392	15
LOG32	2147	12	264	14
LOG1032	2308	2001	31	1
SIN32	2408	1338	462	11
COS32	2405	1256	462	11
SINCOS32	3432	2328	482	15
POW32	5574	4280	699	29
FLOOR32	45	30	138	8
RAND32	117	117	25	4
TALTB32	33	8	59	8
TALEB32	31	8	54	8
TAGTB32	33	8	59	8
TAGEB32	31	8	54	8
TAEQB32	14	4	13	8
TANEB32	14	4	13	8
RND4032	23	3	22	6

Note: Program and data memory values do not include dependency requirements.

TABLE A-2: PIC16CXXX ELEMENTARY FUNCTION PERFORMANCE DATA

Routine	Max Cycles	Min Cycles	Program Memory	Data Memory
SQRT24	2968	7	197	6
EXP24	2600	1990	349	6
EXP1024	2561	2043	355	6
LOG24	3555	1662	261	10
LOG1024	3567	1674	259	10
SIN24	4494	2564	368	11
COS24	4505	2736	368	11
SINCOS24	6478	4525	397	15
FLOOR24	55	37	107	8
TALTB24	28	9	48	6
TALEB24	26	9	44	6
TAGTB24	28	9	48	6
TAGEB24	26	9	44	6
TAEQB24	14	5	13	6
TANEB24	14	5	13	6
RND3224	26	3	25	5
SQRT32	4966	7	142	10
EXP32	5411	16	401	14
EXP1032	5384	3515	401	14
LOG32	5406	4797	297	14
LOG1032	5949	5208	16	1
SIN32	6121	4030	474	11
COS32	6098	3568	474	11
SINCOS32	8858	6611	503	15
FLOOR32	61	41	159	10
RAND32	487	487	37	4
TALTB32	34	9	60	8
TALEB32	32	9	56	8
TAGTB32	34	9	60	8
TAGEB32	32	9	56	8
TAEQB32	18	5	17	8
TANEB32	18	5	17	8
RND4032	29	3	28	8

Note: Program and data memory values do not include dependency requirements.

REFERENCES

1. Cavanagh, J.J.F., “Digital Computer Arithmetic,” McGraw-Hill,1984.
2. Hwang, K., “Computer Arithmetic,” John Wiley & Sons, 1979.
3. Scott, N.R., “Computer Number Systems & Arithmetic,” Prentice Hall, 1985.
4. Knuth, D.E., “The Art of Computer Programming, Volume 2,” Addison-Wesley, 1981.
5. F.J.Testa, “IEEE 754 Compliant Floating Point Routines,” AN575, Embedded Control Handbook, Microchip Technology Inc., 1995.

TABLE A-3: PIC17CXXX ELEMENTARY FUNCTION DEPENDENCIES

Routine	Dependencies						
SQRT24	FPA32	FPD32	FXM1616U	RND3224			
EXP24	FPX32	FXM2416U	FLOOR24	INT2416	RND3224		
EXP1024	FPX32	FXM2416U	FLOOR24	INT2416	RND3224		
LOG24	FPX32	FLO1624	FXM2424U	RND3224			
LOG1024	FPX32	FLO1624	FXM2424U	RND3224			
SIN24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
COS24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
SINCOS24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
POW24	LOG32	EXP32	RND3224				
SQRT32	FPA32	FPD32	FXM2424U	RND4032			
EXP32	FPX32	FXM3224U	FLOOR32	FXM2416U	FXM2424U	INT2416	RND4032
EXP1032	FPX32	FXM3224U	FLOOR32	FXM2416U	FXM2424U	INT2416	RND4032
LOG32	FPX32	FLO1624	FXM2424U	FXM2416U	RND4032		
LOG1032	LOG32	FXM3232U	RND4032				
SIN32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	FXM3232U	RND4032
COS32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	FXM3232U	RND4032
SINCOS32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	FXM3232U	RND4032
POW32	FPX32	TAXXB32	FLO1624	INT3224	FLOOR32	RND4032	
RAND32	FXM3224U						

TABLE A-4: PIC16CXXX ELEMENTARY FUNCTION DEPENDENCIES

Routine	Dependencies						
SQRT24	FPX32	FXM2424U	RND3224				
EXP24	FPX32	FXM2416U	FLOOR24	INT2416	RND3224		
EXP1024	FPX32	FXM2416U	FLOOR24	INT2416	RND3224		
LOG24	FPX32	FLO1624	FXM2424U	RND3224			
LOG1024	FPX32	FLO1624	FXM2424U	RND3224			
SIN24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
COS24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
SINCOS24	FPX32	FXM2416U	INT3224	FLO2432	FXM2424U	RND3224	
SQRT32	FPX32	FXM3232U	RND4032				
EXP32	FPX32	FXM3224U	FLOOR32	INT2416	RND4032		
EXP1032	FPX32	FXM3224U	FLOOR32	INT2416	RND4032		
LOG32	FPX32	FLO1624	FXM2424U	FXM2416U	RND4032		
LOG1032	LOG32						
SIN32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	RND4032	
COS32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	RND4032	
SINCOS32	FPX32	FXM3224U	INT3224	FLO2432	FXM2416U	RND4032	
RAND32	FXM3224U						

NOTES:

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX B:

B.1 Device Family Include File

```
;        RCS Header $Id: dev_fam.inc 1.2 1997/03/24 23:25:07 F.J.Testa Exp $

;        $Revision: 1.2 $

;  DEV_FAM.INC   Device Family Type File, Version 1.00    Microchip Technology, Inc.
;
; This file takes the defined device from the LIST directive, and specifies a
; device family type and the Reset Vector Address (in RESET_V).
;
;*******
;*******   Device Family Type, Returns one of these three Symbols (flags) set
;*******   (other two are cleared) depending on processor selected in LIST Directive:
;*******          P16C5X, P16CXX, or P17CXX
;*******   Also sets the Reset Vector Address in symbol RESET_V
;*******
;*******   File Name:   DEV_FAM.INC
;*******   Revision:    1.00.00        08/24/95       MP
;*******                1.00.01        03/21/97       AL
;*******
;
TRUE                EQU   1
FALSE               EQU   0
;
P16C5X      SET    FALSE       ; If P16C5X, use INHX8M file format.
P16CXX      SET    FALSE       ; If P16CXX, use INHX8M file format.
P17CXX      SET    FALSE       ; If P17CXX, the INHX32 file format is required
;                              ;            in the LIST directive
RESET_V     SET    0x0000      ; Default Reset Vector address of 0h
                               ;   (16Cxx and 17Cxx devices)
P16_MAP1    SET    FALSE       ; FOR 16C60/61/70/71/710/711/715/84 Memory Map
P16_MAP2    SET    FALSE       ; For all other 16Cxx Memory Maps
;
;******    16CXX  ***********
;
    IFDEF   __14000
P16CXX      SET    TRUE        ; If P14000, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C554
P16CXX      SET    TRUE        ; If P16C554, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C556
P16CXX      SET    TRUE        ; If P16C556, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C558
P16CXX      SET    TRUE        ; If P16C558, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C61
P16CXX      SET    TRUE        ; If P16C61, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
```

```
    IFDEF   __16C62
P16CXX      SET    TRUE         ; If P16C62, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C62A
P16CXX      SET    TRUE         ; If P16C62A, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C63
P16CXX      SET    TRUE         ; If P16C63, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C64
P16CXX      SET    TRUE         ; If P16C64, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C64A
P16CXX      SET    TRUE         ; If P16C64A, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C65
P16CXX      SET    TRUE         ; If P16C65, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C65A
P16CXX      SET    TRUE         ; If P16C65A, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C620
P16CXX      SET    TRUE         ; If P16C620, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C621
P16CXX      SET    TRUE         ; If P16C621, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C622
P16CXX      SET    TRUE         ; If P16C622, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C642
P16CXX      SET    TRUE         ; If P16C642, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C662
P16CXX      SET    TRUE         ; If P16C662, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C710
P16CXX      SET    TRUE         ; If P16C710, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16C71
```

```
P16CXX      SET    TRUE        ; If P16C71, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16C711
P16CXX      SET    TRUE        ; If P16C711, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16C72
P16CXX      SET    TRUE        ; If P16C72, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C73
P16CXX      SET    TRUE        ; If P16C73, use INHX8M file format.
P16_MAP2    SET    TRUE        ;
    ENDIF
;
    IFDEF   __16C73A
P16CXX      SET    TRUE        ; If P16C73A, use INHX8M file format.
P16_MAP2    SET    TRUE        ;
    ENDIF
;
    IFDEF   __16C74
P16CXX      SET    TRUE        ; If P16C74, use INHX8M file format.
P16_MAP2    SET    TRUE        ;
    ENDIF
;
    IFDEF   __16C74A
P16CXX      SET    TRUE        ; If P16C74A, use INHX8M file format.
P16_MAP2    SET    TRUE        ;
    ENDIF
;
    IFDEF   __16C84
P16CXX      SET    TRUE        ; If P16C84, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16F84
P16CXX      SET    TRUE        ; If P16F84, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16F83
P16CXX      SET    TRUE        ; If P16F83, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16CR83
P16CXX      SET    TRUE        ; If P16CR83, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16CR84
P16CXX      SET    TRUE        ; If P16CR84, use INHX8M file format.
P16_MAP1    SET    TRUE
    ENDIF
;
    IFDEF   __16C923
P16CXX      SET    TRUE        ; If P16C923, use INHX8M file format.
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16C924
P16CXX      SET    TRUE        ; If P16C924, use INHX8M file format.
```

```
P16_MAP2    SET    TRUE
    ENDIF
;
    IFDEF   __16CXX             ; Generic Processor Type
P16CXX      SET    TRUE         ; If P16CXX, use INHX8M file format.
P16_MAP2    SET    TRUE         ;
    ENDIF
;
;
;
;******    17CXX  ***********
;
;
    IFDEF   __17C42
P17CXX      SET    TRUE         ; If P17C42, the INHX32 file format is required
;                               ;           in the LIST directive
    ENDIF
;
    IFDEF   __17C43
P17CXX      SET    TRUE         ; If P17C43, the INHX32 file format is required
;                               ;           in the LIST directive
    ENDIF
;
    IFDEF   __17C44
P17CXX      SET    TRUE         ; If P17C44, the INHX32 file format is required
;                               ;           in the LIST directive
    ENDIF
;
    IFDEF   __17CXX             ; Generic Processor Type
P17CXX      SET    TRUE         ; If P17CXX, the INHX32 file format is required
;                               ;           in the LIST directive
    ENDIF
;
;******    16C5X  ***********
;
;
    IFDEF   __16C54
P16C5X      SET    TRUE         ; If P16C54, use INHX8M file format.
RESET_V     SET    0x01FF       ; Reset Vector at end of 512 words
    ENDIF
;
    IFDEF   __16C54A
P16C5X      SET    TRUE         ; If P16C54A, use INHX8M file format.
RESET_V     SET    0x01FF       ; Reset Vector at end of 512 words
    ENDIF
;
    IFDEF   __16C55
P16C5X      SET    TRUE         ; If P16C55, use INHX8M file format.
RESET_V     SET    0x01FF       ; Reset Vector at end of 512 words
    ENDIF
;
    IFDEF   __16C56
P16C5X      SET    TRUE         ; If P16C56, use INHX8M file format.
RESET_V     SET    0x03FF       ; Reset Vector at end of 1K words
    ENDIF
;
    IFDEF   __16C57
P16C5X      SET    TRUE         ; If P16C57, use INHX8M file format.
RESET_V     SET    0x07FF       ; Reset Vector at end of 2K words
    ENDIF
;
    IFDEF   __16C58A
P16C5X      SET    TRUE         ; If P16C58A, use INHX8M file format.
RESET_V     SET    0x07FF       ; Reset Vector at end of 2K words
    ENDIF
;
```

```
    IFDEF   __16C5X             ; Generic Processor Type
P16C5X      SET    TRUE         ; If P16C5X, use INHX8M file format.
RESET_V     SET    0x07FF       ; Reset Vector at end of 2K words
    ENDIF
;
;
    if ( P16C5X + P16CXX + P17CXX != 1 )
MESSG  “WARNING - USER DEFINED: One and only one device family can be selected”
MESSG  “                        May be NEW processor not defined in this file”
    endif
;
```

B.2 Math16 Include File

```
;        RCS Header $Id: math16.inc 2.4 1997/02/11 16:58:49 F.J.Testa Exp $

;        $Revision: 2.4 $

;       MATH16 INCLUDE FILE
;
;       IMPORTANT NOTE: The math library routines can be used in a dedicated application on
;       an individual basis and memory allocation may be modified with the stipulation that
;       on the PIC17, P type registers must remain so since P type specific instructions
;       were used to realize some performance improvements.

;*********************************************************************************************
;
;       GENERAL MATH LIBRARY DEFINITIONS
;
;        general literal constants

;        define assembler constants

B0                equ        0
B1                equ        1
B2                equ        2
B3                equ        3
B4                equ        4
B5                equ        5
B6                equ        6
B7                equ        7

MSB               equ        7
LSB               equ        0


;     define commonly used bits

;     STATUS bit definitions

#define        _C                STATUS,0
#define        _Z                STATUS,2

;
;       general register variables
;
    IF ( P16_MAP1 )

ACCB7           equ     0x0C
ACCB6           equ     0x0D
ACCB5           equ     0x0E
ACCB4           equ     0x0F
ACCB3           equ     0x10
ACCB2           equ     0x11
ACCB1           equ     0x12
ACCB0           equ     0x13
ACC             equ     0x13    ; most significant byte of contiguous 8 byte accumulator
;
SIGN            equ     0x15    ; save location for sign in MSB
;
TEMPB3          equ     0x1C
TEMPB2          equ     0x1D
TEMPB1          equ     0x1E
TEMPB0          equ     0x1F
TEMP            equ     0x1F    ; temporary storage
;
```



```
;       binary operation arguments
;
AARGB7          equ     0x0C
AARGB6          equ     0x0D
AARGB5          equ     0x0E
AARGB4          equ     0x0F
AARGB3          equ     0x10
AARGB2          equ     0x11
AARGB1          equ     0x12
AARGB0          equ     0x13
AARG            equ     0x13    ; most significant byte of argument A
;
BARGB3          equ     0x17
BARGB2          equ     0x18
BARGB1          equ     0x19
BARGB0          equ     0x1A
BARG            equ     0x1A    ; most significant byte of argument B
;
;       Note that AARG and ACC reference the same storage location
;
;*********************************************************************************************
;
;       FIXED POINT SPECIFIC DEFINITIONS
;
;       remainder storage
;
REMB3           equ     0x0C
REMB2           equ     0x0D
REMB1           equ     0x0E
REMB0           equ     0x0F    ; most significant byte of remainder

LOOPCOUNT       equ     0x20    ; loop counter
;
;*********************************************************************************************
;
;       FLOATING POINT SPECIFIC DEFINITIONS
;
;       literal constants
;
EXPBIAS       equ     D'127'
;
;       biased exponents
;
EXP           equ     0x14    ; 8 bit biased exponent
AEXP          equ     0x14    ; 8 bit biased exponent for argument A
BEXP          equ     0x1B    ; 8 bit biased exponent for argument B
;
;       floating point library exception flags
;
FPFLAGS       equ     0x16    ; floating point library exception flags
IOV           equ     0       ; bit0 = integer overflow flag
FOV           equ     1       ; bit1 = floating point overflow flag
FUN           equ     2       ; bit2 = floating point underflow flag
FDZ           equ     3       ; bit3 = floating point divide by zero flag
NAN           equ     4       ; bit4 = not-a-number exception flag
DOM           equ     5       ; bit5 = domain error exception flag
RND           equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                              ; 1 = unbiased rounding to nearest LSB

SAT           equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                              ; exception without saturation, 1 = terminate on
                              ; exception with saturation to appropriate value

    ENDIF
;
;
```

```
    IF ( P16_MAP2 )

ACCB7           equ     0x20
ACCB6           equ     0x21
ACCB5           equ     0x22
ACCB4           equ     0x23
ACCB3           equ     0x24
ACCB2           equ     0x25
ACCB1           equ     0x26
ACCB0           equ     0x27
ACC             equ     0x27    ; most significant byte of contiguous 8 byte accumulator
;
SIGN            equ     0x29    ; save location for sign in MSB
;
TEMPB3          equ     0x30
TEMPB2          equ     0x31
TEMPB1          equ     0x32
TEMPB0          equ     0x33
TEMP            equ     0x33    ; temporary storage
;
;       binary operation arguments
;
AARGB7          equ     0x20
AARGB6          equ     0x21
AARGB5          equ     0x22
AARGB4          equ     0x23
AARGB3          equ     0x24
AARGB2          equ     0x25
AARGB1          equ     0x26
AARGB0          equ     0x27
AARG            equ     0x27    ; most significant byte of argument A
;
BARGB3          equ     0x2B
BARGB2          equ     0x2C
BARGB1          equ     0x2D
BARGB0          equ     0x2E
BARG            equ     0x2E    ; most significant byte of argument B
;
;       Note that AARG and ACC reference the same storage location
;
;*********************************************************************************************
;
;       FIXED POINT SPECIFIC DEFINITIONS
;
;       remainder storage
;
REMB3           equ     0x20
REMB2           equ     0x21
REMB1           equ     0x22
REMB0           equ     0x23    ; most significant byte of remainder

LOOPCOUNT       equ     0x34    ; loop counter
;
;*********************************************************************************************
;
;       FLOATING POINT SPECIFIC DEFINITIONS
;
;       literal constants
;
EXPBIAS         equ     D'127'
;
;       biased exponents
;
EXP             equ     0x28    ; 8 bit biased exponent
AEXP            equ     0x28    ; 8 bit biased exponent for argument A
BEXP            equ     0x2F    ; 8 bit biased exponent for argument B
```

```
;
;       floating point library exception flags
;
FPFLAGS         equ     0x2A    ; floating point library exception flags
IOV             equ     0       ; bit0 = integer overflow flag
FOV             equ     1       ; bit1 = floating point overflow flag
FUN             equ     2       ; bit2 = floating point underflow flag
FDZ             equ     3       ; bit3 = floating point divide by zero flag
NAN             equ     4       ; bit4 = not-a-number exception flag
DOM             equ     5       ; bit5 = domain error exception flag
RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = unbiased rounding to nearest LSb
SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;**********************************************************************************************

;        ELEMENTARY FUNCTION MEMORY

CEXP            equ        0x35
CARGB0          equ        0x36
CARGB1          equ        0x37
CARGB2          equ        0x38
CARGB3          equ        0x39

DEXP            equ        0x3A
DARGB0          equ        0x3B
DARGB1          equ        0x3C
DARGB2          equ        0x3D
DARGB3          equ        0x3E

EEXP            equ        0x3F
EARGB0          equ        0x40
EARGB1          equ        0x41
EARGB2          equ        0x42
EARGB3          equ        0x43

ZARGB0          equ        0x44
ZARGB1          equ        0x45
ZARGB2          equ        0x46
ZARGB3          equ        0x47

RANDB0          equ        0x48
RANDB1          equ        0x49
RANDB2          equ        0x4A

RANDB3          equ        0x4B

;**********************************************************************************************

;        24-BIT FLOATING POINT CONSTANTS

;        Machine precision

MACHEP24EXP     equ        0x6F               ; 1.52587890625e-5 = 2**-16
MACHEP24B0      equ        0x00
MACHEP24B1      equ        0x00

;        Maximum argument to EXP24

MAXLOG24EXP     equ        0x85               ; 88.7228391117 = log(2**128)
MAXLOG24B0      equ        0x31
MAXLOG24B1      equ        0x72
```



```
;        Minimum argument to EXP24

MINLOG24EXP     equ        0x85                  ; -87.3365447506 = log(2**-126)
MINLOG24B0      equ        0xAE
MINLOG24B1      equ        0xAC

;        Maximum argument to EXP1024

MAXLOG1024EXP   equ        0x84                  ; 38.531839445 = log10(2**128)
MAXLOG1024B0    equ        0x1A
MAXLOG1024B1    equ        0x21

;        Minimum argument to EXP1024

MINLOG1024EXP   equ        0x84                  ; -37.9297794537 = log10(2**-126)
MINLOG1024B0    equ        0x97
MINLOG1024B1    equ        0xB8

;        Maximum representable number before overflow

MAXNUM24EXP     equ        0xFF                  ; 6.80554349248E38 = (2**128) * (2 - 2**-15)
MAXNUM24B0      equ        0x7F
MAXNUM24B1      equ        0xFF

;        Minimum representable number before underflow

MINNUM24EXP     equ        0x01                  ; 1.17549435082E-38 = (2**-126) * 1
MINNUM24B0      equ        0x00
MINNUM24B1      equ        0x00

;        Loss threshold for argument to SIN24 and COS24

LOSSTHR24EXP    equ        0x8B                  ; 4096 = sqrt(2**24)
LOSSTHR24B0     equ        0x00
LOSSTHR24B1     equ        0x00

;**********************************************************************************************

;        32-BIT FLOATING POINT CONSTANTS

;        Machine precision

MACHEP32EXP     equ        0x67                  ; 5.96046447754E-8 = 2**-24
MACHEP32B0      equ        0x00
MACHEP32B1      equ        0x00
MACHEP32B2      equ        0x00

;        Maximum argument to EXP32

MAXLOG32EXP     equ        0x85                  ; 88.7228391117 = log(2**128)
MAXLOG32B0      equ        0x31
MAXLOG32B1      equ        0x72
MAXLOG32B2      equ        0x18

;        Minimum argument to EXP32

MINLOG32EXP     equ        0x85                  ; -87.3365447506 = log(2**-126)
MINLOG32B0      equ        0xAE
MINLOG32B1      equ        0xAC
MINLOG32B2      equ        0x50

;        Maximum argument to EXP1032

MAXLOG1032EXP   equ        0x84                  ; 38.531839445 = log10(2**128)
MAXLOG1032B0    equ        0x1A
MAXLOG1032B1    equ        0x20
MAXLOG1032B2    equ        0x9B
```

```
;        Minimum argument to EXP1032

MINLOG1032EXP   equ        0x84                ; -37.9297794537 = log10(2**-126)
MINLOG1032B0    equ        0x97
MINLOG1032B1    equ        0xB8
MINLOG1032B2    equ        0x18

;        Maximum representable number before overflow

MAXNUM32EXP     equ        0xFF                ; 6.80564774407E38 = (2**128) * (2 - 2**-23)
MAXNUM32B0      equ        0x7F
MAXNUM32B1      equ        0xFF
MAXNUM32B2      equ        0xFF

;        Minimum representable number before underflow

MINNUM32EXP     equ        0x01                ; 1.17549435082E-38 = (2**-126) * 1
MINNUM32B0      equ        0x00
MINNUM32B1      equ        0x00
MINNUM32B2      equ        0x00

;        Loss threshold for argument to SIN32 and COS32

LOSSTHR32EXP    equ        0x8B                ; 4096 = sqrt(2**24)
LOSSTHR32B0     equ        0x00
LOSSTHR32B1     equ        0x00
LOSSTHR32B2     equ        0x00


    ENDIF
```

B.3 Math17 Include File

```
;       RCS Header $Id: math17.inc 2.9 1997/01/31 02:23:41 F.J.Testa Exp $

;       $Revision: 2.9 $

;       MATH17 INCLUDE FILE
;
;       IMPORTANT NOTE: The math library routines can be used in a dedicated application on
;       an individual basis and memory allocation may be modified with the stipulation that
;       P type registers must remain so since P type specific instructions were used to
;       realize some performance improvements. This applies only to the PIC17.

;*********************************************************************************************

;       GENERAL MATH LIBRARY DEFINITIONS


;       general literal constants

;       define assembler constants

B0              equ     0
B1              equ     1
B2              equ     2
B3              equ     3
B4              equ     4
B5              equ     5
B6              equ     6
B7              equ     7

MSB             equ     7
LSB             equ     0


;     define commonly used bits

;     STATUS bit definitions

#define  _C      ALUSTA,0
#define  _DC     ALUSTA,1
#define  _Z      ALUSTA,2
#define  _OV     ALUSTA,3

;     general register variables

ACCB7           equ     0x18
ACCB6           equ     0x19
ACCB5           equ     0x1A
ACCB4           equ     0x1B
ACCB3           equ     0x1C
ACCB2           equ     0x1D
ACCB1           equ     0x1E
ACCB0           equ     0x1F
ACC             equ     0x1F    ; most significant byte of contiguous 8 byte accumulator

SIGN            equ     0x21    ; save location for sign in MSB

TEMPB3          equ     0x28
TEMPB2          equ     0x29
TEMPB1          equ     0x2A
TEMPB0          equ     0x2B
TEMP            equ     0x2B    ; temporary storage
```

```
;       binary operation arguments

AARGB7          equ     0x18
AARGB6          equ     0x19
AARGB5          equ     0x1A
AARGB4          equ     0x1B
AARGB3          equ     0x1C
AARGB2          equ     0x1D
AARGB1          equ     0x1E
AARGB0          equ     0x1F
AARG            equ     0x1F    ; most significant byte of argument A

BARGB3          equ     0x23
BARGB2          equ     0x24
BARGB1          equ     0x25
BARGB0          equ     0x26
BARG            equ     0x26    ; most significant byte of argument B


;       Note that AARG and ACC reference the same storage location

;**********************************************************************************************

;       FIXED POINT SPECIFIC DEFINITIONS

;       remainder storage

REMB3           equ     0x18
REMB2           equ     0x19
REMB1           equ     0x1A
REMB0           equ     0x1B    ; most significant byte of remainder

;**********************************************************************************************

;       FLOATING POINT SPECIFIC DEFINITIONS

;       literal constants

EXPBIAS         equ     D'127'

;       biased exponents

EXP             equ     0x20    ; 8 bit biased exponent
AEXP            equ     0x20    ; 8 bit biased exponent for argument A
BEXP            equ     0x27    ; 8 bit biased exponent for argument B

;       floating point library exception flags

FPFLAGS         equ     0x22    ; floating point library exception flags
IOV             equ     0       ; bit0 = integer overflow flag
FOV             equ     1       ; bit1 = floating point overflow flag
FUN             equ     2       ; bit2 = floating point underflow flag
FDZ             equ     3       ; bit3 = floating point divide by zero flag
NAN             equ     4       ; bit4 = not-a-number exception flag
DOM             equ     5       ; bit5 = domain error flag
RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = unbiased rounding to nearest LSB
SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;**********************************************************************************************
```

```
;       ELEMENTARY FUNCTION MEMORY

CEXP            equ     0x34
CARGB0          equ     0x33
CARGB1          equ     0x32
CARGB2          equ     0x31
CARGB3          equ     0x30

DEXP            equ     0x39
DARGB0          equ     0x38
DARGB1          equ     0x37
DARGB2          equ     0x36
DARGB3          equ     0x35

EEXP            equ     0x3E
EARGB0          equ     0x3D
EARGB1          equ     0x3C
EARGB2          equ     0x3B
EARGB3          equ     0x3A

FEXP            equ     0x43
FARGB0          equ     0x42
FARGB1          equ     0x41
FARGB2          equ     0x40
FARGB3          equ     0x3F

GEXP            equ     0x48
GARGB0          equ     0x47
GARGB1          equ     0x46
GARGB2          equ     0x45
GARGB3          equ     0x44

ZARGB0          equ     0x2F
ZARGB1          equ     0x2E
ZARGB2          equ     0x2D
ZARGB3          equ     0x2C

RANDB0          equ     0x4C
RANDB1          equ     0x4B
RANDB2          equ     0x4A
RANDB3          equ     0x49

;**********************************************************************************************

;       24-BIT FLOATING POINT CONSTANTS

;       Machine precision

MACHEP24EXP     equ     0x6F            ; 1.52587890625e-5 = 2**-16
MACHEP24B0      equ     0x00
MACHEP24B1      equ     0x00

;       Maximum argument to EXP24

MAXLOG24EXP     equ     0x85            ; 88.7228391117 = log(2**128)
MAXLOG24B0      equ     0x31
MAXLOG24B1      equ     0x72

;       Minimum argument to EXP24

MINLOG24EXP     equ     0x85            ; -87.3365447506 = log(2**-126)
MINLOG24B0      equ     0xAE
MINLOG24B1      equ     0xAC
```

```
;       Maximum argument to EXP1024

MAXLOG1024EXPe   qu      0x84            ; 38.531839445 = log10(2**128)
MAXLOG1024B0     equ     0x1A
MAXLOG1024B1     equ     0x21

;       Minimum argument to EXP1024

MINLOG1024EXP    equ     0x84            ; -37.9297794537 = log10(2**-126)
MINLOG1024B0     equ     0x97
MINLOG1024B1     equ     0xB8

;       Maximum representable number before overflow

MAXNUM24EXP      equ     0xFF            ; 6.80554349248E38 = (2**128) * (2 - 2**-15)
MAXNUM24B0       equ     0x7F
MAXNUM24B1       equ     0xFF

;       Minimum representable number before underflow

MINNUM24EXP      equ     0x01            ; 1.17549435082E-38 = (2**-126) * 1
MINNUM24B0       equ     0x00
MINNUM24B1       equ     0x00

;       Loss threshold for argument to SIN24 and COS24

LOSSTHR24EXP     equ     0x8A            ; LOSSTHR = sqrt(2**24)*PI/4
LOSSTHR24B0      equ     0x49
LOSSTHR24B1      equ     0x10

;**********************************************************************************************

;       32-BIT FLOATING POINT CONSTANTS

;       Machine precision

MACHEP32EXP      equ     0x67            ; 5.96046447754E-8 = 2**-24
MACHEP32B0       equ     0x00
MACHEP32B1       equ     0x00
MACHEP32B2       equ     0x00

;       Maximum argument to EXP32

MAXLOG32EXP      equ     0x85            ; 88.7228391117 = log(2**128)
MAXLOG32B0       equ     0x31
MAXLOG32B1       equ     0x72
MAXLOG32B2       equ     0x18

;       Minimum argument to EXP32

MINLOG32EXP      equ     0x85            ; -87.3365447506 = log(2**-126)
MINLOG32B0       equ     0xAE
MINLOG32B1       equ     0xAC
MINLOG32B2       equ     0x50

;       Maximum argument to EXP1032

MAXLOG1032EXP    equ     0x84            ; 38.531839445 = log10(2**128)
MAXLOG1032B0     equ     0x1A
MAXLOG1032B1     equ     0x20
MAXLOG1032B2     equ     0x9B
```

```
;       Minimum argument to EXP1032

MINLOG1032EXP    equ     0x84            ; -37.9297794537 = log10(2**-126)
MINLOG1032B0     equ     0x97
MINLOG1032B1     equ     0xB8
MINLOG1032B2     equ     0x18

;       Maximum representable number before overflow

MAXNUM32EXP      equ     0xFF            ; 6.80564774407E38 = (2**128) * (2 - 2**-23)
MAXNUM32B0       equ     0x7F
MAXNUM32B1       equ     0xFF
MAXNUM32B2       equ     0xFF

;       Minimum representable number before underflow

MINNUM32EXP      equ     0x01            ; 1.17549435082E-38 = (2**-126) * 1
MINNUM32B0       equ     0x00
MINNUM32B1       equ     0x00
MINNUM32B2       equ     0x00

;       Loss threshold for argument to SIN32 and COS32

LOSSTHR32EXP     equ     0x8A            ; LOSSTHR = sqrt(2**24)*PI/4
LOSSTHR32B0      equ     0x49
LOSSTHR32B1      equ     0x0F
LOSSTHR32B2      equ     0xDB
```

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX C: PIC16CXXX 24-BIT ELEMENTARY FUNCTION LIBRARY

```
;       RCS Header $Id: math16.mac 1.3 1996/10/05 19:52:32 F.J.Testa Exp $

;       $Revision: 1.3 $

;**********************************************************************************************
;**********************************************************************************************

;       polynomial evaluation macros

POLL124 macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       with leading coefficient of one, and where AARG is assumed have been be saved
;       in DARG when N>1.  The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set

        local           i,j
        variable i = N, j = 0

        variable i = i - 1

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPA32

        variable i = i - 1

        while           i >= 0

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
```

```
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

        if      i == 0

                if      ROUND == 2

                BCF             FPFLAGS,RND

                endif

                if      ROUND == 3

                BSF             FPFLAGS,RND

                endif

                if      ROUND == 4

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                endif

                if      ROUND == 5

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND

                endif

        endif

                CALL            FPA32

        variable i = i - 1

        endw

        endm

POL24   macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       and where AARG is assumed have been be saved in DARG when N>1.
;       The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
```

```
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set

        local           i,j
        variable i = N, j = 0

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPM32

        variable i = i - 1

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPA32

        variable i = i - 1

        while           i >= 0

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVLW           COF#v(i)
                MOVWF           BEXP
```

```
        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

        if      i == 0

                if      ROUND == 2

                BCF             FPFLAGS,RND

                endif

                if      ROUND == 3

                BSF             FPFLAGS,RND

                endif

                if      ROUND == 4

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                endif

                if      ROUND == 5

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND

                endif

        endif

                CALL            FPA32

        variable i = i - 1

        endw

        endm


POLL132 macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       with leading coefficient of one, and where AARG is assumed have been be saved
;       in DARG when N>1.  The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set
```



```
        local           i,j
        variable i = N, j = 0

        variable i = i - 1

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPA32

        variable i = i - 1

        while           i >= 0

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

        if      i == 0

                if      ROUND == 2

                BCF             FPFLAGS,RND

                endif

                if      ROUND == 3
```

```
                BSF             FPFLAGS,RND

                endif

                if      ROUND == 4

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                endif

                if      ROUND == 5

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND

                endif

        endif

                CALL            FPA32

        variable i = i - 1

        endw

        endm


POL32   macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       and where AARG is assumed have been be saved in DARG when N>1.
;       The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set
;       ROUND = 6rounding is performed by RND4032 and then disabled before last add

        local           i,j
        variable i = N, j = 0

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        while           j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw
```

```
        CALL            FPM32

if      ROUND == 6

        CALL            RND4032

endif

variable i = i - 1

        MOVLW           COF#v(i)
        MOVWF           BEXP

variable j = 0

while           j <= 2

        MOVLW           COF#v(i)#v(j)
        MOVWF           BARGB#v(j)

variable j = j + 1

endw

        CALL            FPA32

if      ROUND == 6

        CALL            RND4032

endif

variable i = i - 1

while           i >= 0

        MOVF            DEXP,W
        MOVWF           BEXP
        MOVF            DARGB0,W
        MOVWF           BARGB0
        MOVF            DARGB1,W
        MOVWF           BARGB1
        MOVF            DARGB2,W
        MOVWF           BARGB2

        CALL            FPM32

if      ROUND == 6

        CALL            RND4032

endif

        MOVLW           COF#v(i)
        MOVWF           BEXP

variable j = 0

while           j <= 2

        MOVLW           COF#v(i)#v(j)
        MOVWF           BARGB#v(j)

variable j = j + 1

endw
```

```
if      i == 0

        if      ROUND == 2

        BCF             FPFLAGS,RND

        endif

        if      ROUND == 3

        BSF             FPFLAGS,RND

        endif

        if      ROUND == 4

        BTFSS           DARGB3,RND
        BCF             FPFLAGS,RND

        endif

        if      ROUND == 5

        BTFSC           DARGB3,RND
        BSF             FPFLAGS,RND

        endif

endif

        CALL            FPA32

if      ROUND == 6  &&  i != 0

        CALL            RND4032

endif

variable i = i - 1

endw

endm
```



```
;       RCS Header $Id: exp24.a16 1.6 1997/02/25 14:23:30 F.J.Testa Exp $

;       $Revision: 1.6 $

;       Evaluate exp10(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    EXP1024

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  EXP10( AARG )

;       Testing on [MINLOG10,MAXLOG10] from 10000 trials:

;               min     max     mean
;       Timing: 2043    2561    2328.7  clks

;               min     max     mean    rms
;       Error:  -0x75   0x77    -0.95   40.34   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the base 10 exponential function is based upon the
;       expansion

;               exp10(x) = 10**x = 2**(x/log10(2)) = 2**z * 2**n

;                       x/log10(2) = z + n,

;       where 0 <= z < 1 and n is an integer, evaluated during range reduction.
;       Segmented third degree minimax polynomial approximations are used to
;       estimate 2**z on the intervals [0,.25], [.25,.5], [.5,.75] and [.75,1].

EXP1024
                MOVLW           0x64            ; test for |x| < 2**(-24)/(2*LOG(10))
                SUBWF           EXP,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            EXP1024ONE      ; return 10**x = 1

                BTFSC           AARGB0,MSB
                GOTO            TNEXP1024
TPEXP1024
                MOVF            AEXP,W
                SUBLW           MAXLOG1024EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVF            AARGB0,W
                SUBLW           MAXLOG1024B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVF            AARGB1,W
                SUBLW           MAXLOG1024B1
                BTFSS           _C
                GOTO            DOMERR24
                GOTO            EXP1024ARGOK

TNEXP1024
```

```
                MOVF            AEXP,W
                SUBLW           MINLOG1024EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVF            AARGB0,W
                SUBLW           MINLOG1024B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVF            AARGB1,W
                SUBLW           MINLOG1024B1
                BTFSS           _C
                GOTO            DOMERR24

EXP1024ARGOK
                MOVF            FPFLAGS,W
                MOVWF           DARGB3          ; save rounding flag
                BCF             FPFLAGS,RND     ; disable rounding

                CALL            RREXP1024

                MOVLW           0x7E
                SUBWF           AEXP,W
                BTFSS           _Z
                GOTO            EXP1024L

EXP1024H        BTFSS           AARGB0,MSB-1
                GOTO            EXP1024HL

                POL24           EXP24HH,3,0     ; minimax approximation on [.75,1]

                MOVF            EARGB3,W
                ADDWF           AEXP,F
                RETLW           0x00

EXP1024HL       POL24           EXP24HL,3,0     ; minimax approximation on [.5,.75]

                MOVF            EARGB3,W
                ADDWF           AEXP,F
                RETLW           0x00

EXP1024L        MOVLW           0x7D
                SUBWF           AEXP,W
                BTFSS           _Z
                GOTO            EXP1024LL

                POL24           EXP24LH,3,0     ; minimax approximation on [.25,.5]

                MOVF            EARGB3,W
                ADDWF           AEXP,F
                RETLW           0x00

EXP1024LL       POL24           EXP24LL,3,0     ; minimax approximation on [0,.25]

EXP1024OK
                MOVF            EARGB3,W
                ADDWF           AEXP,F
                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
```

```
                GOTO            RND3224

EXP1024ONE      MOVLW           EXPBIAS         ; return e**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0
                CLRF            AARGB1
                CLRF            AARGB2
                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       Range reduction routine for the exponential function

;               x/log10(2) = z + n

RREXP1024
                MOVF            AARGB0,W
                MOVWF           DARGB0
                BSF             AARGB0,MSB

                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1

                MOVLW           0xD4            ; 1/log10(2) = 3.32192809489
                MOVWF           AARGB0
                MOVLW           0x9A
                MOVWF           AARGB1
                MOVLW           0x78
                MOVWF           AARGB2

                CALL            FXM2416U        ; x * (1/log10(2))

                INCF            AEXP,F
                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP1024YOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RREXP1024YOK
                BTFSS           DARGB0,MSB
                BCF             AARGB0,MSB

                MOVF            AEXP,W
                MOVWF           BEXP            ; save y in BARG
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                CALL            FLOOR24

                MOVF            AEXP,W
                MOVWF           DEXP            ; save k in DARG
                MOVF            AARGB0,W
```

```
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1

                CALL            INT2416         ; k = [ x * (1/ln2) ]

                MOVF            AARGB1,W
                MOVWF           EARGB3          ; save k in EARG

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0
                MOVF            DARGB1,W
                MOVWF           AARGB1
                CLRF            AARGB2

                MOVLW           0x80
                XORWF           AARGB0,F

                CALL            FPA32

                MOVF            AEXP,W
                MOVWF           DEXP            ; save y in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00
;----------------------------------------------------------------------------------------------

;       third degree minimax polynomial coefficients for 2**(x) on [.75,1]

EXP24HH0        EQU             0x7E            ; EXP24HH0 = .99103284632
EXP24HH00       EQU             0x7D
EXP24HH01       EQU             0xB4
EXP24HH02       EQU             0x54

EXP24HH1        EQU             0x7E            ; EXP24HH1 = .73346850266
EXP24HH10       EQU             0x3B
EXP24HH11       EQU             0xC4
EXP24HH12       EQU             0x97

EXP24HH2        EQU             0x7C            ; EXP24HH2 = .17374128273
EXP24HH20       EQU             0x31
EXP24HH21       EQU             0xE9
EXP24HH22       EQU             0x3C

EXP24HH3        EQU             0x7B            ; EXP24HH3 = .10175678143
EXP24HH30       EQU             0x50
EXP24HH31       EQU             0x65
EXP24HH32       EQU             0xDC

;       third degree minimax polynomial coefficients for 2**(x) on [.5,.75]

EXP24HL0        EQU             0x7E            ; EXP24HL0 = .99801686089
EXP24HL00       EQU             0x7F
EXP24HL01       EQU             0x7E
EXP24HL02       EQU             0x08

EXP24HL1        EQU             0x7E            ; EXP24HL1 = .70586404164
EXP24HL10       EQU             0x34
EXP24HL11       EQU             0xB3
```

```
EXP24HL12       EQU             0x81

EXP24HL2        EQU             0x7C            ; EXP24HL2 = .21027360637
EXP24HL20       EQU             0x57
EXP24HL21       EQU             0x51
EXP24HL22       EQU             0xF7

EXP24HL3        EQU             0x7B            ; EXP24HL3 = .85566912730E-1
EXP24HL30       EQU             0x2F
EXP24HL31       EQU             0x3D
EXP24HL32       EQU             0xB5

;       third degree minimax polynomial coefficients for 2**(x) on [.25,.5]

EXP24LH0        EQU             0x7E            ; EXP24LH0 = .99979384559
EXP24LH00       EQU             0x7F
EXP24LH01       EQU             0xF2
EXP24LH02       EQU             0x7D

EXP24LH1        EQU             0x7E            ; EXP24LH1 = .69545887384
EXP24LH10       EQU             0x32
EXP24LH11       EQU             0x09
EXP24LH12       EQU             0x98

EXP24LH2        EQU             0x7C            ; EXP24LH2 = .23078300446
EXP24LH20       EQU             0x6C
EXP24LH21       EQU             0x52
EXP24LH22       EQU             0x61

EXP24LH3        EQU             0x7B            ; EXP24LH3 = .71952910179E-1
EXP24LH30       EQU             0x13
EXP24LH31       EQU             0x5C
EXP24LH32       EQU             0x0C

;       third degree minimax polynomial coefficients for 2**(x) on [0,.25]

EXP24LL0        EQU             0x7E            ; EXP24LL0 = .99999970657
EXP24LL00       EQU             0x7F
EXP24LL01       EQU             0xFF
EXP24LL02       EQU             0xFB

EXP24LL1        EQU             0x7E            ; EXP24LL1 = .69318585159
EXP24LL10       EQU             0x31
EXP24LL11       EQU             0x74
EXP24LL12       EQU             0xA1

EXP24LL2        EQU             0x7C            ; EXP24LL2 = .23944330933
EXP24LL20       EQU             0x75
EXP24LL21       EQU             0x30
EXP24LL22       EQU             0xA0

EXP24LL3        EQU             0x7A            ; EXP24LL3 = .60504944237E-1
EXP24LL30       EQU             0x77
EXP24LL31       EQU             0xD4
EXP24LL32       EQU             0x08

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate exp(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    EXP24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1
```

```
;       Result:  AARG  <--  EXP( AARG )

;       Testing on [MINLOG,MAXLOG] from 100000 trials:

;               min     max     mean
;       Timing: 1990    2600    2347.6  clks

;               min     max     mean    rms
;       Error:  -0x43   0x40    -.77    16.75   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp(x) = e**x = 2**(x/log(2)) = 2**z * 2**n,

;                       x/log(2) = z + n,

;       where 0 <= z < 1 and n is an integer, evaluated during range reduction.
;       Segmented third degree minimax polynomial approximations are used to
;       estimate 2**z on the intervals [0,.25], [.25,.5], [.5,.75] and [.75,1].

EXP24
                MOVLW           0x66            ; test for |x| < 2**(-24)/2
                SUBWF           EXP,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            EXP24ONE        ; return e**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP24
TPEXP24
                MOVF            AEXP,W          ; positive domain check
                SUBLW           MAXLOG24EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVF            AARGB0,W
                SUBLW           MAXLOG24B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVF            AARGB1,W
                SUBLW           MAXLOG24B1
                BTFSS           _C
                GOTO            DOMERR24
                GOTO            EXP24ARGOK

TNEXP24
                MOVF            AEXP,W          ; negative domain check
                SUBLW           MINLOG24EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVF            AARGB0,W
                SUBLW           MINLOG24B0
                BTFSS           _C
                GOTO            DOMERR24
```

```
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVF            AARGB1,W
                SUBLW           MINLOG24B1
                BTFSS           _C
                GOTO            DOMERR24

EXP24ARGOK
                MOVF            FPFLAGS,W
                MOVWF           DARGB3          ; save rounding flag
                BCF             FPFLAGS,RND     ; disable rounding

                CALL            RREXP24         ; range reduction

                MOVLW           0x7E
                SUBWF           AEXP,W
                BTFSS           _Z
                GOTO            EXP24L

EXP24H          BTFSS           AARGB0,MSB-1
                GOTO            EXP24HL

                POL24           EXP24HH,3,0     ; minimax approximation on [.75,1]

                GOTO            EXP24OK

EXP24HL         POL24           EXP24HL,3,0     ; minimax approximation on [.5,.75]

                GOTO            EXP24OK

EXP24L          MOVLW           0x7D
                SUBWF           AEXP,W
                BTFSS           _Z
                GOTO            EXP24LL

                POL24           EXP24LH,3,0     ; minimax approximation on [.25,.5]

                GOTO            EXP24OK

EXP24LL         POL24           EXP24LL,3,0     ; minimax approximation on [0,.25]

EXP24OK
                MOVF            EARGB3,W
                ADDWF           AEXP,F
                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                GOTO            RND3224

EXP24ONE        MOVLW           EXPBIAS         ; return e**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0
                CLRF            AARGB1
                CLRF            AARGB2
                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       Range reduction routine for the exponential function

;               x/log(2) = z + n
```

```
RREXP24
                MOVF        AARGB0,W        ; save sign
                MOVWF       DARGB0
                BSF         AARGB0,MSB      ; make MSB explicit

                MOVF        AARGB0,W
                MOVWF       BARGB0
                MOVF        AARGB1,W
                MOVWF       BARGB1

                MOVLW       0xB8            ; 1/ln(2) = 1.44269504089
                MOVWF       AARGB0
                MOVLW       0xAA
                MOVWF       AARGB1
                MOVLW       0x3B
                MOVWF       AARGB2

                CALL        FXM2416U        ; x * (1/ln2)

                INCF        AEXP,F

                BTFSC       AARGB0,MSB
                GOTO        RREXP24YOK
                RLF         AARGB3,F
                RLF         AARGB2,F
                RLF         AARGB1,F
                RLF         AARGB0,F
                DECF        AEXP,F

RREXP24YOK      BTFSS       DARGB0,MSB
                BCF         AARGB0,MSB

                CALL        RND4032

                MOVF        AEXP,W
                MOVWF       BEXP            ; save z in BARG
                MOVF        AARGB0,W
                MOVWF       BARGB0
                MOVF        AARGB1,W
                MOVWF       BARGB1
                MOVF        AARGB2,W
                MOVWF       BARGB2

                CALL        FLOOR24

                MOVF        AEXP,W
                MOVWF       DEXP            ; save float(n) in DARG
                MOVF        AARGB0,W
                MOVWF       DARGB0
                MOVF        AARGB1,W
                MOVWF       DARGB1

                CALL        INT2416         ; n = [ x * (1/ln2) ]

                MOVF        AARGB1,W
                MOVWF       EARGB3          ; save n in EARG

                MOVF        DEXP,W
                MOVWF       AEXP
                MOVF        DARGB0,W
                MOVWF       AARGB0
                MOVF        DARGB1,W
                MOVWF       AARGB1
                CLRF        AARGB2
```

```
                MOVLW           0x80            ; toggle sign
                XORWF           AARGB0,F

                CALL            FPA32

                CALL            RND4032

                MOVF            AEXP,W
                MOVWF           DEXP            ; save z in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       third degree minimax polynomial coefficients for 2**(x) on [.75,1]

EXP24HH0        EQU             0x7E            ; EXP24HH0 = .99103284632
EXP24HH00       EQU             0x7D
EXP24HH01       EQU             0xB4
EXP24HH02       EQU             0x54

EXP24HH1        EQU             0x7E            ; EXP24HH1 = .73346850266
EXP24HH10       EQU             0x3B
EXP24HH11       EQU             0xC4
EXP24HH12       EQU             0x97

EXP24HH2        EQU             0x7C            ; EXP24HH2 = .17374128273
EXP24HH20       EQU             0x31
EXP24HH21       EQU             0xE9
EXP24HH22       EQU             0x3C

EXP24HH3        EQU             0x7B            ; EXP24HH3 = .10175678143
EXP24HH30       EQU             0x50
EXP24HH31       EQU             0x65
EXP24HH32       EQU             0xDC

;       third degree minimax polynomial coefficients for 2**(x) on [.5,.75]

EXP24HL0        EQU             0x7E            ; EXP24HL0 = .99801686089
EXP24HL00       EQU             0x7F
EXP24HL01       EQU             0x7E
EXP24HL02       EQU             0x08

EXP24HL1        EQU             0x7E            ; EXP24HL1 = .70586404164
EXP24HL10       EQU             0x34
EXP24HL11       EQU             0xB3
EXP24HL12       EQU             0x81

EXP24HL2        EQU             0x7C            ; EXP24HL2 = .21027360637
EXP24HL20       EQU             0x57
EXP24HL21       EQU             0x51
EXP24HL22       EQU             0xF7

EXP24HL3        EQU             0x7B            ; EXP24HL3 = .85566912730E-1
EXP24HL30       EQU             0x2F
EXP24HL31       EQU             0x3D
EXP24HL32       EQU             0xB5
```

```
;       third degree minimax polynomial coefficients for 2**(x) on [.25,.5]

EXP24LH0        EQU             0x7E            ; EXP24LH0 = .99979384559
EXP24LH00       EQU             0x7F
EXP24LH01       EQU             0xF2
EXP24LH02       EQU             0x7D

EXP24LH1        EQU             0x7E            ; EXP24LH1 = .69545887384
EXP24LH10       EQU             0x32
EXP24LH11       EQU             0x09
EXP24LH12       EQU             0x98

EXP24LH2        EQU             0x7C            ; EXP24LH2 = .23078300446
EXP24LH20       EQU             0x6C
EXP24LH21       EQU             0x52
EXP24LH22       EQU             0x61

EXP24LH3        EQU             0x7B            ; EXP24LH3 = .71952910179E-1
EXP24LH30       EQU             0x13
EXP24LH31       EQU             0x5C
EXP24LH32       EQU             0x0C

;       third degree minimax polynomial coefficients for 2**(x) on [0,.25]

EXP24LL0        EQU             0x7E            ; EXP24LL0 = .99999970657
EXP24LL00       EQU             0x7F
EXP24LL01       EQU             0xFF
EXP24LL02       EQU             0xFB

EXP24LL1        EQU             0x7E            ; EXP24LL1 = .69318585159
EXP24LL10       EQU             0x31
EXP24LL11       EQU             0x74
EXP24LL12       EQU             0xA1

EXP24LL2        EQU             0x7C            ; EXP24LL2 = .23944330933
EXP24LL20       EQU             0x75
EXP24LL21       EQU             0x30
EXP24LL22       EQU             0xA0

EXP24LL3        EQU             0x7A            ; EXP24LL3 = .60504944237E-1
EXP24LL30       EQU             0x77
EXP24LL31       EQU             0xD4
EXP24LL32       EQU             0x08

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate floor(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    FLOOR24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  FLOOR( AARG )

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 37      55      42.7    clks

;               min     max     mean    rms
;       Error:  0x00    0x00    0.0     0.0     nsb

;----------------------------------------------------------------------------------------------
```



```
;       floor(x) evaluates the largest integer, as a float, not greater than x.

FLOOR24
                CLRF            AARGB2          ; test for zero argument
                MOVF            AEXP,W
                BTFSC           _Z
                RETLW           0x00

                MOVF            AARGB0,W
                MOVWF           AARGB3          ; save mantissa
                MOVF            AARGB1,W
                MOVWF           AARGB4

                MOVLW           EXPBIAS         ; computed unbiased exponent
                SUBWF           AEXP,W
                MOVWF           TEMPB1
                BTFSC           TEMPB1,MSB
                GOTO            FLOOR24ZERO

                SUBLW           0x10-1
                MOVWF           TEMPB0          ; save number of zero bits in TEMPB0
                MOVWF           TEMPB1

                BTFSC           TEMPB1,LSB+3    ; divide by eight
                GOTO            FLOOR24MASKH

FLOOR24MASKL
                MOVLW           0x07            ; get remainder for mask pointer
                ANDWF           TEMPB0,F
                MOVLW           LOW FLOOR24MASKTABLE
                ADDWF           TEMPB0,F
                MOVLW           HIGH FLOOR24MASKTABLE
                BTFSC           _C
                ADDLW           0x01
                MOVWF           PCLATH
                INCF            TEMPB0,W

                CALL            FLOOR24MASKTABLE ; access table for mask

                ANDWF           AARGB1,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVF            AARGB4,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                GOTO            FLOOR24RNDL
                RETLW           0x00

FLOOR24RNDL
                COMF            AARGB7,W
                MOVWF           TEMPB1
                INCF            TEMPB1,W
                ADDWF           AARGB1,F
                BTFSC           _Z
                INCF            AARGB0, F
                BTFSS           _Z              ; has rounding caused carryout?
                RETLW           0x00
                RRF             AARGB0,F
                RRF             AARGB1,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV24
```

```
FLOOR24MASKH
                MOVLW           0x07            ; get remainder for mask pointer
                ANDWF           TEMPB0,F
                MOVLW           LOW FLOOR24MASKTABLE
                ADDWF           TEMPB0,F
                MOVLW           HIGH FLOOR24MASKTABLE
                BTFSC           _C
                ADDLW           0x01
                MOVWF           PCLATH
                INCF            TEMPB0,W

                CALL            FLOOR24MASKTABLE ; access table for mask

                ANDWF           AARGB0,F
                CLRF            AARGB1
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVF            AARGB4,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                GOTO            FLOOR24RNDH
                MOVF            AARGB3,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                GOTO            FLOOR24RNDH
                RETLW           0x00

FLOOR24RNDH
                COMF            AARGB7,W
                MOVWF           TEMPB1
                INCF            TEMPB1,W
                ADDWF           AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRF             AARGB0,F
                RRF             AARGB1,F
                INCFSZ          AEXP,F
                RETLW           0x00
                GOTO            SETFOV24        ; check for overflow

FLOOR24ZERO
                BTFSC           AARGB0,MSB
                GOTO            FLOOR24MINUSONE
                CLRF            AEXP
                CLRF            AARGB0
                CLRF            AARGB1
                RETLW           0x00

FLOOR24MINUSONE
                MOVLW           0x7F
                MOVWF           AEXP
                MOVLW           0x80
                MOVWF           AARGB0
                CLRF            AARGB1
                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       table for least significant byte requiring masking, using pointer from
;       the remainder of the number of zero bits divided by eight.

FLOOR24MASKTABLE
                MOVWF           PCL
                RETLW           0xFF
```

```
                RETLW           0xFE
                RETLW           0xFC
                RETLW           0xF8
                RETLW           0xF0
                RETLW           0xE0
                RETLW           0xC0
                RETLW           0x80
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate log10(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    LOG1024

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  LOG( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 1674    3567    3383.3  clks

;               min     max     mean    rms
;       Error:  -0x01   0x00    -0.32   0.57    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log10(x) = log10(2) * log2(x) = log10(2) * ( n + log2(f) )

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |   2*f-1, f < 1/sqrt(2), n=n-1
;               z = |
;                   |   f-1, otherwise

;       produces a naturally segmented representation of log2(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.

LOG1024
                CLRF            AARGB2          ; clear next significant byte
                MOVF            AEXP,W
                BTFSS           AARGB0,MSB      ; test for negative argument
                BTFSC           _Z              ; test for zero argument
                GOTO            DOMERR24

                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVF            AEXP,W
                MOVWF           EARGB3
                MOVLW           EXPBIAS-1
                SUBWF           EARGB3,F
                MOVWF           AEXP

                MOVLW           0xF3            ; .70710678118655 = 7E3504F3
```

```
              SUBWF          AARGB2,W
              MOVLW          0x04
              MOVWF          TEMPB0
              BTFSS          _C
              INCFSZ         TEMPB0,W
              SUBWF          AARGB1,W

              MOVLW          0x35
              MOVWF          TEMPB0
              BTFSS          _C
              INCFSZ         TEMPB0,W
              SUBWF          AARGB0,W

              BTFSS          _C
              GOTO           LOG1024L

;       minimax rational approximation on [0,.sqrt(2)-1]

LOG1024H
              MOVLW          0x7F
              MOVWF          BEXP
              CLRF           BARGB0
              CLRF           BARGB1
              CLRF           BARGB2

              CALL           FPS32

              MOVF           AEXP,W
              MOVWF          DEXP
              MOVF           AARGB0,W
              MOVWF          DARGB0
              MOVF           AARGB1,W
              MOVWF          DARGB1
              MOVF           AARGB2,W
              MOVWF          DARGB2

              POLL124        LOG24HQ,2,0

              MOVF           AEXP,W
              MOVWF          CEXP
              MOVF           AARGB0,W
              MOVWF          CARGB0
              MOVF           AARGB1,W
              MOVWF          CARGB1
              MOVF           AARGB2,W
              MOVWF          CARGB2

              MOVF           DEXP,W
              MOVWF          AEXP
              MOVF           DARGB0,W
              MOVWF          AARGB0
              MOVF           DARGB1,W
              MOVWF          AARGB1
              MOVF           DARGB2,W
              MOVWF          AARGB2

              POL24          LOG24HP,1,0

              MOVF           CEXP,W
              MOVWF          BEXP
              MOVF           CARGB0,W
              MOVWF          BARGB0
              MOVF           CARGB1,W
              MOVWF          BARGB1
              MOVF           CARGB2,W
              MOVWF          BARGB2
```

```
                CALL            FPD32

                GOTO            LOG1024OK

;       minimax rational approximation on [1/sqrt(2)-1,0]

LOG1024L
                INCF            AEXP,F
                MOVLW           0x7F
                MOVWF           BEXP
                CLRF            BARGB0
                CLRF            BARGB1
                CLRF            BARGB2

                CALL            FPS32

                DECF            EARGB3,F

                MOVF            AEXP,W
                MOVWF           DEXP
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                POLL124         LOG24LQ,2,0

                MOVF            AEXP,W
                MOVWF           CEXP
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                POL24           LOG24LP,1,0

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPD32

LOG1024OK
                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
```

```
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            AEXP,W
                MOVWF           DEXP
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                CLRF            AARGB0
                MOVF            EARGB3,W
                MOVWF           AARGB1
                BTFSC           AARGB1,MSB
                COMF            AARGB0,F
                CALL            FLO1624
                CLRF            AARGB2

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

;       fixed point multiplication by log10(2)

                MOVF            AARGB0,W
                MOVWF           EARGB3
                BSF             AARGB0,MSB

                MOVLW           0x9A
                MOVWF           BARGB0
                MOVLW           0x20
                MOVWF           BARGB1
                MOVLW           0x9B
                MOVWF           BARGB2

                CALL            FXM2424U
                DECF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            LOG1024DONE
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

LOG1024DONE     BTFSS           EARGB3,MSB
                BCF             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND      ; restore rounding flag
```



```
                CALL            RND3224
                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+x)/x on [1/sqrt(2)-1,0]

LOG24HP0        EQU             0x81            ; LOG24HP0 = .73551298732E+1
LOG24HP00       EQU             0x6B
LOG24HP01       EQU             0x5D
LOG24HP02       EQU             0x39

LOG24HP1        EQU             0x81            ; LOG24HP1 = .40900513905E+1
LOG24HP10       EQU             0x02
LOG24HP11       EQU             0xE1
LOG24HP12       EQU             0xB3

LOG24HQ0        EQU             0x81            ; LOG24HQ0 = .50982159260E+1
LOG24HQ00       EQU             0x23
LOG24HQ01       EQU             0x24
LOG24HQ02       EQU             0x96

LOG24HQ1        EQU             0x81            ; LOG24HQ1 = .53849258895E+1
LOG24HQ10       EQU             0x2C
LOG24HQ11       EQU             0x51
LOG24HQ12       EQU             0x50

LOG24HQ2        EQU             0x7F            ; LOG24HQ2 = 1.0
LOG24HQ20       EQU             0x00
LOG24HQ21       EQU             0x00
LOG24HQ22       EQU             0x00

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+x)/x on [0,sqrt(2)-1]

LOG24LP0        EQU             0x82            ; LOG24LP0 = .103115556038E+2
LOG24LP00       EQU             0x24
LOG24LP01       EQU             0xFC
LOG24LP02       EQU             0x22

LOG24LP1        EQU             0x81            ; LOG24LP1 = .457749066375E+1
LOG24LP10       EQU             0x12
LOG24LP11       EQU             0x7A
LOG24LP12       EQU             0xCE

LOG24LQ0        EQU             0x81            ; LOG24LQ0 = .714746549793E+1
LOG24LQ00       EQU             0x64
LOG24LQ01       EQU             0xB8
LOG24LQ02       EQU             0x0A

LOG24LQ1        EQU             0x81            ; LOG24LQ1 = .674551124538E+1
LOG24LQ10       EQU             0x57
LOG24LQ11       EQU             0xDB
LOG24LQ12       EQU             0x3A

LOG24LQ2        EQU             0x7F            ; LOG24LQ2 = 1.0
LOG24LQ20       EQU             0x00
LOG24LQ21       EQU             0x00
LOG24LQ22       EQU             0x00

;**********************************************************************************************
```

```
;**********************************************************************************************

;       Evaluate log(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    LOG24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  LOG( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 1662    3555    3371.2  clks

;               min     max     mean    rms
;       Error:  -0x02   0x00    -0.80   0.92    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log(x) = log(2) * log2(x) = log(2) * ( n + log2(f) )

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |   2*f-1, f < 1/sqrt(2), n=n-1
;               z = |
;                   |   f-1, otherwise

;       produces a naturally segmented representation of log2(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.

LOG24
                CLRF            AARGB2
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR24
                MOVF            AEXP,W          ; test for zero argument
                BTFSC           _Z
                GOTO            DOMERR24

                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3
                BCF             FPFLAGS,RND     ; disable rounding

                MOVF            AEXP,W
                MOVWF           EARGB3
                MOVLW           EXPBIAS-1
                SUBWF           EARGB3,F
                MOVWF           AEXP

                MOVLW           0xF3            ; .70710678118655 = 7E3504F3
                SUBWF           AARGB2,W
                MOVLW           0x04
                MOVWF           TEMP
                BTFSS          _C
                INCFSZ         TEMP,W
                SUBWF           AARGB1,W
                MOVLW           0x35
                MOVWF           TEMP
                BTFSS           _C
                INCFSZ          TEMP,W
```



```
                SUBWF           AARGB0,W

                BTFSS           _C
                GOTO            LOG24L

;       minimax rational approximation on [0,.sqrt(2)-1]

LOG24H
                MOVLW           0x7F
                MOVWF           BEXP
                CLRF            BARGB0
                CLRF            BARGB1
                CLRF            BARGB2

                CALL            FPS32

                MOVF            AEXP,W
                MOVWF           DEXP
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                POLL124         LOG24HQ,2,0

                MOVF            AEXP,W
                MOVWF           CEXP
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                POL24           LOG24HP,1,0

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPD32

                GOTO            LOG24OK

;       minimax rational approximation on [1/sqrt(2)-1,0]

LOG24L
                INCF            AEXP,F
                MOVLW           0x7F
                MOVWF           BEXP
```

```
            CLRF            BARGB0
            CLRF            BARGB1
            CLRF            BARGB2

            CALL            FPS32

            DECF            EARGB3,F

            MOVF            AEXP,W
            MOVWF           DEXP
            MOVF            AARGB0,W
            MOVWF           DARGB0
            MOVF            AARGB1,W
            MOVWF           DARGB1
            MOVF            AARGB2,W
            MOVWF           DARGB2

            POLL124         LOG24LQ,2,0

            MOVF            AEXP,W
            MOVWF           CEXP
            MOVF            AARGB0,W
            MOVWF           CARGB0
            MOVF            AARGB1,W
            MOVWF           CARGB1
            MOVF            AARGB2,W
            MOVWF           CARGB2

            MOVF            DEXP,W
            MOVWF           AEXP
            MOVF            DARGB0,W
            MOVWF           AARGB0
            MOVF            DARGB1,W
            MOVWF           AARGB1
            MOVF            DARGB2,W
            MOVWF           AARGB2

            POL24           LOG24LP,1,0

            MOVF            CEXP,W
            MOVWF           BEXP
            MOVF            CARGB0,W
            MOVWF           BARGB0
            MOVF            CARGB1,W
            MOVWF           BARGB1
            MOVF            CARGB2,W
            MOVWF           BARGB2

            CALL            FPD32

LOG24OK
            MOVF            DEXP,W
            MOVWF           BEXP
            MOVF            DARGB0,W
            MOVWF           BARGB0
            MOVF            DARGB1,W
            MOVWF           BARGB1
            MOVF            DARGB2,W
            MOVWF           BARGB2

            CALL            FPM32

            MOVF            AEXP,W
            MOVWF           DEXP
            MOVF            AARGB0,W
            MOVWF           DARGB0
```

```
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                CLRF            AARGB0
                MOVF            EARGB3,W
                MOVWF           AARGB1
                BTFSC           AARGB1,MSB
                COMF            AARGB0,F
                CALL            FLO1624
                CLRF            AARGB2

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

;       fixed point multiplication by log(2)

                MOVF            AEXP,W
                BTFSC           _Z
                RETLW           0x00

                MOVF            AARGB0,W
                MOVWF           EARGB3
                BSF             AARGB0,MSB

                MOVLW           0xB1
                MOVWF           BARGB0
                MOVLW           0x72
                MOVWF           BARGB1
                MOVLW           0x18
                MOVWF           BARGB2

                CALL            FXM2424U

                BTFSC           AARGB0,MSB
                GOTO            LOG24DONE
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

LOG24DONE       BTFSS           EARGB3,MSB
                BCF             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND
                GOTO            RND3224

DOMERR24        BSF             FPFLAGS,DOM      ; domain error
                RETLW           0xFF

;----------------------------------------------------------------------------------------------
```

```
;       minimax rational coefficients for log2(1+x)/x on [1/sqrt(2)-1,0]

LOG24HP0        EQU             0x81            ; LOG24HP0 = .73551298732E+1
LOG24HP00       EQU             0x6B
LOG24HP01       EQU             0x5D
LOG24HP02       EQU             0x39

LOG24HP1        EQU             0x81            ; LOG24HP1 = .40900513905E+1
LOG24HP10       EQU             0x02
LOG24HP11       EQU             0xE1
LOG24HP12       EQU             0xB3

LOG24HQ0        EQU             0x81            ; LOG24HQ0 = .50982159260E+1
LOG24HQ00       EQU             0x23
LOG24HQ01       EQU             0x24
LOG24HQ02       EQU             0x96

LOG24HQ1        EQU             0x81            ; LOG24HQ1 = .53849258895E+1
LOG24HQ10       EQU             0x2C
LOG24HQ11       EQU             0x51
LOG24HQ12       EQU             0x50

LOG24HQ2        EQU             0x7F            ; LOG24HQ2 = 1.0
LOG24HQ20       EQU             0x00
LOG24HQ21       EQU             0x00
LOG24HQ22       EQU             0x00

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+x)/x on [0,sqrt(2)-1]

LOG24LP0        EQU             0x82            ; LOG24LP0 = .103115556038E+2
LOG24LP00       EQU             0x24
LOG24LP01       EQU             0xFC
LOG24LP02       EQU             0x22

LOG24LP1        EQU             0x81            ; LOG24LP1 = .457749066375E+1
LOG24LP10       EQU             0x12
LOG24LP11       EQU             0x7A
LOG24LP12       EQU             0xCE

LOG24LQ0        EQU             0x81            ; LOG24LQ0 = .714746549793E+1
LOG24LQ00       EQU             0x64
LOG24LQ01       EQU             0xB8
LOG24LQ02       EQU             0x0A

LOG24LQ1        EQU             0x81            ; LOG24LQ1 = .674551124538E+1
LOG24LQ10       EQU             0x57
LOG24LQ11       EQU             0xDB
LOG24LQ12       EQU             0x3A

LOG24LQ2        EQU             0x7F            ; LOG24LQ2 = 1.0
LOG24LQ20       EQU             0x00
LOG24LQ21       EQU             0x00
LOG24LQ22       EQU             0x00

;**********************************************************************************************
;**********************************************************************************************

;       Nearest neighbor rounding

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    RND3224

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1
```

```
;       Result: AARG  <-- RND( AARG )

;       Testing on [MINNUM,MAXNUM] from 10000 trials:

;               min     max     mean
;       Timing: 3       17              clks

;               min     max     mean
;       Error:  0       0       0       nsb

;----------------------------------------------------------------------------------------------

RND3224
                BTFSS           AARGB2,MSB      ; is NSB < 0x80?
                RETLW           0x00

                BSF             _C              ; set carry for rounding
                MOVLW           0x7F
                ANDWF           AARGB2,W
                BTFSC           _Z
                RRF             AARGB1,W        ; select even if NSB = 0x80

                MOVF            AARGB0,W
                MOVWF           SIGN            ; save sign
                BSF             AARGB0,MSB      ; make MSB explicit

                BCF             _Z
                BTFSC           _C              ; round
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

                BTFSS           _Z              ; has rounding caused carryout?
                GOTO            RND3224OK
                RRF             AARGB0,F        ; if so, right shift
                RRF             AARGB1,F
                INCF            EXP,F           ; test for floating point overflow
                BTFSC           _Z
                GOTO            SETFOV24
RND3224OK
                BTFSS           SIGN,MSB
                BCF             AARGB0,MSB      ; clear sign bit if positive
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate cos(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    COS24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  COS( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 2736    4505    4134.3  clks
```

```
;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.13   20.90   nsb

;-----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations.

COS24
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                CLRF            CARGB3          ; initialize sign in CARGB3

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS24
RRCOS24OK
                RRF             EARGB3,W
                XORWF           EARGB3,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,LSB
                GOTO            COSZSIN24

                CALL            ZCOS24

                GOTO            COSSIGN24

COSZSIN24       CALL            ZSIN24

COSSIGN24
                MOVLW           0x80
                BTFSC           EARGB3,LSB+1
                XORWF           CARGB3,F

                BTFSC           CARGB3,MSB
                XORWF           AARGB0,F

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;***********************************************************************************************

;       Evaluate sin(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SIN24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  SIN( AARG )
```

```
;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 2564    4494    4134.5  clks

;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.12   20.89   nsb

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations.

SIN24
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding
                CLRF            CARGB3          ; initialize sign in CARGB3

                BTFSC           AARGB0,MSB      ; toggle sign if x < 0
                BSF             CARGB3,MSB

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS24
RRSIN24OK
                RRF             EARGB3,W
                XORWF           EARGB3,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,LSB
                GOTO            SINZCOS24

                CALL            ZSIN24
                GOTO            SINSIGN24

SINZCOS24       CALL            ZCOS24

SINSIGN24
                MOVLW           0x80
                BTFSC           CARGB3,MSB
                XORWF           AARGB0,F

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;**********************************************************************************************

;       Evaluate sin(x) and cos(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SINCOS24
```

```
;       Output: 24 bit floating point numbers in AEXP, AARGB0, AARGB1 and
;               BEXP, BARGB0, BARGB1

;       Result: AARG  <--  COS( AARG )
;               BARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 4525    6478    6032.2  clks

;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.12   20.89   nsb     sine
;               -0x56   0x13    -7.13   20.90           cosine

;-----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations. In this case,
;       only one range reduction is necessary.

SINCOS24
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVF            AEXP,W          ; save x in EARG
                MOVWF           EEXP
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                CLRF            EARGB2

                BCF             AARGB0,MSB      ; use |x|

                CLRF            CARGB3          ; initialize sign in CARGB3

                CALL            RRSINCOS24      ; range reduction

                MOVF            CARGB3,W        ; save sign from range reduction
                MOVWF           ZARGB3

                MOVLW           0x80
                BTFSC           EARGB0,MSB      ; toggle sign if x < 0
                XORWF           CARGB3,F

                CALL            RRSIN24OK

                BTFSC           DARGB3,RND
                CALL            RND3224

                MOVF            AEXP,W          ; save sin(x) in EARG
                MOVWF           EEXP
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
```

```
            MOVF            AARGB2,W
            MOVWF           EARGB2

            MOVF            DEXP,W          ; restore z*z in AARG
            MOVWF           AEXP
            MOVF            DARGB0,W
            MOVWF           AARGB0
            MOVF            DARGB1,W
            MOVWF           AARGB1
            MOVF            DARGB2,W
            MOVWF           AARGB2

            MOVF            ZARGB3,W        ; restore sign from range reduction
            MOVWF           CARGB3

            CALL            RRCOS24OK

            MOVF            EEXP,W          ; restore sin(x) in BARG
            MOVWF           BEXP
            MOVF            EARGB0,W
            MOVWF           BARGB0
            MOVF            EARGB1,W
            MOVWF           BARGB1
            MOVF            EARGB2,W
            MOVWF           BARGB2

            BTFSS           DARGB3,RND
            RETLW           0x00

            BSF             FPFLAGS,RND     ; restore rounding flag
            CALL            RND3224
            RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for trigonometric functions

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition

;               z = x mod pi/4,

;       produced by first evaluating y and j through the relations

;               y = floor(x/(pi/4)), j = y - 8*[y/8].

;       where j equals the correct octant.  For j odd, adding one to j
;       and y eliminates the odd octants.  Additional logic on j and the
;       sign of the result leads to appropriate use of the sine or cosine
;       routine in each case.

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x mod pi/4 = x - y*(pi/4) = ((x - p1*y)-p2*y)-p3*y

;       where pi/4 = p1 + p2 + p3, with p1 close to pi/4 and p2 close to
;       pi/4 - p1. The numbers p1 and p2 are chosen to have an exact
;       machine representation with slightly more than the lower half of
;       the mantissa bits zero, typically leading to no error in computing
;       the terms in parenthesis.  This calculation breaks down leading to
;       a loss of precision for |x| > LOSSTHR = sqrt(2**24)*pi/4, or for |x|
;       close to an integer multiple of pi/4. This loss threshold has been
;       chosen based on the efficacy of this calculation, with a domain error
;       reported if this threshold is exceeded.
```

```
RRSINCOS24
                MOVF            AEXP,W          ; loss threshold check
                SUBLW           LOSSTHR24EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            RRSINCOS24ARGOK

                MOVF            AARGB0,W
                SUBLW           LOSSTHR24B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            RRSINCOS24ARGOK

                MOVF            AARGB1,W
                SUBLW           LOSSTHR24B1
                BTFSS           _C
                GOTO            DOMERR24

RRSINCOS24ARGOK
                MOVF            AEXP,W
                MOVWF           CEXP            ; save |x| in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                CLRF            CARGB2

;       fixed point multiplication by 4/pi

                BSF             AARGB0,MSB
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1

                MOVLW           0xA2            ; 4/pi = 1.27323954474
                MOVWF           AARGB0
                MOVLW           0xF9
                MOVWF           AARGB1
                MOVLW           0x83
                MOVWF           AARGB2

                CALL            FXM2416U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24YOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS24YOK
                BCF             AARGB0,MSB

                CALL            INT3224         ; y = [ |x| * (4/pi) ]

                BTFSS           AARGB2,LSB
                GOTO            SAVEY24

                INCF            AARGB2,F
```

```
                BTFSC           _Z
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

SAVEY24         MOVF            AARGB0,W
                MOVWF           DARGB0          ; save y in DARG
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                MOVLW           0x07            ; j = y mod 8
                ANDWF           AARGB2,F

                MOVLW           0x03
                SUBWF           AARGB2,W

                MOVLW           0x80
                BTFSS           _C
                GOTO            JOK24
                XORWF           CARGB3,F
                MOVLW           0x04
                SUBWF           AARGB2,F

JOK24
                MOVF            AARGB2,W
                MOVWF           EARGB3          ; save j in EARGB3

                MOVF            DARGB0,W
                MOVWF           AARGB0          ; restore y to AARG
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                CALL            FLO2432

                MOVF            AEXP,W
                MOVWF           DEXP            ; save y in DARG
                BTFSC           _Z
                GOTO            RRSINCOS24ZEQX
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

;       Cody-Waite extended precision calculation of |x| - y * pi/4 using
;       fixed point multiplication. Since y >= 1, underflow is not possible
;       in any of the products.

                BSF             AARGB0,MSB

                MOVLW           0xC9            ; - p1 = -.78515625
                MOVWF           BARGB0
                CLRF            BARGB1

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z1OK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
```

```
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z1OK
                MOVF            CEXP,W          ; restore x to BARG
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                CLRF            BARGB2

                CALL            FPA32           ; z1 = |x| - y * (p1)

                MOVF            AEXP,W
                MOVWF           CEXP            ; save z1 in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0          ; restore y to AARG
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xFD            ; - p2 = -.000241875648498535156 24
                MOVWF           BARGB0
                MOVLW           0xA0
                MOVWF           BARGB1

                CALL            FXM2416U

                MOVLW           0x0D - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z2OK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z2OK
                SUBWF           AEXP,F

                MOVF            CEXP,W          ; restore z1 to BARG
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32           ; z2 = z1 - y * (p2)

                MOVF            AEXP,W
```

```
                MOVWF           CEXP            ; save z2 in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0          ; restore y to AARG
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xA2            ; - p3 = -3.77489497744597636E-8
                MOVWF           BARGB0
                MOVLW           0x21
                MOVWF           BARGB1
                MOVLW           0x69
                MOVWF           BARGB2

                CALL            FXM2424U

                MOVLW           0x19 - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z3OK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z3OK
                SUBWF           AEXP,F

                MOVF            CEXP,W          ; restore z2 to BARG
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32           ; z = z2 - y * (p3)

                MOVF            AEXP,W
                MOVWF           CEXP            ; save z in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
```

```
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                CALL            FPM32           ; z * z

                MOVF            AEXP,W
                MOVWF           DEXP            ; save z * z in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00

RRSINCOS24ZEQX
                MOVF            CEXP,W
                MOVWF           AEXP
                MOVF            CARGB0,W
                MOVWF           AARGB0
                MOVF            CARGB1,W
                MOVWF           AARGB1
                MOVF            CARGB2,W
                MOVWF           AARGB2

                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                CALL            FPM32           ; z * z

                MOVF            AEXP,W
                MOVWF           DEXP            ; save z * z in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       minimax polynomial approximation p(x**2) on [0,pi/4]

ZCOS24          POL24           COS24,3,0

                RETLW           0x00

;**********************************************************************************************

;       minimax polynomial approximation x*p(x**2) on [0,pi/4]

ZSIN24          POL24           SIN24,2,0
```

```
            MOVF        CEXP,W
            MOVWF       BEXP
            MOVF        CARGB0,W
            MOVWF       BARGB0
            MOVF        CARGB1,W
            MOVWF       BARGB1
            MOVF        CARGB2,W
            MOVWF       BARGB2

            CALL        FPM32

            RETLW       0x00

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for sin(z)/z = p(z**2) on [0,pi/4]

SIN240      EQU         0x7E        ; LP0 = .73551298732E+1*******
SIN2400     EQU         0x7F
SIN2401     EQU         0xFF
SIN2402     EQU         0xAC

SIN241      EQU         0x7C        ; LP1 = .40900513905E+1
SIN2410     EQU         0xAA
SIN2411     EQU         0x99
SIN2412     EQU         0x9D

SIN242      EQU         0x78        ; LQ0 = .50982159260E+1
SIN2420     EQU         0x05
SIN2421     EQU         0x10
SIN2422     EQU         0x48


;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for cos(z) = q(z**2) on [0,pi/4]
;       with COS240 constrained to be 1.

COS240      EQU         0x7F        ; LP0 = .73551298732E+1*******
COS2400     EQU         0x00
COS2401     EQU         0x00
COS2402     EQU         0x00

COS241      EQU         0x7D        ; LP1 = .40900513905E+1
COS2410     EQU         0xFF
COS2411     EQU         0xFF
COS2412     EQU         0xD0

COS242      EQU         0x7A        ; LQ0 = .50982159260E+1
COS2420     EQU         0x2A
COS2421     EQU         0x9E
COS2422     EQU         0x76

COS243      EQU         0x75        ; LQ1 = .53849258895E+1
COS2430     EQU         0xB2
COS2431     EQU         0x12
COS2432     EQU         0xBF
;**********************************************************************************************
;**********************************************************************************************

;       Evaluate sqrt(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SQRT24
```

```
;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  SQRT( AARG )

;       Testing on [0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 7       2968    2517.5  clks

;               min     max     mean    rms
;       Error:  -0x0b   0x08    -1.35   3.60    nsb

;----------------------------------------------------------------------------------------------

;       Range reduction for the square root function is naturally produced by
;       the floating point representation,

;               x = f * 2**e, where 1 <= f < 2,

;       leading to the expression

;                          |  sqrt(f) * 2**(e/2),e even
;               sqrt(x) =  |
;                          |  sqrt(f) * sqrt(2) * 2**(e/2),e odd

;       The function sqrt(f) is then approximated by a segmented fourth degree
;       minimax polynomial on the intervals [1,1.5] and [1.5,2].

SQRT24
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR24

                CLRF            AARGB2          ; return if argument zero
                MOVF            AEXP,W
                BTFSC           _Z
                RETLW           0x00

                MOVF            AEXP,W          ; save exponent in CEXP
                MOVWF           CEXP

                MOVF            FPFLAGS,W       ; save RND flag in DARGB3
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVLW           EXPBIAS         ; compute z
                MOVWF           AEXP

                MOVF            AEXP,W          ; save z in DARG
                MOVWF           DEXP
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                CLRF            DARGB2

                BTFSS           AARGB0,MSB-1
                GOTO            SQRT24L

SQRT24H         POL24           SQRT24H,4,0     ; minimax approximation on [1.5,2]

                GOTO            SQRT24OK

SQRT24L
                POL24           SQRT24L,4,0     ; minimax approximation on [1,1.5]
```

```
SQRT24OK
                BTFSC           CEXP,LSB        ; is CEXP even or odd?
                GOTO            RRSQRTOK24

;       fixed point multiplication by sqrt(2)

                BSF             AARGB0,MSB

                MOVLW           0xB5            ; sqrt(2) = 1.41421356237
                MOVWF           BARGB0
                MOVLW           0x04
                MOVWF           BARGB1
                MOVLW           0xF3
                MOVWF           BARGB2

                CALL            FXM2424U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSQRTOK24
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSQRTOK24
                BCF             AARGB0,MSB      ; make  MSB implicit

                MOVLW           EXPBIAS         ; divide exponent by two
                ADDWF           CEXP,F
                RRF             CEXP,W
                MOVWF           AEXP

                BTFSS           DARGB3,RND
                RETLW           0x00
                BSF             FPFLAGS,RND
                CALL            RND3224
                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;----------------------------------------------------------------------------------------------

;       fourth degree minimax polynomial coefficients for sqrt(x) on [1.5,2]

SQRT24H0        EQU             0x7D            ; SQRT24H0 = 3.5963132863E-1
SQRT24H00       EQU             0x38
SQRT24H01       EQU             0x21
SQRT24H02       EQU             0x99

SQRT24H1        EQU             0x7E            ; SQRT24H1 = 8.3106978456E-1
SQRT24H10       EQU             0x54
SQRT24H11       EQU             0xC0
SQRT24H12       EQU             0xFD

SQRT24H2        EQU             0x7C            ; SQRT24H2 = -2.3944355047E-1
SQRT24H20       EQU             0xF5
SQRT24H21       EQU             0x30
SQRT24H22       EQU             0xB1

SQRT24H3        EQU             0x7A            ; SQRT24H3 = 5.5047377031E-2
SQRT24H30       EQU             0x61
SQRT24H31       EQU             0x79
```

```
SQRT24H32       EQU             0x5C

SQRT24H4        EQU             0x77            ; SQRT24H4 = -5.6351436252E-3
SQRT24H40       EQU             0xB8
SQRT24H41       EQU             0xA7
SQRT24H42       EQU             0x03

;       fourth degree minimax polynomial coefficients for sqrt(x) on [1,1.5]

SQRT24L0        EQU             0x7D            ; SQRT24L0 = 3.0221977303E-1
SQRT24L00       EQU             0x1A
SQRT24L01       EQU             0xBC
SQRT24L02       EQU             0x8B

SQRT24L1        EQU             0x7E            ; SQRT24L1 = 9.8831235597E-1
SQRT24L10       EQU             0x7D
SQRT24L11       EQU             0x02
SQRT24L12       EQU             0x0A

SQRT24L2        EQU             0x7D            ; SQRT24L2 = -4.0192034196E-1
SQRT24L20       EQU             0xCD
SQRT24L21       EQU             0xC8
SQRT24L22       EQU             0x81

SQRT24L3        EQU             0x7C            ; SQRT24L3 = 1.3009144111E-1
SQRT24L30       EQU             0x05
SQRT24L31       EQU             0x36
SQRT24L32       EQU             0xB1

SQRT24L4        EQU             0x79            ; SQRT24L4 = -1.8702682470E-2
SQRT24L40       EQU             0x99
SQRT24L41       EQU             0x36
SQRT24L42       EQU             0x36

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A < B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TALTB24

;       Output: logical result in W

;       Result: if A < B TRUE, W = 0x01
;               if A < B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 9       28      14.6    clks

TALTB24         MOVF            AARGB0,W        ; test if signs opposite
                XORWF           BARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TALTB24O

                BTFSC           AARGB0,MSB
                GOTO            TALTB24N

TALTB24P        MOVF            AEXP,W          ; compare positive arguments
                SUBWF           BEXP,W
                BTFSS           _C
```

```
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB24N        MOVF            BEXP,W          ; compare negative arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB24O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A <= B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TALEB24

;       Output: logical result in W

;       Result: if A <= B TRUE, W = 0x01
;               if A <= B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 9       26      14.4    clks
```



```
TALEB24         MOVF            AARGB0,W        ; test if signs opposite
                XORWF           BARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TALEB24O

                BTFSC           AARGB0,MSB
                GOTO            TALEB24N

TALEB24P        MOVF            AEXP,W          ; compare positive arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TALEB24N        MOVF            BEXP,W          ; compare negative arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TALEB24O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A > B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAGTB24

;       Output: logical result in W
```

```
;       Result: if A > B TRUE, W = 0x01
;               if A > B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 9       28      14.6    clks

TAGTB24         MOVF            BARGB0,W        ; test if signs opposite
                XORWF           AARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TAGTB24O

                BTFSC           BARGB0,MSB
                GOTO            TAGTB24N

TAGTB24P        MOVF            BEXP,W          ; compare positive arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB24N        MOVF            AEXP,W          ; compare negative arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB24O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
```



```
;**********************************************************************************************

;       Floating Point Relation A >= B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAGEB24

;       Output: logical result in W

;       Result: if A >= B TRUE, W = 0x01
;               if A >= B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 9       26      14.4    clks

TAGEB24         MOVF            BARGB0,W        ; test if signs opposite
                XORWF           AARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TAGEB24O

                BTFSC           BARGB0,MSB
                GOTO            TAGEB24N

TAGEB24P        MOVF            BEXP,W          ; compare positive arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB24N        MOVF            AEXP,W          ; compare negative arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
```

```
                RETLW           0x01

TAGEB24O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A == B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAEQB24

;       Output: logical result in W

;       Result: if A == B TRUE, W = 0x01
;               if A == B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       14      6.9     clks

TAEQB24         MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _Z
                RETLW           0x00

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                RETLW           0x00

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                RETLW           0x00
                RETLW           0x01

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A =! B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TANEB24

;       Output: logical result in W

;       Result: if A =! B TRUE, W = 0x01
;               if A =! B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       14      6.9     clks

TANEB24         MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _Z
```

```
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************
```


Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX D: PIC16CXXX 32-BIT ELEMENTARY FUNCTION LIBRARY

```
;       RCS Header $Id: exp32.a16 1.4 1997/02/25 14:23:57 F.J.Testa Exp $

;       $Revision: 1.4 $

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate exp10(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    EXP1032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  EXP10( AARG )

;       Testing on [MINLOG10,MAXLOG10] from 100000 trials:

;               min     max     mean
;       Timing: 3515    5384    5001.5  clks

;               min     max     mean    rms
;       Error:  -0xB3   0x14E   25.78   65.54   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp10(x) = 10**x = 10**(z + n*log10(2)) = 10**z * 2**n,

;       where -log10(2)/2 <= z <= log10(2)/2 and n is an integer, evaluated during
;       range reduction. Segmented fifth degree minimax polynomial approximations
;       are used to estimate 10**z on the intervals [-log10(2)/2,0] and [0,log10(2)/2].

EXP1032
                MOVLW           0x5C            ; test for |x| < 2**(-32)/(2*LOG(10))
                SUBWF           EXP,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            EXP1032ONE      ; return e**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP1032
TPEXP1032
                MOVF            AEXP,W          ; positive domain check
                SUBLW           MAXLOG1032EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVF            AARGB0,W
                SUBLW           MAXLOG1032B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK
```



```
                MOVF            AARGB1,W
                SUBLW           MAXLOG1032B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVF            AARGB2,W
                SUBLW           MAXLOG1032B2
                BTFSS           _C
                GOTO            DOMERR32
                GOTO            EXP1032ARGOK

TNEXP1032
                MOVF            AEXP,W          ; negative domain check
                SUBLW           MINLOG1032EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVF            AARGB0,W
                SUBLW           MINLOG1032B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVF            AARGB1,W
                SUBLW           MINLOG1032B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVF            AARGB2,W
                SUBLW           MINLOG1032B2
                BTFSS           _C
                GOTO            DOMERR32

EXP1032ARGOK
                MOVF            FPFLAGS,W       ; save RND flag
                MOVWF           DARGB3

                BSF             FPFLAGS,RND     ; enable rounding
                CALL            RREXP1032

                BTFSC           DARGB0,MSB
                GOTO            EXP1032L

EXP1032H
                POL32           EXP1032H,5,4    ; minimax approximation on [0,log10(2)/2]

                GOTO            EXP1032OK

EXP1032L
                POL32           EXP1032L,5,4    ; minimax approximation on [-log10(2)/2,0]

EXP1032OK
                MOVF            EARGB3,W
                ADDWF           AEXP,F
                RETLW           0x00

EXP1032ONE      MOVLW           EXPBIAS         ; return 10**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0
```

```
                CLRF            AARGB1
                CLRF            AARGB2
                CLRF            AARGB3
                RETLW           0x00

DOMERR32        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       Range reduction routine for the exponential function

;       The evaluation of z and n through the decomposition

;               x = z + n*log10(2)

;       is performed by first evaluating n through the relation

;               n = floor(x*log2(10) + .5)

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x - n*log10(2) = (x - n*c1) - n*c2

;       where c1 is close to log10(2) and has an exact machine representation,
;       typically leading to no error in computing the term in parenthesis.

RREXP1032
                MOVF            AEXP,W
                MOVWF           CEXP            ; save x in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                BSF             AARGB0,MSB

                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                MOVLW           0xD4            ; 1/log10(2) = 3.32192809489
                MOVWF           AARGB0
                MOVLW           0x9A
                MOVWF           AARGB1
                MOVLW           0x78
                MOVWF           AARGB2
                MOVLW           0x47
                MOVWF           AARGB3

                CALL            FXM3224U        ; x * (1/log10(2))

                INCF            AEXP,F
                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP1032YOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
```

```
                RLF             AARGB0,F
                DECF            AEXP,F

RREXP1032YOK    BTFSS           CARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVLW           0x7E            ; k = [ x / log10(2) + .5 ]
                MOVWF           BEXP
                CLRF            BARGB0
                CLRF            BARGB1
                CLRF            BARGB2

                CALL            FPA32

                CALL            FLOOR32

                MOVF            AEXP,W
                MOVWF           EEXP            ; save float k in EARG
                BTFSC           _Z
                GOTO            RREXP1032FEQX
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                MOVF            AARGB2,W
                MOVWF           EARGB2

                MOVLW           0x7D
                MOVWF           BEXP
                MOVLW           0x9A            ; c1 = -.301025390625
                MOVWF           BARGB0
                MOVLW           0x20
                MOVWF           BARGB1
                CLRF            BARGB2

                CALL            FPM32

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                MOVF            AEXP,W
                MOVWF           DEXP            ; save f1 in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                MOVLW           0x6D
                MOVWF           BEXP
                MOVLW           0x9A            ; c2 = 4.6050389811952113E-6
                MOVWF           BARGB0
                MOVLW           0x84
                MOVWF           BARGB1
                MOVLW           0xFC
```

```
                MOVWF       BARGB2

                MOVF        EEXP,W
                MOVWF       AEXP
                MOVF        EARGB0,W
                MOVWF       AARGB0
                MOVF        EARGB1,W
                MOVWF       AARGB1
                MOVF        EARGB2,W
                MOVWF       AARGB2

                CALL        FPM32

                MOVF        DEXP,W
                MOVWF       BEXP
                MOVF        DARGB0,W
                MOVWF       BARGB0
                MOVF        DARGB1,W
                MOVWF       BARGB1
                MOVF        DARGB2,W
                MOVWF       BARGB2

                CALL        FPA32

                MOVF        AEXP,W
                MOVWF       DEXP            ; save f in DARG
                MOVF        AARGB0,W
                MOVWF       DARGB0
                MOVF        AARGB1,W
                MOVWF       DARGB1
                MOVF        AARGB2,W
                MOVWF       DARGB2

                MOVF        EEXP,W
                MOVWF       AEXP
                MOVF        EARGB0,W
                MOVWF       AARGB0
                MOVF        EARGB1,W
                MOVWF       AARGB1

                BCF         FPFLAGS,RND
                CALL        INT2416         ; k = [ x / log10(2) + .5 ]
                BSF         FPFLAGS,RND

                MOVF        AARGB1,W
                MOVWF       EARGB3          ; save integer k in EARGB3

                MOVF        DEXP,W
                MOVWF       AEXP            ; restore f in AARG
                MOVF        DARGB0,W
                MOVWF       AARGB0
                MOVF        DARGB1,W
                MOVWF       AARGB1
                MOVF        DARGB2,W
                MOVWF       AARGB2

                RETLW       0x00

RREXP1032FEQX
                MOVF        CEXP,W
                MOVWF       DEXP
                MOVWF       AEXP            ; save f = x in DARG, AARG
                MOVF        CARGB0,W
                MOVWF       DARGB0
                MOVWF       AARGB0
                MOVF        CARGB1,W
```

```
                MOVWF           DARGB1
                MOVWF           AARGB1
                MOVF            CARGB2,W
                MOVWF           DARGB2
                MOVWF           AARGB2

                CLRF            EARGB3

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       fifth degree minimax polynomial coefficients for 10**(x) on [0,(log10(2))/2]

EXP1032H0       EQU             0x7F            ; EXP1032H0 = 1.0
EXP1032H00      EQU             0x00
EXP1032H01      EQU             0x00
EXP1032H02      EQU             0x00

EXP1032H1       EQU             0x80            ; EXP1032H1 = 2.302585504840E0
EXP1032H10      EQU             0x13
EXP1032H11      EQU             0x5D
EXP1032H12      EQU             0x90

EXP1032H2       EQU             0x80            ; EXP1032H2 = 2.650909138708E0
EXP1032H20      EQU             0x29
EXP1032H21      EQU             0xA8
EXP1032H22      EQU             0x7F

EXP1032H3       EQU             0x80            ; EXP1032H3 = 2.035920309947E0
EXP1032H30      EQU             0x02
EXP1032H31      EQU             0x4C
EXP1032H32      EQU             0x85

EXP1032H4       EQU             0x7F            ; EXP1032H4 = 1.154596329197E0
EXP1032H40      EQU             0x13
EXP1032H41      EQU             0xC9
EXP1032H42      EQU             0xD0

EXP1032H5       EQU             0x7E            ; EXP1032H5 = 6.388992868121E-1
EXP1032H50      EQU             0x23
EXP1032H51      EQU             0x8E
EXP1032H52      EQU             0xE7

;       fifth degree minimax polynomial coefficients for 10**(x) on [-(log10(2))/2,0]

EXP1032L0       EQU             0x7F            ; EXP1032L0 = 1.0
EXP1032L00      EQU             0x00
EXP1032L01      EQU             0x00
EXP1032L02      EQU             0x00

EXP1032L1       EQU             0x80            ; EXP1032L1 = 2.302584716116E0
EXP1032L10      EQU             0x13
EXP1032L11      EQU             0x5D
EXP1032L12      EQU             0x8C

EXP1032L2       EQU             0x80            ; EXP1032L2 = 2.650914554552E0
EXP1032L20      EQU             0x29
EXP1032L21      EQU             0xA8
EXP1032L22      EQU             0x96

EXP1032L3       EQU             0x80            ; EXP1032L3 = 2.033640565225E0
EXP1032L30      EQU             0x02
EXP1032L31      EQU             0x27
EXP1032L32      EQU             0x2B
```

```
EXP1032L4       EQU             0x7F            ; EXP1032L4 = 1.157459289066E0
EXP1032L40      EQU             0x14
EXP1032L41      EQU             0x27
EXP1032L42      EQU             0xA0

EXP1032L5       EQU             0x7D            ; EXP1032L5 = 4.544952589676E-1
EXP1032L50      EQU             0x68
EXP1032L51      EQU             0xB3
EXP1032L52      EQU             0x9A

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate exp(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    EXP32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  EXP( AARG )

;       Testing on [MINLOG,MAXLOG] from 100000 trials:

;               min     max     mean
;       Timing: 16      5411    4985.0  clks

;               min     max     mean    rms
;       Error:  -0xD2   0xF7    2.50    63.99   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp(x) = e**x = e**(z + n*log(2)) = e**z * 2**n,

;       where -log(2)/2 <= z <= log(2)/2 and n is an integer, evaluated during
;       range reduction. Segmented fifth degree minimax polynomial approximations
;       are used to estimate e**z on the intervals [-log(2)/2,0] and [0,log(2)/2].

EXP32
                MOVLW           0x5E            ; test for |x| < 2**(-32)/2
                SUBWF           EXP,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            EXP32ONE        ; return e**x = 1

                BTFSC           AARGB0,MSB
                GOTO            TNEXP32
TPEXP32
                MOVF            AEXP,W
                SUBLW           MAXLOG32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVF            AARGB0,W
                SUBLW           MAXLOG32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK
```

```
                MOVF            AARGB1,W
                SUBLW           MAXLOG32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVF            AARGB2,W
                SUBLW           MAXLOG32B2
                BTFSS           _C
                GOTO            DOMERR32
                GOTO            EXP32ARGOK

TNEXP32
                MOVF            AEXP,W
                SUBLW           MINLOG32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVF            AARGB0,W
                SUBLW           MINLOG32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVF            AARGB1,W
                SUBLW           MINLOG32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVF            AARGB2,W
                SUBLW           MINLOG32B2
                BTFSS           _C
                GOTO            DOMERR32

EXP32ARGOK
                MOVF            FPFLAGS,W
                MOVWF           DARGB3          ; save rounding flag
                BCF             FPFLAGS,RND     ; disable rounding

                CALL            RREXP32

                BTFSC           DARGB0,MSB
                GOTO            EXP32L

EXP32H
                POL32           EXP32H,5,0

                MOVF            EARGB3,W
                ADDWF           AEXP,F
                RETLW           0x00

EXP32L
                POL32           EXP32L,5,0

EXP32OK
                MOVF            EARGB3,W
                ADDWF           AEXP,F
                BTFSS           DARGB3,RND
                RETLW           0x00
```

```
                BSF             FPFLAGS,RND     ; restore rounding flag
                GOTO            RND4032

EXP32ONE        MOVLW           EXPBIAS         ; return e**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0
                CLRF            AARGB1
                CLRF            AARGB2
                CLRF            AARGB3
                RETLW           0x00

DOMERR32        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       Range reduction routine for the exponential function

;       The evaluation of z and n through the decomposition

;               x = z + n*log(2)

;       is performed by first evaluating n through the relation

;               n = floor(x*log2(e) + .5)

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x - n*log(2) = (x - n*c1) + n*c2

;       where c1 is close to log(2) and has an exact machine representation,
;       typically leading to no error in computing the term in parenthesis.

RREXP32
                MOVF            AEXP,W
                MOVWF           CEXP            ; save x in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                BSF             AARGB0,MSB

                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                MOVLW           0xB8            ; 1/ln(2) = 1.44269504089
                MOVWF           AARGB0
                MOVLW           0xAA
                MOVWF           AARGB1
                MOVLW           0x3B
                MOVWF           AARGB2
                MOVLW           0x29
                MOVWF           AARGB3

                CALL            FXM3224U        ; x * (1/ln2)

                INCF            AEXP,F
```

```
                BTFSC           AARGB0,MSB
                GOTO            RREXP32YOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RREXP32YOK      BTFSS           CARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVLW           0x7E                    ; k = [ x / ln2 + .5 ]
                MOVWF           BEXP
                CLRF            BARGB0
                CLRF            BARGB1
                CLRF            BARGB2

                CALL            FPA32

                CALL            FLOOR32

                MOVF            AEXP,W
                MOVWF           EEXP                    ; save float k in EARG
                BTFSC           _Z
                GOTO            RREXP32FEQX
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                MOVF            AARGB2,W
                MOVWF           EARGB2

                MOVLW           0x7E
                MOVWF           BEXP
                MOVLW           0xB1                    ; c1 = .693359375
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1
                CLRF            BARGB2

                CALL            FPM32

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                MOVF            AEXP,W
                MOVWF           DEXP                    ; save f1 in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                MOVLW           0x72
                MOVWF           BEXP
```



```
                MOVLW           0x5E            ; c2 = .00021219444005
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1
                MOVLW           0x83
                MOVWF           BARGB2

                MOVF            EEXP,W
                MOVWF           AEXP
                MOVF            EARGB0,W
                MOVWF           AARGB0
                MOVF            EARGB1,W
                MOVWF           AARGB1
                MOVF            EARGB2,W
                MOVWF           AARGB2

                CALL            FPM32

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                CALL            RND4032

                MOVF            AEXP,W
                MOVWF           DEXP            ; save f in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                MOVF            EEXP,W
                MOVWF           AEXP
                MOVF            EARGB0,W
                MOVWF           AARGB0
                MOVF            EARGB1,W
                MOVWF           AARGB1

                CALL            INT2416         ; k = [ x / ln2 + .5 ]

                MOVF            AARGB1,W
                MOVWF           EARGB3          ; save integer k in EARGB3

                MOVF            DEXP,W
                MOVWF           AEXP            ; restore f in AARG
                MOVF            DARGB0,W
                MOVWF           AARGB0
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                RETLW           0x00

RREXP32FEQX
                MOVF            CEXP,W
                MOVWF           DEXP
```

```
                MOVWF           AEXP            ; save f = x in DARG, AARG
                MOVF            CARGB0,W
                MOVWF           DARGB0
                MOVWF           AARGB0
                MOVF            CARGB1,W
                MOVWF           DARGB1
                MOVWF           AARGB1
                MOVF            CARGB2,W
                MOVWF           DARGB2
                MOVWF           AARGB2

                CLRF            EARGB3

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       fifth degree minimax polynomial coefficients for e**(x) on [0,(ln2)/2]

EXP32H0         EQU             0x7F            ; EXP32H0 = 1.0
EXP32H00        EQU             0x00
EXP32H01        EQU             0x00
EXP32H02        EQU             0x00

EXP32H1         EQU             0x7F            ; EXP32H1 = 1.00000025499
EXP32H10        EQU             0x00
EXP32H11        EQU             0x00
EXP32H12        EQU             0x02

EXP32H2         EQU             0x7D            ; EXP32H2 = .499991163105
EXP32H20        EQU             0x7F
EXP32H21        EQU             0xFE
EXP32H22        EQU             0xD7

EXP32H3         EQU             0x7C            ; EXP32H3 = .166777360103
EXP32H30        EQU             0x2A
EXP32H31        EQU             0xC7
EXP32H32        EQU             0xAF

EXP32H4         EQU             0x7A            ; EXP32H4 = .410473706887E-1
EXP32H40        EQU             0x28
EXP32H41        EQU             0x21
EXP32H42        EQU             0x4A

EXP32H5         EQU             0x78            ; EXP32H5 = .989943653774E-2
EXP32H50        EQU             0x22
EXP32H51        EQU             0x31
EXP32H52        EQU             0x3F

;       fifth degree minimax polynomial coefficients for e**(x) on [-(ln2)/2,0]

EXP32L0         EQU             0x7F            ; EXP32L0 = 1.0
EXP32L00        EQU             0x00
EXP32L01        EQU             0x00
EXP32L02        EQU             0x00

EXP32L1         EQU             0x7E            ; EXP32L1 = .999999766814
EXP32L10        EQU             0x7F
EXP32L11        EQU             0xFF
EXP32L12        EQU             0xFC

EXP32L2         EQU             0x7D            ; EXP32L2 = .499992371926
EXP32L20        EQU             0x7F
EXP32L21        EQU             0xFF
EXP32L22        EQU             0x00
```

```
EXP32L3         EQU             0x7C            ; EXP32L3 = .166574299807
EXP32L30        EQU             0x2A
EXP32L31        EQU             0x92
EXP32L32        EQU             0x74

EXP32L4         EQU             0x7A            ; EXP32L4 = .411548782678E-1
EXP32L40        EQU             0x28
EXP32L41        EQU             0x92
EXP32L42        EQU             0x05

EXP32L5         EQU             0x77            ; EXP32L5 = .699995870637E-2
EXP32L50        EQU             0x65
EXP32L51        EQU             0x5F
EXP32L52        EQU             0xE9

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate floor(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    FLOOR32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  FLOOR( AARG )

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 41      61      47.32   clks

;               min     max     mean    rms
;       Error:  0x00    0x00    0.0     0.0     nsb

;----------------------------------------------------------------------------------------------

;       floor(x) evaluates the largest integer, as a float, not greater than x.

FLOOR32
                CLRF            AARGB3          ; test for zero argument
                MOVF            AEXP,W
                BTFSC           _Z
                RETLW           0x00

                MOVF            AARGB0,W
                MOVWF           AARGB4          ; save mantissa
                MOVF            AARGB1,W
                MOVWF           AARGB5
                MOVF            AARGB2,W
                MOVWF           AARGB6

                MOVLW           EXPBIAS
                SUBWF           AEXP,W
                MOVWF           TEMPB1
                BTFSC           TEMPB1,MSB
                GOTO            FLOOR32ZERO

                SUBLW           0x18-1
                MOVWF           TEMPB0          ; save number of zero bits in TEMPB0
                MOVWF           TEMPB1

                BTFSC           TEMPB1,LSB+1+3  ; divide by eight
                GOTO            FLOOR32MASKH
                BTFSC           TEMPB1,LSB+3
```

```
                GOTO            FLOOR32MASKM

FLOOR32MASKL
                MOVLW           0x07            ; get remainder for mask pointer
                ANDWF           TEMPB0,F
                MOVLW           LOW FLOOR32MASKTABLE
                ADDWF           TEMPB0,F
                MOVLW           HIGH FLOOR32MASKTABLE
                BTFSC           _C
                ADDLW           0x01
                MOVWF           PCLATH
                INCF            TEMPB0,W

                CALL            FLOOR32MASKTABLE ; access table for mask

                ANDWF           AARGB2,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVF            AARGB6,W
                SUBWF           AARGB2,W
                BTFSS           _Z
                GOTO            FLOOR32RNDL
                RETLW           0x00

FLOOR32RNDL
                COMF            AARGB7,W
                MOVWF           TEMPB1
                INCF            TEMPB1,W
                ADDWF           AARGB2,F
                BTFSC           _Z
                INCF            AARGB1, F
                BTFSC           _Z
                INCF            AARGB0, F
                BTFSS           _Z              ; has rounding caused carryout?
                RETLW           0x00
                RRF             AARGB0,F
                RRF             AARGB1,F
                RRF             AARGB2,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32MASKM
                MOVLW           0x07            ; get remainder for mask pointer
                ANDWF           TEMPB0,F
                MOVLW           LOW FLOOR32MASKTABLE
                ADDWF           TEMPB0,F
                MOVLW           HIGH FLOOR32MASKTABLE
                BTFSC           _C
                ADDLW           0x01
                MOVWF           PCLATH
                INCF            TEMPB0,W

                CALL            FLOOR32MASKTABLE ; access table for mask

                ANDWF           AARGB1,F
                CLRF            AARGB2
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVF            AARGB6,W
                SUBWF           AARGB2,W
                BTFSS           _Z
```



```
                GOTO            FLOOR32RNDM
                MOVF            AARGB5,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                GOTO            FLOOR32RNDM
                RETLW           0x00

FLOOR32RNDM
                COMF            AARGB7,W
                MOVWF           TEMPB1
                INCF            TEMPB1,W
                ADDWF           AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F
                BTFSS           _Z              ; has rounding caused carryout?
                RETLW           0x00
                RRF             AARGB0,F
                RRF             AARGB1,F
                RRF             AARGB2,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32MASKH
                MOVLW           0x07            ; get remainder for mask pointer
                ANDWF           TEMPB0,F
                MOVLW           LOW FLOOR32MASKTABLE
                ADDWF           TEMPB0,F
                MOVLW           HIGH FLOOR32MASKTABLE
                BTFSC           _C
                ADDLW           0x01
                MOVWF           PCLATH
                INCF            TEMPB0,W

                CALL            FLOOR32MASKTABLE ; access table for mask

                ANDWF           AARGB0,F
                CLRF            AARGB1
                CLRF            AARGB2
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVF            AARGB6,W
                SUBWF           AARGB2,W
                BTFSS           _Z
                GOTO            FLOOR32RNDH
                MOVF            AARGB5,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                GOTO            FLOOR32RNDH
                MOVF            AARGB4,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                GOTO            FLOOR32RNDH
                RETLW           0x00

FLOOR32RNDH
                COMF            AARGB7,W
                MOVWF           TEMPB1
                INCF            TEMPB1,W
                ADDWF           AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRF             AARGB0,F
                RRF             AARGB1,F
```

```
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32ZERO
                BTFSC           AARGB0,MSB
                GOTO            FLOOR32MINUSONE
                CLRF            AEXP
                CLRF            AARGB0
                CLRF            AARGB1
                CLRF            AARGB2
                RETLW           0x00

FLOOR32MINUSONE
                MOVLW           0x7F
                MOVWF           AEXP
                MOVLW           0x80
                MOVWF           AARGB0
                CLRF            AARGB1
                CLRF            AARGB2
                RETLW           0x00

;----------------------------------------------------------------------------------------------

FLOOR32MASKTABLE
                MOVWF           PCL
                RETLW           0xFF
                RETLW           0xFE
                RETLW           0xFC
                RETLW           0xF8
                RETLW           0xF0
                RETLW           0xE0
                RETLW           0xC0
                RETLW           0x80
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate log10(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    LOG1032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  LOG10( AARG )

;       Testing on (0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5208    5949    5605.7  clks

;               min     max     mean    rms
;       Error:  -0x96   0xAC    59.20   87.50   nsb

;----------------------------------------------------------------------------------------------

LOG1032         MOVF            FPFLAGS,W
                MOVWF           ZARGB0
                BSF             FPFLAGS,RND

                CALL            LOG32

                MOVLW           0x7D
```



```
                MOVWF           BEXP
                MOVLW           0x5E            ; log10(e) = .43429448190325
                MOVWF           BARGB0
                MOVLW           0x5B
                MOVWF           BARGB1
                MOVLW           0xD9
                MOVWF           BARGB2

                BTFSS           ZARGB0,RND
                BCF             FPFLAGS,RND

                CALL            FPM32

                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate log(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    LOG32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  LOG( AARG )

;       Testing on (0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 4797    5406    5090.6  clks

;               min     max     mean    rms
;       Error:  -0xF0   0x02    0.57    1.12    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log(x) = log(f) + log(2**n) = log(f) + n*log(2)

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |   2*f-1, f < 1/sqrt(2), n=n-1
;               z = |
;                   |   f-1, otherwise

;       produces a naturally segmented representation of log(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.  The final evaluation of

;               log(1+z) + n*log(2) = (log(1+z) - n*c2) + n*c1

;       is performed in pseudo extended precision where c1 is close to log(2)
;       and has an exact machine representation.

LOG32
                CLRF            AARGB3
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR32
                MOVF            AEXP,W          ; test for zero argument
                BTFSC           _Z
                GOTO            DOMERR32
```

```
                MOVF        FPFLAGS,W       ; save rounding flag
                MOVWF       DARGB3
                BCF         FPFLAGS,RND     ; disable rounding

                MOVF        AEXP,W
                MOVWF       EARGB3
                MOVLW       EXPBIAS-1
                SUBWF       EARGB3,F
                MOVWF       AEXP

                MOVLW       0xF3            ; .70710678118655 = 7E3504F3
                SUBWF       AARGB2,W
                MOVLW       0x04
                MOVWF       TEMP
                BTFSS       _C
                INCFSZ      TEMP,W
                SUBWF       AARGB1,W
                MOVLW       0x35
                MOVWF       TEMP
                BTFSS       _C
                INCFSZ      TEMP,W
                SUBWF       AARGB0,W

                BTFSS       _C
                GOTO        LOG32FLOW

LOG32FHIGH      MOVLW       0x7F
                MOVWF       BEXP
                CLRF        BARGB0
                CLRF        BARGB1
                CLRF        BARGB2

                CALL        FPS32

                GOTO        LOGZ32OK

LOG32FLOW       INCF        AEXP,F
                MOVLW       0x7F
                MOVWF       BEXP
                CLRF        BARGB0
                CLRF        BARGB1
                CLRF        BARGB2

                CALL        FPS32

                DECF        EARGB3,F

LOGZ32OK
                MOVF        AEXP,W          ; save z
                MOVWF       DEXP
                MOVF        AARGB0,W
                MOVWF       DARGB0
                MOVF        AARGB1,W
                MOVWF       DARGB1
                MOVF        AARGB2,W
                MOVWF       DARGB2

                POLL132     LOG32Q,2,0      ; Q(z)

                MOVF        AEXP,W
                MOVWF       CEXP
                MOVF        AARGB0,W
                MOVWF       CARGB0
                MOVF        AARGB1,W
                MOVWF       CARGB1
                MOVF        AARGB2,W
```

```
        MOVWF           CARGB2

        MOVF            DEXP,W
        MOVWF           AEXP
        MOVF            DARGB0,W
        MOVWF           AARGB0
        MOVF            DARGB1,W
        MOVWF           AARGB1
        MOVF            DARGB2,W
        MOVWF           AARGB2

        POL32           LOG32P,1,0      ; P(z)

        MOVF            CEXP,W
        MOVWF           BEXP
        MOVF            CARGB0,W
        MOVWF           BARGB0
        MOVF            CARGB1,W
        MOVWF           BARGB1
        MOVF            CARGB2,W
        MOVWF           BARGB2

        CALL            FPD32           ; P(z)/Q(z)

        MOVF            AEXP,W          ; save in CARG
        MOVWF           CEXP
        MOVF            AARGB0,W
        MOVWF           CARGB0
        MOVF            AARGB1,W
        MOVWF           CARGB1
        MOVF            AARGB2,W
        MOVWF           CARGB2

        MOVF            DEXP,W
        MOVWF           BEXP
        MOVF            DARGB0,W
        MOVWF           BARGB0
        MOVF            DARGB1,W
        MOVWF           BARGB1
        MOVF            DARGB2,W
        MOVWF           BARGB2

        MOVF            DEXP,W
        MOVWF           AEXP
        MOVF            DARGB0,W
        MOVWF           AARGB0
        MOVF            DARGB1,W
        MOVWF           AARGB1
        MOVF            DARGB2,W
        MOVWF           AARGB2

        CALL            FPM32           ; z*z

        MOVF            AEXP,W          ; save in EARG
        MOVWF           EEXP
        MOVF            AARGB0,W
        MOVWF           EARGB0
        MOVF            AARGB1,W
        MOVWF           EARGB1
        MOVF            AARGB2,W
        MOVWF           EARGB2

        MOVF            CEXP,W          ; z*z*P(z)/Q(z)
        MOVWF           BEXP
        MOVF            CARGB0,W
        MOVWF           BARGB0
```

```
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            DEXP,W          ; z*(z*z*P(z)/Q(z))
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            EARGB0,W
                MOVWF           BARGB0
                MOVF            EARGB1,W
                MOVWF           BARGB1
                MOVF            EARGB2,W
                MOVWF           BARGB2
                MOVF            EEXP,W          ; -.5*z*z + z*(z*z*P(z)/Q(z))
                MOVWF           BEXP
                BTFSS           _Z
                DECF            BEXP,F

                CALL            FPS32

                CALL            RND4032

                MOVF            DEXP,W          ; z -.5*z*z + z*(z*z*P(z)/Q(z))
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                BTFSC           DARGB3,RND
                CALL            RND4032

                MOVF            EARGB3,W
                BTFSS           _Z
                GOTO            ADJLOG32
                RETLW           0x00

ADJLOG32
                CALL            RND4032

                MOVF            AEXP,W          ; save in EARG
                MOVWF           EEXP
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                MOVF            AARGB2,W
                MOVWF           EARGB2

                CLRF            AARGB0
                MOVF            EARGB3,W
```



```
                MOVWF           AARGB1
                BTFSC           AARGB1,MSB
                COMF            AARGB0,F

                CALL            FLO1624
                CLRF            AARGB2

                MOVF            AEXP,W          ; save k in DARG
                MOVWF           DEXP
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                BSF             AARGB0,MSB
                MOVLW           0x0D-1          ; .000212194440055
                SUBWF           AEXP,F
                MOVLW           0xDE
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1
                MOVLW           0x83
                MOVWF           BARGB2

                CALL            FXM2424U

                BTFSC           AARGB0,MSB
                GOTO            LOG32F1OK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

LOG32F1OK
                BTFSC           DARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVF            EEXP,W          ; log(1+z) + k*log(2)
                MOVWF           BEXP
                MOVF            EARGB0,W
                MOVWF           BARGB0
                MOVF            EARGB1,W
                MOVWF           BARGB1
                MOVF            EARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                CALL            RND4032

                MOVF            AEXP,W          ; save in EARG
                MOVWF           EEXP
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                MOVF            AARGB2,W
                MOVWF           EARGB2

                MOVLW           0xB1            ; .693359375
                MOVWF           BARGB0
```

```
                MOVLW           0x80
                MOVWF           BARGB1

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                BSF             AARGB0,MSB

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            LOG32FOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

LOG32FOK
                BTFSS           DARGB0,MSB
                BCF             AARGB0,MSB

                MOVF            EEXP,W          ; log(1+z) + k*log(2)
                MOVWF           BEXP
                MOVF            EARGB0,W
                MOVWF           BARGB0
                MOVF            EARGB1,W
                MOVWF           BARGB1
                MOVF            EARGB2,W
                MOVWF           BARGB2

                CALL            FPA32

                BTFSC           DARGB3,RND
                GOTO            RND4032

DOMERR32        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;---------------------------------------------------------------------------------------------

;        minimax rational approximation z-.5*z*z+z*(z*z*P(z)/Q(z))

LOG32P0         EQU             0x7E            ; LOG32P0 = .83311400452
LOG32P00        EQU             0x55
LOG32P01        EQU             0x46
LOG32P02        EQU             0xF6

LOG32P1         EQU             0x7D            ; LOG32P1 = .48646956294
LOG32P10        EQU             0x79
LOG32P11        EQU             0x12
LOG32P12        EQU             0x8A

LOG32Q0         EQU             0x80            ; LOG32Q0 = .24993759223E1
LOG32Q00        EQU             0x1F
LOG32Q01        EQU             0xF5
LOG32Q02        EQU             0xC6

LOG32Q1         EQU             0x80            ; LOG32Q1 = .33339502905E+1
LOG32Q10        EQU             0x55
```

```
LOG32Q11        EQU             0x5F
LOG32Q12        EQU             0x72

LOG32Q2         EQU             0x7F            ; LOG32Q2 = 1.0
LOG32Q20        EQU             0x00
LOG32Q21        EQU             0x00
LOG32Q22        EQU             0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate rand(x)

;       Input:  32 bit initial integer seed in AARGB0, AARGB1, AARGB2, AARGB3

;       Use:    CALL    RAND32

;       Output: 32 bit random integer in AARGB0, AARGB1, AARGB2, AARGB3

;       Result: AARG  <--  RAND32( AARG )

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 487     487     487     clks

;               min     max     mean
;       Error:  0x00    0x00    0x00    nsb

;----------------------------------------------------------------------------------------------

;       Linear congruential random number generator

;               X <- (a * X + c) mod m

;       The calculation is performed exactly, with multiplier a, increment c, and
;       modulus m, selected to achieve high ratings from standard spectral tests.

RAND32
                MOVF            RANDB0,W
                MOVWF           AARGB0
                MOVF            RANDB1,W
                MOVWF           AARGB1
                MOVF            RANDB2,W
                MOVWF           AARGB2
                MOVF            RANDB3,W
                MOVWF           AARGB3

                MOVLW           0x0D            ; multiplier a = 1664525
                MOVWF           BARGB2
                MOVLW           0x66
                MOVWF           BARGB1
                MOVLW           0x19
                MOVWF           BARGB0

                CALL            FXM3224U

                INCF            AARGB6,F        ; c = 1
                BTFSC           _Z
                INCF            AARGB5,F
                BTFSC           _Z
                INCF            AARGB4,F
                BTFSC           _Z
                INCF            AARGB3,F
                BTFSC           _Z
                INCF            AARGB2,F
```

```
                BTFSC           _Z
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

                MOVF            AARGB3,W
                MOVWF           RANDB0          ; m = 2**32
                MOVF            AARGB4,W
                MOVWF           RANDB1
                MOVF            AARGB5,W
                MOVWF           RANDB2
                MOVF            AARGB6,W
                MOVWF           RANDB3

                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Nearest neighbor rounding

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    RND3224

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <-- RND( AARG )

;       Testing on [MINNUM,MAXNUM] from 10000 trials:

;               min     max     mean
;       Timing: 3       17              clks

;               min     max     mean
;       Error:  0       0       0       nsb

;----------------------------------------------------------------------------------------------

RND3224
                BTFSS           AARGB2,MSB      ; is NSB < 0x80?
                RETLW           0x00

                BSF             _C              ; set carry for rounding
                MOVLW           0x7F
                ANDWF           AARGB2,W
                BTFSC           _Z
                RRF             AARGB1,W        ; select even if NSB = 0x80

                MOVF            AARGB0,W
                MOVWF           SIGN            ; save sign
                BSF             AARGB0,MSB      ; make MSB explicit

                BCF             _Z
                BTFSC           _C              ; round
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

                BTFSS           _Z              ; has rounding caused carryout?
                GOTO            RND3224OK
                RRF             AARGB0,F        ; if so, right shift
                RRF             AARGB1,F
                INCF            EXP,F           ; test for floating point overflow
                BTFSC           _Z
                GOTO            SETFOV24
```

```
RND3224OK
                BTFSS           SIGN,MSB
                BCF             AARGB0,MSB      ; clear sign bit if positive
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Nearest neighbor rounding

;       Input:  40 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2, AARGB3

;       Use:    CALL    RND4032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <-- RND( AARG )

;       Testing on [MINNUM,MAXNUM] from 10000 trials:

;               min     max     mean
;       Timing: 3       17              clks

;               min     max     mean
;       Error:  0       0       0       nsb

;----------------------------------------------------------------------------------------------

RND4032
                BTFSS           AARGB3,MSB      ; is NSB < 0x80?
                RETLW           0x00

                BSF             _C              ; set carry for rounding
                MOVLW           0x7F
                ANDWF           AARGB3,W
                BTFSC           _Z
                RRF             AARGB2,W        ; select even if NSB = 0x80

                MOVF            AARGB0,W
                MOVWF           SIGN            ; save sign
                BSF             AARGB0,MSB      ; make MSB explicit

                BCF             _Z
                BTFSC           _C              ; round
                INCF            AARGB2,F
                BTFSC           _Z
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

                BTFSS           _Z              ; has rounding caused carryout?
                GOTO            RND4032OK
                RRF             AARGB0,F        ; if so, right shift
                RRF             AARGB1,F
                RRF             AARGB2,F
                INCF            EXP,F           ; test for floating point overflow
                BTFSC           _Z
                GOTO            SETFOV32
RND4032OK
                BTFSS           SIGN,MSB
                BCF             AARGB0,MSB      ; clear sign bit if positive
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************
```

```
;**********************************************************************************************
;**********************************************************************************************

;       Evaluate cos(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    COS32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  COS( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 3568    6098    5545.9  clks

;               min     max     mean    rms
;       Error:  -0x225  0x1E5   -10.42  98.36   nsb

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations.

COS32
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                CLRF            CARGB3          ; initialize sign in CARGB3

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS32      ; range reduction

RRCOS32OK
                RRF             EARGB3,W
                XORWF           EARGB3,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,LSB
                GOTO            COSZSIN32

                CALL            ZCOS32

                GOTO            COSSIGN32

COSZSIN32       CALL            ZSIN32

COSSIGN32
                MOVLW           0x80
                BTFSC           EARGB3,LSB+1
                XORWF           CARGB3,F

                BTFSC           CARGB3,MSB
                XORWF           AARGB0,F

                BTFSS           DARGB3,RND
```



```
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND4032
                RETLW           0x00

;*********************************************************************************************

;       Evaluate sin(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SIN32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 4030    6121    5545.7  clks

;               min     max     mean    rms
;       Error:  -0x22D  0x1F1   -9.55   97.87   nsb

;---------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations.

SIN32
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                CLRF            CARGB3          ; initialize sign in CARGB3

                BTFSC           AARGB0,MSB
                BSF             CARGB3,MSB

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS32      ; range reduction

RRSIN32OK
                RRF             EARGB3,W
                XORWF           EARGB3,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,LSB
                GOTO            SINZCOS32

                CALL            ZSIN32

                GOTO            SINSIGN32

SINZCOS32       CALL            ZCOS32
```

```
SINSIGN32
                MOVLW           0x80
                BTFSC           CARGB3,MSB
                XORWF           AARGB0,F

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND4032
                RETLW           0x00

;*********************************************************************************************

;       Evaluate sin(x) and cos(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SINCOS32

;       Output: 32 bit floating point cos(x) in AEXP, AARGB0, AARGB1, AARGB2 and
;               sin(x) BEXP, BARGB0, BARGB1, BARGB2

;       Result: AARG  <--  COS( AARG )
;               BARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 6611    8858    8382.6  clks

;               min     max     mean    rms
;       Error:  -0x225  0x1E5   -10.42  98.36   nsb     cos(x)
;               -0x22D  0x1F1   -9.55   97.87           sin(x)

;---------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations. In this case,
;       only one range reduction is necessary.

SINCOS32
                MOVF            FPFLAGS,W       ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                MOVF            AEXP,W          ; save x in EARG
                MOVWF           EEXP
                MOVF            AARGB0,W
                MOVWF           EARGB0
                MOVF            AARGB1,W
                MOVWF           EARGB1
                MOVF            AARGB2,W
                MOVWF           EARGB2

                BCF             AARGB0,MSB      ; use |x|

                CLRF            CARGB3          ; initialize sign in CARGB3
```

```
            CALL            RRSINCOS32          ; range reduction

            MOVF            CARGB3,W            ; save sign from range reduction
            MOVWF           ZARGB2

            MOVLW           0x80
            BTFSC           EARGB0,MSB          ; toggle sign if x < 0
            XORWF           CARGB3,F

            CALL            RRSIN32OK

            MOVF            AEXP,W              ; save sin(x) in EARG
            MOVWF           EEXP
            MOVF            AARGB0,W
            MOVWF           EARGB0
            MOVF            AARGB1,W
            MOVWF           EARGB1
            MOVF            AARGB2,W
            MOVWF           EARGB2
            MOVF            AARGB3,W
            MOVWF           ZARGB3

            BSF             FPFLAGS,RND         ; enable rounding

            MOVF            DEXP,W              ; restore z*z in AARG
            MOVWF           AEXP
            MOVF            DARGB0,W
            MOVWF           AARGB0
            MOVF            DARGB1,W
            MOVWF           AARGB1
            MOVF            DARGB2,W
            MOVWF           AARGB2

            MOVF            ZARGB2,W            ; restore sign from range reduction
            MOVWF           CARGB3

            CALL            RRCOS32OK

            MOVF            EEXP,W              ; restore sin(x) in BARG
            MOVWF           BEXP
            MOVF            EARGB0,W
            MOVWF           BARGB0
            MOVF            EARGB1,W
            MOVWF           BARGB1
            MOVF            EARGB2,W
            MOVWF           BARGB2
            MOVF            ZARGB3,W
            MOVWF           BARGB3

            RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for trigonometric functions

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition

;               z = x mod pi/4,

;       produced by first evaluating y and j through the relations

;               y = floor(x/(pi/4)), j = y - 8*[y/8].
```

```
;       where j equals the correct octant.  For j odd, adding one to j
;       and y eliminates the odd octants.  Additional logic on j and the
;       sign of the result leads to appropriate use of the sine or cosine
;       routine in each case.

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;       z = x mod pi/4 = x - y*(pi/4) = (((x - p1*y)-p2*y)-p3*y)-p4*y

;       where pi/4 = p1 + p2 + p3 + p4, with p1 close to pi/4, p2 close to
;       pi/4 - p1, and p3 close to pi/4 - p1 - p2. The numbers p1, p2 and p3
;       are chosen to have an exact machine representation with slightly more
;       than the lower half of the mantissa bits zero, typically leading to no
;       error in computing the terms in parenthesis. This calculation breaks
;       down leading to a loss of precision for |x| > LOSSTHR = sqrt(2**24)*pi/4,
;       or for |x| close to an integer multiple of pi/4. This loss threshold has
;       been chosen based on the efficacy of this calculation, with a domain error
;       reported if this threshold is exceeded.

RRSINCOS32
                MOVF            AEXP,W          ; loss threshold check
                SUBLW           LOSSTHR32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVF            AARGB0,W
                SUBLW           LOSSTHR32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVF            AARGB1,W
                SUBLW           LOSSTHR32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVF            AARGB2,W
                SUBLW           LOSSTHR32B2
                BTFSS           _C
                GOTO            DOMERR32

RRSINCOS32ARGOK
                MOVF            AEXP,W
                MOVWF           CEXP            ; save |x| in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

;       fixed point multiplication by 4/pi

                BSF             AARGB0,MSB
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2
```

```
                MOVLW           0xA2            ; 4/pi = 1.27323954474
                MOVWF           AARGB0
                MOVLW           0xF9
                MOVWF           AARGB1
                MOVLW           0x83
                MOVWF           AARGB2
                MOVLW           0x6E
                MOVWF           AARGB3

                CALL            FXM3224U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32YOK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS32YOK
                BCF             AARGB0,MSB

                BCF             FPFLAGS,RND
                CALL            INT3224         ; y = [ |x| * (4/pi) ]
                BSF             FPFLAGS,RND

                BTFSS           AARGB2,LSB
                GOTO            SAVEY32

                INCF            AARGB2,F
                BTFSC           _Z
                INCF            AARGB1,F
                BTFSC           _Z
                INCF            AARGB0,F

SAVEY32         MOVF            AARGB0,W
                MOVWF           DARGB0          ; save y in DARG
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                MOVLW           0x07            ; j = y mod 8
                ANDWF           AARGB2,F

                MOVLW           0x03
                SUBWF           AARGB2,W

                MOVLW           0x80
                BTFSS           _C
                GOTO            JOK32
                XORWF           CARGB3,F
                MOVLW           0x04
                SUBWF           AARGB2,F

JOK32           MOVF            AARGB2,W
                MOVWF           EARGB3          ; save j in EARGB3

                MOVF            DARGB0,W
                MOVWF           AARGB0          ; restore y to AARG
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
```

```
                MOVWF           AARGB2

                CALL            FLO2432

                MOVF            AEXP,W
                MOVWF           DEXP            ; save y in DARG
                BTFSC           _Z
                GOTO            RRSINCOS32ZEQX
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

;       Cody-Waite extended precision calculation of |x| - y * pi/4 using
;       fixed point multiplication. Since y >= 1, underflow is not possible
;       in any of the products.

                BSF             AARGB0,MSB

                MOVLW           0xC9            ; - p1 = -.78515625
                MOVWF           BARGB0
                CLRF            BARGB1

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z1OK
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z1OK
                MOVF            CEXP,W          ; restore x to BARG
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32           ; z1 = |x| - y * (p1)

                MOVF            AEXP,W
                MOVWF           CEXP            ; save z1 in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           AARGB0          ; restore y to AARG
                MOVF            DARGB1,W
                MOVWF           AARGB1
                MOVF            DARGB2,W
                MOVWF           AARGB2

                BSF             AARGB0,MSB
```

```
                MOVLW       0xFD            ; - p2 = -.000241875648498535156244
                MOVWF       BARGB0
                MOVLW       0xA0
                MOVWF       BARGB1

                CALL        FXM2416U

                MOVLW       0x0D - 1

                BTFSC       AARGB0,MSB
                GOTO        RRSINCOS32Z2OK
                RLF         AARGB3,F
                RLF         AARGB2,F
                RLF         AARGB1,F
                RLF         AARGB0,F
                DECF        AEXP,F

RRSINCOS32Z2OK
                SUBWF       AEXP,F

                MOVF        CEXP,W          ; restore z1 to BARG
                MOVWF       BEXP
                MOVF        CARGB0,W
                MOVWF       BARGB0
                MOVF        CARGB1,W
                MOVWF       BARGB1
                MOVF        CARGB2,W
                MOVWF       BARGB2

                CALL        FPA32           ; z2 = z1 - y * (p2)

                MOVF        AEXP,W
                MOVWF       CEXP            ; save z2 in CARG
                MOVF        AARGB0,W
                MOVWF       CARGB0
                MOVF        AARGB1,W
                MOVWF       CARGB1
                MOVF        AARGB2,W
                MOVWF       CARGB2

                MOVF        DEXP,W
                MOVWF       AEXP
                MOVF        DARGB0,W
                MOVWF       AARGB0          ; restore y to AARG
                MOVF        DARGB1,W
                MOVWF       AARGB1
                MOVF        DARGB2,W
                MOVWF       AARGB2

                BSF         AARGB0,MSB

                MOVLW       0xA2            ; - p3 = -3.7747668102383613583E-8
                MOVWF       BARGB0
                MOVLW       0x20
                MOVWF       BARGB1

                CALL        FXM2416U

                MOVLW       0x19 - 1

                BTFSC       AARGB0,MSB
                GOTO        RRSINCOS32Z3OK
                RLF         AARGB3,F
                RLF         AARGB2,F
                RLF         AARGB1,F
```

```
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z3OK
                SUBWF           AEXP,F

                MOVF            CEXP,W          ; restore z2 to BARG
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32           ; z3 = z2 - y * (p3)

                MOVF            AEXP,W
                MOVWF           CEXP            ; save z3 in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            DEXP,W
                MOVWF           AEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0          ; restore y to BARG
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                BSF             BARGB0,MSB

                MOVLW           0xB4            ; - p4 = -3.77489497744597636E-8
                MOVWF           AARGB0
                MOVLW           0x61
                MOVWF           AARGB1
                MOVLW           0x1A
                MOVWF           AARGB2
                MOVLW           0x63
                MOVWF           AARGB3

                CALL            FXM3224U

                MOVLW           0x28 - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z4OK
                RLF             AARGB4,F
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z4OK
                SUBWF           AEXP,F

                CALL            RND4032

                MOVF            CEXP,W          ; restore z3 to BARG
                MOVWF           BEXP
```

```
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPA32           ; z = z3 - y * (p4)

RRSINCOS32OK
                MOVF            AEXP,W
                MOVWF           CEXP            ; save z in CARG
                MOVF            AARGB0,W
                MOVWF           CARGB0
                MOVF            AARGB1,W
                MOVWF           CARGB1
                MOVF            AARGB2,W
                MOVWF           CARGB2

                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            AEXP,W
                MOVWF           DEXP            ; save z * z in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00

RRSINCOS32ZEQX
                MOVF            CEXP,W
                MOVWF           AEXP
                MOVF            CARGB0,W
                MOVWF           AARGB0
                MOVF            CARGB1,W
                MOVWF           AARGB1
                MOVF            CARGB2,W
                MOVWF           AARGB2

                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            AARGB0,W
                MOVWF           BARGB0
                MOVF            AARGB1,W
                MOVWF           BARGB1
                MOVF            AARGB2,W
                MOVWF           BARGB2

                CALL            FPM32           ; z * z

                MOVF            AEXP,W
                MOVWF           DEXP            ; save z * z in DARG
                MOVF            AARGB0,W
                MOVWF           DARGB0
```

```
                MOVF            AARGB1,W
                MOVWF           DARGB1
                MOVF            AARGB2,W
                MOVWF           DARGB2

                RETLW           0x00

DOMERR32        BSF             FPFLAGS,DOM      ; domain error
                RETLW           0xFF

;**********************************************************************************************

ZCOS32          POL32           COS32D,2,1

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2
                DECF            BEXP,F

                CALL            FPS32

                MOVLW           EXPBIAS
                MOVWF           BEXP
                CLRF            BARGB0
                CLRF            BARGB1
                CLRF            BARGB2

                BCF             FPFLAGS,RND
                CALL            FPA32

                RETLW           0x00

ZSIN32
                POL32           SIN32D,3,1

                MOVF            DEXP,W
                MOVWF           BEXP
                MOVF            DARGB0,W
                MOVWF           BARGB0
```

```
                MOVF            DARGB1,W
                MOVWF           BARGB1
                MOVF            DARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                CALL            FPM32

                MOVF            CEXP,W
                MOVWF           BEXP
                MOVF            CARGB0,W
                MOVWF           BARGB0
                MOVF            CARGB1,W
                MOVWF           BARGB1
                MOVF            CARGB2,W
                MOVWF           BARGB2

                BCF             FPFLAGS,RND

                CALL            FPA32

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for sin(z) = z+z*(z**2)*p(z**2) on [-pi/4,pi/4]

SIN32D0         EQU             0x7C            ; SIN32D0 = -1.666666664079712E-1
SIN32D00        EQU             0xAA
SIN32D01        EQU             0xAA
SIN32D02        EQU             0xAB

SIN32D1         EQU             0x78            ; SIN32D1 = 8.333329304850749E-3
SIN32D10        EQU             0x08
SIN32D11        EQU             0x88
SIN32D12        EQU             0x84

SIN32D2         EQU             0x72            ; SIN32D2 = -1.983931227180460E-4
SIN32D20        EQU             0xD0
SIN32D21        EQU             0x07
SIN32D22        EQU             0xC0

SIN32D3         EQU             0x6C            ; SIN32D3 = 2.718121647219611E-6
SIN32D30        EQU             0x36
SIN32D31        EQU             0x68
SIN32D32        EQU             0xF9

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for cos(z) = 1 -.5*z**2 + z**4*q(z**2)
;       on [-pi/4,pi/4]

COS32D0         EQU             0x7A            ; COS32D0 = 4.166664568297614E-2
COS32D00        EQU             0x2A
COS32D01        EQU             0xAA
COS32D02        EQU             0xA5
```

```
COS32D1         EQU             0x75            ; COS32D1 = -1.388731625438419E-3
COS32D10        EQU             0xB6
COS32D11        EQU             0x06
COS32D12        EQU             0x1A

COS32D2         EQU             0x6F            ; COS32D2 = 2.443315706066392E-5
COS32D20        EQU             0x4C
COS32D21        EQU             0xF5
COS32D22        EQU             0xCE
;**********************************************************************************************
;**********************************************************************************************

;       Evaluate sqrt(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SQRT32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  SQRT( AARG )

;       Testing on [0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 7       4966    4290.2  clks

;               min     max     mean    rms
;       Error:  -0xC7   0xDF    -15.18  37.95   nsb

;----------------------------------------------------------------------------------------------

;       Range reduction for the square root function is naturally produced by
;       the floating point representation,

;               x = f * 2**e,where 1 <= f < 2,

;       leading to the expression

;                           |  sqrt(f) * 2**(e/2),e even
;              sqrt(x) =  |
;                           |  sqrt(f) * sqrt(2) * 2**(e/2),e odd

;       With f=1+z, the function sqrt(1+z) is then approximated by a
;       minimax rational function on the interval [0,1].

SQRT32
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR32

                CLRF            AARGB3          ; return if argument zero
                MOVF            AEXP,W
                BTFSC           _Z
                RETLW           0x00

                MOVF            AEXP,W          ; save exponent in CEXP
                MOVWF           CEXP

                MOVF            FPFLAGS,W       ; save RND flag in DARGB3
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVLW           EXPBIAS         ; compute z
                MOVWF           AEXP
```

```
        MOVWF           BEXP
        CLRF            BARGB0
        CLRF            BARGB1
        CLRF            BARGB2
        CALL            FPS32

        MOVF            AEXP,W          ; save z in DARG
        MOVWF           DEXP
        MOVF            AARGB0,W
        MOVWF           DARGB0
        MOVF            AARGB1,W
        MOVWF           DARGB1
        MOVF            AARGB2,W
        MOVWF           DARGB2

        POLL132         SQRT32Q,3,0     ; Q(z)

        MOVF            AEXP,W          ; save Q(z) in EARG
        MOVWF           EEXP
        MOVF            AARGB0,W
        MOVWF           EARGB0
        MOVF            AARGB1,W
        MOVWF           EARGB1
        MOVF            AARGB2,W
        MOVWF           EARGB2

        MOVF            DEXP,W          ; restore z
        MOVWF           AEXP
        MOVF            DARGB0,W
        MOVWF           AARGB0
        MOVF            DARGB1,W
        MOVWF           AARGB1
        MOVF            DARGB2,W
        MOVWF           AARGB2

        POL32           SQRT32P,2,0     ; P(z)

        MOVF            EEXP,W
        MOVWF           BEXP
        MOVF            EARGB0,W
        MOVWF           BARGB0
        MOVF            EARGB1,W
        MOVWF           BARGB1
        MOVF            EARGB2,W
        MOVWF           BARGB2

        CALL            FPD32           ; P(z)/Q(z)

        MOVF            DEXP,W          ; restore z
        MOVWF           BEXP
        MOVF            DARGB0,W
        MOVWF           BARGB0
        MOVF            DARGB1,W
        MOVWF           BARGB1
        MOVF            DARGB2,W
        MOVWF           BARGB2

        CALL            FPM32           ; z*P(z)/Q(z)
        MOVLW           EXPBIAS
        MOVWF           BEXP
        CLRF            BARGB0
        CLRF            BARGB1
        CLRF            BARGB2

        CALL            FPA32           ; sqrt(1+z)=1+z*P(z)/Q(z)
```



```
SQRT32OK
                BTFSC           CEXP,LSB        ; is CEXP even or odd?
                GOTO            RRSQRTOK32

;       fixed point multiplication by sqrt(2)

                BSF             AARGB0,MSB

                MOVLW           0xB5            ; sqrt(2) = 1.41421356237
                MOVWF           BARGB0
                MOVLW           0x04
                MOVWF           BARGB1
                MOVLW           0xF3
                MOVWF           BARGB2
                MOVLW           0x33
                MOVWF           BARGB3

                CALL            FXM3232U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSQRTOK32
                RLF             AARGB4,F
                RLF             AARGB3,F
                RLF             AARGB2,F
                RLF             AARGB1,F
                RLF             AARGB0,F
                DECF            AEXP,F

RRSQRTOK32
                BCF             AARGB0,MSB      ; make  MSB implicit

                MOVLW           EXPBIAS         ; divide exponent by two
                ADDWF           CEXP,F
                RRF             CEXP,W
                MOVWF           AEXP

                BTFSS           DARGB3,RND
                RETLW           0x00
                BSF             FPFLAGS,RND
                CALL            RND4032
                RETLW           0x00

DOMERR32        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;-----------------------------------------------------------------------------------------------

;       minimax rational coefficients for (sqrt(1+z)-1)/z on [0,1]

SQRT32P0        EQU             0x84            ; SQRT32P0 = 6.054736157E1
SQRT32P00       EQU             0x72
SQRT32P01       EQU             0x30
SQRT32P02       EQU             0x80

SQRT32P1        EQU             0x84            ; SQRT32P1 = 5.154073142E1
SQRT32P10       EQU             0x4E
SQRT32P11       EQU             0x29
SQRT32P12       EQU             0xB5

SQRT32P2        EQU             0x81            ; SQRT32P2 = 7.370062896E0
SQRT32P20       EQU             0x6B
SQRT32P21       EQU             0xD7
SQRT32P22       EQU             0x8E
```

```
SQRT32Q0        EQU             0x85            ; SQRT32Q0 = 1.210947497E2
SQRT32Q00       EQU             0x72
SQRT32Q01       EQU             0x30
SQRT32Q02       EQU             0x83

SQRT32Q1        EQU             0x86            ; SQRT32Q1 = 1.333554439E2
SQRT32Q10       EQU             0x05
SQRT32Q11       EQU             0x5A
SQRT32Q12       EQU             0xBC

SQRT32Q2        EQU             0x84            ; SQRT32Q2 = 3.294831307E1
SQRT32Q20       EQU             0x03
SQRT32Q21       EQU             0xCB
SQRT32Q22       EQU             0x13

SQRT32Q3        EQU             0x7F            ; SQRT32Q3 = 1.0
SQRT32Q30       EQU             0x00
SQRT32Q31       EQU             0x00
SQRT32Q32       EQU             0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A < B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TALTB32

;       Output: logical result in W

;       Result: if A < B TRUE, W = 0x01
;               if A < B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 59      34      15.4    clks

TALTB32         MOVF            AARGB0,W
                XORWF           BARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TALTB32O

                BTFSC           AARGB0,MSB
                GOTO            TALTB32N

TALTB32P        MOVF            AEXP,W
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
```

```
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB2,W
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB32N        MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB2,W
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB32O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A <= B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TALEB32

;       Output: logical result in W

;       Result: if A <= B TRUE, W = 0x01
;               if A <= B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 9       32      15.1    clks
```

```
TALEB32     MOVF     AARGB0,W
            XORWF    BARGB0,W
            MOVWF    TEMPB0
            BTFSC    TEMPB0,MSB
            GOTO     TALEB32O

            BTFSC    AARGB0,MSB
            GOTO     TALEB32N

TALEB32P    MOVF     AEXP,W
            SUBWF    BEXP,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     AARGB0,W
            SUBWF    BARGB0,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     AARGB1,W
            SUBWF    BARGB1,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     AARGB2,W
            SUBWF    BARGB2,W
            BTFSS    _C
            RETLW    0x00
            RETLW    0x01

TALEB32N    MOVF     BEXP,W
            SUBWF    AEXP,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     BARGB0,W
            SUBWF    AARGB0,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     BARGB1,W
            SUBWF    AARGB1,W
            BTFSS    _C
            RETLW    0x00
            BTFSS    _Z
            RETLW    0x01

            MOVF     BARGB2,W
            SUBWF    AARGB2,W
            BTFSS    _C
            RETLW    0x00
            RETLW    0x01
```

```
TALEB32O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A > B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAGTB32

;       Output: logical result in W

;       Result: if A > B TRUE, W = 0x01
;               if A > B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       9       34      15.4    clks

TAGTB32         MOVF            BARGB0,W
                XORWF           AARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TAGTB32O

                BTFSC           BARGB0,MSB
                GOTO            TAGTB32N

TAGTB32P        MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB2,W
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB32N        MOVF            AEXP,W
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
```



```
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB2,W
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB32O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A >= B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAGEB32

;       Output: logical result in W

;       Result: if A >= B TRUE, W = 0x01
;               if A >= B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       32      15.1    clks

TAGEB32         MOVF            BARGB0,W
                XORWF           AARGB0,W
                MOVWF           TEMPB0
                BTFSC           TEMPB0,MSB
                GOTO            TAGEB32O

                BTFSC           BARGB0,MSB
                GOTO            TAGEB32N

TAGEB32P        MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
```

```
                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB2,W
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB32N        MOVF            AEXP,W
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB0,W
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB1,W
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVF            AARGB2,W
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB32O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A == B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAEQB32

;       Output: logical result in W

;       Result: if A == B TRUE, W = 0x01
;               if A == B FALSE, W = 0x00
```

```
;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       5       18      7.4     clks

TAEQB32         MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _Z
                RETLW           0x00

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                RETLW           0x00

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                RETLW           0x00

                MOVF            BARGB2,W
                SUBWF           AARGB2,W
                BTFSS           _Z
                RETLW           0x00
                RETLW           0x01
;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A =! B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TANEB32

;       Output: logical result in W

;       Result: if A =! B TRUE, W = 0x01
;               if A =! B FALSE, W = 0x00

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 5       18      7.4     clks

TANEB32         MOVF            BEXP,W
                SUBWF           AEXP,W
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _Z
                RETLW           0x01

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _Z
                RETLW           0x01
```

```
                MOVF            BARGB2,W
                SUBWF           AARGB2,W
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

;***********************************************************************************************
;***********************************************************************************************
```

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX E: PIC17CXXX 24-BIT ELEMENTARY FUNCTION LIBRARY

```
;       RCS Header $Id: ef24.a17 1.55 1997/02/25 14:32:22 F.J.Testa Exp $

;       $Revision: 1.55 $

;       PIC17 24-BIT ELEMENTARY FUNCTION LIBRARY

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Test statistics are typically from 100000 trials, with timing in cycles
;       and error in the next significant byte. In all cases, the floating point
;       routines satisfy a half unit in the last position (.5*ulp) accuracy
;       requirement, resulting in |nsb error| <= 0x7F.  The integer and logical
;       routines are exact.

;       Routine  Function         Timing in cycles         Error in nsb
;                                 min      max      mean     min      max      mean     rms

;       SQRT24    24 bit sqrt(x)  6        327      292.7    -0x10    0x05     -3.56    5.20

;       EXP24     24 bit exp(x)   645      999      859.3    -0x6E    0x69     -0.97    35.75

;       EXP1024   24 bit exp10(x) 646      1002     859.5    -0x75    0x77     -0.94    40.34

;       LOG24     24 bit log(x)   12       1442     1316.5   -0x02    0x00     -0.81    0.92

;       LOG1024   24 bit log10(x) 12       1457     1317.7   -0x01    0x00     -0.32    0.57

;       SIN24     24 bit sin(x)   834      1625     1465.7   -0x56    0x13     -7.12    20.89

;       COS24     24 bit cos(x)   942      1637     1465.7   -0x56    0x13     -7.13    20.90

;       SINCOS24 24 bit sin(x),cos(x) 15162248    2128.2   -0x56    0x13     -7.12    20.89
;                                                          -0x56    0x13     -7.13    20.90

;       POW24    24 bit pow(x,y)=x**y

;       FLOOR24  24 bit floor(x)  18       39       30.11    0x00     0x00     0.0      0.0

;-----------------------------------------------------------------------------------------------

;       TALTB24  24 bit A < B     8        27       11.5

;       TALEB24  24 bit A <= B    8        25       11.5

;       TAGTB24  24 bit A > B     8        27       11.5

;       TAGEB24  24 bit A >= B    8        25       11.5

;       TAEQB24  24 bit A == B    4        11       6.0

;       TANEB24  24 bit A != B    4        11       6.0

;***********************************************************************************************
;***********************************************************************************************
;
;       24 bit floating point representation
;
;       EXPONENT        8 bit biased exponent
;
;                       It is important to note that the use of biased exponents produces
```

```
;                        a unique representation of a floating point 0, given by
;                        EXP = HIGHBYTE = LOWBYTE = 0x00, with 0 being the only
;                        number with EXP = 0.
;
;       HIGHBYTE         8 bit most significant byte of fraction in sign-magnitude representation,
;                        with SIGN = MSB, implicit MSB = 1 and radix point to the right of MSB
;
;       LOWBYTE          8 bit least significant byte of sign-magnitude fraction
;
;       EXPONENT         HIGHBYTE         LOWBYTE
;
;       xxxxxxxx         S.xxxxxxx        xxxxxxxx
;
;                         |
;                       RADIX
;                       POINT

;**********************************************************************************************
;**********************************************************************************************

;       polynomial evaluation macros

POLL124  macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       with leading coefficient of one, and where AARG is assumed have been saved
;       in DARG when N > 1.  The result is in AARG.

;        ROUND = 0no rounding is enabled; can be previously enabled
;        ROUND = 1rounding is enabled
;        ROUND = 2rounding is enabled then disabled before last add
;        ROUND = 3rounding is assumed disabled then enabled before last add
;        ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear

         local    i,j
         variable i = N, j = 0

         variable i = i - 1

         if       ROUND == 1  ||  ROUND == 2

                  BSF              FPFLAGS,RND

         endif

                  MOVLW            COF#v(i)
                  MOVWF            BEXP

         variable j = 0

         while    j <= 2

                  MOVLW            COF#v(i)#v(j)
                  MOVWF            BARGB#v(j)

         variable j = j + 1

         endw

                  CALL             FPA32

         variable i = i - 1

         while    i >= 0
```

```
                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while   j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

        if      i == 0

                if      ROUND == 2

                BCF             FPFLAGS,RND

                endif

                if      ROUND == 3

                BSF             FPFLAGS,RND

                endif

                if      ROUND == 4

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                endif

                if      ROUND == 5

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND

                endif

        endif

                CALL            FPA32

        variable i = i - 1

        endw

        endm

POL24   macro           COF,N,ROUND
```

```
;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       and where AARG is assumed have been be saved in DARG when N > 1.
;       The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set

        local    i,j
        variable i = N, j = 0

        if       ROUND == 1  ||  ROUND == 2

                 BSF              FPFLAGS,RND

        endif

                 MOVLW            COF#v(i)
                 MOVWF            BEXP

        while    j <= 2

                 MOVLW            COF#v(i)#v(j)
                 MOVWF            BARGB#v(j)

        variable j = j + 1

        endw

                 CALL             FPM32

        variable i = i - 1

                 MOVLW            COF#v(i)
                 MOVWF            BEXP

        variable j = 0

        while    j <= 2

                 MOVLW            COF#v(i)#v(j)
                 MOVWF            BARGB#v(j)

        variable j = j + 1

        endw

                 CALL             FPA32

        variable i = i - 1

        while    i >= 0

                 MOVFP            DEXP,WREG
                 MOVPF            WREG,BEXP
                 MOVFP            DARGB0,WREG
                 MOVPF            WREG,BARGB0
                 MOVFP            DARGB1,WREG
                 MOVPF            WREG,BARGB1
                 MOVFP            DARGB2,WREG
                 MOVPF            WREG,BARGB2
```

```
        CALL            FPM32

        MOVLW           COF#v(i)
        MOVWF           BEXP

variable j = 0

while   j <= 2

        MOVLW           COF#v(i)#v(j)
        MOVWF           BARGB#v(j)

variable j = j + 1

endw

if      i == 0

        if      ROUND == 2

        BCF             FPFLAGS,RND

        endif

        if      ROUND == 3

        BSF             FPFLAGS,RND

        endif

        if      ROUND == 4

        BTFSS           DARGB3,RND
        BCF             FPFLAGS,RND

        endif

        if      ROUND == 5

        BTFSC           DARGB3,RND
        BSF             FPFLAGS,RND

        endif

endif

        CALL            FPA32

variable i = i - 1

endw

endm
;**********************************************************************************************
;**********************************************************************************************

;       Evaluate exp(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    EXP24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1
```

```
;       Result:  AARG  <--  EXP( AARG )

;       Testing on [MINLOG,MAXLOG] from 100000 trials:

;               min      max      mean
;       Timing: 645      999      859.3    clks

;               min      max      mean     rms
;       Error:  -0x6E    0x69     -0.97    35.75    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp(x) = e**x = 2**(x/log(2)) = 2**z * 2**n,

;                       x/log(2) = z + n,

;       where 0 <= z < 1 and n is an integer, evaluated during range reduction.
;       Segmented third degree minimax polynomial approximations are used to
;       estimate 2**z on the intervals [0,.25], [.25,.5], [.5,.75] and [.75,1].

EXP24
                MOVLW           0x66            ; test for |x| < 2**(-24)/2
                CPFSGT          EXP
                GOTO            EXP24ONE        ; return e**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP24
TPEXP24
                MOVFP           AEXP,WREG       ; positive domain check
                SUBLW           MAXLOG24EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MAXLOG24B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MAXLOG24B1
                BTFSS           _C
                GOTO            DOMERR24
                GOTO            EXP24ARGOK

TNEXP24
                MOVFP           AEXP,WREG       ; negative domain check
                SUBLW           MINLOG24EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MINLOG24B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP24ARGOK
```

```
                MOVFP           AARGB1,WREG
                SUBLW           MINLOG24B1
                BTFSS           _C
                GOTO            DOMERR24

EXP24ARGOK
                MOVFP           FPFLAGS,WREG
                MOVWF           DARGB3          ; save rounding flag

                BCF             FPFLAGS,RND     ; disable rounding

                CALL            RREXP24         ; range reduction

                MOVLW           0x7E
                CPFSEQ          AEXP
                GOTO            EXP24L

EXP24H          BTFSS           AARGB0,MSB-1
                GOTO            EXP24HL

                POL24           EXP24HH,3,0     ; minimax approximation on [.75,1]

                GOTO            EXP24OK

EXP24HL         POL24           EXP24HL,3,0     ; minimax approximation on [.5,.75]

                GOTO            EXP24OK

EXP24L          MOVLW           0x7D
                CPFSEQ          AEXP
                GOTO            EXP24LL

                POL24           EXP24LH,3,0     ; minimax approximation on [.25,.5]

                GOTO            EXP24OK

EXP24LL
                POL24           EXP24LL,3,0     ; minimax approximation on [0,.25]

EXP24OK
                MOVFP           EARGB3,WREG
                ADDWF           AEXP,F

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

EXP24ONE        MOVLW           EXPBIAS         ; return e**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                RETLW           0x00

DOMERR24        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF

;**********************************************************************************************

;       Range reduction routine for the exponential function

;               x/log(2) = z + n
```

```
RREXP24
                MOVPF           AARGB0,DARGB0    ; save sign
                BSF             AARGB0,MSB       ; make MSB explicit

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1

                MOVLW           0xB8             ; 1/ln(2) = 1.44269504089
                MOVPF           WREG,AARGB0
                MOVLW           0xAA
                MOVPF           WREG,AARGB1
                MOVLW           0x3B
                MOVPF           WREG,AARGB2

                CALL            FXM2416U         ; x * (1/ln2)

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP24YOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP24YOK      BTFSS           DARGB0,MSB       ; restore sign
                BCF             AARGB0,MSB

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP        ; save x/ln2 in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                CALL            FLOOR24

                MOVFP           AEXP,WREG
                MOVPF           WREG,BEXP        ; save float(n) in BARG
                BTFSC           _Z
                GOTO            RREXP24ZOK       ; done if n = 0
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                CLRF            BARGB2,F

                CALL            INT2416          ; n = [ x * (1/ln2) ]

                MOVPF           AARGB1,EARGB3    ; save n in EARG

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CALL            FPS32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP        ; save z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00
```



```
RREXP24ZOK
                MOVFP           DEXP,WREG
                MOVWF           AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CLRF            EARGB3,F

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       third degree minimax polynomial coefficients for 2**(x) on [.75,1]

EXP24HH0        EQU             0x7E            ; EXP24HH0 = .99103284632
EXP24HH00       EQU             0x7D
EXP24HH01       EQU             0xB4
EXP24HH02       EQU             0x54

EXP24HH1        EQU             0x7E            ; EXP24HH1 = .73346850266
EXP24HH10       EQU             0x3B
EXP24HH11       EQU             0xC4
EXP24HH12       EQU             0x97

EXP24HH2        EQU             0x7C            ; EXP24HH2 = .17374128273
EXP24HH20       EQU             0x31
EXP24HH21       EQU             0xE9
EXP24HH22       EQU             0x3C

EXP24HH3        EQU             0x7B            ; EXP24HH3 = .10175678143
EXP24HH30       EQU             0x50
EXP24HH31       EQU             0x65
EXP24HH32       EQU             0xDC

;       third degree minimax polynomial coefficients for 2**(x) on [.5,.75]

EXP24HL0        EQU             0x7E            ; EXP24HL0 = .99801686089
EXP24HL00       EQU             0x7F
EXP24HL01       EQU             0x7E
EXP24HL02       EQU             0x08

EXP24HL1        EQU             0x7E            ; EXP24HL1 = .70586404164
EXP24HL10       EQU             0x34
EXP24HL11       EQU             0xB3
EXP24HL12       EQU             0x81

EXP24HL2        EQU             0x7C            ; EXP24HL2 = .21027360637
EXP24HL20       EQU             0x57
EXP24HL21       EQU             0x51
EXP24HL22       EQU             0xF7

EXP24HL3        EQU             0x7B            ; EXP24HL3 = .85566912730E-1
EXP24HL30       EQU             0x2F
EXP24HL31       EQU             0x3D
EXP24HL32       EQU             0xB5

;       third degree minimax polynomial coefficients for 2**(x) on [.25,.5]

EXP24LH0        EQU             0x7E            ; EXP24LH0 = .99979384559
EXP24LH00       EQU             0x7F
EXP24LH01       EQU             0xF2
EXP24LH02       EQU             0x7D

EXP24LH1        EQU             0x7E            ; EXP24LH1 = .69545887384
EXP24LH10       EQU             0x32
```

```
EXP24LH11       EQU             0x09
EXP24LH12       EQU             0x98

EXP24LH2        EQU             0x7C            ; EXP24LH2 = .23078300446
EXP24LH20       EQU             0x6C
EXP24LH21       EQU             0x52
EXP24LH22       EQU             0x61

EXP24LH3        EQU             0x7B            ; EXP24LH3 = .71952910179E-1
EXP24LH30       EQU             0x13
EXP24LH31       EQU             0x5C
EXP24LH32       EQU             0x0C

;       third degree minimax polynomial coefficients for 2**(x) on [0,.25]

EXP24LL0        EQU             0x7E            ; EXP24LL0 = .99999970657
EXP24LL00       EQU             0x7F
EXP24LL01       EQU             0xFF
EXP24LL02       EQU             0xFB

EXP24LL1        EQU             0x7E            ; EXP24LL1 = .69318585159
EXP24LL10       EQU             0x31
EXP24LL11       EQU             0x74
EXP24LL12       EQU             0xA1

EXP24LL2        EQU             0x7C            ; EXP24LL2 = .23944330933
EXP24LL20       EQU             0x75
EXP24LL21       EQU             0x30
EXP24LL22       EQU             0xA0

EXP24LL3        EQU             0x7A            ; EXP24LL3 = .60504944237E-1
EXP24LL30       EQU             0x77
EXP24LL31       EQU             0xD4
EXP24LL32       EQU             0x08

;**********************************************************************************************

;       Evaluate exp10(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    EXP1024

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  EXP10( AARG )

;       Testing on [MINLOG10,MAXLOG10] from 10000 trials:

;               min     max     mean
;       Timing: 646     1002    859.5   clks

;               min     max     mean    rms
;       Error:  -0x75   0x77    -0.94   40.34   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the base 10 exponential function is based upon the
;       expansion

;               exp10(x) = 10**x = 2**(x/log10(2)) = 2**z * 2**n

;                       x/log10(2) = z + n,
```

```
;        where 0 <= z < 1 and n is an integer, evaluated during range reduction.
;        Segmented third degree minimax polynomial approximations are used to
;        estimate 2**z on the intervals [0,.25], [.25,.5], [.5,.75] and [.75,1].

EXP1024
                MOVLW           0x66            ; test for |x| < 2**(-24)/2
                CPFSGT          EXP
                GOTO            EXP1024ONE      ; return 10**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP1024
TPEXP1024
                MOVFP           AEXP,WREG       ; positive domain check
                SUBLW           MAXLOG1024EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MAXLOG1024B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MAXLOG1024B1
                BTFSS           _C
                GOTO            DOMERR24
                GOTO            EXP1024ARGOK

TNEXP1024
                MOVFP           AEXP,WREG       ; negative domain check
                SUBLW           MINLOG1024EXP
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MINLOG1024B0
                BTFSS           _C
                GOTO            DOMERR24
                BTFSS           _Z
                GOTO            EXP1024ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MINLOG1024B1
                BTFSS           _C
                GOTO            DOMERR24

EXP1024ARGOK
                MOVFP           FPFLAGS,WREG
                MOVWF           DARGB3          ; save rounding flag

                BCF             FPFLAGS,RND     ; disable rounding

                CALL            RREXP1024       ; range reduction

                MOVLW           0x7E
                CPFSEQ          AEXP
                GOTO            EXP1024L

EXP1024H        BTFSS           AARGB0,MSB-1
                GOTO            EXP1024HL
```

```
                POL24           EXP24HH,3,0     ; minimax approximation on [.75,1]

                GOTO            EXP1024OK

EXP1024HL       POL24           EXP24HL,3,0     ; minimax approximation on [.5,.75]

                GOTO            EXP1024OK

EXP1024L        MOVLW           0x7D
                CPFSEQ          AEXP
                GOTO            EXP1024LL

                POL24           EXP24LH,3,0     ; minimax approximation on [.25,.5]

                GOTO            EXP1024OK

EXP1024LL       POL24           EXP24LL,3,0     ; minimax approximation on [0,.25]


EXP1024OK
                MOVFP           EARGB3,WREG
                ADDWF           AEXP,F

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

EXP1024ONE      MOVLW           EXPBIAS         ; return 10**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for the exponential function

;               x/log10(2) = z + n

RREXP1024
                MOVPF           AARGB0,DARGB0
                BSF             AARGB0,MSB

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1

                MOVLW           0xD4            ; 1/log10(2) = 3.32192809489
                MOVPF           WREG,AARGB0
                MOVLW           0x9A
                MOVPF           WREG,AARGB1
                MOVLW           0x78
                MOVPF           WREG,AARGB2

                CALL            FXM2416U        ; x * (1/log10(2))

                INCF            AEXP,F
                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP24YOK
                RLCF            AARGB3,F
```



```
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP1024YOK    BTFSS           DARGB0,MSB      ; restore sign
                BCF             AARGB0,MSB

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save x/log10(2) in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                CALL            FLOOR24

                MOVFP           AEXP,WREG
                MOVPF           WREG,BEXP       ; save float(n) in BARG
                BTFSC           _Z
                GOTO            RREXP1024ZOK    ; done if n = 0
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                CLRF            BARGB2,F

                CALL            INT2416         ; n = [ x * (1/log10(2)) ]

                MOVPF           AARGB1,EARGB3   ; save n in EARG

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CALL            FPS32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00

RREXP1024ZOK
                MOVFP           DEXP,WREG
                MOVWF           AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CLRF            EARGB3,F

                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate log(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    LOG24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1
```

```
;       Result:  AARG  <--  LOG( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 12      1442    1316.5   clks

;               min     max     mean     rms
;       Error:  -0x02   0x00    -0.81    0.92     nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log(x) = log(2) * log2(x) = log(2) * ( n + log2(f) )

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |    2*f-1, f < 1/sqrt(2), n=n-1
;               z = |
;                   |    f-1, otherwise

;       produces a naturally segmented representation of log2(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.

LOG24
                CLRF            AARGB2,W        ; clear next significant byte
                BTFSS           AARGB0,MSB      ; test for negative argument
                CPFSGT          AEXP            ; test for zero argument
                GOTO            DOMERR24

                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVFP           AEXP,WREG
                MOVPF           WREG,EARGB3
                MOVLW           EXPBIAS-1
                SUBWF           EARGB3,F
                MOVWF           AEXP

                MOVLW           0xF3            ; .70710678118655 = 7E3504F3
                SUBWF           AARGB2,W
                MOVLW           0x04
                SUBWFB          AARGB1,W
                MOVLW           0x35
                SUBWFB          AARGB0,W

                BTFSS           _C
                GOTO            LOG24L

;       minimax rational approximation on [0,.sqrt(2)-1]

LOG24H
                MOVLW           0x7F
                MOVPF           WREG,BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPS32

                MOVFP           AEXP,WREG
```



```
            MOVPF       WREG,DEXP
            MOVPF       AARGB0,DARGB0
            MOVPF       AARGB1,DARGB1
            MOVPF       AARGB2,DARGB2

            POLL124     LOG24HQ,2,0

            MOVFP       AEXP,WREG
            MOVPF       WREG,CEXP
            MOVPF       AARGB0,CARGB0
            MOVPF       AARGB1,CARGB1
            MOVPF       AARGB2,CARGB2

            MOVFP       DEXP,WREG
            MOVPF       WREG,AEXP
            MOVFP       DARGB0,AARGB0
            MOVFP       DARGB1,AARGB1
            MOVFP       DARGB2,AARGB2

            POL24       LOG24HP,1,0

            MOVFP       CEXP,WREG
            MOVPF       WREG,BEXP
            MOVFP       CARGB0,WREG
            MOVPF       WREG,BARGB0
            MOVFP       CARGB1,WREG
            MOVPF       WREG,BARGB1
            MOVFP       CARGB2,WREG
            MOVPF       WREG,BARGB2

            CALL        FPD32

            GOTO        LOG24OK

;       minimax rational approximation on [1/sqrt(2)-1,0]

LOG24L
            INCF        AEXP,F
            MOVLW       0x7F
            MOVPF       WREG,BEXP
            CLRF        BARGB0,F
            CLRF        BARGB1,F
            CLRF        BARGB2,F

            CALL        FPS32

            DECF        EARGB3,F

            MOVFP       AEXP,WREG
            MOVPF       WREG,DEXP
            MOVPF       AARGB0,DARGB0
            MOVPF       AARGB1,DARGB1
            MOVPF       AARGB2,DARGB2

            POLL124     LOG24LQ,2,0

            MOVFP       AEXP,WREG
            MOVPF       WREG,CEXP
            MOVPF       AARGB0,CARGB0
            MOVPF       AARGB1,CARGB1
            MOVPF       AARGB2,CARGB2

            MOVFP       DEXP,WREG
            MOVPF       WREG,AEXP
            MOVFP       DARGB0,AARGB0
            MOVFP       DARGB1,AARGB1
```

```
                MOVFP           DARGB2,AARGB2

                POL24           LOG24LP,1,0

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPD32

LOG24OK
                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                CLRF            AARGB0,F
                MOVFP           EARGB3,AARGB1
                BTFSC           AARGB1,MSB
                SETF            AARGB0,F
                CALL            FLO1624
                CLRF            AARGB2,F

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

;       fixed point multiplication by log(2)

                MOVPF           AARGB0,EARGB3
                BSF             AARGB0,MSB

                MOVLW           0xB1
                MOVPF           WREG,BARGB0
                MOVLW           0x72
                MOVPF           WREG,BARGB1
                MOVLW           0x18
                MOVPF           WREG,BARGB2

                CALL            FXM2424U

                BTFSC           AARGB0,MSB
```



```
                GOTO            LOG24DONE
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

LOG24DONE       BTFSS           EARGB3,MSB
                BCF             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+z)/z on [1/sqrt(2)-1,0]

LOG24HP0        EQU             0x81            ; LOG24HP0 = .73551298732E+1
LOG24HP00       EQU             0x6B
LOG24HP01       EQU             0x5D
LOG24HP02       EQU             0x39

LOG24HP1        EQU             0x81            ; LOG24HP1 = .40900513905E+1
LOG24HP10       EQU             0x02
LOG24HP11       EQU             0xE1
LOG24HP12       EQU             0xB3

LOG24HQ0        EQU             0x81            ; LOG24HQ0 = .50982159260E+1
LOG24HQ00       EQU             0x23
LOG24HQ01       EQU             0x24
LOG24HQ02       EQU             0x96

LOG24HQ1        EQU             0x81            ; LOG24HQ1 = .53849258895E+1
LOG24HQ10       EQU             0x2C
LOG24HQ11       EQU             0x51
LOG24HQ12       EQU             0x50

LOG24HQ2        EQU             0x7F            ; LOG24HQ2 = 1.0
LOG24HQ20       EQU             0x00
LOG24HQ21       EQU             0x00
LOG24HQ22       EQU             0x00

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+z)/z on [0,.sqrt(2)-1]

LOG24LP0        EQU             0x82            ; LOG24LP0 = .103115556038E+2
LOG24LP00       EQU             0x24
LOG24LP01       EQU             0xFC
LOG24LP02       EQU             0x22

LOG24LP1        EQU             0x81            ; LOG24LP1 = .457749066375E+1
LOG24LP10       EQU             0x12
LOG24LP11       EQU             0x7A
LOG24LP12       EQU             0xCE

LOG24LQ0        EQU             0x81            ; LOG24LQ0 = .714746549793E+1
LOG24LQ00       EQU             0x64
LOG24LQ01       EQU             0xB8
LOG24LQ02       EQU             0x0A

LOG24LQ1        EQU             0x81            ; LOG24LQ1 = .674551124538E+1
```

```
LOG24LQ10       EQU             0x57
LOG24LQ11       EQU             0xDB
LOG24LQ12       EQU             0x3A

LOG24LQ2        EQU             0x7F            ; LOG24LQ2 = 1.0
LOG24LQ20       EQU             0x00
LOG24LQ21       EQU             0x00
LOG24LQ22       EQU             0x00

;**********************************************************************************************

;       Evaluate log10(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    LOG1024

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  LOG( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 12      1457    1317.7  clks

;               min     max     mean    rms
;       Error:  -0x01   0x00    -0.32   0.57    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log10(x) = log10(2) * log2(x) = log10(2) * ( n + log2(f) )

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |   2 * f - 1, f < 1/sqrt(2), n = n - 1
;               z = |
;                   |   f - 1, otherwise

;       produces a naturally segmented representation of log2(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.

LOG1024
                CLRF            AARGB2,W        ; clear next significant byte
                BTFSS           AARGB0,MSB      ; test for negative argument
                CPFSGT          AEXP            ; test for zero argument
                GOTO            DOMERR24

                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVFP           AEXP,WREG
                MOVPF           WREG,EARGB3
                MOVLW           EXPBIAS-1
                SUBWF           EARGB3,F
                MOVWF           AEXP

                MOVLW           0xF3            ; .70710678118655 = 7E3504F3
                SUBWF           AARGB2,W
                MOVLW           0x04
```

```
                SUBWFB          AARGB1,W
                MOVLW           0x35
                SUBWFB          AARGB0,W

                BTFSS           _C
                GOTO            LOG1024L

;       minimax rational approximation on [0,.sqrt(2)-1]

LOG1024H
                MOVLW           0x7F
                MOVPF           WREG,BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPS32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                POLL124         LOG24HQ,2,0

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                POL24           LOG24HP,1,0

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPD32

                GOTO            LOG1024OK

;       minimax rational approximation on [1/sqrt(2)-1,0]

LOG1024L
                INCF            AEXP,F
                MOVLW           0x7F
                MOVPF           WREG,BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPS32

                DECF            EARGB3,F
```

```
                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                POLL124         LOG24LQ,2,0

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                POL24           LOG24LP,1,0

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPD32

LOG1024OK
                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                CLRF            AARGB0,F
                MOVFP           EARGB3,AARGB1
                BTFSC           AARGB1,MSB
                SETF            AARGB0,F
                CALL            FLO1624
                CLRF            AARGB2,F

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
```

```
                MOVPF           WREG,BARGB2

                CALL            FPA32

;       fixed point multiplication by log10(2)

                MOVPF           AARGB0,EARGB3
                BSF             AARGB0,MSB

                MOVLW           0x9A
                MOVPF           WREG,BARGB0
                MOVLW           0x20
                MOVPF           WREG,BARGB1
                MOVLW           0x9B
                MOVPF           WREG,BARGB2

                CALL            FXM2424U
                DECF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            LOG1024DONE
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

LOG1024DONE     BTFSS           EARGB3,MSB
                BCF             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate cos(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    COS24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  COS( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 912     1618    1458.6  clks

;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.05   0.00    nsb

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations
```

```
;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations.

COS24
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                CLRF            CARGB3,F        ; initialize sign in CARGB3

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS24
RRCOS24OK
                RRCF            EARGB3,W
                XORWF           EARGB3,W
                BTFSC           WREG,LSB
                GOTO            COSZSIN24

                CALL            ZCOS24

                GOTO            COSSIGN24

COSZSIN24       CALL            ZSIN24

COSSIGN24
                BTFSC           EARGB3,LSB+1
                BTG             CARGB3,MSB

                BTFSC           CARGB3,MSB
                BTG             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;**********************************************************************************************

;       Evaluate sin(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SIN24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 942     1637    1465.7  clks

;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.13   20.90   nsb

;----------------------------------------------------------------------------------------------
```

```
;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations.

SIN24
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding
                CLRF            CARGB3,F        ; initialize sign in CARGB3

                BTFSC           AARGB0,MSB      ; toggle sign if x < 0
                BSF             CARGB3,MSB

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS24
RRSIN24OK
                RRCF            EARGB3,W
                XORWF           EARGB3,W
                BTFSC           WREG,LSB
                GOTO            SINZCOS24

                CALL            ZSIN24
                GOTO            SINSIGN24

SINZCOS24       CALL            ZCOS24

SINSIGN24       BTFSC           CARGB3,MSB
                BTG             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND3224
                RETLW           0x00

;**********************************************************************************************

;       Evaluate sin(x) and cos(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SINCOS24

;       Output: 24 bit floating point numbers in AEXP, AARGB0, AARGB1 and
;               BEXP, BARGB0, BARGB1

;       Result: AARG  <--  COS( AARG )
;               BARG  <--  SIN( AARG )
```

```
;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 1516    2248    2128.2  clks

;               min     max     mean    rms
;       Error:  -0x56   0x13    -7.12   20.89   nsb     sine
;               -0x56   0x13    -7.13   20.90           cosine

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;               sin(z) = z * p(z**2),cos(z) = q(z**2)

;       where p and q are minimax polynomial approximations. In this case,
;       only one range reduction is necessary.

SINCOS24
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVFP           AEXP,WREG       ; save x in EARG
                MOVWF           EEXP
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                CLRF            EARGB2,F

                BCF             AARGB0,MSB      ; use |x|

                CLRF            CARGB3,F        ; initialize sign in CARGB3

                CALL            RRSINCOS24      ; range reduction

                MOVFP           CARGB3,WREG     ; save sign from range reduction
                MOVWF           ZARGB3

                BTFSC           EARGB0,MSB      ; toggle sign if x < 0
                BTG             CARGB3,MSB

                CALL            RRSIN24OK

                BTFSC           DARGB3,RND
                CALL            RND3224

                MOVFP           AEXP,WREG       ; save sin(x) in EARG
                MOVWF           EEXP
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                MOVFP           DEXP,WREG       ; restore z*z in AARG
                MOVWF           AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                MOVFP           ZARGB3,WREG     ; restore sign from range reduction
                MOVWF           CARGB3
```

```
            CALL            RRCOS24OK

            MOVFP           EEXP,WREG       ; restore sin(x) in BARG
            MOVPF           WREG,BEXP
            MOVFP           EARGB0,WREG
            MOVPF           WREG,BARGB0
            MOVFP           EARGB1,WREG
            MOVPF           WREG,BARGB1
            MOVFP           EARGB2,WREG
            MOVPF           WREG,BARGB2

            BTFSS           DARGB3,RND
            RETLW           0x00

            BSF             FPFLAGS,RND     ; restore rounding flag
            CALL            RND3224
            RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for trigonometric functions

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition

;               z = x mod pi/4,

;       produced by first evaluating y and j through the relations

;               y = floor(x/(pi/4)), j = y - 8*[y/8].

;       where j equals the correct octant.  For j odd, adding one to j
;       and y eliminates the odd octants.  Additional logic on j and the
;       sign of the result leads to appropriate use of the sine or cosine
;       routine in each case.

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x mod pi/4 = x - y*(pi/4) = ((x - p1*y)-p2*y)-p3*y

;       where pi/4 = p1 + p2 + p3, with p1 close to pi/4 and p2 close to
;       pi/4 - p1. The numbers p1 and p2 are chosen to have an exact
;       machine representation with slightly more than the lower half of
;       the mantissa bits zero, typically leading to no error in computing
;       the terms in parenthesis.  This calculation breaks down leading to
;       a loss of precision for |x| > LOSSTHR = sqrt(2**24)*pi/4, or for |x|
;       close to an integer multiple of pi/4. This loss threshold has been
;       chosen based on the efficacy of this calculation, with a domain error
;       reported if this threshold is exceeded.

RRSINCOS24
            MOVFP           AEXP,WREG       ; loss threshold check
            SUBLW           LOSSTHR24EXP
            BTFSS           _C
            GOTO            DOMERR24
            BTFSS           _Z
            GOTO            RRSINCOS24ARGOK

            MOVFP           AARGB0,WREG
            SUBLW           LOSSTHR24B0
            BTFSS           _C
            GOTO            DOMERR24
            BTFSS           _Z
            GOTO            RRSINCOS24ARGOK
```

```
                MOVFP           AARGB1,WREG
                SUBLW           LOSSTHR24B1
                BTFSS           _C
                GOTO            DOMERR24

RRSINCOS24ARGOK
                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP           ; save |x| in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                CLRF            CARGB2,F

;       fixed point multiplication by 4/pi

                BSF             AARGB0,MSB
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1

                MOVLW           0xA2                ; 4/pi = 1.27323954474
                MOVPF           WREG,AARGB0
                MOVLW           0xF9
                MOVPF           WREG,AARGB1
                MOVLW           0x83
                MOVPF           WREG,AARGB2

                CALL            FXM2416U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24YOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS24YOK
                BCF             AARGB0,MSB

                CALL            INT3224             ; y = [ |x| * (4/pi) ]

                BTFSS           AARGB2,LSB
                GOTO            SAVEY24
                INCF            AARGB2,F
                CLRF            WREG,F
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F

SAVEY24         MOVPF           AARGB0,DARGB0       ; save y in DARG
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVLW           0x07                ; j = y mod 8
                ANDWF           AARGB2,F

                MOVLW           0x03
                CPFSGT          AARGB2
                GOTO            JOK24
                BTG             CARGB3,MSB
                MOVLW           0x04
                SUBWF           AARGB2,F

JOK24
                MOVPF           AARGB2,EARGB3       ; save j in EARGB3
```

```
                MOVFP           DARGB0,AARGB0    ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CALL            FLO2432

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP        ; save y in DARG
                BTFSC           _Z
                GOTO            RRSINCOS24ZEQX
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

;       Cody-Waite extended precision calculation of |x| - y * pi/4 using
;       fixed point multiplication. Since y >= 1, underflow is not possible
;       in any of the products.

                BSF             AARGB0,MSB

                MOVLW           0xC9             ; - p1 = -.78515625
                MOVPF           WREG,BARGB0
                CLRF            BARGB1,F

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z1OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z1OK
                MOVFP           CEXP,WREG        ; restore x to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                CLRF            BARGB2,F

                CALL            FPA32            ; z1 = |x| - y * (p1)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP        ; save z1 in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0    ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xFD             ; - p2 = -.00024187564849853515624
                MOVPF           WREG,BARGB0
                MOVLW           0xA0
                MOVPF           WREG,BARGB1

                CALL            FXM2416U
```

```
                MOVLW           0x0D - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z2OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z2OK
                SUBWF           AEXP,F

                MOVFP           CEXP,WREG       ; restore z1 to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32           ; z2 = z1 - y * (p2)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z2 in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0   ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xA2            ; - p3 = -3.77489497744597636E-8
                MOVPF           WREG,BARGB0
                MOVLW           0x21
                MOVPF           WREG,BARGB1
                MOVLW           0x69
                MOVPF           WREG,BARGB2

                CALL            FXM2424U

                MOVLW           0x19 - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS24Z3OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS24Z3OK
                SUBWF           AEXP,F

                MOVFP           CEXP,WREG       ; restore z2 to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
```



```
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32           ; z = z2 - y * (p3)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           AEXP,WREG
                MOVPF           WREG,BEXP
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                CALL            FPM32           ; z * z

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save z * z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00

RRSINCOS24ZEQX
                MOVFP           CEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           CARGB0,AARGB0
                MOVFP           CARGB1,AARGB1
                MOVFP           CARGB2,AARGB2

                MOVFP           AEXP,WREG
                MOVPF           WREG,BEXP
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                CALL            FPM32           ; z * z

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save z * z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00

;**********************************************************************************************

;       minimax polynomial approximation p(x**2) on [0,pi/4]

ZCOS24          POL24           COS24,3,0

                RETLW           0x00

;**********************************************************************************************

;       minimax polynomial approximation x*p(x**2) on [0,pi/4]

ZSIN24          POL24           SIN24,2,0

                MOVFP           CEXP,WREG
```

```
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for sin(z)/z = p(z**2) on [0,pi/4]

SIN240          EQU             0x7E            ; LP0 = .73551298732E+1*******
SIN2400         EQU             0x7F
SIN2401         EQU             0xFF
SIN2402         EQU             0xAC

SIN241          EQU             0x7C            ; LP1 = .40900513905E+1
SIN2410         EQU             0xAA
SIN2411         EQU             0x99
SIN2412         EQU             0x9D

SIN242          EQU             0x78            ; LQ0 = .50982159260E+1
SIN2420         EQU             0x05
SIN2421         EQU             0x10
SIN2422         EQU             0x48


;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for cos(z) = q(z**2) on [0,pi/4]
;       with COS240 constrained to be 1.

COS240          EQU             0x7F            ; LP0 = .73551298732E+1*******
COS2400         EQU             0x00
COS2401         EQU             0x00
COS2402         EQU             0x00

COS241          EQU             0x7D            ; LP1 = .40900513905E+1
COS2410         EQU             0xFF
COS2411         EQU             0xFF
COS2412         EQU             0xD0

COS242          EQU             0x7A            ; LQ0 = .50982159260E+1
COS2420         EQU             0x2A
COS2421         EQU             0x9E
COS2422         EQU             0x76

COS243          EQU             0x75            ; LQ1 = .53849258895E+1
COS2430         EQU             0xB2
COS2431         EQU             0x12
COS2432         EQU             0xBF

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate sqrt(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    SQRT24
```

```
;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  SQRT( AARG )

;       Testing on [0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 6       327     292.7   clks

;               min     max     mean    rms
;       Error:  -0x10   0x05    -3.56   5.20    nsb

;----------------------------------------------------------------------------------------------

;       Range reduction for the square root function is naturally produced by
;       the floating point representation,

;               x = f * 2**e,where 1 <= f < 2,

;       leading to the expression

;                          |  sqrt(f) * 2**(e/2),e even
;               sqrt(x) =  |
;                          |  sqrt(f) * sqrt(2) * 2**(e/2),e odd

;       The approximation of sqrt(f) utilizes a table lookup of 16 bit zeroth
;       degree minimax estimates of the square root as a seed to a single
;       Newton-Raphson iteration,

;               y = (y0 + f/y0)/2,

;       where the precision of the result is guaranteed by the precision of the
;       seed and the quadratic conversion of the method.

SQRT24
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR24

                CLRF            AARGB2,W        ; return if argument zero
                CPFSGT          AEXP
                RETLW           0x00

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save x in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1

                MOVFP           FPFLAGS,WREG    ; save RND flag in DARGB3
                MOVPF           WREG,DARGB3

                BCF             FPFLAGS,RND     ; disable rounding

                MOVLW           EXPBIAS         ; initialize exponent
                MOVPF           WREG,AEXP


;       generation of y0 using 16 bit zeroth degree minimax approximations to the
;       square root of AARG, with the top 8 explicit bits of AARG as a pointer.

                MOVLW           HIGH (RATBL256M) ; access table for y0
                MOVWF           TBLPTRH
                RLCF            AARGB1,W
                RLCF            AARGB0,W
                ADDLW           LOW (RATBL256M)
                MOVWF           TBLPTRL
                BTFSC           _C
```



```
                INCF            TBLPTRH,F
                TABLRD          0,1,AARGB0
                TLRD            1,AARGB0
                TLRD            0,AARGB1

                BTFSC           CEXP,LSB        ; is CEXP even or odd?
                GOTO            RRSOK24

;       fixed point multiplication by sqrt(2)

                BSF             AARGB0,MSB      ; make MSB explicit

                MOVLW           0xB5            ; sqrt(2) = 1.41421356237
                MOVPF           WREG,BARGB0
                MOVLW           0x05
                MOVPF           WREG,BARGB1

                CALL            FXM1616U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSOK24
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSOK24
                BCF             AARGB0,MSB      ; make  MSB implicit

                MOVLW           EXPBIAS         ; divide exponent by two
                ADDWF           CEXP,W
                RRCF            WREG,F

                MOVPF           WREG,AEXP
                MOVPF           WREG,BEXP
                MOVPF           WREG,DEXP

                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1

                MOVFP           CEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           CARGB0,AARGB0
                MOVFP           CARGB1,AARGB1

                CALL            FPD24           ; Newton-Raphson iteration

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                CLRF            BARGB2,F

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            FPA32
```

```
            DECF            AEXP,F

            RETLW           0x00

;----------------------------------------------------------------------------------------------

;       Zeroth degree minimax approximations to sqrt(f), with pointer from
;       the 8 most significant explicit bits of f, the mantissa of x.

RATBL256M
            DATA    0x001F
            DATA    0x005F
            DATA    0x009F
            DATA    0x00DE
            DATA    0x011E
            DATA    0x015D
            DATA    0x019D
            DATA    0x01DC
            DATA    0x021B
            DATA    0x025A
            DATA    0x0298
            DATA    0x02D7
            DATA    0x0316
            DATA    0x0354
            DATA    0x0392
            DATA    0x03D1
            DATA    0x040F
            DATA    0x044D
            DATA    0x048B
            DATA    0x04C8
            DATA    0x0506
            DATA    0x0544
            DATA    0x0581
            DATA    0x05BE
            DATA    0x05FB
            DATA    0x0639
            DATA    0x0675
            DATA    0x06B2
            DATA    0x06EF
            DATA    0x072C
            DATA    0x0768
            DATA    0x07A5
            DATA    0x07E1
            DATA    0x081D
            DATA    0x0859
            DATA    0x0896
            DATA    0x08D1
            DATA    0x090D
            DATA    0x0949
            DATA    0x0985
            DATA    0x09C0
            DATA    0x09FC
            DATA    0x0A37
            DATA    0x0A72
            DATA    0x0AAD
            DATA    0x0AE8
            DATA    0x0B23
            DATA    0x0B5E
            DATA    0x0B99
            DATA    0x0BD3
            DATA    0x0C0E
            DATA    0x0C48
            DATA    0x0C83
            DATA    0x0CBD
            DATA    0x0CF7
            DATA    0x0D31
```

```
DATA    0x0D6B
DATA    0x0DA5
DATA    0x0DDF
DATA    0x0E18
DATA    0x0E52
DATA    0x0E8C
DATA    0x0EC5
DATA    0x0EFE
DATA    0x0F38
DATA    0x0F71
DATA    0x0FAA
DATA    0x0FE3
DATA    0x101C
DATA    0x1055
DATA    0x108D
DATA    0x10C6
DATA    0x10FE
DATA    0x1137
DATA    0x116F
DATA    0x11A7
DATA    0x11E0
DATA    0x1218
DATA    0x1250
DATA    0x1288
DATA    0x12C0
DATA    0x12F7
DATA    0x132F
DATA    0x1367
DATA    0x139E
DATA    0x13D6
DATA    0x140D
DATA    0x1444
DATA    0x147C
DATA    0x14B3
DATA    0x14EA
DATA    0x1521
DATA    0x1558
DATA    0x158E
DATA    0x15C5
DATA    0x15FC
DATA    0x1632
DATA    0x1669
DATA    0x169F
DATA    0x16D6
DATA    0x170C
DATA    0x1742
DATA    0x1778
DATA    0x17AE
DATA    0x17E4
DATA    0x181A
DATA    0x1850
DATA    0x1886
DATA    0x18BB
DATA    0x18F1
DATA    0x1927
DATA    0x195C
DATA    0x1991
DATA    0x19C7
DATA    0x19FC
DATA    0x1A31
DATA    0x1A66
DATA    0x1A9B
DATA    0x1AD0
DATA    0x1B05
DATA    0x1B3A
DATA    0x1B6F
```

```
DATA    0x1BA3
DATA    0x1BD8
DATA    0x1C0C
DATA    0x1C41
DATA    0x1C75
DATA    0x1CAA
DATA    0x1CDE
DATA    0x1D12
DATA    0x1D46
DATA    0x1D7A
DATA    0x1DAE
DATA    0x1DE2
DATA    0x1E16
DATA    0x1E4A
DATA    0x1E7D
DATA    0x1EB1
DATA    0x1EE5
DATA    0x1F18
DATA    0x1F4C
DATA    0x1F7F
DATA    0x1FB2
DATA    0x1FE6
DATA    0x2019
DATA    0x204C
DATA    0x207F
DATA    0x20B2
DATA    0x20E5
DATA    0x2118
DATA    0x214B
DATA    0x217E
DATA    0x21B0
DATA    0x21E3
DATA    0x2215
DATA    0x2248
DATA    0x227A
DATA    0x22AD
DATA    0x22DF
DATA    0x2311
DATA    0x2344
DATA    0x2376
DATA    0x23A8
DATA    0x23DA
DATA    0x240C
DATA    0x243E
DATA    0x2470
DATA    0x24A1
DATA    0x24D3
DATA    0x2505
DATA    0x2536
DATA    0x2568
DATA    0x2599
DATA    0x25CB
DATA    0x25FC
DATA    0x262E
DATA    0x265F
DATA    0x2690
DATA    0x26C1
DATA    0x26F2
DATA    0x2723
DATA    0x2754
DATA    0x2785
DATA    0x27B6
DATA    0x27E7
DATA    0x2818
DATA    0x2848
DATA    0x2879
```

```
DATA    0x28AA
DATA    0x28DA
DATA    0x290B
DATA    0x293B
DATA    0x296B
DATA    0x299C
DATA    0x29CC
DATA    0x29FC
DATA    0x2A2C
DATA    0x2A5D
DATA    0x2A8D
DATA    0x2ABD
DATA    0x2AED
DATA    0x2B1C
DATA    0x2B4C
DATA    0x2B7C
DATA    0x2BAC
DATA    0x2BDC
DATA    0x2C0B
DATA    0x2C3B
DATA    0x2C6A
DATA    0x2C9A
DATA    0x2CC9
DATA    0x2CF9
DATA    0x2D28
DATA    0x2D57
DATA    0x2D87
DATA    0x2DB6
DATA    0x2DE5
DATA    0x2E14
DATA    0x2E43
DATA    0x2E72
DATA    0x2EA1
DATA    0x2ED0
DATA    0x2EFF
DATA    0x2F2D
DATA    0x2F5C
DATA    0x2F8B
DATA    0x2FB9
DATA    0x2FE8
DATA    0x3017
DATA    0x3045
DATA    0x3074
DATA    0x30A2
DATA    0x30D0
DATA    0x30FF
DATA    0x312D
DATA    0x315B
DATA    0x3189
DATA    0x31B7
DATA    0x31E5
DATA    0x3213
DATA    0x3241
DATA    0x326F
DATA    0x329D
DATA    0x32CB
DATA    0x32F9
DATA    0x3327
DATA    0x3354
DATA    0x3382
DATA    0x33B0
DATA    0x33DD
DATA    0x340B
DATA    0x3438
DATA    0x3466
DATA    0x3493
```

```
                DATA    0x34C0
                DATA    0x34EE

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate floor(x)

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;       Use:    CALL    FLOOR24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <--  FLOOR( AARG )

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 18      39      30.11   clks

;               min     max     mean    rms
;       Error:  0x00    0x00    0.0     0.0     nsb

FLOOR24
                CLRF            AARGB2,W        ; clear next significant byte
                CPFSGT          AEXP            ; test for zero argument
                RETLW           0x00

                MOVFP           AARGB0,AARGB3   ; save mantissa
                MOVFP           AARGB1,AARGB4

                MOVLW           EXPBIAS         ; compute unbiased exponent
                SUBWF           AEXP,W
                BTFSC           WREG,MSB
                GOTO            FLOOR24ZERO

                SUBLW           0x10-1
                MOVWF           TEMPB0          ; save number of zero bits in TEMPB0

                BTFSC           WREG,LSB+3      ; divide by eight
                GOTO            FLOOR24MASKH

FLOOR24MASKL
                CLRF            TBLPTRH,F

                MOVFP           TEMPB0,WREG     ; get remainder for mask pointer
                ANDLW           0x07

                ADDLW           LOW (FLOOR24MASKTABLE)
                MOVWF           TBLPTRL
                MOVLW           HIGH (FLOOR24MASKTABLE); access table for F0
                ADDWFC          TBLPTRH,F
                TABLRD          0,1,WREG
                TLRD            0,WREG

                ANDWF           AARGB1,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVFP           AARGB4,WREG
                CPFSEQ          AARGB1
                GOTO            FLOOR24RNDL
                RETLW           0x00
```

```
FLOOR24RNDL
                COMF            AARGB7,W
                INCF            WREG,F
                ADDWF           AARGB1,F
                CLRF            WREG,F
                ADDWFC          AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRCF            AARGB0,F
                RRCF            AARGB1,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV24

FLOOR24MASKH
                CLRF            TBLPTRH,F

                MOVFP           TEMPB0,WREG     ; get remainder for mask pointer
                ANDLW           0x07

                ADDLW           LOW (FLOOR24MASKTABLE)
                MOVWF           TBLPTRL
                MOVLW           HIGH (FLOOR24MASKTABLE); access table for F0
                ADDWFC          TBLPTRH,F
                TABLRD          0,1,WREG
                TLRD            0,WREG

                ANDWF           AARGB0,F
                CLRF            AARGB1,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVFP           AARGB4,WREG
                CPFSEQ          AARGB1
                GOTO            FLOOR24RNDH
                MOVFP           AARGB3,WREG
                CPFSEQ          AARGB0
                GOTO            FLOOR24RNDH
                RETLW           0x00

FLOOR24RNDH
                COMF            AARGB7,W
                INCF            WREG,F
                ADDWF           AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRCF            AARGB0,F
                RRCF            AARGB1,F
                INCFSZ          AEXP,F
                RETLW           0x00
                GOTO            SETFOV24        ; check for overflow

FLOOR24ZERO
                BTFSC           AARGB0,MSB
                GOTO            FLOOR24MINUSONE
                CLRF            AEXP,F
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                RETLW           0x00

FLOOR24MINUSONE
                MOVLW           0x7F
                MOVWF           AEXP
                MOVLW           0x80
                MOVWF           AARGB0
```

```
                CLRF            AARGB1,F
                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       table for least significant byte requiring masking, using pointer from
;       the remainder of the number of zero bits divided by eight.

FLOOR24MASKTABLE
                DATA            0xFF
                DATA            0xFE
                DATA            0xFC
                DATA            0xF8
                DATA            0xF0
                DATA            0xE0
                DATA            0xC0
                DATA            0x80
                DATA            0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A < B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TALTB24

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A < B TRUE, WREG = 0x01
;               if A < B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       27      11.6    clks

TALTB24         MOVFP           AARGB0,WREG     ; test if signs opposite
                XORWF           BARGB0,W
                BTFSC           WREG,MSB
                GOTO            TALTB24O

                BTFSC           AARGB0,MSB
                GOTO            TALTB24N

TALTB24P        MOVFP           AEXP,WREG       ; compare positive arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
```

```
                RETLW           0x01
                RETLW           0x00

TALTB24N        MOVFP           BEXP,WREG       ; compare negative arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB24O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A <= B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TALEB24

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A <= B TRUE, WREG = 0x01
;               if A <= B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       25      11.5    clks

TALEB24         MOVFP           AARGB0,WREG     ; test if signs opposite
                XORWF           BARGB0,W
                BTFSC           WREG,MSB
                GOTO            TALEB24O

                BTFSC           AARGB0,MSB
                GOTO            TALEB24N

TALEB24P        MOVFP           AEXP,WREG       ; compare positive arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
```



```
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TALEB24N        MOVFP           BEXP,WREG       ; compare negative arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TALEB24O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A > B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAGTB24

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A > B TRUE, WREG = 0x01
;               if A > B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       27      11.5    clks

TAGTB24         MOVFP           BARGB0,WREG     ; test if signs opposite
                XORWF           AARGB0,W
                BTFSC           WREG,MSB
                GOTO            TAGTB24O

                BTFSC           BARGB0,MSB
                GOTO            TAGTB24N

TAGTB24P        MOVFP           BEXP,WREG       ; compare positive arguments
                SUBWF           AEXP,W
                BTFSS           _C
```

```
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB24N        MOVFP           AEXP,WREG       ; compare negative arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB24O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A >= B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAGEB24

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A >= B TRUE, WREG = 0x01
;               if A >= B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       25      11.5    clks

TAGEB24         MOVFP           BARGB0,WREG     ; test if signs opposite
```

```
                XORWF           AARGB0,W
                BTFSC           WREG,MSB
                GOTO            TAGEB24O

                BTFSC           BARGB0,MSB
                GOTO            TAGEB24N

TAGEB24P        MOVFP           BEXP,WREG       ; compare positive arguments
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB24N        MOVFP           AEXP,WREG       ; compare negative arguments
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB24O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A == B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1
;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TAEQB24

;       Output: logical result in WREG
```

```
;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A == B TRUE, WREG = 0x01
;               if A == B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 4       11      6.0     clks

TAEQB24         MOVFP           AEXP,WREG
                CPFSEQ          BEXP
                RETLW           0x00
                MOVFP           AARGB0,WREG
                CPFSEQ          BARGB0
                RETLW           0x00
                MOVFP           AARGB1,WREG
                CPFSEQ          BARGB1
                RETLW           0x00
                RETLW           0x01

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A =! B

;       Input:  24 bit floating point number in AEXP, AARGB0, AARGB1

;               24 bit floating point number in BEXP, BARGB0, BARGB1

;       Use:    CALL    TANEB24

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A =! B TRUE, WREG = 0x01
;               if A =! B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 4       11      6.0     clks

TANEB24         MOVFP           AEXP,WREG
                CPFSEQ          BEXP
                RETLW           0x01
                MOVFP           AARGB0,WREG
                CPFSEQ          BARGB0
                RETLW           0x01
                MOVFP           AARGB1,WREG
                CPFSEQ          BARGB1
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Nearest neighbor rounding

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    RND3224

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1

;       Result: AARG  <-- RND( AARG )
```

```
;       Testing on [MINNUM,MAXNUM] from 10000 trials:

;               min     max     mean
;       Timing: 3       21              clks

;               min     max     mean
;       Error:  0       0       0       nsb

;----------------------------------------------------------------------------------------------

RND3224
                BTFSS           AARGB2,MSB      ; is NSB < 0x80?
                RETLW           0x00

                BSF             _C              ; set carry for rounding
                MOVLW           0x80
                CPFSGT          AARGB2
                RRCF            AARGB1,W        ; select even if NSB = 0x80

                MOVPF           AARGB0,SIGN     ; save sign
                BSF             AARGB0,MSB      ; make MSB explicit

                CLRF            WREG,F          ; round
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F

                BTFSS           _C              ; has rounding caused carryout?
                GOTO            RND3224OK
                RRCF            ACCB0, F        ; if so, right shift
                RRCF            ACCB1, F
                INFSNZ          EXP, F          ; test for floating point overflow
                GOTO            SETFOV24
RND3224OK
                BTFSS           SIGN,MSB
                BCF             AARGB0,MSB      ; clear sign bit if positive
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************
```

NOTES:

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX F: PIC17CXXX 32-BIT ELEMENTARY FUNCTION LIBRARY

```
;       RCS Header $Id: ef32.a17 1.61 1997/03/11 15:48:45 F.J.Testa Exp $

;       $Revision: 1.61 $

;       PIC17 32-BIT ELEMENTARY FUNCTION LIBRARY

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Test statistics are typically from 100000 trials, with timing in cycles
;       and error in the next significant byte. In almost all cases, the floating
;       point routines satisfy a unit in the last position (1*ulp) accuracy
;       requirement, resulting in |nsb error| <= 0xFF. The integer and logical
;       routines are exact.

;       Routine  Function         Timing in cycles          Error in nsb
;                                 min      max      mean     min      max      mean     rms

;       SQRT32   32 bit sqrt(x)   10       568      494.0    -0x41    0x41     0.04     36.87

;       EXP32    32 bit exp(x)    14       2024     1834.7   -0xA2    0x9A     2.20     29.18

;       EXP1032  32 bit exp10(x)  14       2084     1845.3   -0x69    0xD9     21.72    39.44

;       LOG32    32 bit log(x)    12       2147     1985.0   -0x01    0x02     0.55     0.77

;       LOG1032  32 bit log10(x)  2001     2308     2135.9   -0x01    0x02     -0.11    0.60

;       SIN32    32 bit sin(x)    1338     2408     2182.5   -0x182   0x18D    -0.91    62.74

;       COS32    32 bit cos(x)    1256     2405     2182.6   -0x19A   0x148    -1.20    62.83

;       SINCOS32 32 bit cos(x),sin(x) 23283432      3217.8   -0x19A   0x148    -1.20    62.83
;                                                            -0x182   0x18D    -0.91    62.74

;       POW24    24 bit pow(x,y)=x**y 28524255      3915.7   -0x6B    0x77     -0.48    16.49

;       POW32    32 bit pow(x,y)=x**y4280 5574      5168.4   -0x270   0x209    8.94     92.21

;       FLOOR32  32 bit floor(x)  30       45       35.2     0x00     0x00     0.0      0.0

;----------------------------------------------------------------------------------------------

;       RAND32   32 bit rand(x)   117      117      117

;       TALTB32  32 bit A < B     8        33       11.6

;       TALEB32  32 bit A <= B    8        31       11.6

;       TAGTB32  32 bit A > B     8        33       11.6

;       TAGEB32  32 bit A >= B    8        31       11.6

;       TAEQB32  32 bit A == B    4        14       5.9

;       TANEB32  32 bit A != B    4        14       5.9

;**********************************************************************************************
;**********************************************************************************************

;       32 bit floating point representation
```

```
;       EXPONENT        8 bit biased exponent

;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = MIDBYTE = LOWBYTE = 0x00, with 0 being
;                       the only number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of fraction in sign-magnitude representation,
;                       with SIGN = MSB, implicit MSB = 1 and radix point to the right of MSB

;       MIDBYTE         8 bit middle significant byte of sign-magnitude fraction

;       LOWBYTE         8 bit least significant byte of sign-magnitude fraction

;       EXPONENT        HIGHBYTE        MIDBYTE         LOWBYTE

;       xxxxxxxx        S.xxxxxxx       xxxxxxxx        xxxxxxxx

;                        |
;                      RADIX
;                      POINT

;**********************************************************************************************
;**********************************************************************************************

;       polynomial evaluation macros

POLL132 macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       with leading coefficient of one, and where AARG is assumed have been be saved
;       in DARG.  The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;               add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;               add if DARGB3,RND is set

        local   i,j
        variable i = N, j = 0

        variable i = i - 1

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while   j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1
```



```
        endw

                CALL            FPA32

        variable i = i - 1

        while   i >= 0

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while   j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

        if      i == 0

                if      ROUND == 2

                BCF             FPFLAGS,RND

                endif

                if      ROUND == 3

                BSF             FPFLAGS,RND

                endif

                if      ROUND == 4

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                endif

                if      ROUND == 5

                BTFSC           DARGB3,RND
                BSF             FPFLAGS,RND

                endif

        endif

                CALL            FPA32

        variable i = i - 1
```

```
        endw

        endm


POL32   macro           COF,N,ROUND

;       32 bit evaluation of polynomial of degree N, PN(AARG), with coefficients COF,
;       and where AARG is assumed have been be saved in DARG.  The result is in AARG.

;       ROUND = 0no rounding is enabled; can be previously enabled
;       ROUND = 1rounding is enabled
;       ROUND = 2rounding is enabled then disabled before last add
;       ROUND = 3rounding is assumed disabled then enabled before last add
;       ROUND = 4rounding is assumed enabled and then disabled before last
;                       add if DARGB3,RND is clear
;       ROUND = 5rounding is assumed disabled and then enabled before last
;                       add if DARGB3,RND is set

        local   i,j
        variable i = N, j = 0

        if      ROUND == 1  ||  ROUND == 2

                BSF             FPFLAGS,RND

        endif

                MOVLW           COF#v(i)
                MOVWF           BEXP

        while   j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPM32

        variable i = i - 1

                MOVLW           COF#v(i)
                MOVWF           BEXP

        variable j = 0

        while   j <= 2

                MOVLW           COF#v(i)#v(j)
                MOVWF           BARGB#v(j)

        variable j = j + 1

        endw

                CALL            FPA32

        variable i = i - 1

        while   i >= 0

                MOVFP           DEXP,WREG
```



```
        MOVPF           WREG,BEXP
        MOVFP           DARGB0,WREG
        MOVPF           WREG,BARGB0
        MOVFP           DARGB1,WREG
        MOVPF           WREG,BARGB1
        MOVFP           DARGB2,WREG
        MOVPF           WREG,BARGB2

        CALL            FPM32

        MOVLW           COF#v(i)
        MOVWF           BEXP

variable j = 0

while   j <= 2

        MOVLW           COF#v(i)#v(j)
        MOVWF           BARGB#v(j)

variable j = j + 1

endw

if      i == 0

        if      ROUND == 2

        BCF             FPFLAGS,RND

        endif

        if      ROUND == 3

        BSF             FPFLAGS,RND

        endif

        if      ROUND == 4

        BTFSS           DARGB3,RND
        BCF             FPFLAGS,RND

        endif

        if      ROUND == 5

        BTFSC           DARGB3,RND
        BSF             FPFLAGS,RND

        endif

endif

        CALL            FPA32

variable i = i - 1

endw

endm

;*********************************************************************************************
```

```
;**********************************************************************************************

;       Evaluate exp(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    EXP32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  EXP( AARG )

;       Testing on [MINLOG,MAXLOG] from 100000 trials:

;               min     max     mean
;       Timing: 14      2024    1834.7  clks

;               min     max     mean    rms
;       Error:  -0xA2   0x9A    2.20    29.18   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp(x) = e**x = e**(z + n*log(2)) = e**z * 2**n,

;       where -log(2)/2 <= z <= log(2)/2 and n is an integer, evaluated during
;       range reduction. Segmented fifth degree minimax polynomial approximations
;       are used to estimate e**z on the intervals [-log(2)/2,0] and [0,log(2)/2].

EXP32
                MOVLW           0x5E            ; test for |x| < 2**(-32)/2
                CPFSGT          EXP
                GOTO            EXP32ONE        ; return e**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP32
TPEXP32
                MOVFP           AEXP,WREG       ; positive domain check
                SUBLW           MAXLOG32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MAXLOG32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MAXLOG32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB2,WREG
                SUBLW           MAXLOG32B2
                BTFSS           _C
                GOTO            DOMERR32
                GOTO            EXP32ARGOK
```

```
TNEXP32
                MOVFP           AEXP,WREG       ; negative domain check
                SUBLW           MINLOG32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MINLOG32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MINLOG32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP32ARGOK

                MOVFP           AARGB2,WREG
                SUBLW           MINLOG32B2
                BTFSS           _C
                GOTO            DOMERR32

EXP32ARGOK
                MOVFP           FPFLAGS,WREG    ; save RND flag
                MOVWF           DARGB3

                BSF             FPFLAGS,RND     ; enable rounding
                CALL            RREXP32

                BTFSC           DARGB0,MSB
                GOTO            EXP32L

EXP32H
                POL32           EXP32H,5,4      ; minimax approximation on [0,log(2)/2]

                GOTO            EXP32OK

EXP32L
                POL32           EXP32L,5,4      ; minimax approximation on [-log(2)/2,0]

EXP32OK
                MOVFP           EARGB3,WREG
                ADDWF           AEXP,F
                RETLW           0x00

EXP32ONE        MOVLW           EXPBIAS         ; return e**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                CLRF            AARGB3,F
                RETLW           0x00

DOMERR32        BSF             FPFLAGS,DOM     ; domain error
                RETLW           0xFF
```

```
;**********************************************************************************************

;       Range reduction routine for the exponential function

;       The evaluation of z and n through the decomposition

;               x = z + n*log(2)

;       is performed by first evaluating n through the relation

;               n = floor(x*log2(e) + .5)

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x - n*log(2) = (x - n*c1) + n*c2

;       where c1 is close to log(2) and has an exact machine representation,
;       typically leading to no error in computing the term in parenthesis.

RREXP32
                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save x in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                BSF             AARGB0,MSB

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                MOVLW           0xB8            ; 1/ln(2) = 1.44269504089
                MOVPF           WREG,AARGB0
                MOVLW           0xAA
                MOVPF           WREG,AARGB1
                MOVLW           0x3B
                MOVPF           WREG,AARGB2
                MOVLW           0x29
                MOVPF           WREG,AARGB3

                CALL            FXM3224U        ; x * (1/ln2)

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP32YOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP32YOK      BTFSS           CARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVLW           0x7E            ; k = [ x / ln2 + .5 ]
                MOVWF           BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPA32
```



```
                CALL            FLOOR32

                MOVFP           AEXP,WREG
                MOVPF           WREG,EEXP        ; save float k in EARG
                BTFSC           _Z
                GOTO            RREXP32FEQX
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                BSF             AARGB0,MSB

                MOVLW           0xB1             ; c1 = .693359375
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RREXP32F1OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP32F1OK     BTFSC           EARGB0,MSB       ; make AARG negative
                BCF             AARGB0,MSB

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP        ; save f1 in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVLW           0xDE             ; c2 = .00021219444005
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1
                MOVLW           0x83
                MOVWF           BARGB2

                MOVFP           EEXP,WREG
                MOVPF           WREG,AEXP
                MOVLW           0x0D-1
                SUBWF           AEXP,F
                MOVFP           EARGB0,AARGB0
                MOVFP           EARGB1,AARGB1
                MOVFP           EARGB2,AARGB2

                BSF             AARGB0,MSB

                CALL            FXM2424U
```

```
                BTFSC           AARGB0,MSB
                GOTO            RREXP32F2OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP32F2OK     BTFSS           EARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save f in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVFP           EEXP,WREG
                MOVWF           AEXP
                MOVFP           EARGB0,AARGB0
                MOVFP           EARGB1,AARGB1

                BCF             FPFLAGS,RND
                CALL            INT2416         ; k = [ x / ln2 + .5 ]
                BSF             FPFLAGS,RND

                MOVPF           AARGB1,EARGB3   ; save integer k in EARGB3

                MOVFP           DEXP,WREG
                MOVWF           AEXP            ; restore f in AARG
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                RETLW           0x00

RREXP32FEQX
                MOVFP           CEXP,WREG
                MOVWF           DEXP
                MOVWF           AEXP            ; save f = x in DARG, AARG
                MOVFP           CARGB0,WREG
                MOVWF           DARGB0
                MOVWF           AARGB0
                MOVFP           CARGB1,WREG
                MOVWF           DARGB1
                MOVWF           AARGB1
                MOVFP           CARGB2,WREG
                MOVWF           DARGB2
                MOVWF           AARGB2

                CLRF            EARGB3,F
```



```
                RETLW           0x00

;---------------------------------------------------------------------------------------------

;       fifth degree minimax polynomial coefficients for e**(x) on [0,(ln2)/2]

EXP32H0         EQU             0x7F            ; EXP32H0 = 1.0
EXP32H00        EQU             0x00
EXP32H01        EQU             0x00
EXP32H02        EQU             0x00

EXP32H1         EQU             0x7F            ; EXP32H1 = 1.00000025499
EXP32H10        EQU             0x00
EXP32H11        EQU             0x00
EXP32H12        EQU             0x02

EXP32H2         EQU             0x7D            ; EXP32H2 = .499991163105
EXP32H20        EQU             0x7F
EXP32H21        EQU             0xFE
EXP32H22        EQU             0xD7

EXP32H3         EQU             0x7C            ; EXP32H3 = .166777360103
EXP32H30        EQU             0x2A
EXP32H31        EQU             0xC7
EXP32H32        EQU             0xAF

EXP32H4         EQU             0x7A            ; EXP32H4 = .410473706887E-1
EXP32H40        EQU             0x28
EXP32H41        EQU             0x21
EXP32H42        EQU             0x4A

EXP32H5         EQU             0x78            ; EXP32H5 = .989943653774E-2
EXP32H50        EQU             0x22
EXP32H51        EQU             0x31
EXP32H52        EQU             0x3F

;       fifth degree minimax polynomial coefficients for e**(x) on [-(ln2)/2,0]

EXP32L0         EQU             0x7F            ; EXP32L0 = 1.0
EXP32L00        EQU             0x00
EXP32L01        EQU             0x00
EXP32L02        EQU             0x00

EXP32L1         EQU             0x7E            ; EXP32L1 = .999999766814
EXP32L10        EQU             0x7F
EXP32L11        EQU             0xFF
EXP32L12        EQU             0xFC

EXP32L2         EQU             0x7D            ; EXP32L2 = .499992371926
EXP32L20        EQU             0x7F
EXP32L21        EQU             0xFF
EXP32L22        EQU             0x00

EXP32L3         EQU             0x7C            ; EXP32L3 = .166574299807
EXP32L30        EQU             0x2A
EXP32L31        EQU             0x92
EXP32L32        EQU             0x75

EXP32L4         EQU             0x7A            ; EXP32L4 = .411548782678E-1
EXP32L40        EQU             0x28
EXP32L41        EQU             0x92
EXP32L42        EQU             0x05
```



```
EXP32L5         EQU             0x77            ; EXP32L5 = .699995870637E-2
EXP32L50        EQU             0x65
EXP32L51        EQU             0x5F
EXP32L52        EQU             0xE9

;**********************************************************************************************

;       Evaluate exp10(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    EXP1032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  EXP10( AARG )

;       Testing on [MINLOG10,MAXLOG10] from 100000 trials:

;               min     max     mean
;       Timing: 14      2084    1845.3  clks

;               min     max     mean    rms
;       Error:  -0x69   0xD9    21.72   39.44   nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the exponential function is based upon the
;       expansion

;               exp10(x) = 10**x = 10**(z + n*log10(2)) = 10**z * 2**n,

;       where -log10(2)/2 <= z <= log10(2)/2 and n is an integer, evaluated during
;       range reduction. Segmented fifth degree minimax polynomial approximations
;       are used to estimate 10**z on the intervals [-log10(2)/2,0] and [0,log10(2)/2].

EXP1032
                MOVLW           0x5E            ; test for |x| < 2**(-32)/2
                CPFSGT          AEXP
                GOTO            EXP1032ONE      ; return 10**x = 1

                BTFSC           AARGB0,MSB      ; determine sign
                GOTO            TNEXP1032
TPEXP1032
                MOVFP           AEXP,WREG       ; positive domain check
                SUBLW           MAXLOG1032EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MAXLOG1032B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MAXLOG1032B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB2,WREG
```



```
                SUBLW           MAXLOG1032B2
                BTFSS           _C
                GOTO            DOMERR32
                GOTO            EXP1032ARGOK

TNEXP1032
                MOVFP           AEXP,WREG        ; negative domain check
                SUBLW           MINLOG1032EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           MINLOG1032B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           MINLOG1032B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            EXP1032ARGOK

                MOVFP           AARGB2,WREG
                SUBLW           MINLOG1032B2
                BTFSS           _C
                GOTO            DOMERR32

EXP1032ARGOK
                MOVFP           FPFLAGS,WREG     ; save RND flag
                MOVWF           DARGB3

                BSF             FPFLAGS,RND      ; enable rounding
                CALL            RREXP1032

                BTFSC           DARGB0,MSB
                GOTO            EXP1032L

EXP1032H
                POL32           EXP1032H,5,4     ; minimax approximation on [0,log10(2)/2]

                GOTO            EXP1032OK

EXP1032L
                POL32           EXP1032L,5,4     ; minimax approximation on [-log10(2)/2,0]

EXP1032OK
                MOVFP           EARGB3,WREG
                ADDWF           AEXP,F
                RETLW           0x00

EXP1032ONE      MOVLW           EXPBIAS          ; return 10**x = 1.0
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                CLRF            AARGB3,F
                RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for the exponential function
```

```
;       The evaluation of z and n through the decomposition

;               x = z + n*log10(2)

;       is performed by first evaluating n through the relation

;               n = floor(x*log2(10) + .5)

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;               z = x - n*log10(2) = (x - n*c1) - n*c2

;       where c1 is close to log10(2) and has an exact machine representation,
;       typically leading to no error in computing the term in parenthesis.

RREXP1032
                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save x in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                BSF             AARGB0,MSB

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                MOVLW           0xD4            ; 1/log10(2) = 3.32192809489
                MOVPF           WREG,AARGB0
                MOVLW           0x9A
                MOVPF           WREG,AARGB1
                MOVLW           0x78
                MOVPF           WREG,AARGB2
                MOVLW           0x47
                MOVPF           WREG,AARGB3

                CALL            FXM3224U        ; x * (1/log10(2))

                INCF            AEXP,F
                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RREXP1032YOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP1032YOK    BTFSS           CARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVLW           0x7E            ; k = [ x / log10(2) + .5 ]
                MOVWF           BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPA32

                CALL            FLOOR32
```



```
                MOVFP           AEXP,WREG
                MOVPF           WREG,EEXP       ; save float k in EARG
                BTFSC           _Z
                GOTO            RREXP1032FEQX
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                BSF             AARGB0,MSB

                MOVLW           0x9A            ; c1 = .301025390625
                MOVWF           BARGB0
                MOVLW           0x20
                MOVWF           BARGB1

                DECF            AEXP,F

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RREXP1032F1OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP1032F1OK   BTFSC           EARGB0,MSB      ; make AARG negative
                BCF             AARGB0,MSB

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save f1 in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVLW           0x9A            ; c2 = 4.6050389811952113E-6
                MOVWF           BARGB0
                MOVLW           0x84
                MOVWF           BARGB1
                MOVLW           0xFC
                MOVWF           BARGB2

                MOVFP           EEXP,WREG
                MOVPF           WREG,AEXP
                MOVLW           0x12-1
                SUBWF           AEXP,F
                MOVFP           EARGB0,AARGB0
                MOVFP           EARGB1,AARGB1
                MOVFP           EARGB2,AARGB2

                BSF             AARGB0,MSB

                CALL            FXM2424U
```

```
                BTFSC           AARGB0,MSB
                GOTO            RREXP1032F2OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RREXP1032F2OK   BTFSC           EARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save f in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVFP           EEXP,WREG
                MOVWF           AEXP
                MOVFP           EARGB0,AARGB0
                MOVFP           EARGB1,AARGB1

                BCF             FPFLAGS,RND
                CALL            INT2416         ; k = [ x / log10(2) + .5 ]
                BSF             FPFLAGS,RND

                MOVPF           AARGB1,EARGB3   ; save integer k in EARGB3

                MOVFP           DEXP,WREG
                MOVWF           AEXP            ; restore f in AARG
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                RETLW           0x00

RREXP1032FEQX
                MOVFP           CEXP,WREG
                MOVWF           DEXP
                MOVWF           AEXP            ; save f = x in DARG, AARG
                MOVFP           CARGB0,WREG
                MOVWF           DARGB0
                MOVWF           AARGB0
                MOVFP           CARGB1,WREG
                MOVWF           DARGB1
                MOVWF           AARGB1
                MOVFP           CARGB2,WREG
                MOVWF           DARGB2
                MOVWF           AARGB2
```

```
                CLRF            EARGB3,F

                RETLW           0x00

;-----------------------------------------------------------------------------------------------

;       fifth degree minimax polynomial coefficients for 10**(x) on [0,(log10(2))/2]

EXP1032H0       EQU             0x7F            ; EXP1032H0 = 1.0
EXP1032H00      EQU             0x00
EXP1032H01      EQU             0x00
EXP1032H02      EQU             0x00

EXP1032H1       EQU             0x80            ; EXP1032H1 = 2.302585504840E0
EXP1032H10      EQU             0x13
EXP1032H11      EQU             0x5D
EXP1032H12      EQU             0x90

EXP1032H2       EQU             0x80            ; EXP1032H2 = 2.650909138708E0
EXP1032H20      EQU             0x29
EXP1032H21      EQU             0xA8
EXP1032H22      EQU             0x7F

EXP1032H3       EQU             0x80            ; EXP1032H3 = 2.035920309947E0
EXP1032H30      EQU             0x02
EXP1032H31      EQU             0x4C
EXP1032H32      EQU             0x85

EXP1032H4       EQU             0x7F            ; EXP1032H4 = 1.154596329197E0
EXP1032H40      EQU             0x13
EXP1032H41      EQU             0xC9
EXP1032H42      EQU             0xD0

EXP1032H5       EQU             0x7E            ; EXP1032H5 = 6.388992868121E-1
EXP1032H50      EQU             0x23
EXP1032H51      EQU             0x8E
EXP1032H52      EQU             0xE7

;       fifth degree minimax polynomial coefficients for 10**(x) on [-(log10(2))/2,0]

EXP1032L0       EQU             0x7F            ; EXP1032L0 = 1.0
EXP1032L00      EQU             0x00
EXP1032L01      EQU             0x00
EXP1032L02      EQU             0x00

EXP1032L1       EQU             0x80            ; EXP1032L1 = 2.302584716116E0
EXP1032L10      EQU             0x13
EXP1032L11      EQU             0x5D
EXP1032L12      EQU             0x8C

EXP1032L2       EQU             0x80            ; EXP1032L2 = 2.650914554552E0
EXP1032L20      EQU             0x29
EXP1032L21      EQU             0xA8
EXP1032L22      EQU             0x96

EXP1032L3       EQU             0x80            ; EXP1032L3 = 2.033640565225E0
EXP1032L30      EQU             0x02
EXP1032L31      EQU             0x27
EXP1032L32      EQU             0x2B

EXP1032L4       EQU             0x7F            ; EXP1032L4 = 1.157459289066E0
EXP1032L40      EQU             0x14
EXP1032L41      EQU             0x27
EXP1032L42      EQU             0xA0
```

```
EXP1032L5       EQU             0x7D            ; EXP1032L5 = 4.544952589676E-1
EXP1032L50      EQU             0x68
EXP1032L51      EQU             0xB3
EXP1032L52      EQU             0x9A

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate log(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    LOG32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  LOG( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 12      2147    1985.0  clks

;               min     max     mean    rms
;       Error:  -0x01   0x02    0.55    0.77    nsb

;----------------------------------------------------------------------------------------------

;       This approximation of the natural log function is based upon the
;       expansion

;               log(x) = log(f) + log(2**n) = log(f) + n*log(2)

;       where .5 <= f < 1 and n is an integer.  The additional transformation

;                   |   2*f-1, f < 1/sqrt(2), n = n - 1
;               z = |
;                   |   f-1, otherwise

;       produces a naturally segmented representation of log(1+z) on the
;       intervals [1/sqrt(2)-1,0] and [0,sqrt(2)-1], utilizing minimax rational
;       approximations.  The final evaluation of

;               log(1+z) + n*log(2) = (log(1+z) - n*c2) + n*c1

;       is performed in pseudo extended precision where c1 is close to log(2)
;       and has an exact machine representation.

LOG32
                CLRF            AARGB3,W
                BTFSS           AARGB0,MSB      ; test for negative argument
                CPFSGT          AEXP            ; test for zero argument
                GOTO            DOMERR32

                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                MOVFP           AEXP,WREG
                MOVPF           WREG,EARGB3
                MOVLW           EXPBIAS-1
                SUBWF           EARGB3,F
                MOVWF           AEXP

                MOVLW           0xF3            ; .70710678118655 = 7E3504F3
                SUBWF           AARGB2,W
```

```
                MOVLW           0x04
                SUBWFB          AARGB1,W
                MOVLW           0x35
                SUBWFB          AARGB0,W

                BTFSS           _C
                GOTO            LOG32FLOW

LOG32FHIGH      MOVLW           0x7F
                MOVPF           WREG,BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPS32

                GOTO            LOGZ32OK

LOG32FLOW       INCF            AEXP,F
                MOVLW           0x7F
                MOVPF           WREG,BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                CALL            FPS32

                DECF            EARGB3,F

LOGZ32OK
                MOVFP           AEXP,WREG       ; save z
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                POLL132         LOG32Q,2,0      ; Q(z)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                POL32           LOG32P,1,0      ; P(z)

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPD32           ; P(z)/Q(z)

                MOVFP           AEXP,WREG       ; save in CARG
                MOVPF           WREG,CEXP
                MOVPF           AARGB0,CARGB0
```

```
        MOVPF       AARGB1,CARGB1
        MOVPF       AARGB2,CARGB2

        MOVFP       DEXP,WREG
        MOVPF       WREG,BEXP
        MOVFP       DARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       DARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       DARGB2,WREG
        MOVPF       WREG,BARGB2

        MOVFP       DEXP,WREG
        MOVPF       WREG,AEXP
        MOVFP       DARGB0,AARGB0
        MOVFP       DARGB1,AARGB1
        MOVFP       DARGB2,AARGB2

        CALL        FPM32           ; z*z

        MOVFP       AEXP,WREG       ; save in EARG
        MOVPF       WREG,EEXP
        MOVPF       AARGB0,EARGB0
        MOVPF       AARGB1,EARGB1
        MOVPF       AARGB2,EARGB2

        MOVFP       CEXP,WREG       ; z*z*P(z)/Q(z)
        MOVPF       WREG,BEXP
        MOVFP       CARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       CARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       CARGB2,WREG
        MOVPF       WREG,BARGB2

        CALL        FPM32

        MOVFP       DEXP,WREG       ; z*(z*z*P(z)/Q(z))
        MOVPF       WREG,BEXP
        MOVFP       DARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       DARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       DARGB2,WREG
        MOVPF       WREG,BARGB2

        CALL        FPM32

        MOVFP       EEXP,WREG       ; -.5*z*z + z*(z*z*P(z)/Q(z))
        MOVPF       WREG,BEXP
        MOVFP       EARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       EARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       EARGB2,WREG
        MOVPF       WREG,BARGB2
        TSTFSZ      BEXP
        DECF        BEXP,F

        CALL        FPS32

        MOVFP       DEXP,WREG       ; z -.5*z*z + z*(z*z*P(z)/Q(z))
        MOVPF       WREG,BEXP
        MOVFP       DARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       DARGB1,WREG
```

```
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                TSTFSZ          EARGB3
                GOTO            ADJLOG32

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND
                CALL            FPA32
                RETLW           0x00

ADJLOG32
                CALL            FPA32

                MOVFP           AEXP,WREG       ; save in EARG
                MOVPF           WREG,EEXP
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                CLRF            AARGB0,F
                MOVFP           EARGB3,AARGB1
                BTFSC           AARGB1,MSB
                SETF            AARGB0,F

                CALL            FLO1624
                CLRF            AARGB2,F

                MOVFP           AEXP,WREG       ; save k in DARG
                MOVPF           WREG,DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                BSF             AARGB0,MSB
                MOVLW           0x0D-1          ; .000212194440055
                SUBWF           AEXP,F
                MOVLW           0xDE
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1
                MOVLW           0x83
                MOVWF           BARGB2

                CALL            FXM2424U

                BTFSC           AARGB0,MSB
                GOTO            LOG32F1OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

LOG32F1OK
                BTFSC           DARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVFP           EEXP,WREG       ; log(1+z) + k*log(2)
                MOVPF           WREG,BEXP
                MOVFP           EARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           EARGB1,WREG
```

```
                MOVPF           WREG,BARGB1
                MOVFP           EARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32

                MOVFP           AEXP,WREG       ; save in EARG
                MOVPF           WREG,EEXP
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                MOVLW           0xB1            ; .693359375
                MOVWF           BARGB0
                MOVLW           0x80
                MOVWF           BARGB1

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                BSF             AARGB0,MSB

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            LOG32FOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

LOG32FOK
                BTFSS           DARGB0,MSB
                BCF             AARGB0,MSB

                CALL            RND4032

                MOVFP           EEXP,WREG       ; log(1+z) + k*log(2)
                MOVPF           WREG,BEXP
                MOVFP           EARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           EARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           EARGB2,WREG
                MOVPF           WREG,BARGB2

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND
                CALL            FPA32
                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax rational approximationz-.5*z*z+z*(z*z*P(z)/Q(z))

LOG32P0         EQU             0x7E            ; LOG32P0 = .83311400452
LOG32P00        EQU             0x55
LOG32P01        EQU             0x46
LOG32P02        EQU             0xF6

LOG32P1         EQU             0x7D            ; LOG32P1 = .48646956294
LOG32P10        EQU             0x79
```



```
LOG32P11        EQU             0x12
LOG32P12        EQU             0x8A

LOG32Q0         EQU             0x80            ; LOG32Q0 = .24993759223E1
LOG32Q00        EQU             0x1F
LOG32Q01        EQU             0xF5
LOG32Q02        EQU             0xC6

LOG32Q1         EQU             0x80            ; LOG32Q1 = .33339502905E+1
LOG32Q10        EQU             0x55
LOG32Q11        EQU             0x5F
LOG32Q12        EQU             0x72

LOG32Q2         EQU             0x7F            ; LOG32Q2 = 1.0
LOG32Q20        EQU             0x00
LOG32Q21        EQU             0x00
LOG32Q22        EQU             0x00

;*********************************************************************************************

;       Evaluate log10(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    LOG1032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  LOG10( AARG )

;       Testing on [MINNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 2001    2308    2135.9  clks

;               min     max     mean    rms
;       Error:  -0x01   0x02    -0.11   0.60    nsb

;---------------------------------------------------------------------------------------------

LOG1032         MOVFP           FPFLAGS,WREG
                MOVWF           ZARGB0
                BCF             FPFLAGS,RND

                CALL            LOG32

                MOVPF           AARGB0,DARGB0
                BSF             AARGB0,MSB

                MOVLW           0xDE            ; log10(e) = .43429448190325
                MOVPF           WREG,BARGB0
                MOVLW           0x5B
                MOVPF           WREG,BARGB1
                MOVLW           0xD8
                MOVPF           WREG,BARGB2
                MOVLW           0xA9
                MOVPF           WREG,BARGB3

                CALL            FXM3232U        ; log(x) * log10(e)

                DECF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            LOG1032OK
                RLCF            AARGB4,F
                RLCF            AARGB3,F
```

```
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

LOG1032OK       BTFSS           DARGB0,MSB
                BCF             AARGB0,MSB

                BTFSS           ZARGB0,RND
                RETLW           0x00

                BSF             FPFLAGS,RND
                CALL            RND4032
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate cos(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    COS32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  COS( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 1256    2405    2182.6  clks

;               min     max     mean    rms
;       Error:  -0x19A  0x148   -1.20   62.83   nsb

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations.

COS32
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                CLRF            CARGB3,F        ; initialize sign in CARGB3

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS32      ; range reduction

RRCOS32OK
                RRCF            EARGB3,W
                XORWF           EARGB3,W
                BTFSC           WREG,LSB
                GOTO            COSZSIN32

                CALL            ZCOS32
```



```
                GOTO            COSSIGN32

COSZSIN32       CALL            ZSIN32

COSSIGN32       BTFSC           EARGB3,LSB+1
                BTG             CARGB3,MSB

                BTFSC           CARGB3,MSB
                BTG             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND4032
                RETLW           0x00

;**********************************************************************************************

;       Evaluate sin(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SIN32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 1338    2408    2182.5  clks

;               min     max     mean    rms
;       Error:  -0x182  0x18D   -0.91   62.74   nsb

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations.

SIN32
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                CLRF            CARGB3,F        ; initialize sign in CARGB3

                BTFSC           AARGB0,MSB
                BSF             CARGB3,MSB

                BCF             AARGB0,MSB      ; use |x|

                CALL            RRSINCOS32      ; range reduction

RRSIN32OK
                RRCF            EARGB3,W
```

```
                XORWF           EARGB3,W
                BTFSC           WREG,LSB
                GOTO            SINZCOS32

                CALL            ZSIN32

                GOTO            SINSIGN32

SINZCOS32       CALL            ZCOS32

SINSIGN32       BTFSC           CARGB3,MSB
                BTG             AARGB0,MSB

                BTFSS           DARGB3,RND
                RETLW           0x00

                BSF             FPFLAGS,RND     ; restore rounding flag
                CALL            RND4032
                RETLW           0x00

;**********************************************************************************************

;       Evaluate sin(x) and cos(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SINCOS32

;       Output: 32 bit floating point cos(x) in AEXP, AARGB0, AARGB1, AARGB2 and
;                                       sin(x) BEXP, BARGB0, BARGB1, BARGB2

;       Result: AARG  <--  COS( AARG )
;               BARG  <--  SIN( AARG )

;       Testing on [-LOSSTHR,LOSSTHR] from 100000 trials:

;               min     max     mean
;       Timing: 2328    3432    3217.8  clks

;               min     max     mean    rms
;       Error:  -0x19A  0x148   -1.20   62.83   nsb     cos(x)
;               -0x182  0x18D   -0.91   62.74           sin(x)

;----------------------------------------------------------------------------------------------

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition z = x mod pi/4, with an additional variable j
;       indicating the correct octant, leading to the appropriate call
;       to either the sine or cosine approximations

;       sin(z) = z * (z**2) * p(z**2),cos(z) = 1 - .5 * z**2 + (z**4) * q(z**2)

;       where p and q are minimax polynomial approximations. In this case,
;       only one range reduction is necessary.

SINCOS32
                MOVFP           FPFLAGS,WREG    ; save rounding flag
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                MOVFP           AEXP,WREG       ; save x in EARG
                MOVWF           EEXP
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2
```



```
            BCF             AARGB0,MSB      ; use |x|

            CLRF            CARGB3,F        ; initialize sign in CARGB3

            CALL            RRSINCOS32      ; range reduction

            MOVFP           CARGB3,WREG     ; save sign from range reduction
            MOVWF           ZARGB2

            BTFSC           EARGB0,MSB      ; toggle sign if x < 0
            BTG             CARGB3,MSB

            CALL            RRSIN32OK

            MOVFP           AEXP,WREG       ; save sin(x) in EARG
            MOVWF           EEXP
            MOVPF           AARGB0,EARGB0
            MOVPF           AARGB1,EARGB1
            MOVPF           AARGB2,EARGB2
            MOVPF           AARGB3,ZARGB3

            BSF             FPFLAGS,RND     ; enable rounding

            MOVFP           DEXP,WREG       ; restore z*z in AARG
            MOVWF           AEXP
            MOVFP           DARGB0,AARGB0
            MOVFP           DARGB1,AARGB1
            MOVFP           DARGB2,AARGB2

            MOVFP           ZARGB2,WREG     ; restore sign from range reduction
            MOVWF           CARGB3

            CALL            RRCOS32OK

            MOVFP           EEXP,WREG       ; restore sin(x) in BARG
            MOVPF           WREG,BEXP
            MOVFP           EARGB0,WREG
            MOVPF           WREG,BARGB0
            MOVFP           EARGB1,WREG
            MOVPF           WREG,BARGB1
            MOVFP           EARGB2,WREG
            MOVPF           WREG,BARGB2
            MOVFP           ZARGB3,WREG
            MOVWF           BARGB3

            RETLW           0x00

;**********************************************************************************************

;       Range reduction routine for trigonometric functions

;       The actual argument x on [-LOSSTHR,LOSSTHR] is mapped to the
;       alternative trigonometric argument z on [-pi/4,pi/4], through
;       the definition

;               z = x mod pi/4,

;       produced by first evaluating y and j through the relations

;               y = floor(x/(pi/4)), j = y - 8*[y/8].

;       where j equals the correct octant.  For j odd, adding one to j
;       and y eliminates the odd octants.  Additional logic on j and the
```

```
;       sign of the result leads to appropriate use of the sine or cosine
;       routine in each case.

;       The calculation of z is then obtained through a pseudo extended
;       precision method

;       z = x mod pi/4 = x - y*(pi/4) = (((x - p1*y)-p2*y)-p3*y)-p4*y

;       where pi/4 = p1 + p2 + p3 + p4, with p1 close to pi/4, p2 close to
;       pi/4 - p1, and p3 close to pi/4 - p1 - p2. The numbers p1, p2 and p3
;       are chosen to have an exact machine representation with slightly more
;       than the lower half of the mantissa bits zero, typically leading to no
;       error in computing the terms in parenthesis. This calculation breaks
;       down leading to a loss of precision for |x| > LOSSTHR = sqrt(2**24)*pi/4,
;       or for |x| close to an integer multiple of pi/4. This loss threshold has
;       been chosen based on the efficacy of this calculation, with a domain error
;       reported if this threshold is exceeded.

RRSINCOS32
                MOVFP           AEXP,WREG       ; loss threshold check
                SUBLW           LOSSTHR32EXP
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVFP           AARGB0,WREG
                SUBLW           LOSSTHR32B0
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVFP           AARGB1,WREG
                SUBLW           LOSSTHR32B1
                BTFSS           _C
                GOTO            DOMERR32
                BTFSS           _Z
                GOTO            RRSINCOS32ARGOK

                MOVFP           AARGB2,WREG
                SUBLW           LOSSTHR32B2
                BTFSS           _C
                GOTO            DOMERR32

RRSINCOS32ARGOK
                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save |x| in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

;       fixed point multiplication by 4/pi

                BSF             AARGB0,MSB
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                MOVLW           0xA2            ; 4/pi = 1.27323954474
                MOVPF           WREG,AARGB0
                MOVLW           0xF9
                MOVPF           WREG,AARGB1
                MOVLW           0x83
                MOVPF           WREG,AARGB2
                MOVLW           0x6E
```

```
                MOVPF           WREG,AARGB3

                CALL            FXM3224U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32YOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS32YOK
                BCF             AARGB0,MSB

                BCF             FPFLAGS,RND
                CALL            INT3224         ; y = [ |x| * (4/pi) ]
                BSF             FPFLAGS,RND

                BTFSS           AARGB2,LSB
                GOTO            SAVEY32
                INCF            AARGB2,F
                CLRF            WREG,F
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F

SAVEY32         MOVPF           AARGB0,DARGB0   ; save y in DARG
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVLW           0x07            ; j = y mod 8
                ANDWF           AARGB2,F

                MOVLW           0x03
                CPFSGT          AARGB2
                GOTO            JOK32
                BTG             CARGB3,MSB
                MOVLW           0x04
                SUBWF           AARGB2,F

JOK32           MOVPF           AARGB2,EARGB3   ; save j in EARGB3

                MOVFP           DARGB0,AARGB0   ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                CALL            FLO2432

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save y in DARG
                BTFSC           _Z
                GOTO            RRSINCOS32ZEQX
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

;       Cody-Waite extended precision calculation of |x| - y * pi/4 using
;       fixed point multiplication. Since y >= 1, underflow is not possible
;       in any of the products.

                BSF             AARGB0,MSB

                MOVLW           0xC9            ; - p1 = -.78515625
                MOVPF           WREG,BARGB0
```

```
                CLRF            BARGB1,F

                CALL            FXM2416U

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z1OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z1OK
                MOVFP           CEXP,WREG       ; restore x to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32           ; z1 = |x| - y * (p1)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z1 in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0   ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xFD            ; - p2 = -.000241875648498535156 24
                MOVPF           WREG,BARGB0
                MOVLW           0xA0
                MOVPF           WREG,BARGB1

                CALL            FXM2416U

                MOVLW           0x0D - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z2OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z2OK
                SUBWF           AEXP,F

                MOVFP           CEXP,WREG       ; restore z1 to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2
```

```
                CALL            FPA32           ; z2 = z1 - y * (p2)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z2 in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0   ; restore y to AARG
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                BSF             AARGB0,MSB

                MOVLW           0xA2            ; - p3 = -3.77476681023836135830E-8
                MOVPF           WREG,BARGB0
                MOVLW           0x20
                MOVPF           WREG,BARGB1

                CALL            FXM2416U

                MOVLW           0x19 - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z3OK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z3OK
                SUBWF           AEXP,F

                MOVFP           CEXP,WREG       ; restore z2 to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPA32           ; z3 = z2 - y * (p3)

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z3 in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,WREG
                MOVWF           BARGB0          ; restore y to BARG
                MOVFP           DARGB1,WREG
                MOVWF           BARGB1
                MOVFP           DARGB2,WREG
                MOVWF           BARGB2

                BSF             BARGB0,MSB

                MOVLW           0xB4            ; - p4 = -3.77489497744597636E-8
```

```
                MOVPF           WREG,AARGB0
                MOVLW           0x61
                MOVPF           WREG,AARGB1
                MOVLW           0x1A
                MOVPF           WREG,AARGB2
                MOVLW           0x63
                MOVPF           WREG,AARGB3

                CALL            FXM3224U

                MOVLW           0x28 - 1

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32Z4OK
                RLCF            AARGB4,F
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSINCOS32Z4OK
                SUBWF           AEXP,F

                CALL            RND4032

                MOVFP           CEXP,WREG       ; restore z3 to BARG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                BCF             FPFLAGS,RND     ; disable rounding
                CALL            FPA32           ; z = z3 - y * (p4)

RRSINCOS32OK
                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save z in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                BTFSS           AARGB3,MSB      ; is NSB < 0x80?
                GOTO            RRSINCOS32ZOK

                BSF             _C              ; set carry for rounding
                MOVLW           0x80
                CPFSGT          AARGB3
                RRCF            AARGB2,W        ; select even if NSB = 0x80

                MOVPF           AARGB0,SIGN     ; save sign
                BSF             CARGB0,MSB      ; make MSB explicit

                CLRF            WREG,F          ; round
                ADDWFC          CARGB2,F
                ADDWFC          CARGB1,F
                ADDWFC          CARGB0,F

                BTFSS           _C              ; has rounding caused carryout?
                GOTO            RRSINCOS32RZOK
                RRCF            CARGB0,F        ; if so, right shift
                RRCF            CARGB1,F
                RRCF            CARGB2,F
```



```
                INFSNZ          CEXP, F         ; test for floating point overflow
                GOTO            SETFOV32
RRSINCOS32RZOK
                BTFSS           SIGN,MSB
                BCF             CARGB0,MSB      ; clear sign bit if positive

RRSINCOS32ZOK
                BSF             AARGB0,MSB      ; make MSB explicit
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2
                MOVPF           AARGB3,BARGB3

                CALL            FXM3232U        ; z * z

                BCF             _C              ; multiply exponent by 2
                RLCF            AEXP,F
                MOVLW           EXPBIAS-1
                SUBWFB          AEXP,F

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSINCOS32ZZOK
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F
RRSINCOS32ZZOK
                BCF             AARGB0,MSB

                CALL            RND4032

                BSF             FPFLAGS,RND     ; enable rounding

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save z * z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00

RRSINCOS32ZEQX
                MOVFP           CEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           CARGB0,AARGB0
                MOVFP           CARGB1,AARGB1
                MOVFP           CARGB2,AARGB2

                MOVFP           AEXP,WREG
                MOVPF           WREG,BEXP
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                CALL            FPM32           ; z * z

                MOVFP           AEXP,WREG
                MOVPF           WREG,DEXP       ; save z * z in DARG
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                RETLW           0x00
```

```
;**********************************************************************************************

ZCOS32          POL32           COS32D,2,1

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2
                DECF            BEXP,F

                CALL            FPS32

                MOVLW           EXPBIAS
                MOVWF           BEXP
                CLRF            BARGB0,F
                CLRF            BARGB1,F
                CLRF            BARGB2,F

                BCF             FPFLAGS,RND
                CALL            FPA32

                RETLW           0x00

ZSIN32
                POL32           SIN32D,3,1

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
```

```
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPM32

                MOVFP           CEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           CARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           CARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           CARGB2,WREG
                MOVPF           WREG,BARGB2

                BCF             FPFLAGS,RND

                CALL            FPA32

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for sin(z) = z+z*(z**2)*p(z**2) on [-pi/4,pi/4]

SIN32D0         EQU             0x7C            ; SIN32D0 = -1.666666664079712E-1
SIN32D00        EQU             0xAA
SIN32D01        EQU             0xAA
SIN32D02        EQU             0xAB

SIN32D1         EQU             0x78            ; SIN32D1 = 8.333329304850749E-3
SIN32D10        EQU             0x08
SIN32D11        EQU             0x88
SIN32D12        EQU             0x84

SIN32D2         EQU             0x72            ; SIN32D2 = -1.983931227180460E-4
SIN32D20        EQU             0xD0
SIN32D21        EQU             0x07
SIN32D22        EQU             0xC0

SIN32D3         EQU             0x6C            ; SIN32D3 = 2.718121647219611E-6
SIN32D30        EQU             0x36
SIN32D31        EQU             0x68
SIN32D32        EQU             0xF9

;----------------------------------------------------------------------------------------------

;       minimax polynomial coefficients for cos(z) = 1 -.5*z**2 + z**4*q(z**2)
;       on [-pi/4,pi/4]

COS32D0         EQU             0x7A            ; COS32D0 = 4.166664568297614E-2
COS32D00        EQU             0x2A
COS32D01        EQU             0xAA
COS32D02        EQU             0xA5

COS32D1         EQU             0x75            ; COS32D1 = -1.388731625438419E-3
COS32D10        EQU             0xB6
COS32D11        EQU             0x06
COS32D12        EQU             0x1A

COS32D2         EQU             0x6F            ; COS32D2 = 2.443315706066392E-5
COS32D20        EQU             0x4C
COS32D21        EQU             0xF5
```

```
COS32D22        EQU             0xCE

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate sqrt(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    SQRT32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  SQRT( AARG )

;       Testing on [0,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 10      568     494.0   clks

;               min     max     mean    rms
;       Error:  -0x41   0x41    0.04    36.87   nsb

;----------------------------------------------------------------------------------------------

;       Range reduction for the square root function is naturally produced by
;       the floating point representation,

;               x = f * 2**e,where 1 <= f < 2,

;       leading to the expression

;                          |  sqrt(f) * 2**(e/2),e even
;               sqrt(x) = |
;                          |  sqrt(f) * sqrt(2) * 2**(e/2),e odd

;       The approximation of sqrt(f) utilizes a table lookup of 16 bit
;       estimates of the square root with linear interpolation between
;       adjacent entries as a seed to a single Newton-Raphson iteration,

;               y = (y0 + f/y0)/2,

;       where the precision of the result is guaranteed by the precision of the
;       seed and the quadratic conversion of the method.

SQRT32
                BTFSC           AARGB0,MSB      ; test for negative argument
                GOTO            DOMERR32

                CLRF            AARGB3,W        ; return if argument zero
                CPFSGT          AEXP
                RETLW           0x00

                MOVFP           AEXP,WREG
                MOVPF           WREG,CEXP       ; save x in CARG
                MOVPF           AARGB0,CARGB0
                MOVPF           AARGB1,CARGB1
                MOVPF           AARGB2,CARGB2

                MOVFP           FPFLAGS,WREG    ; save RND flag in DARGB3
                MOVPF           WREG,DARGB3

                BSF             FPFLAGS,RND     ; enable rounding

                MOVLW           EXPBIAS         ; initialize exponent
                MOVPF           WREG,AEXP
```



```
;       generation of y0 by interpolating between consecutive 16 bit approximations
;       to the square root of AARG, with the top 8 explicit bits of AARG as a pointer
;       and the remaining 15 explicit bits as the argument to linear interpolation.

                MOVLW           HIGH (RATBL256I); access table for y0
                MOVWF           TBLPTRH
                RLCF            AARGB1,W
                RLCF            AARGB0,W
                ADDLW           LOW (RATBL256I)
                MOVWF           TBLPTRL
                BTFSC           _C
                INCF            TBLPTRH,F
                TABLRD          0,1,TEMPB0
                TLRD            1,TEMPB0
                TABLRD          0,0,TEMPB1
                TLRD            0,AARGB5

                MOVFP           TEMPB1,WREG      ; calculate difference
                SUBWF           AARGB5,W
                MOVWF           AARGB5

                BCF             _C               ; interpolate
                RLCF            AARGB2,W
                MULWF           AARGB5
                MOVPF           PRODH,TBLPTRH
                RLCF            AARGB1,W
                MULWF           AARGB5
                MOVPF           PRODL,WREG
                ADDWF           TBLPTRH,F
                BTFSC           _C
                INCF            PRODH,F

                CLRF            TEMPB2,F
                MOVFP           TBLPTRH,WREG
                ADDWF           TEMPB2,F
                MOVPF           PRODH,WREG
                ADDWFC          TEMPB1,F
                CLRF            WREG,F
                ADDWFC          TEMPB0,F         ; y0

                MOVFP           TEMPB0,AARGB0
                MOVFP           TEMPB1,AARGB1
                MOVFP           TEMPB2,AARGB2

                BTFSC           CEXP,LSB         ; is CEXP even or odd?
                GOTO            RRSQRT32OK

;       fixed point multiplication by sqrt(2)

                BSF             AARGB0,MSB       ; make MSB explicit

                MOVLW           0xB5             ; sqrt(2) = 1.41421356237
                MOVPF           WREG,BARGB0
                MOVLW           0x04
                MOVPF           WREG,BARGB1
                MOVLW           0xF3
                MOVPF           WREG,BARGB2

                CALL            FXM2424U

                INCF            AEXP,F

                BTFSC           AARGB0,MSB
                GOTO            RRSQRT32OK
```

```
                RLCF            AARGB3,F
                RLCF            AARGB2,F
                RLCF            AARGB1,F
                RLCF            AARGB0,F
                DECF            AEXP,F

RRSQRT32OK
                BCF             AARGB0,MSB      ; make  MSB implicit

                CALL            RND4032

                MOVLW           EXPBIAS         ; divide exponent by two
                ADDWF           CEXP,W
                RRCF            WREG,F

                MOVPF           WREG,AEXP
                MOVPF           WREG,BEXP
                MOVPF           WREG,DEXP

                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                MOVFP           CEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           CARGB0,AARGB0
                MOVFP           CARGB1,AARGB1
                MOVFP           CARGB2,AARGB2

                CALL            FPD32           ; Newton-Raphson iteration

                MOVFP           DEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           DARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           DARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           DARGB2,WREG
                MOVPF           WREG,BARGB2

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                CALL            FPA32

                DECF            AEXP,F

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       Rounded to the nearest approximations to sqrt(f), with pointer from
;       the 8 most significant explicit bits of f, the mantissa of x. Linear
;       interpolation is performed between adjacent entries using the remaining
;       explicit bits of f.

RATBL256I
                DATA    0x0000
                DATA    0x0040
                DATA    0x0080
                DATA    0x00BF
                DATA    0x00FF
```



```
DATA    0x013E
DATA    0x017E
DATA    0x01BD
DATA    0x01FC
DATA    0x023B
DATA    0x027A
DATA    0x02B9
DATA    0x02F7
DATA    0x0336
DATA    0x0374
DATA    0x03B2
DATA    0x03F0
DATA    0x042F
DATA    0x046C
DATA    0x04AA
DATA    0x04E8
DATA    0x0526
DATA    0x0563
DATA    0x05A0
DATA    0x05DE
DATA    0x061B
DATA    0x0658
DATA    0x0695
DATA    0x06D2
DATA    0x070E
DATA    0x074B
DATA    0x0787
DATA    0x07C4
DATA    0x0800
DATA    0x083C
DATA    0x0878
DATA    0x08B4
DATA    0x08F0
DATA    0x092C
DATA    0x0968
DATA    0x09A3
DATA    0x09DF
DATA    0x0A1A
DATA    0x0A55
DATA    0x0A90
DATA    0x0ACB
DATA    0x0B06
DATA    0x0B41
DATA    0x0B7C
DATA    0x0BB7
DATA    0x0BF1
DATA    0x0C2C
DATA    0x0C66
DATA    0x0CA1
DATA    0x0CDB
DATA    0x0D15
DATA    0x0D4F
DATA    0x0D89
DATA    0x0DC3
DATA    0x0DFC
DATA    0x0E36
DATA    0x0E70
DATA    0x0EA9
DATA    0x0EE2
DATA    0x0F1C
DATA    0x0F55
DATA    0x0F8E
DATA    0x0FC7
DATA    0x1000
DATA    0x1039
DATA    0x1072
```

```
DATA    0x10AA
DATA    0x10E3
DATA    0x111B
DATA    0x1154
DATA    0x118C
DATA    0x11C4
DATA    0x11FC
DATA    0x1235
DATA    0x126D
DATA    0x12A4
DATA    0x12DC
DATA    0x1314
DATA    0x134C
DATA    0x1383
DATA    0x13BB
DATA    0x13F2
DATA    0x1429
DATA    0x1461
DATA    0x1498
DATA    0x14CF
DATA    0x1506
DATA    0x153D
DATA    0x1574
DATA    0x15AB
DATA    0x15E1
DATA    0x1618
DATA    0x164E
DATA    0x1685
DATA    0x16BB
DATA    0x16F2
DATA    0x1728
DATA    0x175E
DATA    0x1794
DATA    0x17CA
DATA    0x1800
DATA    0x1836
DATA    0x186C
DATA    0x18A1
DATA    0x18D7
DATA    0x190D
DATA    0x1942
DATA    0x1977
DATA    0x19AD
DATA    0x19E2
DATA    0x1A17
DATA    0x1A4C
DATA    0x1A81
DATA    0x1AB6
DATA    0x1AEB
DATA    0x1B20
DATA    0x1B55
DATA    0x1B8A
DATA    0x1BBE
DATA    0x1BF3
DATA    0x1C27
DATA    0x1C5C
DATA    0x1C90
DATA    0x1CC4
DATA    0x1CF9
DATA    0x1D2D
DATA    0x1D61
DATA    0x1D95
DATA    0x1DC9
DATA    0x1DFD
DATA    0x1E31
DATA    0x1E64
```

```
DATA    0x1E98
DATA    0x1ECC
DATA    0x1EFF
DATA    0x1F33
DATA    0x1F66
DATA    0x1F99
DATA    0x1FCD
DATA    0x2000
DATA    0x2033
DATA    0x2066
DATA    0x2099
DATA    0x20CC
DATA    0x20FF
DATA    0x2132
DATA    0x2165
DATA    0x2198
DATA    0x21CA
DATA    0x21FD
DATA    0x222F
DATA    0x2262
DATA    0x2294
DATA    0x22C7
DATA    0x22F9
DATA    0x232B
DATA    0x235D
DATA    0x238F
DATA    0x23C2
DATA    0x23F4
DATA    0x2425
DATA    0x2457
DATA    0x2489
DATA    0x24BB
DATA    0x24ED
DATA    0x251E
DATA    0x2550
DATA    0x2581
DATA    0x25B3
DATA    0x25E4
DATA    0x2616
DATA    0x2647
DATA    0x2678
DATA    0x26A9
DATA    0x26DA
DATA    0x270B
DATA    0x273D
DATA    0x276D
DATA    0x279E
DATA    0x27CF
DATA    0x2800
DATA    0x2831
DATA    0x2861
DATA    0x2892
DATA    0x28C3
DATA    0x28F3
DATA    0x2924
DATA    0x2954
DATA    0x2984
DATA    0x29B5
DATA    0x29E5
DATA    0x2A15
DATA    0x2A45
DATA    0x2A75
DATA    0x2AA5
DATA    0x2AD5
DATA    0x2B05
DATA    0x2B35
```

```
                DATA    0x2B65
                DATA    0x2B95
                DATA    0x2BC4
                DATA    0x2BF4
                DATA    0x2C24
                DATA    0x2C53
                DATA    0x2C83
                DATA    0x2CB2
                DATA    0x2CE2
                DATA    0x2D11
                DATA    0x2D40
                DATA    0x2D70
                DATA    0x2D9F
                DATA    0x2DCE
                DATA    0x2DFD
                DATA    0x2E2C
                DATA    0x2E5B
                DATA    0x2E8A
                DATA    0x2EB9
                DATA    0x2EE8
                DATA    0x2F17
                DATA    0x2F45
                DATA    0x2F74
                DATA    0x2FA3
                DATA    0x2FD1
                DATA    0x3000
                DATA    0x302F
                DATA    0x305D
                DATA    0x308B
                DATA    0x30BA
                DATA    0x30E8
                DATA    0x3116
                DATA    0x3145
                DATA    0x3173
                DATA    0x31A1
                DATA    0x31CF
                DATA    0x31FD
                DATA    0x322B
                DATA    0x3259
                DATA    0x3287
                DATA    0x32B5
                DATA    0x32E3
                DATA    0x3310
                DATA    0x333E
                DATA    0x336C
                DATA    0x3399
                DATA    0x33C7
                DATA    0x33F5
                DATA    0x3422
                DATA    0x3450
                DATA    0x347D
                DATA    0x34AA
                DATA    0x34D8
                DATA    0x3505

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate pow(x,y) = X**Y

;       Input:  24 bit floating point number X in AEXP, AARGB0, AARGB1 and
;               24 bit floating point number Y in BEXP, BARGB0, BARGB1.

;       Use:    CALL    POW24

;       Output: 24 bit floating point number in AEXP, AARGB0, AARGB1
```

```
;       Result: AARG  <--  POW( AARG )

;       Testing on [1/26,26] from 100000 trials:

;               min     max     mean
;       Timing: 2852    4255    3915.7  clks

;               min     max     mean    rms
;       Error:  -0x6B   0x77    -0.48   16.49   nsb

;----------------------------------------------------------------------------------------------

;       Because of the availability of extended precision routines, the 24 bit
;       power function can be estimated directly using the identity

;                       x**y = exp(y*log(x))

;       where the 32 bit exponential and natural log functions are called. A test
;       for overflow from the product y*log(x) is performed explicitly, but the
;       actual domain check is done in the exponential function.

POW24
                CLRF            AARGB2,W        ; clear NSB

                BTFSC           AARGB0,MSB      ; test if AARG < 0
                GOTO            DOMERR32

                CPFSGT          BEXP            ; if BARG=0, return 1.0
                GOTO            POW24ONE
                MOVFP           BEXP,WREG       ; save Y in ZARG
                MOVWF           ZARGB2
                MOVFP           BARGB0,WREG
                MOVWF           ZARGB0
                MOVFP           BARGB1,WREG
                MOVWF           ZARGB1

                CLRF            WREG,F          ; if AARG=0, return 0.0
                CPFSGT          AEXP
                GOTO            POW24AZERO

                MOVFP           FPFLAGS,WREG    ; save RND flag in ZARGB3
                MOVWF           ZARGB3
                BSF             FPFLAGS,RND     ; enable rounding

                CALL            LOG32           ; log(x)

                MOVFP           ZARGB2,WREG
                MOVWF           BEXP
                MOVFP           ZARGB0,WREG
                MOVWF           BARGB0
                MOVFP           ZARGB1,WREG
                MOVWF           BARGB1
                CLRF            BARGB2,F

                CALL            FPM32           ; y*log(x)

                TSTFSZ          WREG            ; test for overflow
                GOTO            DOMERR32

                BCF             FPFLAGS,RND     ; disable rounding

                CALL            EXP32           ; exp(y*log(x))

                BTFSS           ZARGB3,RND
                RETLW           0x00
```

```
                BSF             FPFLAGS,RND
                CALL            RND4032
                RETLW           0x00

POW24ONE        MOVLW           EXPBIAS
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                RETLW           0x00

POW24AZERO
                BTFSS           BARGB0,MSB      ; if x=0 and y<0, set overflow flag
                RETLW           0x00
                GOTO            SETFOV24

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate pow(x,y) = X**Y

;       Input:  32 bit floating point number X in AEXP, AARGB0, AARGB1, AARGB2 and
;               32 bit floating point number Y in BEXP, BARGB0, BARGB1, BARGB2.

;       Use:    CALL    POW32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2.

;       Result: AARG  <--  POW( AARG )

;       Testing on [1/26,26] from 70000 trials:

;               min     max     mean
;       Timing: 4280    5574    5168.4  clks

;               min     max     mean    rms
;       Error:  -0x270  0x209   8.94    92.21   nsb

;----------------------------------------------------------------------------------------------

;       The unavailability of extended precision routines for the 32 bit format
;       requires considerably more effort with more sophisticated pseudo extended
;       precision methods to control error propagation. Because the relative error
;       in the exponential function is proportional to the absolute error of its
;       argument, great care must be taken in any algorithm based on an exponential
;       identity. Such methods generally rely on extracting as much of the result
;       as an integer power of two as possible, followed by computations requiring
;       approximations over a relatively small interval. To that end, consider the
;       representation of the argument x given by

;                       x=f*2**e,  where  .5 <= f < 1.

;       The power function can then be expressed in the form

;                       x**y = 2**(y*log2(x)),

;       with the  base 2 log of x represented as

;               log2(x) =  log2(f*2**e) = e + log2(a) + log2(1+v),  v = (f-a)/a,

;       where a is chosen so that v is small. We choose a set of values of a defined
;       by a(k)=2**(-k/16), k=0,1,...16, and for a given f, the value of a(k) for
;       even k, nearest to f is chosen, resulting in an argument v to the natural
;       log function

;               log(1+v),  2**-(1/16)-1 < v < 2**(1/16)-1.
```

```
;       Since the numbers a(k) cannot be represented exactly in full precision, psuedo
;       extended precision evaluation of v is performed through the expansion

;               v = (f-a(k))/a(k) = (f-A(k)-f*C(k))/A(k),  C(k) = B(k)/A(k)

;       where a(k) = A(k)+B(k). The number A(k) is equal to a(k) rounded to machine
;       precision, and then B(k) is the difference computed in higher precision.
;       This method assures evaluation of v with a maximum relative error less than
;       1 ulp. A minimax approximation of the form

;               log(1+v) = v - .5*v**2 + (v**3)*(p(v)/q(v)),

;       with first degree polynomials p and q, followed by conversion to the required
;       function log2(1+v), leading to the result

;               log2(x) = e - k/16 + log2(1+v).

;       The product y*log2(x) is now carefully computed by reducing the number y into
;       a sum of two parts with one less than 1/16 and first evaluating small products
;       of similar magnitude and collecting terms. Each stage of this strategy is
;       followed by a similar reduction operation where the large part is an integer
;       plus a number of 16ths. The final form of the product is then expressed as an
;       integer plus a number of 16ths plus a number on the interval [-.0625,0],
;       leading to a final result expressed in the form

;               x**y = 2**(y(log2(x)) = (2**i)*(2**(-n/16))*(2**h),

;       where 2**h is evaluated by a minimax approximation of the form

;               (2**h)-1 = h + h*p(h),

;       with a second degree polynomial p.

POW32
                CLRF            AARGB3,W        ; clear NSB

                BTFSC           AARGB0,MSB      ; test if AARG < 0
                GOTO            DOMERR32

                CPFSGT          BEXP            ; if BARG=0, return 1.0
                GOTO            POW32ONE
                MOVFP           BEXP,WREG       ; save Y in CARG
                MOVWF           CEXP
                MOVFP           BARGB0,WREG
                MOVWF           CARGB0
                MOVFP           BARGB1,WREG
                MOVWF           CARGB1
                MOVFP           BARGB2,WREG
                MOVWF           CARGB2

                CLRF            WREG,F          ; if AARG=0, return 0.0
                CPFSGT          AEXP
                GOTO            POW32AZERO

                MOVFP           FPFLAGS,WREG    ; save RND flag in DARGB3
                MOVWF           DARGB3
                BSF             FPFLAGS,RND     ; enable rounding


;       evaluate log2(x)


                MOVFP           AEXP,WREG
                MOVPF           WREG,TMR0L
                MOVLW           EXPBIAS-1
                SUBWF           TMR0L,F
```

```
            MOVWF       AEXP

            MOVLW       0x01
            MOVWF       AARGB7

            MOVLW       0x09
            MOVWF       TEMPB0

            CALL        POW32GETA

            CALL        TALEB32

            TSTFSZ      WREG
            MOVFP       TEMPB0,AARGB7

            MOVLW       0x04
            ADDWF       AARGB7,W
            MOVWF       TEMPB0

            CALL        POW32GETA

            CALL        TALEB32

            TSTFSZ      WREG
            MOVFP       TEMPB0,AARGB7

            MOVLW       0x02
            ADDWF       AARGB7,W
            MOVWF       TEMPB0

            CALL        POW32GETA

            CALL        TALEB32

            TSTFSZ      WREG
            MOVFP       TEMPB0,AARGB7

            MOVLW       0x01
            MOVWF       TEMPB0

            CALL        POW32GETA

            CALL        TAGEB32

            MOVWF       TEMPB0
            CLRF        WREG,F
            CPFSGT      TEMPB0
            GOTO        POW32INCI
            MOVLW       0xFF
            MOVWF       AARGB7
POW32INCI
            INCF        AARGB7,F
            MOVPF       AARGB7,ZARGB0

            MOVFP       AEXP,WREG         ; DARG = X
            MOVWF       DEXP
            MOVPF       AARGB0,DARGB0
            MOVPF       AARGB1,DARGB1
            MOVPF       AARGB2,DARGB2

            MOVPF       AARGB7,TEMPB0
            CALL        POW32GETA
            CALL        FPS32

            MOVFP       AEXP,WREG         ; EARG = X-A1
            MOVWF       EEXP
```

```
                MOVPF           AARGB0,EARGB0
                MOVPF           AARGB1,EARGB1
                MOVPF           AARGB2,EARGB2

                MOVFP           DEXP,WREG
                MOVWF           AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                MOVPF           AARGB7,TEMPB0
                CALL            POW32GETD
                CALL            FPM32

;               TSTFSZ          AEXP
;               BTG             AARGB0,MSB

                MOVFP           EEXP,WREG
                MOVWF           BEXP
                MOVFP           EARGB0,WREG
                MOVWF           BARGB0
                MOVFP           EARGB1,WREG
                MOVWF           BARGB1
                MOVFP           EARGB2,WREG
                MOVWF           BARGB2

                CALL            FPA32           ; X - A1 - X * (A2/A1)

                MOVFP           ZARGB0,WREG
                MOVWF           TEMPB0
                CALL            POW32GETA
                CALL            FPD32

                MOVFP           AEXP,WREG       ; DARG = v = (X - A1 - X * (A2/A1))/A1
                MOVWF           DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                POLL132         LOG32BQ,1,0     ; Q(z)

                MOVFP           AEXP,WREG
                MOVPF           WREG,FEXP
                MOVPF           AARGB0,FARGB0
                MOVPF           AARGB1,FARGB1
                MOVPF           AARGB2,FARGB2

                MOVFP           DEXP,WREG
                MOVPF           WREG,AEXP
                MOVFP           DARGB0,AARGB0
                MOVFP           DARGB1,AARGB1
                MOVFP           DARGB2,AARGB2

                POL32           LOG32BP,1,0     ; P(z)

                MOVFP           FEXP,WREG
                MOVPF           WREG,BEXP
                MOVFP           FARGB0,WREG
                MOVPF           WREG,BARGB0
                MOVFP           FARGB1,WREG
                MOVPF           WREG,BARGB1
                MOVFP           FARGB2,WREG
                MOVPF           WREG,BARGB2

                CALL            FPD32           ; P(z)/Q(z)
```

```
        MOVFP       AEXP,WREG       ; save in CARG
        MOVPF       WREG,FEXP
        MOVPF       AARGB0,FARGB0
        MOVPF       AARGB1,FARGB1
        MOVPF       AARGB2,FARGB2

        MOVFP       DEXP,WREG
        MOVPF       WREG,BEXP
        MOVFP       DARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       DARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       DARGB2,WREG
        MOVPF       WREG,BARGB2

        MOVFP       DEXP,WREG
        MOVPF       WREG,AEXP
        MOVFP       DARGB0,AARGB0
        MOVFP       DARGB1,AARGB1
        MOVFP       DARGB2,AARGB2

        CALL        FPM32           ; z*z

        MOVFP       AEXP,WREG       ; save in EARG
        MOVPF       WREG,EEXP
        MOVPF       AARGB0,EARGB0
        MOVPF       AARGB1,EARGB1
        MOVPF       AARGB2,EARGB2

        MOVFP       FEXP,WREG       ; z*z*P(z)/Q(z)
        MOVPF       WREG,BEXP
        MOVFP       FARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       FARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       FARGB2,WREG
        MOVPF       WREG,BARGB2

        CALL        FPM32

        MOVFP       DEXP,WREG       ; z*(z*z*P(z)/Q(z))
        MOVPF       WREG,BEXP
        MOVFP       DARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       DARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       DARGB2,WREG
        MOVPF       WREG,BARGB2

        CALL        FPM32

        MOVFP       EEXP,WREG       ; -.5*z*z + z*(z*z*P(z)/Q(z))
        MOVPF       WREG,BEXP
        MOVFP       EARGB0,WREG
        MOVPF       WREG,BARGB0
        MOVFP       EARGB1,WREG
        MOVPF       WREG,BARGB1
        MOVFP       EARGB2,WREG
        MOVPF       WREG,BARGB2
        TSTFSZ      BEXP
        DECF        BEXP,F

        CALL        FPS32

        MOVFP       AEXP,WREG       ; save in EARG
        MOVWF       EEXP
```

```
        MOVPF           AARGB0,EARGB0
        MOVPF           AARGB1,EARGB1
        MOVPF           AARGB2,EARGB2

        MOVLW           0x7D            ; LOG2(e) - 1
        MOVWF           BEXP
        MOVLW           0x62
        MOVWF           BARGB0
        MOVLW           0xA8
        MOVWF           BARGB1
        MOVLW           0xED
        MOVWF           BARGB2

        CALL            FPM32

        MOVFP           EEXP,WREG
        MOVWF           BEXP
        MOVFP           EARGB0,WREG
        MOVWF           BARGB0
        MOVFP           EARGB1,WREG
        MOVWF           BARGB1
        MOVFP           EARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32

        MOVFP           AEXP,WREG       ; save in EARG
        MOVWF           EEXP
        MOVPF           AARGB0,EARGB0
        MOVPF           AARGB1,EARGB1
        MOVPF           AARGB2,EARGB2

        MOVFP           DEXP,WREG
        MOVWF           AEXP
        MOVFP           DARGB0,AARGB0
        MOVFP           DARGB1,AARGB1
        MOVFP           DARGB2,AARGB2

        MOVLW           0x7D            ; LOG2(e) - 1
        MOVWF           BEXP
        MOVLW           0x62
        MOVWF           BARGB0
        MOVLW           0xA8
        MOVWF           BARGB1
        MOVLW           0xED
        MOVWF           BARGB2

        CALL            FPM32

        MOVFP           EEXP,WREG
        MOVWF           BEXP
        MOVFP           EARGB0,WREG
        MOVWF           BARGB0
        MOVFP           EARGB1,WREG
        MOVWF           BARGB1
        MOVFP           EARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32

        MOVFP           DEXP,WREG
        MOVWF           BEXP
        MOVFP           DARGB0,WREG
        MOVWF           BARGB0
        MOVFP           DARGB1,WREG
        MOVWF           BARGB1
```

```
        MOVFP           DARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32

        MOVFP           AEXP,WREG       ; save z in EARG
        MOVWF           EEXP
        MOVPF           AARGB0,EARGB0
        MOVPF           AARGB1,EARGB1
        MOVPF           AARGB2,EARGB2

        MOVFP           ZARGB0,AARGB1   ; w = - i / 16
        CLRF            AARGB0,F
        CALL            FLO1624
        CLRF            AARGB2,F
        TSTFSZ          AEXP
        BSF             AARGB0,MSB
        MOVLW           0x04
        TSTFSZ          AEXP
        SUBWF           AEXP,F

        MOVFP           AEXP,WREG       ; save w in BARG
        MOVWF           BEXP
        MOVPF           AARGB0,BARGB0
        MOVPF           AARGB1,BARGB1
        MOVPF           AARGB2,BARGB2

        MOVFP           TMR0L,AARGB1    ; w = w + e
        CLRF            AARGB0,F
        BTFSC           AARGB1,MSB
        COMF            AARGB0,F
        CALL            FLO1624
        CLRF            AARGB2,F
        CALL            FPA32

        MOVFP           AEXP,WREG       ; save w in FARG
        MOVWF           FEXP
        MOVPF           AARGB0,FARGB0
        MOVPF           AARGB1,FARGB1
        MOVPF           AARGB2,FARGB2

        MOVFP           CEXP,WREG
        MOVWF           AEXP
        MOVFP           CARGB0,AARGB0
        MOVFP           CARGB1,AARGB1
        MOVFP           CARGB2,AARGB2

        CALL            REDUCE          ; AARG = Yb, DARG = Ya

        MOVFP           FEXP,WREG
        MOVWF           BEXP
        MOVFP           FARGB0,WREG
        MOVWF           BARGB0
        MOVFP           FARGB1,WREG
        MOVWF           BARGB1
        MOVFP           FARGB2,WREG
        MOVWF           BARGB2

        CALL            FPM32

        MOVFP           AEXP,WREG       ; save w * Yb in GARG
        MOVWF           GEXP
        MOVPF           AARGB0,GARGB0
        MOVPF           AARGB1,GARGB1
        MOVPF           AARGB2,GARGB2
```

```
        MOVFP       EEXP,WREG
        MOVWF       AEXP
        MOVFP       EARGB0,AARGB0
        MOVFP       EARGB1,AARGB1
        MOVFP       EARGB2,AARGB2

        MOVFP       CEXP,WREG
        MOVWF       BEXP
        MOVFP       CARGB0,WREG
        MOVWF       BARGB0
        MOVFP       CARGB1,WREG
        MOVWF       BARGB1
        MOVFP       CARGB2,WREG
        MOVWF       BARGB2

        CALL        FPM32

        MOVFP       GEXP,WREG
        MOVWF       BEXP
        MOVFP       GARGB0,WREG
        MOVWF       BARGB0
        MOVFP       GARGB1,WREG
        MOVWF       BARGB1
        MOVFP       GARGB2,WREG
        MOVWF       BARGB2

        CALL        FPA32

        MOVFP       DEXP,WREG        ; move Ya to CARG
        MOVWF       CEXP
        MOVFP       DARGB0,WREG
        MOVWF       CARGB0
        MOVFP       DARGB1,WREG
        MOVWF       CARGB1
        MOVFP       DARGB2,WREG
        MOVWF       CARGB2

        CALL        REDUCE           ; AARG = Fb, DARG = Fa

        MOVFP       AEXP,WREG        ; save Fb in EARG
        MOVWF       EEXP
        MOVPF       AARGB0,EARGB0
        MOVPF       AARGB1,EARGB1
        MOVPF       AARGB2,EARGB2

        MOVFP       FEXP,WREG
        MOVWF       BEXP
        MOVFP       FARGB0,WREG
        MOVWF       BARGB0
        MOVFP       FARGB1,WREG
        MOVWF       BARGB1
        MOVFP       FARGB2,WREG
        MOVWF       BARGB2

        MOVFP       CEXP,WREG
        MOVWF       AEXP
        MOVFP       CARGB0,AARGB0
        MOVFP       CARGB1,AARGB1
        MOVFP       CARGB2,AARGB2

        CALL        FPM32

        MOVFP       DEXP,WREG
        MOVWF       BEXP
        MOVFP       DARGB0,WREG
        MOVWF       BARGB0
```

```
        MOVFP           DARGB1,WREG
        MOVWF           BARGB1
        MOVFP           DARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32

        CALL            REDUCE          ; AARG = Gb, DARG = Ga

        MOVFP           EEXP,WREG
        MOVWF           BEXP
        MOVFP           EARGB0,WREG
        MOVWF           BARGB0
        MOVFP           EARGB1,WREG
        MOVWF           BARGB1
        MOVFP           EARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32

        MOVFP           DEXP,WREG       ; move Ga to CARG
        MOVWF           CEXP
        MOVFP           DARGB0,WREG
        MOVWF           CARGB0
        MOVFP           DARGB1,WREG
        MOVWF           CARGB1
        MOVFP           DARGB2,WREG
        MOVWF           CARGB2

        CALL            REDUCE          ; AARG = Hb, DARG = Ha

        MOVFP           AEXP,WREG       ; save Hb in EARG
        MOVWF           EEXP
        MOVPF           AARGB0,EARGB0
        MOVPF           AARGB1,EARGB1
        MOVPF           AARGB2,EARGB2

        MOVFP           CEXP,WREG
        MOVWF           AEXP
        MOVFP           CARGB0,AARGB0
        MOVFP           CARGB1,AARGB1
        MOVFP           CARGB2,AARGB2

        MOVFP           DEXP,WREG
        MOVWF           BEXP
        MOVFP           DARGB0,WREG
        MOVWF           BARGB0
        MOVFP           DARGB1,WREG
        MOVWF           BARGB1
        MOVFP           DARGB2,WREG
        MOVWF           BARGB2

        CALL            FPA32
        MOVLW           0x04
        TSTFSZ          AEXP
        ADDWF           AEXP,F

        BCF             FPFLAGS,RND
        CALL            INT3224
        BSF             FPFLAGS,RND

        MOVFP           AARGB1,WREG     ; test for overflow
        BTFSC           AARGB1,MSB
        NEGW            WREG,F
        BTFSC           WREG,4          ; is |e| < 2048 ?
        GOTO            DOMERR32
```



```
            MOVPF       AARGB1,ZARGB0   ; save e in ZARGB0,ZARGB1
            MOVPF       AARGB2,ZARGB1

            BTFSC       EARGB0,MSB
            GOTO        POW32HBOK

            CLRF        WREG,F
            INCF        ZARGB1,F
            ADDWFC      ZARGB0,F

            MOVFP       EEXP,WREG
            MOVWF       AEXP
            MOVFP       EARGB0,AARGB0
            MOVFP       EARGB1,AARGB1
            MOVFP       EARGB2,AARGB2

            MOVLW       0x7B
            MOVWF       BEXP
            MOVLW       0x80
            MOVWF       BARGB0
            CLRF        BARGB1,F
            CLRF        BARGB2,F
            CALL        FPA32

            MOVFP       AEXP,WREG       ; save Hb in EARG
            MOVWF       EEXP
            MOVPF       AARGB0,EARGB0
            MOVPF       AARGB1,EARGB1
            MOVPF       AARGB2,EARGB2

POW32HBOK
            MOVFP       EEXP,WREG
            MOVWF       AEXP
            MOVFP       EARGB0,AARGB0
            MOVFP       EARGB1,AARGB1
            MOVFP       EARGB2,AARGB2

            MOVFP       AEXP,WREG
            MOVWF       DEXP
            MOVPF       AARGB0,DARGB0
            MOVPF       AARGB1,DARGB1
            MOVPF       AARGB2,DARGB2

            BSF         FPFLAGS,RND

            POL32       EXP232,2,0      ; z = 2**Hb - 1

            MOVFP       DEXP,WREG
            MOVWF       BEXP
            MOVFP       DARGB0,WREG
            MOVWF       BARGB0
            MOVFP       DARGB1,WREG
            MOVWF       BARGB1
            MOVFP       DARGB2,WREG
            MOVWF       BARGB2

            CALL        FPM32

            MOVFP       ZARGB0,WREG
            MOVWF       ZARGB2
            MOVFP       ZARGB1,WREG
            MOVWF       ZARGB3

            CLRF        GARGB3,F
            BTFSS       ZARGB0,MSB
```

```
                INCF            GARGB3,F
                BCF             _C
                RRCF            ZARGB2,F
                RRCF            ZARGB3,F
                RRCF            ZARGB2,F
                RRCF            ZARGB3,F
                RRCF            ZARGB2,F
                RRCF            ZARGB3,F
                RRCF            ZARGB2,F
                RRCF            ZARGB3,F
                BTFSC           ZARGB0,MSB
                INCF            ZARGB3,F
                MOVFP           ZARGB3,WREG

                ADDWF           GARGB3,F

                MOVFP           GARGB3,WREG
                MULLW           0x10
                MOVLW           0x10
                BTFSC           GARGB3,MSB
                SUBWF           PRODH,F

                MOVFP           ZARGB1,WREG
                SUBWF           PRODL,F
                MOVFP           ZARGB0,WREG
                SUBWFB          PRODH,F

                MOVFP           PRODL,WREG
                MOVWF           ZARGB3
                MOVWF           TEMPB0

                CALL            POW32GETA

                CALL            FPM32

                MOVFP           ZARGB3,WREG
                MOVWF           TEMPB0
                CALL            POW32GETC
                TSTFSZ          BEXP
                INCF            BEXP,F

                CALL            FPA32

                MOVFP           ZARGB3,WREG
                MOVWF           TEMPB0
                CALL            POW32GETA

                BTFSS           DARGB3,RND
                BCF             FPFLAGS,RND

                CALL            FPA32

                MOVFP           GARGB3,WREG
                TSTFSZ          AEXP
                ADDWF           AEXP,F

                RETLW           0x00

POW32ONE        MOVLW           EXPBIAS
                MOVWF           AEXP
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                RETLW           0x00
```



```
POW32AZERO
                BTFSS           BARGB0,MSB      ; if x=0 and y<0, set overflow flag
                RETLW           0x00
                GOTO            SETFOV32

;**********************************************************************************************

REDUCE
                MOVFP           AEXP,WREG       ; BARG = X
                MOVWF           BEXP
                MOVPF           AARGB0,BARGB0
                MOVPF           AARGB1,BARGB1
                MOVPF           AARGB2,BARGB2

                MOVLW           0x04
                ADDWF           AEXP,F
                CALL            FLOOR32
                MOVLW           0x04
                TSTFSZ          AEXP
                SUBWF           AEXP,F

                MOVFP           AEXP,WREG       ; DARG = Xa
                MOVWF           DEXP
                MOVPF           AARGB0,DARGB0
                MOVPF           AARGB1,DARGB1
                MOVPF           AARGB2,DARGB2

                BTG             AARGB0,MSB

                CALL            FPA32           ; AARG = Xb

                RETLW           0x00


;**********************************************************************************************

POW32GETA
                MOVLW           HIGH (POW32TABLEA); access table for A
                MOVWF           TBLPTRH
                RLNCF           TEMPB0,W
                ADDLW           LOW (POW32TABLEA)
                MOVWF           TBLPTRL
                BTFSC           _C
                INCF            TBLPTRH,F
                TABLRD          0,1,BEXP
                TLRD            1,BEXP
                TABLRD          0,1,BARGB0
                TLRD            1,BARGB1
                TABLRD          0,0,BARGB2

                RETLW           0x00


POW32GETC
                MOVLW           HIGH (POW32TABLEC); access table for A
                MOVWF           TBLPTRH
                RLNCF           TEMPB0,W
                ADDLW           LOW (POW32TABLEC)
                MOVWF           TBLPTRL
                BTFSC           _C
                INCF            TBLPTRH,F
                TABLRD          0,1,BEXP
                TLRD            1,BEXP
                TABLRD          0,1,BARGB0
                TLRD            1,BARGB1
                TABLRD          0,0,BARGB2
```

```
                RETLW           0x00

POW32GETD
                MOVLW           HIGH (POW32TABLED); access table for A
                MOVWF           TBLPTRH
                RLNCF           TEMPB0,W
                ADDLW           LOW (POW32TABLED)
                MOVWF           TBLPTRL
                BTFSC           _C
                INCF            TBLPTRH,F
                TABLRD          0,1,BEXP
                TLRD            1,BEXP
                TABLRD          0,1,BARGB0
                TLRD            1,BARGB1
                TABLRD          0,0,BARGB2

                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       minimax rational coefficients for log2(1+z)/z on [-.0625,.0625]

LOG232P0        EQU             0x81            ; LOG232P0 = .73551298732E+1******
LOG232P00       EQU             0x19
LOG232P01       EQU             0xB1
LOG232P02       EQU             0xA6

LOG232P1        EQU             0x80            ; LOG232P1 = .40900513905E+1
LOG232P10       EQU             0x57
LOG232P11       EQU             0x5A
LOG232P12       EQU             0x68

LOG232P2        EQU             0x7C            ; LOG232P1 = .40900513905E+1
LOG232P20       EQU             0x24
LOG232P21       EQU             0x58
LOG232P22       EQU             0x44

LOG232Q0        EQU             0x80            ; LOG232Q0 = .50982159260E+1
LOG232Q00       EQU             0x55
LOG232Q01       EQU             0x10
LOG232Q02       EQU             0xA7

LOG232Q1        EQU             0x80            ; LOG232Q1 = .53849258895E+1
LOG232Q10       EQU             0x7F
LOG232Q11       EQU             0xCD
LOG232Q12       EQU             0xD0

LOG232Q2        EQU             0x7F            ; LOG232Q2 = 1.0
LOG232Q20       EQU             0x00
LOG232Q21       EQU             0x00
LOG232Q22       EQU             0x00

;----------------------------------------------------------------------------------------------

;       minimax rational approximationz-.5*z*z+z*(z*z*P(z)/Q(z))

LOG32AP0        EQU             0x7D            ; LOG32AP0 = .4165382203229886
LOG32AP00       EQU             0x55
LOG32AP01       EQU             0x44
LOG32AP02       EQU             0x7F

LOG32AP1        EQU             0x79            ; LOG32AP1 = .02090135006173772
LOG32AP10       EQU             0x2B
LOG32AP11       EQU             0x39
LOG32AP12       EQU             0x4F
```



```
LOG32AQ0        EQU             0x7F            ; LOG32AQ0 = 1.249615003891314
LOG32AQ00       EQU             0x1F
LOG32AQ01       EQU             0xF3
LOG32AQ02       EQU             0x62

LOG32AQ1        EQU             0x7F            ; LOG32AQ1 = 1.0
LOG32AQ10       EQU             0x00
LOG32AQ11       EQU             0x00
LOG32AQ12       EQU             0x00

;----------------------------------------------------------------------------------------------

;       minimax rational approximationz-.5*z*z+z*(z*z*P(z)/Q(z))

LOG32BP0        EQU             0x7D            ; LOG32BP0 = .4165382203229886****
LOG32BP00       EQU             0x55
LOG32BP01       EQU             0x57
LOG32BP02       EQU             0x8F

LOG32BP1        EQU             0x79            ; LOG32BP1 = .02090135006173772
LOG32BP10       EQU             0x2A
LOG32BP11       EQU             0x72
LOG32BP12       EQU             0xAE

LOG32BQ0        EQU             0x7F            ; LOG32BQ0 = 1.249615003891314
LOG32BQ00       EQU             0x20
LOG32BQ01       EQU             0x01
LOG32BQ02       EQU             0xAB

LOG32BQ1        EQU             0x7F            ; LOG32BQ1 = 1.0
LOG32BQ10       EQU             0x00
LOG32BQ11       EQU             0x00
LOG32BQ12       EQU             0x00

;----------------------------------------------------------------------------------------------

;       second degree minimax polynomial coefficients for 2**(x)-1 on [-.0625,0]

EXP2320         EQU             0x7E            ; EXP2320 = .693146757796576
EXP23200        EQU             0x31
EXP23201        EQU             0x72
EXP23202        EQU             0x11

EXP2321         EQU             0x7C            ; EXP2321 = .2401853543026017
EXP23210        EQU             0x75
EXP23211        EQU             0xF3
EXP23212        EQU             0x26

EXP2322         EQU             0x7A            ; EXP2322 = .05436330184989159
EXP23220        EQU             0x5E
EXP23221        EQU             0xAC
EXP23222        EQU             0x0E
;----------------------------------------------------------------------------------------------

;       second degree minimax polynomial coefficients for 2**(x)-1 on [-.0625,0]

EXP232A0        EQU             0x7E            ; EXP232A0 = .693146757796576****
EXP232A00       EQU             0x31
EXP232A01       EQU             0x72
EXP232A02       EQU             0x11

EXP232A1        EQU             0x7C            ; EXP232A1 = .2401853543026017
EXP232A10       EQU             0x75
EXP232A11       EQU             0xF3
EXP232A12       EQU             0x26
```

```
EXP232A2        EQU         0x7A        ; EXP232A2 = .05436330184989159
EXP232A20       EQU         0x5E
EXP232A21       EQU         0xAC
EXP232A22       EQU         0x0E

;----------------------------------------------------------------------------------------------

POW32TABLEA
                DATA    0x7F00
                DATA    0x0000
                DATA    0x7E75
                DATA    0x257D
                DATA    0x7E6A
                DATA    0xC0C7
                DATA    0x7E60
                DATA    0xCCDF
                DATA    0x7E57
                DATA    0x44FD
                DATA    0x7E4E
                DATA    0x248C
                DATA    0x7E45
                DATA    0x672A
                DATA    0x7E3D
                DATA    0x08A4
                DATA    0x7E35
                DATA    0x04F3
                DATA    0x7E2D
                DATA    0x583F
                DATA    0x7E25
                DATA    0xFED7
                DATA    0x7E1E
                DATA    0xF532
                DATA    0x7E18
                DATA    0x37F0
                DATA    0x7E11
                DATA    0xC3D3
                DATA    0x7E0B
                DATA    0x95C2
                DATA    0x7E05
                DATA    0xAAC3
                DATA    0x7E00
                DATA    0x0000

POW32TABLEC
                DATA    0x0000
                DATA    0x0000
                DATA    0x6329
                DATA    0x2436
                DATA    0x63C1
                DATA    0x16DE
                DATA    0x639E
                DATA    0xAB59
                DATA    0x64D4
                DATA    0xA58A
                DATA    0x6328
                DATA    0xFC24
                DATA    0x630A
                DATA    0xA837
                DATA    0x65C1
                DATA    0x4FE8
                DATA    0x644F
                DATA    0xE77A
                DATA    0x63AD
                DATA    0xEAF6
                DATA    0x65AC
```

```
                DATA    0x9D5E
                DATA    0x6541
                DATA    0x2342
                DATA    0x6523
                DATA    0x1B71
                DATA    0x6567
                DATA    0x5624
                DATA    0x63E0
                DATA    0xABA1
                DATA    0x654F
                DATA    0x9891
                DATA    0x0000
                DATA    0x0000

POW32TABLED
                DATA    0x0000
                DATA    0x0000
                DATA    0x63B0
                DATA    0xA146
                DATA    0x6352
                DATA    0x90BE
                DATA    0x6334
                DATA    0xB0DA
                DATA    0x647C          ; +1 647CE183
                DATA    0xE182
                DATA    0x63D1
                DATA    0xDAF2
                DATA    0x63B3
                DATA    0xD0E5
                DATA    0x6602
                DATA    0xE5A2
                DATA    0x6593          ; -1 659302AE
                DATA    0x02AF
                DATA    0x6400
                DATA    0x6C56
                DATA    0x6605
                DATA    0x1AA9
                DATA    0x669B
                DATA    0x85F2
                DATA    0x6689
                DATA    0x2801
                DATA    0x66CB
                DATA    0x2482
                DATA    0x644E          ; +1 644E0611
                DATA    0x0610
                DATA    0x66C6
                DATA    0xCB6A
                DATA    0x0000
                DATA    0x0000

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate floor(x)

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Use:    CALL    FLOOR32

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <--  FLOOR( AARG )
```

```
;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 30      45      35.2    clks

;               min     max     mean    rms
;       Error:  0x00    0x00    0.0     0.0     nsb

;----------------------------------------------------------------------------------------------

FLOOR32
                CLRF            AARGB3,W        ; test for zero argument
                CPFSGT          AEXP
                RETLW           0x00

                MOVFP           AARGB0,AARGB4   ; save mantissa
                MOVFP           AARGB1,AARGB5
                MOVFP           AARGB2,AARGB6

                MOVLW           EXPBIAS
                SUBWF           AEXP,W
                BTFSC           WREG,MSB
                GOTO            FLOOR32ZERO

                SUBLW           0x18-1
                MOVWF           TEMPB0          ; save number of zero bits in TEMPB0

                BTFSC           WREG,LSB+1+3    ; divide by eight
                GOTO            FLOOR32MASKH
                BTFSC           WREG,LSB+3
                GOTO            FLOOR32MASKM

FLOOR32MASKL
                CLRF            TBLPTRH,F

                MOVFP           TEMPB0,WREG     ; get remainder for mask pointer
                ANDLW           0x07

                ADDLW           LOW (FLOOR32MASKTABLE)
                MOVWF           TBLPTRL
                MOVLW           HIGH (FLOOR32MASKTABLE); access table for F0
                ADDWFC          TBLPTRH,F
                TABLRD          0,1,WREG
                TLRD            0,WREG

                ANDWF           AARGB2,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVFP           AARGB6,WREG
                CPFSEQ          AARGB2
                GOTO            FLOOR32RNDL
                RETLW           0x00

FLOOR32RNDL
                COMF            AARGB7,W
                INCF            WREG,F
                ADDWF           AARGB2,F
                CLRF            WREG,F
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRCF            AARGB0,F
                RRCF            AARGB1,F
```

```
                RRCF            AARGB2,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32MASKM
                CLRF            TBLPTRH,F

                MOVFP           TEMPB0,WREG
                ANDLW           0x07

                ADDLW           LOW (FLOOR32MASKTABLE)
                MOVWF           TBLPTRL
                MOVLW           HIGH (FLOOR32MASKTABLE); access table for F0
                ADDWFC          TBLPTRH,F
                TABLRD          0,1,WREG
                TLRD            0,WREG

                ANDWF           AARGB1,F
                CLRF            AARGB2,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00

                MOVWF           AARGB7
                MOVFP           AARGB6,WREG
                CPFSEQ          AARGB2
                GOTO            FLOOR32RNDM
                MOVFP           AARGB5,WREG
                CPFSEQ          AARGB1
                GOTO            FLOOR32RNDM
                RETLW           0x00

FLOOR32RNDM
                COMF            AARGB7,W
                INCF            WREG,F
                ADDWF           AARGB1,F
                CLRF            WREG,F
                ADDWFC          AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRCF            AARGB0,F
                RRCF            AARGB1,F
                RRCF            AARGB2,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32MASKH
                CLRF            TBLPTRH,F

                MOVFP           TEMPB0,WREG
                ANDLW           0x07

                ADDLW           LOW (FLOOR32MASKTABLE)
                MOVWF           TBLPTRL
                MOVLW           HIGH (FLOOR32MASKTABLE); access table for F0
                ADDWFC          TBLPTRH,F
                TABLRD          0,1,WREG
                TLRD            0,WREG

                ANDWF           AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                BTFSS           AARGB0,MSB      ; if negative, round down
                RETLW           0x00
```

```
                MOVWF           AARGB7
                MOVFP           AARGB6,WREG
                CPFSEQ          AARGB2
                GOTO            FLOOR32RNDH
                MOVFP           AARGB5,WREG
                CPFSEQ          AARGB1
                GOTO            FLOOR32RNDH
                MOVFP           AARGB4,WREG
                CPFSEQ          AARGB0
                GOTO            FLOOR32RNDH
                RETLW           0x00

FLOOR32RNDH
                COMF            AARGB7,W
                INCF            WREG,F
                ADDWF           AARGB0,F
                BTFSS           _C              ; has rounding caused carryout?
                RETLW           0x00
                RRCF            AARGB0,F
                RRCF            AARGB1,F
                INCFSZ          AEXP,F          ; check for overflow
                RETLW           0x00
                GOTO            SETFOV32

FLOOR32ZERO
                BTFSC           AARGB0,MSB
                GOTO            FLOOR32MINUSONE
                CLRF            AEXP,F
                CLRF            AARGB0,F
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                RETLW           0x00

FLOOR32MINUSONE
                MOVLW           0x7F
                MOVWF           AEXP
                MOVLW           0x80
                MOVWF           AARGB0
                CLRF            AARGB1,F
                CLRF            AARGB2,F
                RETLW           0x00

;----------------------------------------------------------------------------------------------

;       table for least significant byte requiring masking, using pointer from
;       the remainder of the number of zero bits divided by eight.

FLOOR32MASKTABLE
                DATA            0xFF
                DATA            0xFE
                DATA            0xFC
                DATA            0xF8
                DATA            0xF0
                DATA            0xE0
                DATA            0xC0
                DATA            0x80
                DATA            0x00

;**********************************************************************************************
;**********************************************************************************************

;       Evaluate rand(x)

;       Input:  32 bit initial integer seed in RANDB0, RANDB1, RANDB2, RANDB3

;       Use:    CALL    RAND32
```

```
;       Output:  32 bit random integer in RANDB0, RANDB1, RANDB2, RANDB3

;       Result:  RAND  <--  RAND32( RAND )

;       Timing:  4+6+2+90+15 = 117 clks

;----------------------------------------------------------------------------------------------

;       Linear congruential random number generator

;               X <- (a * X + c) mod m

;       The calculation is performed exactly, with multiplier a, increment c, and
;       modulus m, selected to achieve high ratings from standard spectral tests.
;       The dedicated storage in RANDBx retains the current number in the sequence
;       and is not used by any other routine in the library. The initial seed, X0,
;       is arbitrary and must be placed in RANDBx.

RAND32
                MOVFP           RANDB0,AARGB0
                MOVFP           RANDB1,AARGB1
                MOVFP           RANDB2,AARGB2
                MOVFP           RANDB3,AARGB3

                MOVLW           0x0D            ; multiplier a = 1664525
                MOVWF           BARGB2
                MOVLW           0x66
                MOVWF           BARGB1
                MOVLW           0x19
                MOVWF           BARGB0

                CALL            FXM3224U

                MOVLW           0x01            ; increment c = 1
                ADDWF           AARGB6,F
                CLRF            WREG,F
                ADDWFC          AARGB5,F
                ADDWFC          AARGB4,F
                ADDWFC          AARGB3,F
                ADDWFC          AARGB2,F
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F

                MOVPF           AARGB3,RANDB0   ; modulus m = 2**32
                MOVPF           AARGB4,RANDB1
                MOVPF           AARGB5,RANDB2
                MOVPF           AARGB6,RANDB3

                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A < B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TALTB32

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:
```

```
;        Result: if A < B TRUE, WREG = 0x01
;                if A < B FALSE, WREG = 0x00

;                min      max      mean
;        Timing: 8        33       11.6    clks

TALTB32          MOVFP             AARGB0,WREG
                 XORWF             BARGB0,W
                 BTFSC             WREG,MSB
                 GOTO              TALTB32O

                 BTFSC             AARGB0,MSB
                 GOTO              TALTB32N

TALTB32P         MOVFP             AEXP,WREG
                 SUBWF             BEXP,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             AARGB0,WREG
                 SUBWF             BARGB0,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             AARGB1,WREG
                 SUBWF             BARGB1,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             AARGB2,WREG
                 SUBWF             BARGB2,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01
                 RETLW             0x00

TALTB32N         MOVFP             BEXP,WREG
                 SUBWF             AEXP,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             BARGB0,WREG
                 SUBWF             AARGB0,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             BARGB1,WREG
                 SUBWF             AARGB1,W
                 BTFSS             _C
                 RETLW             0x00
                 BTFSS             _Z
                 RETLW             0x01

                 MOVFP             BARGB2,WREG
                 SUBWF             AARGB2,W
```

```
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TALTB32O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A <= B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TALEB32

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A <= B TRUE, WREG = 0x01
;               if A <= B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       31      11.6    clks

TALEB32         MOVFP           AARGB0,WREG
                XORWF           BARGB0,W
                BTFSC           WREG,MSB
                GOTO            TALEB32O

                BTFSC           AARGB0,MSB
                GOTO            TALEB32N

TALEB32P        MOVFP           AEXP,WREG
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB2,WREG
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01
```

```
TALEB32N        MOVFP           BEXP,WREG
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB2,WREG
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TALEB32O        BTFSS           BARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A > B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAGTB32

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A > B TRUE, WREG = 0x01
;               if A > B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       33      11.6    clks

TAGTB32         MOVFP           BARGB0,WREG
                XORWF           AARGB0,W
                BTFSC           WREG,MSB
                GOTO            TAGTB32O

                BTFSC           BARGB0,MSB
                GOTO            TAGTB32N

TAGTB32P        MOVFP           BEXP,WREG
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
```



```
                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB2,WREG
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB32N        MOVFP           AEXP,WREG
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB2,WREG
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01
                RETLW           0x00

TAGTB32O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A >= B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAGEB32

;       Output: logical result in WREG
```

```
;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A >= B TRUE, WREG = 0x01
;               if A >= B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 8       31      11.6    clks

TAGEB32         MOVFP           BARGB0,WREG
                XORWF           AARGB0,W
                BTFSC           WREG,MSB
                GOTO            TAGEB32O

                BTFSC           BARGB0,MSB
                GOTO            TAGEB32N

TAGEB32P        MOVFP           BEXP,WREG
                SUBWF           AEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB0,WREG
                SUBWF           AARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB1,WREG
                SUBWF           AARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           BARGB2,WREG
                SUBWF           AARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB32N        MOVFP           AEXP,WREG
                SUBWF           BEXP,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB0,WREG
                SUBWF           BARGB0,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB1,WREG
                SUBWF           BARGB1,W
                BTFSS           _C
                RETLW           0x00
                BTFSS           _Z
                RETLW           0x01

                MOVFP           AARGB2,WREG
```

```
                SUBWF           BARGB2,W
                BTFSS           _C
                RETLW           0x00
                RETLW           0x01

TAGEB32O        BTFSS           AARGB0,MSB
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A == B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TAEQB32

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A == B TRUE, WREG = 0x01
;               if A == B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 4       14      5.9     clks

TAEQB32         MOVFP           AEXP,WREG
                CPFSEQ          BEXP
                RETLW           0x00
                MOVFP           AARGB0,WREG
                CPFSEQ          BARGB0
                RETLW           0x00
                MOVFP           AARGB1,WREG
                CPFSEQ          BARGB1
                RETLW           0x00
                MOVFP           AARGB2,WREG
                CPFSEQ          BARGB2
                RETLW           0x00
                RETLW           0x01

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Relation A =! B

;       Input:  32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2
;               32 bit floating point number in BEXP, BARGB0, BARGB1, BARGB2

;       Use:    CALL    TANEB32

;       Output: logical result in WREG

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;       Result: if A =! B TRUE, WREG = 0x01
;               if A =! B FALSE, WREG = 0x00

;               min     max     mean
;       Timing: 4       14      5.9     clks

TANEB32         MOVFP           AEXP,WREG
                CPFSEQ          BEXP
                RETLW           0x01
```

```
                MOVFP           AARGB0,WREG
                CPFSEQ          BARGB0
                RETLW           0x01
                MOVFP           AARGB1,WREG
                CPFSEQ          BARGB1
                RETLW           0x01
                MOVFP           AARGB2,WREG
                CPFSEQ          BARGB2
                RETLW           0x01
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************

;       Nearest neighbor rounding

;       Input:  40 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2, AARGB3

;       Use:    CALL    RND4032

;       Output: 32 bit floating point number in AEXP, AARGB0, AARGB1, AARGB2

;       Result: AARG  <-- RND( AARG )

;       Testing on [-MAXNUM,MAXNUM] from 100000 trials:

;               min     max     mean
;       Timing: 3       23              clks

;               min     max     mean
;       Error:  0       0       0       nsb

;----------------------------------------------------------------------------------------------

RND4032
                BTFSS           AARGB3,MSB      ; is NSB < 0x80?
                RETLW           0x00

                BSF             _C              ; set carry for rounding
                MOVLW           0x80
                CPFSGT          AARGB3
                RRCF            AARGB2,W        ; select even if NSB = 0x80

                MOVPF           AARGB0,SIGN     ; save sign
                BSF             AARGB0,MSB      ; make MSB explicit

                CLRF            WREG,F          ; round
                ADDWFC          AARGB2,F
                ADDWFC          AARGB1,F
                ADDWFC          AARGB0,F

                BTFSS           _C              ; has rounding caused carryout?
                GOTO            RND4032OK
                RRCF            AARGB0,F        ; if so, right shift
                RRCF            AARGB1,F
                RRCF            AARGB2,F
                INFSNZ          EXP, F          ; test for floating point overflow
                GOTO            SETFOV32
RND4032OK
                BTFSS           SIGN,MSB
                BCF             AARGB0,MSB      ; clear sign bit if positive
                RETLW           0x00

;**********************************************************************************************
;**********************************************************************************************
```

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc. USA
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.