A Collection of Functions for the TMS320C30

APPLICATION REPORT: SPRA117

Gary Sitton
Gaslight Software

Digital Signal Processing Solutions

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

A Collection of Functions for the TMS320C30

Abstract

This book presents a collection of efficient machine language programs for advanced applications with the TMS320C30 family of digital processors. These programs include the following categories:

- Normal precision floating point math functions
- Extended precision floating point math functions
- Integer arithmetic routines
- Vector utility routines
- Radix 2 FFT routines
- Linear algebra routines

The names and short descriptions of these routine categories (and special notations) are included.

The book contains detailed information about:

- Extended vs. Normal Precision
- Program utilization
- Function approximation techniques
- Math function details
- Integer arithmetic program details
- Vector utility routines
- Ftp routines

- Linear algebra routines

The book concludes with a list of references and an appendix of source code for the described routines.

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to dsph@ti.com. Questions receive prompt attention and are usually answered within one business day.

Introduction

This report presents a collection of efficient machine language programs for advanced applications with the TMS320C30. These programs provide basic math and transcendental functions. Other routines include vector functions, FFTs and linear algebra.

Library Overview

The set of programs fall into six categories:

I. Normal precision floating point math functions,
II. Extended precision floating point math functions,
III. Integer arithmetic routines,
IV. Vector utility routines,
V. Radix 2 FFT routines, and
VI. Linear algebra routines.

Categories I and II are programs which implement a minimal set of elementary mathematical functions for advanced applications. In these categories, the functions **FPINV** and **SQRT** are improved versions of the programs in the *TMS320C3x User's Guide* [1]. In category III, **IMULT** and **IDIV** are improved versions of the programs **EXTMPY** and **DIVI** in [1]. In category IV, ***FMIEEE** and ***TOIEE** are array versions of the **TOIEEE** and **FMIEEE** scalar programs from the User's Guide.

The names and short descriptions of these routines use some special notation:

Categories I and II:	**xd** — indicates that the relative accuracy of the implemented function is x decimal digits.
Categories IV and VI:	* — program name prefix stands for M or R.
	M — selects the memory based parameter entry point.
	R — selects the register based parameter entry point.
Categories II and VI:	**X** — indicates the extended precision program version.

Consult the program source listings for more details.

The following are brief descriptions of the programs by category:

I. Normal floating-point (32-bit) math functions (**$MATH.ASM**):

A.	**SIN**	—computes a 7d sine(x) for all x in radians.
B.	**COS**	—computes a 7d cosine(x) for all x in radians.
C.	**EXP**	—computes a 7d exp(x) for all $\|x\| \leq 88$.
D.	**LN**	—computes a 7d ln(x) for all $x > 0$.
E.	**ATAN**	—computes a 7d atan(x) in radians for all x.
F.	**SQRT**	—computes an 8d sqrt(x) for all $x \geq 0$.
G.	**FPINV**	—computes an 8d 1/x for all $x \neq 0$.
H.	**FDIV**	—computes an 8d x/y for all x and all $y \neq 0$.

II. Extended-precision, floating-point (40-bit) math functions (**$MATHX.ASM**):

A.	**SINX**	—computes a 9d sine(x) for all x in radians.
B.	**COSX**	—computes a 9d cosine(x) for all x in radians.
C.	**EXPX**	—computes a 9d exp(x) for all $\|x\| \leq 88$.
D.	**LNX**	—computes an 8d ln(x) for all $x > 0$.
E.	**ATANX**	—computes an 8d atan(x) in radians for all x.
F.	**SQRTX**	—computes a 10d sqrt(x) for all $x \geq 0$.
G.	**FPINVX**	—computes a 10d 1/x for all $x \neq 0$.
H.	**FDIVX**	—computes a 10d x/y for all x and all $y \neq 0$.
I.	**FMULTX**	—computes a 10d x*y for all x and y.

III. Integer (32-bit) math routines (**$MATHI.ASM**):

A.	**ILOG2**	—computes $m = \log2(n)$, $n \leq 2^m$ for use with radix 2 FFT programs.
B.	**IMULT**	—computes 64-bit product of two 32-bit numbers.
C.	**IDIV**	—computes quotient and remainder of two 32-bit numbers.

IV. Vector utilities (**$VECTOR.ASM**):

A.	***CORMULT**	—in-place computation of the complex vector product of two complex arrays using the complex conjugate of the second array.
B.	***CONMULT**	—in-place computation of the complex vector product of two complex arrays.
C.	***CBITREV**	—in-place bit reverse permutation on a complex array with separate real and imaginary arrays.
D.	***FMIEEE**	—in-place fast conversion of an IEEE array to a TMS320C30 array.

E. ***TOIEEE** —in-place fast conversion of a TMS320C30 array to an IEEE array.

F. ***VECMULT** —in-place multiplies a constant times an array.

G. ***CONMOV** —moves (fills) a constant into an array.

H. ***VECMOV** —moves (copies) an array into another array.

V. Radix 2 FFT routines (**$FFT2.ASM**):

A. **CFFFT2** —Complex DIF forward radix 2 FFT using separate real and imaginary arrays and 3/4 cycle sine table.

B. **CIFFT2** —Complex DIT inverse radix 2 FFT using separate real and imaginary arrays and 3/4 cycle sine table (does not include the 1/N scale factor).

VI. Linear algebra routines (**$LINALG.ASM**):

A. ***SOLUTN** —Solves a well conditioned system of linear equations with any number of dependent variable sets. Uses no (diagonal) pivoting with normal-precision floating-point math.

B. ***SOLUTNX** —Solves a well conditioned system of linear equations with any number of dependent variable sets. Uses no (diagonal) pivoting with extended-precision floating-point math.

Extended vs. Normal Precision

Categories I, II, and VI represent a dual collection of programs implemented with 32-bit single- or normal-precision TMS320C30 floating-point arithmetic, and with 40-bit extended-precision TMS320C30 floating-point arithmetic. Some of the normal-precision programs (category I, for example) have been written using the TMS320C30 **RND** instruction for rounding to obtain the optimal precision from the standard floating point TMS320C30 instruction set. This has been done with a slight loss of speed. Such rounding can be carefully eliminated by the user if the additional speed is necessary at the expense of some accuracy.

Extended-precision was implemented on the TMS320C30 by the simple implementation of the 40-by-40 floating-point multiply routine, **FMULTX**. This was necessary since the TMS320C30 has 40-bit addition and subtraction instructions, but the multiply operates only on 32-bit inputs. By using the native add and subtract **FMULTX** and the extended-precision registers R0 to R7, 40-bit floating-point math was effected. All 40-bit constants are stored in two consecutive words in memory. The first word is the normal truncated 32-bit floating-point number. The least significant byte of the second word contains the remaining bottom 8 bits of the extended mantissa. The programs are coded to properly load extended-precision registers with these double-word constants.

The extended-precision versions of the programs in this report may be slower than their normal precision counterparts. When using extended-precision results in R0 from category II programs, note that the results may be stored in memory with or without rounding, A more accurate normal-precision result will generally be obtained by rounding. You should never round before using an extended-precision result as input to another extended-precision program unless special circumstances exist. Note that truncation, not rounding, will occur if an extended-precision register is moved to any 32-bit register or any memory location. This will generally cause loss of accuracy in the amount of the value of the least significant bit of the mantissa.

Program Utilization

Since all programs in this collection are intended to be invoked by a **CALL** instruction, you must have the stack pointer (SP register) appropriately set to an available memory area, preferably in internal RAM. Programs in categories I and II save and restore the data page register DP by using the stack area pointed to by SP. Programs in category III do not alter or use the DP register at all. The programs in categories IV through VI alter but do not restore the DP register.

All of the programs in categories I through III, except for **ILOG2**, are implemented as straight line code. You may wish to disable the instruction cache while these programs are being executing. This will cause no loss of execution speed and will avoid flushing out potentially reusable instructions in the cache. It is beneficial to have the cache enabled when using most of the remaining programs (categories IV through VI) as they generally contain multi-instruction loops.

Programs in categories IV through VI allow input through externally defined variables addresses. The **.global** references indicate these addresses, where the input variable values and/or addresses are located. The starting address of these memory locations is given by the external variable **$PARAMS**. All of the addresses are assumed to be in the same TMS320C30 memory page as **$PARAMS**. If this is not the case, the addresses or the programs should be changed assure that the DP register gets set properly.

Programs in categories IV and VI also allow the use of registers to hold input parameters. The exact registers to be used are found in the program source listings. When using the register input entry point, refer to the program using the *R* prefix on the program name, e.g. *R*SOLUTN. The memory based parameter input entry uses the *M* prefix, e.g. *M*SOLUTN. The **.global** references to the *R* prefix entry points may be deleted if they are not needed.

Function Approximation Techniques

Categories I and II are made up of a collection of elementary mathematical functions numerically approximated using two basic methods. The functions **SIN, COS, EXP, LN,** and **ATAN** are approximated by using polynomials fitted to the various functions over a limited range of the independent variable. The functions **SQRT** and **FPINV** are approximated by iteratively solving a particular non-linear equation. The extended precision versions of these programs (category II) use the same approach with extended-precision arithmetic and resort to more accurate polynomials or more iterations to achieve the desired precision.

Polynomial Approximations

The polynomial approximation method is fundamentally very simple. A limited part of a function is approximated by a polynomial of some order sufficient to obtain the desired accuracy. The polynomial is generally a series of the form:

$$P(n, x) = \sum_{i=0}^{n} \{a[i]x^i\}, \tag{1}$$

where x is the independent variable, n the polynomial order (a fixed integer), and a[i] is a set of n+1 fixed coefficients.

The desired function, say f(x), is then approximated by a particular P(n, x) such that:

$$f(x) = P(n, x) + e(x), \quad x1 < x < xu, \tag{2}$$

where x1 and xu are the limits of the domain of x, and e(x) or e(x)/f(x) is the error function which has been usually minimized in the min-max (equi-ripple) sense. This is done by selecting an appropriate means of calculating the coefficients a[i].

Various techniques and schemes are used in the selection of:

- the approximation interval,
- transformations on the function,
- selection of the polynomial form,
- error minimization criteria, and
- calculation of the coefficients.

See Hastings [2] for an excellent tutorial on this numerical methodology. All of the polynomial approximations used in here were obtained from the National Bureau of Standards reference edited by Abramowitz and Stegun [3].

Non-Linear Equation Approximation

The second method of approximation, using the solution of non-linear equations, is easier to understand. This method requires that a solution for the equation $g(x) = 0$ be found. One means for solving this equation is by Newton-Raphson iteration. This can be understood by considering the Taylor series expansion for g(x):

$$g(x + h) = g(x) + hg'(x) + r(x, h), \tag{3}$$

where r(x, h) is the remainder of the series (which can be assumed to be small), and g'(x) is the derivative of the function g(x). Leaving off the remainder in (3) we get, in terms of incremental values of x, the approximation:

$$g(x[i+1]) = g(x[i]) + \{x[i+1]-x[i]\}g'(x[i]). \tag{4}$$

Solving for x[i+1] in (4) with $g(x[i+1]) = 0$ yields the approximation:

$$x[i+1] = x[i] - g(x[i])/g'(x[i]). \tag{5}$$

Thus, x[i+1] will converge to a solution of $g(x) = 0$. Convergence can be shown to be quadratic, i.e. the error in the approximation at each iteration is proportional to the square of the error in the previous iteration. Minimally, this requires a sufficiently close starting value for x[0] and the condition that $|g'(x)| > 0$ for all iterated values of x.

Math Functions Details

The approximation techniques can be applied to each of the classes of functions. The following sections describe the approximations as they are applied to each function.

Inverse and Square Root Functions

For the problem of computing good approximations to sqrt(c) (**SQRT** and **SQRTX** routines) and 1/c (**FPINV** and **FPINVX** routines), both g(x) and g'(x) must be derived and then use the iteration of equation (5). This is complicated by the restriction that division should be avoided since the TMS320C30 has no divide instructions. For the iteration to find the inverse of c, you can write:

$$g(x[i]) = 1/x[i] - c = 0, \tag{6}$$

which is solved when $1/x = c$ or $x = 1/c$. Taking the derivative of (6) and substituting into (5) and simplifying gives us:

$$x[i+1] = x[i]\{2 - cx[i]\}, \tag{7}$$

which needs no division.

Thus, (7) will converge to 1/c with the accuracy (in digits) for each iteration equal to twice that of the preceding one. Thus, if x[0] approximates 1/c to 3 bits of precision, only three iterations of (7) will yield about $24 = 3(2^3)$ bits of accuracy.

A similar iteration from $f(x) = x^2$ for sqrt(c) can be derived from the formulation:

$$g(x[i]) = x[i]^2 - c = 0, \tag{8}$$

which is solved when $x^2 = c$ or $x = \text{sqrt}(c)$. The solution for (8) leads to the classic square root formula:

$$x[i+1] = 0.5\{c/x[i] + x[i]\}, \tag{9}$$

but this equation uses division. However, the iteration from $f(x) = 1/x^2$ for 1/sqrt(c) can be shown to be:

$$x[i+1] = x[i]\{1.5 - c'x[i]^2\}, \tag{10}$$

where $c' = c/2 = 0.5c$. Though (10) needs no division, the final desired result must be transformed by an extra multiplication by the input c because:

$$\text{sqrt}(c) = c\{1/\text{sqrt}(c)\}. \tag{11}$$

Formula (10) will also converge, in the precision doubling fashion of the Newton-Raphson iteration, given a suitable close starting value for x[0] and the use of sufficiently accurate arithmetic. Note that the extended-precision version routines **FPINVX** and **SQRTX** both use an extra iteration (for a total of 4) to achieve the needed 32-bit accuracy for the 40-bit format.

The initial guess x[0], for the iterations of 1/sqrt(c) and 1/c, may be obtained using an interesting approximation. A TMS320C30 floating-point number $c = (1 + m)2^e$, where $0 \leq m < 1$ and $-127 \leq e \leq 127$. The extra 1, added to the fractional mantissa m, is the implied bit. Then we can write the inverse of c as:

$$1/c = 1/(1 + m)2^{-e}. \tag{12}$$

An excellent approximation for the inverse of the mantissa is:

$$1/(1 + m) = 1 - m/2, \tag{13}$$

which is exact at the end points: $m = 0$ and $m = 1$. Then the approximation for the reciprocal would be:

$$1/c = (1 - m/2)2^{-e}. \tag{14}$$

It turns out that this approximation can be achieved in a single logical operation. If you compute the unlikely value of c' = c XOR 0FF7FFFFFFh, you would complement all bits in c except the sign bit. Including the implied bit and taking the effect of one's complement arithmetic into account results in a final value of:

$$c' = \{1 + (1 - m)\}2^{-(e + 1)}, \tag{15}$$

or the desired approximation:

$$c' = (1 - m/)2^{-e} = 1/c. \tag{16}$$

c' gives about 3 bits of precision, which is an excellent seed x[0] for the 1/c iteration. Using e/2, you have a start for the 1/sqrt(c) iteration as well.

Sine and Cosine Functions

The **SIN**, **COS**, **SINX**, and **COSX** (sine and cosine) routines all use the same basic approximation (section 4.3.98, p. 76 in [3]). The series is for sin(x)/x but is obviously transformed by multiplying by x. The polynomial of even terms then is of the form:

$$\sin(x) = x \sum_{i=0}^{5} \{a[2i]x^{2i}\} + xe(x), \tag{16}$$

where $|x| \leq Pi/2$ and $|xe(x)| \leq 2(10^{-9})$. Instead of using another power series for cos(x), you can use the fact that:

$$\cos(x) = \sin(x + Pi/2). \tag{17}$$

The series given by (16) is only accurate in the 1st and 4th quadrants, i.e. $|x| \leq$ Pi/2. Sin(x) in the other two quadrants is found from:

$$\sin(x) = \sin(Pi - x). \tag{18}$$

The case for $x < 0$ is expediently handled by using |x| for all calculations except for the final multiply by x in (16).

Exponential Functions

The **EXP** and **EXPX** (exponential) routines use an approximation (see Section 4.2.45, p. 71, in [3]). The expansion is of the form

$$\exp(x) = \sum_{i=0}^{7} \{a[i]x^{i}\} + e(x), \tag{19}$$

where $0 \leq x \leq \ln(2)$ and $|e(x)| \leq 2(10^{-10})$. The series for 2^y is found by substituting $y = x/\ln(2)$ since:

$$\exp(x) = \exp(\ln(2)y) = 2^{y}. \tag{20}$$

The new expansion then becomes:

$$2^y = \sum_{i=0}^{7} \{b[i]y^i\} + e(x), \tag{21}$$

where $b[i] = a[i](\ln(2)^i)$. See the coefficients in the **EXP** routine.

Values of exp(x) for x outside the convergent range are found by two means. First for $x < 0$, note the relationship:

$$\exp(-x) = 1/\exp(x), \tag{22}$$

which does require an inverse (see the **FPINV** and **FPINVX** routines). For $y > 1$, let $y = n + f$ where $n = 1, 2, \ldots$ and $0 \leq f < 1$. By substituting y in (20), you get

$$\exp(x) = 2^{n+f} = (2^f)(2^n). \tag{23}$$

Natural Log Functions

The **LN** and **LNX** (natural or base e logarithm) routines use the approximation from [3] (section 4.1.44, p. 69). The expansion comes in the form:

$$\ln(1 + x) = \sum_{i=1}^{8} \{a[i]x^i\} + e(x), \tag{24}$$

where $0 \leq x \leq 1$ and $|e(x)| \leq 3(10^{-8})$. The expansion for ln(y) can be used if the transformation $y = x - 1$ is applied.

Values of ln(x) for x outside the convergent range are found in the following way. First, make the substitution $x = f(2^n)$ for $1 \leq f < 2$ and $n = 0, 1, \ldots$, and then write:

$$\log2(x) = \log2(f2^n) = n + \log2(f), \tag{25}$$

where log2(x) is the log base 2 of x. Using the relationship that $\log2(x) = \ln(x)/\ln(2)$, you get the equation

$$\ln(x) = \ln(f) + n\ln(2). \tag{26}$$

Arctangent Functions

The **ATAN** and **ATANX** (arc or inverse tangent) routines use the approximation from section 4.4.49, p. 81 in [3]. The series with only even terms for atan(x)/x is transformed to

$$\text{atan}(x) = x \sum_{i=0}^{8} \{a[2i]x^{2i}\} + xe(x), \tag{27}$$

where $-1 \leq x \leq 1$ and $|xe(x)| \leq 2(10^{-8})$. Values for atan(x) for x outside the convergent range are obtained by noting the following identity:

$$\text{atan}(x) = \text{atan}((x - 1)/(x + 1)) + \text{Pi}/4. \tag{28}$$

Using the bilinear transformation $y = (x - 1)/(x + 1)$ assures, at the expense of a divide operation, that $y \leq 1$ for $x \geq 1$. The case for $x < 0$ is expediently handled by using $|x|$ for all calculations except for the final multiply by x in (27).

Divide and Multiply Functions

The last group of routines in category I and II are those for the additional arithmetic functions **FDIV** and **FDIVX** (floating-point divides), and **FMULTX** (extended-precision floating-point multiply). The divide operation for the TMS320C30, a = b/c is done by calculating the reciprocal or inverse of the divisor c. Then you compute

$$a = b(1/c). \tag{29}$$

For a normal-precision divide, FDIV finds 1/c by a call to FPINV. A subsequent normal TMS320C30 floating-point multiply of the rounded inverse provides a suitable quotient. For an extended-precision divide, FDIVX finds 1/c by a call to FPINVX. The inverse is then extended-precision multiplied by the dividend using FMULTX.

The extended-precision floating-point multiply simulated by FMULTX is the key to the implementation of virtually all of the extended-precision functions. The extended multiply is achieved using the normal floating-point multiply of the TMS320C30. For two extended-precision numbers **xa** and **xb**, you can represent each as the sum of two floating-point numbers: $xa = a + ea(2^{-24})$ and $xb = b + eb(2^{-24})$. The quantities **ea** and **eb** are the one-byte extensions of **xa** and **xb** respectively.

Thus the complete product xc = (xa)(xb) can be expanded and written as

$$xc = (a)(b) + [(a)(eb) + (b)(ea)]2^{-24} + (ea)(eb)2^{-48}. \tag{30}$$

The last term in (30) is always less than the 32-bit precision in the mantissa of the final result. Therefore, you need only to compute the first two terms in the product **xc**. Also, note that all the indicated products in (30) may be computed using a normal-precision native TMS320C30 multiply as long as the terms are collected in extended-precision registers. The additions are also done using the native TMS320C30 add as it is implemented in extended-precision.

Integer Arithmetic Program Details

Integer routines differ from the floating-point versions because they produce only integer results. If the computation can produce fractional values, then the fraction must be truncated to leave only the integer result.

Integer Result Log Base 2

The routine **ILOG2** is a useful utility for computing integer value **m** of the log base 2 of the integer **n**. The result is computed by successive multiplies by 2 (implemented as shifts by 1). The resulting relationship is $n \leq 2m$, such that if log2(n) is not an exact integer, m is rounded up to the next largest integer. This is useful as it allows the determination of m from any value $n > 0$ (e.g. not a power of two) which might require the padding of additional values (zeros) for a radix 2 FFT. This program is very fast because of a delayed branch loop and internally requires only $4(m+1)$ cycles (cached) to do the calculation.

Extended Precision Integer Multiply

The IMULT routine is a modified version of the program EXTMPY in the *TMS320C3x User's Guide* [1]. It has been modified and slightly speeded up. The negation of the final 64-bit product is done in two instructions by direct two's complement negation rather than by using one's complement to simulate the same result. The product is computed by breaking the multiplier and multiplicand up into two 16 bit integers each. Thus the full product **c** of the numbers $a = au(2^{16}) + al$, and $b = au(2^{16}) + bl$ is

$$c = (au)(bu)2^{32} + [(au)(bl) + (bu)(al)]2^{16} + (al)(bl), \tag{31}$$

where the powers of two indicated are accomplished by shifts. Note that each product in (31) must be represented as a 32-bit integer. The adds in the sum must be done with care to facilitate the carry between the two final 32-bit components of the product.

Integer Divide

The IDIV routine is a modified version of the program DIVI in the *TMS320C3x User's Guide* [1]. It has been modified to return the absolute value of the remainder of the integer division. The remainder was originally computed, but was discarded during the extraction process for the quotient. A few more instructions allow the extraction of both the quotient and remainder from the result of the **SUBC** process. The program **IDIV** may be used for the computation of the modulo function. The output of **IDIV** is the pair $\{q, |r|\} = a/b$, with the property:

$$0 \leq r = (a \text{ modulo } b) < a, \tag{32}$$

for $a > 0$ and $b > 0$. The complete relationship is, by definition, $a = bq + r$, for positive a and b.

Vector Utility Routines

Vector utilities are functions which operate on arrays of numbers. Some utilities, like dot products and convolutions, are simple. Other utilities, like those presented here, are more involved.

Complex and Complex Conjugate Array Multiplies

The array routine ***CORMULT** computes the point-by-point complex conjugate multiply of two complex arrays. If the arrays are c1 and c2, and are of length n, then:

$$c1[k] \leftarrow c1[k]\text{conj}(c2[k]),\ k = 1, \ldots, n, \tag{33}$$

where $\leftarrow$ means replaces. Each complex array is assumed to be stored as two separate arrays, i.e. $\{c1\} = \{x1, y1\}$ and $\{c2\} = \{x2, y2\}$. In cartesian complex representation, (33) becomes

$$(x1 + iy1) \leftarrow (x1 + iy1)(x2 - iy2), \tag{34}$$

where i represents the imaginary constant $\text{sqrt}(-1)$. Separating the real and imaginary parts, we have:

$$x1 \leftarrow x1x2 + y1y2,\ y1 \leftarrow y1x2 - y2x1 \tag{35}$$

This operation can be used for the frequency domain correlation of two FFTs to implement time domain correlation.

On the other hand, the array routine ***CONMULT** computes the point-by-point complex multiply of two complex arrays. If the arrays are c1 and c2, and are each of length n, then

$$c1[k] \leftarrow c1[k](c2[k]),\ k = 1, \ldots, n, \tag{36}$$

In cartesian complex representation, (36) becomes

$$(x1 + iy1) \leftarrow (x1 + iy1)(x2 + iy2). \tag{37}$$

Separating the real and imaginary parts results in

$$x1 \leftarrow x1x2 - y1y2,\ y1 \leftarrow y1x2 + y2x1. \tag{38}$$

This operation can be used for the frequency domain convolution of two FFTs to implement digital filtering.

Complex Array Bit Reversal

The array routine ***CBITREV** executes an in-place bit reverse permutation on two arrays simultaneously. This operation is generally used for index scrambling before a DIT FFT (decimation in time, see CIFFT2), or after a DIF FFT (decimation in frequency, see CFFFT2) for index unscrambling. Therefore, ***CBITREV** is useful in permuting complex arrays stored as two separate arrays which are associated with radix 2 FFTs. The program uses the bit reverse indexing feature of the TMS320C30 to achieve this function. The loop in ***CBITREV** is nearly as efficient in permuting two arrays together as permuting one array alone. This is due to the use of parallel load and store instructions and a delayed (single cycle) conditional branch.

Floating Point Conversions

The array routines ***FMIEEE** and ***TOIEEE** are vectorized versions of their original scalar counterparts **FMIEEE** and **TOIEEE**. Both routines do fast conversions from or to IEEE format by avoiding dealing with special rare cases. Also, both programs convert the numbers in the arrays in-place which destroys the original data. These array versions of the format conversion routines are much faster than calling the scalar version routines in a special loop. These routines also have their own internal, shared constant table for conversions.

Vector Primitives

The array routines ***VECMULT**, ***CONMOV**, and ***VECMOV** are a useful suite of efficient programs for simple array operations. The first routine, ***VECMULT**, performs the simple operation x[k] ← x[k]c which is a scalar-vector multiply useful in uniformly scaling an array by a constant c. You can use this for scaling arrays after an inverse FFT by choosing c = 1/n. The next routine, ***CONMOV**, performs the operation x[k] ← c which is useful in filling or initializing any portion of an array to a single constant c. The last routine, ***VECMOV** performs the simple operation x[k] ← y[k], an array move, and is, therefore, generally useful.

FFT Routines

This category contains the two complementary radix 2 complex FFT programs **CFFFT2** and **CIFFT2**. These programs differ from previously available TMS320C30 FFT programs in that they operate on complex arrays which are stored as two separate and independent real arrays. Both routines do the FFTs in-place and do no index permutations or constant scaling (multiplication). Also these programs require only a 3/4 cycle external, pre-computed sine table. As with previous FFT programs, these, too, have a special multiply-less butterfly loop for the occurrence of unity twiddle or complex rotation factors.

The routine **CFFFT2** is a DIF radix 2 complex forward FFT program and thus assumes a normally indexed pair of input arrays. The output array is bit-reverse permuted and normally must be unscrambled to be of any use (see ***CBITREV**). The routine **CIFFT2** is a DIT radix 2 inverse FFT program and thus assumes a bit-reverse indexed pair of input arrays. A normally indexed complex frequency spectrum must be bit-reverse scrambled before using **CIFFT2** (again, see ***CBITREV**). On the other hand, the output from this inverse FFT is in normal indexed order, but lacks the traditional scaling by the factor of 1/n. Therefore, back-to-back calls of **CFFFT2** and **CIFFT2** will return the original complex array (in proper order) but multiplied by a factor of n. Consult the handbook by Burrus and Parks [4] for additional FFT algorithm details.

Linear Algebra Routines

The routines ***SOLUTN** and ***SOLUTNX** are the normal- and extended-precision implementations of the algorithm for solving simultaneous linear equations. This algorithm is the modified Gauss-Jordan elimination without (off diagonal) pivoting. This is a simple algorithm which is intended for use with *well-conditioned* systems of dense linear equations of moderate size. Well conditioned means that the system of linear equations is linearly independent or non-singular. This subject and further algorithm details are to be found in chapter 2 of [5] by Press et al, or any other book on the numerical techniques of linear algebra. This algorithm is suitable for a wide range of problems requiring the solution of a system of linear equations, e.g. exact or least squares polynomial fitting.

A simple system of linear equations has the form:

$$\begin{aligned} A[1,1]x[1] + A[1,2]x[2] + \ldots + A[1,n]x[n] &= y[1], \\ A[2,1]x[1] + A[2,2]x[2] + \ldots + A[2,n]x[n] &= y[2], \\ \vdots \qquad\quad \vdots \qquad\quad & \vdots \qquad \vdots \\ A[n,1]x[1] + A[n,2]x[2] + \ldots + A[n,n]x[n] &= y[n]. \end{aligned} \tag{39}$$

Symbolically, you may write A = A[i, j] as the n x n matrix of coefficients, and x = x[i] as the unknown independent variable (column) vector, and y = y[j] as the dependent variable (row) vector. Thus (39) can be written in short hand form as Ax = y or Ax − y = 0, where the multiplication indicated is a matrix-vector multiply. The fundamental problem in linear algebra, then, is to find the solution vector x. In fact, you may desire to find the m different solutions to m sets of linear equations which share the same coefficient matrix A, i.e. Ax[k] = y[k], for k = 1, . . . , m.

You can solve the general problem just stated by using *SOLUTN, or with more accuracy with *SOLUTNX. This is done by constructing a tableau B (table of coefficients) which is simply the coefficient matrix A (in row major storage format) with the negative of the y vector(s) appended (:) as m extra columns to A. Thus you would have B = A : −y, as your problem, where B is a n by n+m matrix and typically m = 1. Thus, for the common case of m = 1, the input array B can be written as:

$$\begin{array}{l} A[1,\ 1],\ A[1,\ 2],\ \ldots,\ A[1,\ n],\ -y[1], \\ A[2,\ 1],\ A[2,\ 2],\ \ldots,\ A[2,\ n],\ -y[2], \\ \quad .\qquad .\qquad\qquad .\qquad . \\ \quad .\qquad .\qquad\qquad .\qquad . \\ \quad .\qquad .\qquad\qquad .\qquad . \\ A[n,\ 1],\ A[n,\ 2],\ \ldots,\ A[n,\ n],\ -y[n]. \end{array} \tag{40}$$

After the ***SOLUTN** routine is executed, the matrix C = A' : x appears, where the column(s) beyond the original coefficients A (the y[k] vectors) have been replaced by the solution vector(s) x[k]. The new matrix A' is a partially computed version of the inverse of the matrix A. The complete inverse of A, which is normally computed by the standard Gauss-Jordan scheme, is rarely needed. Therefore, a faster modified algorithm has been used which does about half the work.

This simple method used for solving systems of linear equations has two restrictions.

1. As the pivoting operation (exchange of x and y variables) always starts with A[1, 1] and proceeds down the diagonal, A[1, 1] must be non-zero. This is because, in the exchange process, you must divide by the pivot element. A zero coefficient at A[1, 1] may be moved by reordering the variable indices by appropriately swapping rows and columns in A and in y.

2. The maximum absolute value of the elements in A must be approximately unity. This is necessary to assure that no pivot element is encountered which is smaller in magnitude than 10^{-8} for ***SOLUTN**, and 10^{-10} for ***SOLUTNX**. This restriction monitors the system condition and assures an adequately accurate solution, but the final solution should always be verified by substitution. This is done by inspecting the elements of the error vector e = Ax − y computed by using the solution x, and the original A and y.

Summary

This report presented a set of routines that can be used in digital signal processing applications. The appendix contains the source code of these routines. This source code can also be obtained from the Texas Instruments Electronic Bulletin Board (713) 274-2323. If there are comments or corrections, please contact the author of this report:

Mr. Gary Sitton
Gas Light Software
5211 Yarwell
Houston, TX 77096
Tel (713) 729-1257

References

(1) *TMS320C3x User's Guide* (literature number SPRU031), Texas Instruments, Dallas, TX, August 1988.

(2) Hastings, C. Jr., "Approximations for Digital Computers", Princeton University Press, Princeton N.J., 1955.

(3) Abramowitz, M. and Stegun, I.A. (Editors), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards (Applied Mathematics Series 55), Washington D.C., 1964.

(4) Burrus, C.S. and Parks, T.W., "DFT/FFT and Convolution Algorithms", John Wiley and Sons, New York N.Y., 1985.

(5) Press, W.H., Flannery, B.P., Teukolsy, S.A., and Vettering, W.T., *Numerical Recipes in C - The Art of Scientific Programming*, Cambridge University Press, Cambridge England, 1988.

Appendix A

Appendix

Program Library

I. $MATH.ASM
II. $MATHX.ASM
III. $MATHI.ASM
IV. $VECTOR.ASM
V. $FFT2.ASM
VI. $LINALG.ASM

```
*****************************************************************
*
*   PROGRAM: $MATH.ASM
*
*   NORMAL FLOATING-POINT (32-BIT) MATH FUNCTIONS
*
*   $MATH.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*       SIN   - COMPUTES A 7D SINE(X) FOR ALL X IN RADIANS.
*
*       COS   - COMPUTES A 7D COSINE(X) FOR ALL X IN RADIANS.
*
*       EXP   - COMPUTES A 7D EXP(X) FOR ALL !X! =< 88.
*
*       LN    - COMPUTES A 7D LN(X) FOR ALL X > 0.
*
*       ATAN  - COMPUTES A 7D ATAN(X) FOR ALL X IN RADIANS.
*
*       SQRT  - COMPUTES AN 8D SQRT(X) FOR ALL X >= 0.
*
*       FPINV - COMPUTES AN 8D 1/X FOR ALL X /= 0.
*
*       FDIV  - COMPUTES AN 8D X/Y FOR ALL X AND ALL Y /= 0.
*
*****************************************************************
```

```
*****************************************************
*   PROGRAM: SIN                                    *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   SINE FUNCTION: R0 <= SIN(R0).                   *
*                                                   *
*   APPROXIMATE ACCURACY: 7 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: NONE.                       *
*   REGISTERS FOR INPUT: R0 (ARGUMENT IN RADIANS).  *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0, IR0, AND R0-4.          *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: NONE.                          *
*   EXECUTION CYCLES (MIN, MAX): 44 , 44.           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  SIN
        .GLOBL  ECOS

;       INTERNAL CONSTANTS

        .DATA

NORM    .FLOAT  0.636619772     ; 2/PI

;       POLYNOMIAL COEFFS. FOR SIN(X*2/PI), -1 < X < 1

SHFT    .FLOAT  1.570796327     ; C1 (PI/2)
        .FLOAT  -0.6459640968   ; C3
        .FLOAT  0.07969260878   ; C5
        .FLOAT  -0.00468166687  ; C7
        .FLOAT  0.00016025884   ; C9
COF     .FLOAT  -0.000003433338 ; C11

ACOF    .WORD   COF             ; ADDRESS OF COEFFS.

CON     .FLOAT  -1.0, 0.0, 1.0, 0.0 ; MAPPING CONSTANTS

ACON    .WORD   CON             ; ADDRESS OF CONSTS.

        .TEXT

;       START OF SIN PROGRAM

SIN:

        PUSH    DP              ; SAVE DP
        LDP     @ACOF           ; LOAD DATA PAGE POINTER
        RND     R0              ; ROUND X
        LDF     R0,R4           ; R4 <= X

;       COSINE ENTRY POINT

ECOS:

;       SCALE AND MAP VARIABLE X

        ABSF    R0              ; R0 <= |X|
        LDF     R0,R1           ; R1 <= RND |X|
        MPYF    @NORM,R1        ; R1 <= X*2/PI
        FIX     R1,IR0          ; IR0 <= INTEGER QUADRANT Q
        FLOAT   IR0,R2          ; R2 <= FLOATING QUADRANT Q
        SUBF    R2,R1,R0        ; R0 <= X, -1 < X < 1
        NEGF    R0,R3           ; R3 <= -X
        ADDI    1,IR0           ; IR0 <= Q + 1
        AND     3,IR0           ; IR0 <= TABLE INDEX
        TSTB    2,IR0           ; LOOK AT 2ND LSB
        LDFNZ   R3,R0           ; IF 1 THEN R0 <= -X
        LDI     @ACON,AR0       ; AR1 -> CONST. TABLE
        ADDF    *+AR0(IR0),R0   ; FINAL MAPPING, R0 <= X + C
        NEGF    R0,R3           ; R3 <= -X
        LDI     @ACOF,AR0       ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED (ODD) SERIES

        MPYF    R0,R0,R2        ; R2 <= X**2
        RND     R2              ; ROUND X**2

        MPYF    *AR0--,R2,R1    ; R1 <= X**2*C11
        ADDF    *AR0--,R1       ; R1 <= C9 + R1

        MPYF    R2,R1           ; R1 <= X**2*(C9 + R1)
        ADDF    *AR0--,R1       ; R1 <= C7 + R1

        MPYF    R2,R1           ; R1 <= X**2*(C7 + R1)
        ADDF    *AR0--,R1       ; R1 <= C5 + R1

        RND     R1              ; ROUND BEFORE *
        MPYF    R2,R1           ; R1 <= X**2*(C5 + R1)
        ADDF    *AR0--,R1       ; R1 <= C3 + R1

        RND     R1              ; ROUND BEFORE *
        MPYF    R2,R1           ; R1 <= X**2*(C3 + R1)
        ADDF    *AR0,R1         ; R1 <= C1 + R1
```

```
;       FINISH UP SERIES AND RETURN

        LDF     R4,R4           ; TEST ORIGINAL X
        LDFN    R3,R0           ; IF X < 0 THEN R0 <= -X
        POP     R2              ; R2 <= RETURN ADDRESS
        BUD     R2              ; RETURN (DELAYED)
        RND     R0              ; ROUND BEFORE *
        RND     R1              ; ROUND BEFORE *
        MPYF    R1,R0           ; R1 <= X*(C1 + R1)
```

```
*****************************************************
*   PROGRAM: COS                                    *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   COSINE FUNCTION: R0 <= COS(R0).                 *
*                                                   *
*   APPROXIMATE ACCURACY: 7 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: NONE.                       *
*   REGISTERS FOR INPUT: R0 (ARGUMENT IN RADIANS).  *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0, IR0, AND R0-4.          *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: ECOS (SIN).                    *
*   EXECUTION CYCLES (MIN, MAX): 46 , 46.           *
*                                                   *
*   NOTE: USES SHFT CONSTANT FROM SIN PROGRAM!      *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  COS
        .GLOBL  ECOS

        .TEXT

;       START OF COS PROGRAM

COS:

        PUSH    DP              ; SAVE DP
        LDP     @ACOF           ; LOAD DATA PAGE POINTER

        BRD     ECOS            ; R0 <= COS(X) = SIN(X'), (DELAYED)
        RND     R0              ; ROUND X
        ADDF    @SHFT,R0        ; R0 <= X' = X + PI/2
        LDF     R0,R4           ; R4 <= X'

;       RETURN OCCURS FROM SIN !
```

```
*****************************************************
*   PROGRAM: EXP                                    *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXPONENTIAL FUNCTION: R0 <= EXP(R0).            *
*                                                   *
*   APPROXIMATE ACCURACY: 7 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: |R0| <= 88.0.               *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-4.                        *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: FPINV.                         *
*   EXECUTION CYCLES (MIN, MAX): 44 (R0 <= 0), 70.  *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  EXP
        .GLOBL  FPINV

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFF. FOR 2**-X

ENRM    .FLOAT  1.442695041     ; 1/LN(2)

;       POLYNOMIAL COEFFS. FOR 2**-X, 0 <= X < 1.

        .FLOAT  1.0000000000    ; C0
        .FLOAT  -0.693147180    ; C1
        .FLOAT  0.240226469     ; C2
        .FLOAT  -0.055503654    ; C3
        .FLOAT  0.009615978     ; C4
        .FLOAT  -0.001328240    ; C5
        .FLOAT  0.000147491     ; C6
C7      .FLOAT  -0.000010863    ; C7

AC7     .WORD   C7

        .TEXT

;       START OF EXP PROGRAM

EXP:

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC7            ; LOAD DATA PAGE POINTER
        RND     R0              ; ROUND X
        NEGF    R0,R2           ; R2 <= -X
        LDF     R0,R1           ; R1 <= X
        LDFN    R2,R1           ; IF X < 0 THEN R1 <= |X|
        MPYF    @ENRM,R1        ; R1 <= X = |X|/LN(2)
        FIX     R1,R3           ; R3 <= I = INTEGER OF X
        FLOAT   R3,R0           ; R0 <= FLT. PT. I
        SUBF    R0,R1           ; R1 <= FRACTION OF |X|, 0 <= X < 1
        NEGI    R3              ; R3 <= -I
        LSH     24,R3           ; MOVE -I TO EXP.
        PUSH    R3              ; SAVE AS INT.
        POPF    R3              ; R3 <= FLT. PT. 2**-I
        LDI     @AC7,AR0        ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED SERIES

        RND     R1              ; ROUND BEFORE *
        MPYF    *AR0--,R1,R0    ; R0 <= X*C7
        ADDF    *AR0--,R0       ; R0 <= C6 + R0

        MPYF    R1,R0           ; R0 <= X*(C6 + R0)
        ADDF    *AR0--,R0       ; R0 <= C5 + R0

        MPYF    R1,R0           ; R0 <= X*(C5 + R0)
        ADDF    *AR0--,R0       ; R0 <= C4 + R0

        MPYF    R1,R0           ; R0 <= X*(C4 + R0)
        ADDF    *AR0--,R0       ; R0 <= C3 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C3 + R0)
        ADDF    *AR0--,R0       ; R0 <= C2 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C2 + R0)
        ADDF    *AR0--,R0       ; R0 <= C1 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C1 + R0)

;       TEST FOR X < 0 AND RETURN

        LDF     R2,R2           ; TEST ORIGINAL -X
        BND     FPINV           ; IF -X < 0 THEN R0 <= 1/X, (DELAYED)
        ADDF    *AR0,R0         ; R0 <= 2**-X = C0 + R0
        RND     R0              ; ROUND BEFORE *
        MPYF    R3,R0           ; R0 <= 2**-(I + X)

        RETS                    ; RETURN (IF NO FPINV BRANCH)
```

```
*****************************************************
*   PROGRAM: LN                                     *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   LOGARITHM FUNCTION BASE E: R0 <= LN(R0).        *
*                                                   *
*   APPROXIMATE ACCURACY: 7 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: R0 > 0.0.                   *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0 AND R0-3.                *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: NONE.                          *
*   EXECUTION CYCLES (MIN, MAX): 43 , 43.           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  LN

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR LN(1+X)

LNRM    .FLOAT  0.6931471806    ; LN(2)
C0      .FLOAT  1.0000000000    ; C0 (1.0)

;       POLYNOMIAL COEFFS. FOR LN(1+X), 0 <= X < 1.

        .FLOAT  0.9999964239    ; TOP OF C1
        .FLOAT  -0.4998741238   ; TOP OF C2
        .FLOAT  0.3317990258    ; TOP OF C3
        .FLOAT  -0.2407338084   ; TOP OF C4
        .FLOAT  0.1676540711    ; TOP OF C5
        .FLOAT  -0.0953293897   ; TOP OF C6
        .FLOAT  0.0360884937    ; TOP OF C7
C8      .FLOAT  -0.0064535442   ; TOP OF C8

AC8     .WORD   C8

        .TEXT

;       START OF LN PROGRAM

LN:

        LDF     R0,R0           ; TEST X
        RETSLE                  ; RETURN NOW IF X <= 0

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC8            ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE AS FLT. PT.
        POP     R3              ; R3 <= INTEGER FORMAT
        ASH     -24,R3          ; R3 <= E = SIGNED EXP.
        FLOAT   R3,R1           ; R1 <= FLT. PT. E VALUE
        LDF     @C0,R2          ; R2 <= 1.0
        LDE     R2,R0           ; EXP. R0 <= 0 (1 <= X < 2)
        SUBRF   R0,R2           ; R2 <= X - 1 (0 <= X < 1)
        LDF     @LNRM,R0        ; R0 <= LN(2)
        MPYF    R1,R0           ; R0 <= E*LN(2)
        LDF     R0,R3           ; R3 <= E*LN(2)
        LDI     @AC8,AR0        ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED SERIES

        RND     R2,R1           ; R1 <= RND X
        MPYF    *AR0--,R1,R0    ; R0 <= X*C8
        ADDF    *AR0--,R0       ; R0 <= C7 + R0

        MPYF    R1,R0           ; R0 <= X*(C7 + R0)
        ADDF    *AR0--,R0       ; R0 <= C6 + R0

        MPYF    R1,R0           ; R0 <= X*(C6 + R0)
        ADDF    *AR0--,R0       ; R0 <= C5 + R0

        MPYF    R1,R0           ; R0 <= X*(C5 + R0)
        ADDF    *AR0--,R0       ; R0 <= C4 + R0

        MPYF    R1,R0           ; R0 <= X*(C4 + R0)
        ADDF    *AR0--,R0       ; R0 <= C3 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C3 + R0)
        ADDF    *AR0--,R0       ; R0 <= C2 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C2 + R0)
        ADDF    *AR0--,R0       ; R0 <= C1 + R0

;       ADD IN SCALED EXPONENT AND RETURN

        POP     R2              ; R2 <= RETURN ADDRESS
        BUD     R2              ; RETURN (DELAYED)
        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X*(C1 + R0)
        ADDF    R3,R0           ; R0 <= LN(X) + E*LN(2)
```

```
*****************************************************
*   PROGRAM: ATAN                                   *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   ARC TANGENT FUNCTION: R0 <= ATAN(R0).           *
*                                                   *
*   APPROXIMATE ACCURACY: 7 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: NONE.                       *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0, IR0, AND R0-4.          *
*   REGISTERS FOR OUTPUT: R0 (IN RADIANS).          *
*   ROUTINES NEEDED: FDIV.                          *
*   EXECUTION CYCLES (MIN, MAX): 30 (|ATAN| <= 1), 69.  *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  ATAN
        .GLOBL  FDIV

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR ATAN(X)

        .FLOAT  -0.7853981635   ; -PI/4
        .FLOAT  0.7853981635    ; PI/4
        .FLOAT  0.0000000000    ; ZERO

;       POLYNOMIAL COEFFS. FOR ATAN(X), -1 <= X <= 1.

C1      .FLOAT  1.0000000000    ; C1
        .FLOAT  -0.3333314528   ; C3
        .FLOAT  0.1999355085    ; C5
        .FLOAT  -0.1420889944   ; C7
        .FLOAT  0.1065626393    ; C9
        .FLOAT  -0.0752896400   ; C11
        .FLOAT  0.0429096138    ; C13
        .FLOAT  -0.0161657367   ; C15
C17     .FLOAT  0.0028662257    ; C17

AC17    .WORD   C17

        .TEXT

;       START OF ATAN PROGRAM

ATAN:

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC17           ; LOAD DATA PAGE POINTER
        ABSF    R0,R2           ; R2 <= |X|
        SUBF    @C1,R2          ; R2 <= |X| - 1
        BLED    SKIP            ; IF |X| > 1 THEN SCALE (DELAYED)
        RND     R0,R3           ; R3 <= RND X
        RND     R0,R1           ; R1 <= RND X
        LDI     0,IR0           ; IR0 <= 0, POST SCALE INDEX

;       SCALE FOR |X| > 1

        PUSHF   R1              ; SAVE RND X
        ABSF    R0,R1           ; R1 <= |X|
        ADDF    @C1,R1          ; R1 <= |X| + 1
        LDF     R2,R0           ; R0 <= |X| - 1
        CALL    FDIV            ; R0 <= (|X| - 1)/(|X| + 1)

;       TEST FOR X' < 0

        POPF    R2              ; GET ORIGINAL X
        BGED    SKIP            ; IF X < 0 THEN R0 <= -X' (DELAYED)
        RND     R0,R3           ; R3 <= RND X'
        RND     R0,R1           ; R1 <= RND X'
        SUBI    1,IR0           ; IR0 <= -1, (PI/4)

        NEGF    R3,R3           ; R3 <= -X'
        SUBI    1,IR0           ; IR0 <= -2, (-PI/4)

SKIP:   MPYF    R1,R1,R0        ; R0 <= X**2
        LDI     @AC17,AR0       ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED (ODD) SERIES

        RND     R0,R1           ; R1 <= RND X**2
        MPYF    *AR0--,R1,R0    ; R0 <= X**2*C17
        ADDF    *AR0--,R0       ; R0 <= C15 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C15 + R0)
        ADDF    *AR0--,R0       ; R0 <= C13 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C13 + R0)
        ADDF    *AR0--,R0       ; R0 <= C11 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C11 + R0)
        ADDF    *AR0--,R0       ; R0 <= C9 + R0

        RND     R0              ; ROUND BEFORE *
        MPYF    R1,R0           ; R0 <= X**2*(C9 + R0)
        ADDF    *AR0--,R0       ; R0 <= C7 + R0
```

```
RND     R0              ; ROUND BEFORE *
MPYF    R1,R0           ; R0 <= X**2*(C7 + R0)
ADDF    *AR0--,R0        ; R0 <= C5 + R0

RND     R0              ; ROUND BEFORE *
MPYF    R1,R0           ; R0 <= X**2*(C5 + R0)
ADDF    *AR0--,R0       ; R0 <= C3 + R0

RND     R0              ; ROUND BEFORE *
MPYF    R1,R0           ; R0 <= X**2*(C3 + R0)
ADDF    *AR0--,R0,R1    ; R1 <= C1 + R0

FINISH UP, POST SCALE BY C AND RETURN

POP     R2              ; R2 <= RETURN ADDRESS
BUD     R2              ; RETURN (DELAYED)
RND     R1              ; ROUND BEFORE *
MPYF    R3,R1,R0        ; R0 <= ATAN(X) = X*(1 + R0)
ADDF    ***AR0(IR0),R0  ; R0 <= ATAN(X) + C (0.0, PI/4 OR -PI/4)
```

```
*****************************************************
*   PROGRAM: SQRT                                   *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   SQUARE ROOT FUNCTION: R0 <= SQRT(R0).           *
*                                                   *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: R0 >= 0.0.                  *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-4.                        *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: NONE.                          *
*   EXECUTION CYCLES (MIN, MAX): 49 , 49.           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  SQRT

;       INTERNAL CONSTANTS

        .DATA

CNST1   .SET    0.5
CNST2   .SET    1.5
CNST3   .FLOAT  1.103553391     ; ADJUSTED 1.0
CNST4   .FLOAT  0.780330086     ; ADJUSTED SQRT(1/2)

SMSK    .WORD   0FF7FFFFFH

        .TEXT

;       START OF SQRT PROGRAM.

SQRT:

        LDF     R0,R3           ; TEST AND SAVE V
        RETSLE                  ; RETURN NOW IF V <= 0

;       GET APPROXIMATION TO 1/V.  FOR V = (1+M)*2**E
;       AND 0 <= M < 1, FOR E EVEN: X[0] = (1-M/2)*2**-E/2
;       AND FOR E ODD: X[0] = SQRT(1/2)*(1-M/2)*2**-E/2

        PUSH    DP              ; SAVE DP
        LDP     @SMSK           ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE V AS FLT. PT. V = (1+M)*2**E
        POP     R2              ; R2 <= V AS INTEGER
        XOR     @SMSK,R2        ; R2 <= COMPLEMENT ALL BUT SIGN
        LDI     R2,R1           ; R1 <= (1-M/2)*2**-E
```

```
        LDI     R2,R4           ; R4 <= R1
        LSH     8,R1            ; R1 <= R1 EXP. REMOVED
        ASH     -1,R2           ; R2 <= R2 WITH -E/2 EXP.
        PUSH    R2              ; SAVE R2 AS INTEGER
        POPF    R2              ; R2 <= FLT. PT.
        LDE     R2,R1           ; R1 <= (1-M/2)*2**-E/2
        LDF     @CNST3,R2       ; R2 <= 1.1... FOR ODD E
        LSH     7,R4            ; TEST LSB OF E (AS SIGN)
        LDFNN   @CNST4,R2       ; IF E EVEN R2 <= 0.78...
        MPYF    R2,R1           ; R1 <= CORRECTED ESTIMATE
        POP     DP              ; UNSAVE DP

;       GENERATE V/2 (USES MPYF).

        MPYF    CNST1,R0        ; R0 <= V/2 TRUNC.
        RND     R0              ; R0 <= RND V/2

;       NEWTON ITERATION FOR Y(X) = X - V**-2 = 0 ...

        MPYF    R1,R1,R2        ; R2 <= X[0]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[0]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[0]**2
        MPYF    R2,R1           ; R1 <= X[1] = X[0] * (1.5 - (V/2)*X[0]**2)
;
        MPYF    R1,R1,R2        ; R2 <= X[1]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[1]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[1]**2
        MPYF    R2,R1           ; R1 <= X[2] = X[1] * (1.5 - (V/2)*X[1]**2)
;
        MPYF    R1,R1,R2        ; R2 <= X[2]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[2]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[2]**2
        MPYF    R2,R1           ; R1 <= X[3] = X[2] * (1.5 - (V/2)*X[2]**2)
;
        RND     R1              ; ROUND BEFORE *
        MPYF    R1,R1,R2        ; R2 <= X[3]**2
        RND     R2              ; ROUND BEFORE *
        MPYF    R0,R2           ; R2 <= (V/2) * X[3]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[3]**2
        RND     R2              ; ROUND BEFORE *
        MPYF    R2,R1           ; R1 <= X[4] = X[3] * (1.5 - (V/2)*X[3]**2)

;       INVERT FINAL RESULT AND RETURN

        POP     R2              ; R2 <= RETURN ADDRESS
        BUD     R2              ; RETURN (DELAYED)
        RND     R3              ; ROUND BEFORE *
        RND     R1              ; ROUND BEFORE *
        MPYF    R1,R3,R0        ; R0 = SQRT(V) = V*SQRT(1/V)
```

```
*****************************************************
*   PROGRAM: FPINV                                  *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   FLOATING POINT INVERSE: R0 <= 1/R0              *
*                                                   *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: R0 != 0.0.                  *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-2 AND R4.                 *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: NONE.                          *
*   EXECUTION CYCLES (MIN, MAX): 33 , 33.           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FPINV

;       INTERNAL CONSTANTS

        .DATA

ONE     .SET    1.0
TWO     .SET    2.0

MSK     .WORD   0FF7FFFFFH

        .TEXT

;       START OF FPINV PROGRAM

FPINV:

        LDF     R0,R0           ; TEST F
        RETSZ                   ; RETURN NOW IF F = 0

;       GET APPROXIMATION TO 1/F.  FOR F = (1+M) * 2**E
;       AND 0 <= M < 1, USE: X[0] = (1-M/2) * 2**-E

        PUSH    DP              ; SAVE DATA PAGE POINTER
        LDP     @MSK            ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE AS FLT. PT. F = (1+M) * 2**E
        POP     R1              ; FETCH BACK AS INTEGER
        XOR     @MSK,R1         ; COMPLEMENT E & M BUT NOT SIGN BIT
        PUSH    R1              ; SAVE AS INTEGER, AND BY MAGIC...
        POPF    R1              ; R1 <= X[0] = (1-M/2) * 2**-E.
        POP     DP              ; UNSAVE DP
```

```
;       NEWTON ITERATION FOR: Y(X) = X - 1/F = 0 ...

        MPYF    R1,R0,R4        ; R4 <= F * X[0]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[0]
        MPYF    R4,R1           ; R1 <= X[1] = X[0] * (2 - F * X[0])

        MPYF    R1,R0,R4        ; R4 <= F * X[1]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[1]
        MPYF    R4,R1           ; R1 <= X[2] = X[1] * (2 - F * X[1])

        MPYF    R1,R0,R4        ; R4 <= F * X[2]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[2]
        MPYF    R4,R1           ; R1 <= X[3] = X[2] * (2 - F * X[2])

;       FOR THE LAST ITERATION: X[4] = (X[3] * (1 - (F * X[3]))) + X[3]

        RND     R0,R4           ; ROUND F BEFORE LAST MULTIPLY
        RND     R1,R0           ; ROUND X[3] BEFORE MULTIPLIES
        MPYF    R0,R4           ; R4 <= F * X[3] = 1 + EPS

;       FINISH ITERATION AND RETURN

        POP     R2              ; R2 <= RETURN ADDRESS
        BUD     R2              ; RETURN (DELAYED)
        SUBRF   ONE,R4          ; R4 <= 1 - F * X[3] = EPS
        MPYF    R0,R4           ; R4 <= X[3] * EPS
        ADDF    R4,R1,R0        ; R0 <= X[4] = (X[3]*(1 - (F*X[3]))) + X[3]
```

```
*****************************************************
*   PROGRAM: FDIV                                   *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               APRIL 1989.                         *
*                                                   *
*   FLOATING POINT DIVIDE FUNCTION: R0 <= R0/R1.    *
*                                                   *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: R1 != 0.0.                  *
*   REGISTERS FOR INPUT: R0 (DIVIDEND) AND R1 (DIVISOR).*
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-4.                        *
*   REGISTERS FOR OUTPUT: R0 (QUOTIENT).            *
*   ROUTINES NEEDED: FPINV.                         *
*   EXECUTION CYCLES (MIN, MAX): 43 , 43.           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FDIV
        .GLOBL  FPINV

        .TEXT

;       START OF FDIV PROGRAM

FDIV:

        RND     R0,R3           ; R3 <= RND X
        LDF     R1,R0           ; R1 <= Y
        CALL    FPINV           ; R0 <= 1/Y
        RND     R0              ; ROUND BEFORE *
        MPYF    R3,R0           ; R0 <= X

        RETS                    ; RETURN

        .END
```

```
*************************************************************************
*
*    PROGRAM: $MATHX.ASM
*
*    EXTENDED-PRECISION, FLOATING-POINT (40-BIT) MATH FUNCTIONS
*
*    $MATHX.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*         SINX   - COMPUTES A 9D SIN(X) FOR ALL X IN RADIANS.
*
*         COSX   - COMPUTES A 9D COSINE(X) FOR ALL X IN RADIANS.
*
*         EXPX   - COMPUTES A 9D EXP(X) FOR ALL :X: =< 88.
*
*         LNX    - COMPUTES AN 8D LN(X) FOR ALL X > 0.
*
*         ATANX  - COMPUTES AN 8D ATAN(X) FOR ALL X IN RADIANS.
*
*         SQRTX  - COMPUTES A 10D SQRT(X) FOR ALL X >= 0.
*
*         FPINVX - COMPUTES A 10D 1/X FOR ALL X /= 0.
*
*         FDIVX  - COMPUTES A 10D X/Y FOR ALL X AND ALL Y /= 0.
*
*         FMULTX - COMPUTES A 10D X*Y FOR ALL X AND ALL Y.
*
*************************************************************************
```

```
*************************************************
*   PROGRAM: SINX                               *
*                                               *
*   WRITTEN BY: GARY A. SITTON                  *
*               GAS LIGHT SOFTWARE              *
*               HOUSTON, TEXAS                  *
*               MARCH 1989.                     *
*                                               *
*   EXTENDED PRECISION SINE FUNCTION: R0 <= SIN(R0).  *
*                                               *
*   APPROXIMATE ACCURACY: 9 DECIMAL DIGITS.     *
*   INPUT RESTRICTIONS: NONE.                   *
*   REGISTERS FOR INPUT: R0 (ARGUMENT IN RADIANS).    *
*   REGISTERS USED AND RESTORED: DP AND SP.     *
*   REGISTERS ALTERED: AR0, IR0, AND R0-7.      *
*   REGISTERS FOR OUTPUT: R0.                   *
*   ROUTINES NEEDED: FMULTX.                    *
*   EXECUTION CYCLES (MIN, MAX): 160, 160.      *
*************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  SINX
        .GLOBL  ECOSX
        .GLOBL  FMULTX

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR SIN(X)

NRM2    .WORD   00000006FH      ; BOTTOM OF 2/PI
NRM1    .WORD   0FF22F983H      ; TOP OF 2/PI

;       POLYNOMIAL COEFFS. FOR SIN(X)

SHF2    .WORD   0000000A3H      ; BOTTOM OF C1 (PI/2)
SHF1    .WORD   000490FDAH      ; TOP OF C1 (PI/2)
        .WORD   0000000D1H      ; BOTTOM OF C3
        .WORD   0FFDAA218H      ; TOP OF C3
        .WORD   0000000E3H      ; BOTTOM OF C5
        .WORD   0FC2335E0H      ; TOP OF C5
        .WORD   0F8E69754H      ; TOP OF C7
        .WORD   0F3280B28H      ; TOP OF C9
COF     .WORD   0ED9997B4H      ; TOP OF C11

ACOF    .WORD   COF             ; ADDRESS OF COEFFS.

CON     .FLOAT  -1.0, 0.0, 1.0, 0.0 ; MAPPING CONSTS.

ACON    .WORD   CON             ; ADDRESS OF CONSTS.

        .TEXT
```

```
;       START OF SINX PROGRAM

SINX:

        PUSH    DP              ; SAVE DP
        LDP     @NRM1           ; LOAD DATA PAGE POINTER

;       COSX ENTRY POINT

ECOSX:

;       SCALE AND MAP VARIABLE X

        PUSHF   R0              ; SAVE ORIGINAL X
        ABSF    R0              ; R0 <= |X|
        LDF     @NRM1,R1        ; R1 <= TOP OF 2/PI
        OR      @NRM2,R1        ; OR IN BOTTOM OF 2/PI
        CALL    FMULTX          ; R0 <= |X|*2/PI
        FIX     R0,IR0          ; IR0 <= INTEGER QUADRANT Q
        FLOAT   IR0,R1          ; R1 <= FLOATING QUADRANT Q
        SUBF    R1,R0           ; R0 <= X, -1 < X < 1
        NEGF    R0,R3           ; R3 <= -X
        ADDI    1,IR0           ; R2 <= Q + 1
        AND     3,IR0           ; IR0 <= TABLE INDEX
        TSTB    2,IR0           ; LOOK AT 2ND LSB
        LDFNZ   R3,R0           ; IF 1 THEN R0 <= -X
        LDP     @ACON           ; LOAD DATA PAGE POINTER
        LDI     @ACON,AR0       ; AR0 -> CONST. TABLE
        ADDF    *+AR0(IR0),R0   ; FINAL MAPPING, R0 <= X + C
        NEGF    R0,R3           ; R3 <= -X
        LDI     @ACOF,AR0       ; AR0 -> COEFF. TABLE

;       EVALUATE TRUNCATED SERIES

        LDF     R0,R1           ; R1 <= X
        CALL    FMULTX          ; R0 <= X**2
        LDF     R0,R1           ; R1 <= X**2

        MPYF    *AR0--,R1,R0    ; R0 <= X**2*C11
        ADDF    *AR0--,R0       ; R0 <= C9 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C9 + R0)
        ADDF    *AR0--,R0       ; R0 <= C7 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C7 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C5
        OR      *AR0--,R2       ; OR IN BOTTOM OF C5
        ADDF    R2,R0           ; R0 <= C5 + R0

        CALL    FMULTX          ; R0 <= X**2*(C5 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C3
        OR      *AR0--,R2       ; OR IN BOTTOM OF C3
        ADDF    R2,R0           ; R0 <= C3 + R0

        CALL    FMULTX          ; R0 <= X**2*(C3 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C1
        OR      *AR0,R2         ; OR IN BOTTOM OF C1
        ADDF    R2,R0,R1        ; R1 <= C1 + R0

;       TEST FOR X < 0 AND RETURN

        NEGF    R3,R0           ; R0 <= X
        BRD     FMULTX          ; R0 <= X*R1 = SIN(X), (DELAYED)
        POPF    R5              ; TEST ORIGINAL X
        LDFN    R3,R0           ; IF X < 0 THEN R0 <= -X
        POP     DP              ; UNSAVE DP

;       RETURN OCCURS FROM FMULTX !
```

```
*****************************************************
*   PROGRAM: COSX                                   *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PRECISION COSINE FUNCTION: R0 <= COS(R0).  *
*                                                   *
*   APPROXIMATE ACCURACY: 9 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: NONE.                       *
*   REGISTERS FOR INPUT: R0 (ARGUMENT IN RADIANS).  *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0, IR0, AND R0-7.          *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: ECOSX (SINX).                  *
*   EXECUTION CYCLES (MIN, MAX): 165, 165.          *
*                                                   *
*   NOTE: USES SHF1 AND SHF2 FROM SINX PROGRAM!     *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  COSX
        .GLOBL  ECOSX

        .TEXT

;       START OF COSX PROGRAM

COSX:

        PUSH    DP              ; SAVE DP
        LDP     @HRM1           ; LOAD DATA PAGE POINTER

        BRD     ECOSX           ; R0 <= COS(X) = SIN(X'), (DELAYED)
        LDF     @SHF1,R1        ; R1 <= TOP OF PI/2
        OR      @SHF2,R1        ; OR IN BOTTOM OF PI/2
        ABSF    R1,R0           ; R0 <= X' = X + PI/2

;       RETURN OCCURS FROM SINX (ALIAS FMULTX) !
```

```
*****************************************************
*   PROGRAM: EXPX                                   *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PREC. EXPONENTIAL: R0 <= EXP(R0).      *
*                                                   *
*   APPROXIMATE ACCURACY: 9 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: |R0| <= 88.0.               *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0 AND R0-7.                *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: FMULTX AND FPINVX.             *
*   EXECUTION CYCLES (MIN, MAX): 115 (R0 <=0 ), 160.  *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  EXPX
        .GLOBL  FMULTX
        .GLOBL  FPINVX

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR 2**X

ENRM2   .WORD   000000029H      ; BOTTOM OF 1/LN(2)
ENRM1   .WORD   00038AA3BH      ; TOP OF 1/LN(2)

;       POLYNOMIAL COEFFS. FOR 2**-X, 0 <= X < 1.

        .WORD   000000000H      ; C0 (1.0)
        .WORD   00000000AH      ; BOTTOM OF C1
        .WORD   0FFCE8DE8H      ; TOP OF C1
        .WORD   00000006EH      ; BOTTOM OF C2
        .WORD   0FD75FDEDH      ; TOP OF C2
        .WORD   000000046H      ; BOTTOM OF C3
        .WORD   0FB9CA833H      ; TOP OF C3
        .WORD   0F91D8C56H      ; TOP OF C4
        .WORD   0F6D1E7A9H      ; TOP OF C5
        .WORD   0F31AA7D7H      ; TOP OF C6
C7      .WORD   0EFC9BD9CH      ; TOP OF C7

AC7     .WORD   C7

        .TEXT

;       START OF EXPX PROGRAM
```

```
EXPX:

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC7            ; LOAD DATA PAGE POINTER
        NEGF    R0,R2           ; R2 <= -X
        LDF     R0,R1           ; R1 <= X
        LDFN    R2,R0           ; IF X < 0 THEN R1 <= |X|
        LDF     @ENRM1,R1       ; R1 <= TOP OF 1/LN(2)
        OR      @ENRM2,R1       ; OR IN BOTTOM OF 1/LN(2)
        CALL    FMULTX          ; R0 <= X * |X|/LN(2)
        FIX     R0,R3           ; R3 <= I = INTEGER OF X
        FLOAT   R3,R1           ; R1 <= FLT. PT. I
        SUBF    R1,R0,R1        ; R1 <= FRACTION OF |X|, 0 <= X < 1
        NEGI    R3              ; R3 <= -I
        LSH     24,R3           ; MOVE -I TO EXP.
        PUSH    R3              ; SAVE AS INT.
        POPF    R3              ; R3 <= FLT. PT. 2**-I
        LDI     @AC7,AR0        ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED SERIES

        MPYF    *AR0--,R1,R0    ; R0 <= X*C7
        ADDF    *AR0--,R0       ; R0 <= C6 + R0

        MPYF    R1,R0           ; R0 <= X*(C6 + R0)
        ADDF    *AR0--,R0       ; R0 <= C5 + R0

        MPYF    R1,R0           ; R0 <= X*(C5 + R0)
        ADDF    *AR0--,R0       ; R0 <= C4 + R0

        MPYF    R1,R0           ; R0 <= X*(C4 + R0)
        LDF     *AR0--,R4       ; R4 <= TOP OF C3
        OR      *AR0--,R4       ; OR IN BOTTOM OF C3
        ADDF    R4,R0           ; R0 <= C3 + R0

        MPYF    R1,R0           ; R0 <= X*(C3 + R0)
        LDF     *AR0--,R4       ; R4 <= TOP OF C2
        OR      *AR0--,R4       ; OR IN BOTTOM OF C2
        ADDF    R4,R0           ; R0 <= C2 + R0

        CALL    FMULTX          ; R0 <= X*(C2 + R0)
        LDF     *AR0--,R4       ; R4 <= TOP OF C1
        OR      *AR0--,R4       ; OR IN BOTTOM OF C1
        ADDF    R4,R0           ; R0 <= C1 + R0

        CALL    FMULTX          ; R0 <= X*(C1 + R0)

;       TEST FOR X < 0 AND RETURN

        LDF     R2,R2           ; TEST ORIGINAL -X

        BND     FPINVX          ; IF -X < 0 THEN R0 <= 1/X, (DELAYED)
        ADDF    *AR0,R0,R1      ; R1 <= 2**-X = C0 + R0
        MPYF    R3,R1,R0        ; R0 <= 2**-(I + X) TRUNC.
        LDM     R1,R0           ; R0 <= FULL MANTISSA

        RETS                    ; RETURN (IF NO FPINVX BRANCH)
```

```
*****************************************************
*   PROGRAM: LNX                                    *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PREC. LOGARITHM BASE E: R0 <= LN(R0).  *
*                                                   *
*   APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.         *
*   INPUT RESTRICTIONS: R0 > 0.0.                   *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: AR0 AND R0-7.                *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: FMULTX.                        *
*   EXECUTION CYCLES (MIN, MAX): 193, 193.          *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  LNX
        .GLOBL  FMULTX

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR LN(1+X)

LNRM2   .WORD   0000000F7H      ; BOTTOM OF LN(2)
LNRM1   .WORD   0FF317217H      ; TOP OF LN(2)

;       POLYNOMIAL COEFFS. FOR LN(1+X), 0 <= X < 1.

C0      .FLOAT  1.0             ; C0 (1.0)

        .WORD   0000000FFH      ; BOTTOM OF C1
        .WORD   0FF7FFFC3H      ; TOP OF C1
        .WORD   000000B4H       ; BOTTOM OF C2
        .WORD   0FEB0107FH      ; TOP OF C2
        .WORD   000000D9H       ; BOTTOM OF C3
        .WORD   0FE29E18FH      ; TOP OF C3
        .WORD   000000097H      ; BOTTOM OF C4
        .WORD   0FD897D13H      ; TOP OF C4
        .WORD   000000041H      ; BOTTOM OF C5
        .WORD   0FD2BAD82H      ; TOP OF C5
        .WORD   0000000E7H      ; BOTTOM OF C6
        .WORD   0FC3CC3F1H      ; TOP OF C6
        .WORD   000000043H      ; BOTTOM OF C7
        .WORD   0FB13D187H      ; TOP OF C7
C8      .WORD   0FBAC87BFH      ; TOP OF C8

AC8     .WORD   C8

        .TEXT

;       START OF LNX PROGRAM

LNX:

        LDF     R0,R0           ; TEST X
        RETSLE                  ; RETURN NOW IF X <= 0

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC8            ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE AS FLT. PT.
        POP     R3              ; R3 <= INTEGER FORMAT
        ASH     -24,R3          ; R3 <= E = SIGNED EXP.
        FLOAT   R3,R1           ; R1 <= FLT. PT. E VALUE
        LDF     @C0,R2          ; R2 <= 1.0
        LDE     R2,R0           ; EXP. R0 <= 0 (1 <= X < 2)
        SUBRF   R0,R2           ; R2 <= X - 1 (0 <= X < 1)
        LDF     @LNRM1,R0       ; R0 <= TOP OF LN(2)
        OR      @LNRM2,R0       ; OR IN BOTTOM OF LN(2)
        CALL    FMULTX          ; R0 <= E*LN(2)
        LDF     R0,R3           ; R3 <= E*LN(2)
        LDI     @AC8,AR0        ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED SERIES

        LDF     R2,R1           ; R1 <= X
        MPYF    *AR0--,R1,R0    ; R0 <= X*C8
        LDF     *AR0--,R2       ; R2 <= TOP OF C7
        OR      *AR0--,R2       ; OR IN BOTTOM OF C7
        ADDF    R2,R0           ; R0 <= C7 + R0

        MPYF    R1,R0           ; R0 <= X*(C7 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C6
        OR      *AR0--,R2       ; OR IN BOTTOM OF C6
        ADDF    R2,R0           ; R0 <= C6 + R0

        MPYF    R1,R0           ; R0 <= X*(C6 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C5
        OR      *AR0--,R2       ; OR IN BOTTOM OF C5
        ADDF    R2,R0           ; R0 <= C5 + R0

        CALL    FMULTX          ; R0 <= X*(C5 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C4
        OR      *AR0--,R2       ; OR IN BOTTOM OF C4
        ADDF    R2,R0           ; R0 <= C4 + R0

        CALL    FMULTX          ; R0 <= X*(C4 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C3
```

```
        OR      *AR0--,R2       ; OR IN BOTTOM OF C3
        ADDF    R2,R0           ; R0 <= C3 + R0

        CALL    FMULTX          ; R0 <= X*(C3 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C2
        OR      *AR0--,R2       ; OR IN BOTTOM OF C2
        ADDF    R2,R0           ; R0 <= C2 + R0

        CALL    FMULTX          ; R0 <= X*(C2 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C1
        OR      *AR0--,R2       ; OR IN BOTTOM OF C1
        ADDF    R2,R0           ; R0 <= C1 + R0

        CALL    FMULTX          ; R0 <= X*(C1 + R0)

;       ADD IN SCALED EXPONENT.

        ADDF    R3,R0           ; R0 <= LN(X) + E*LN(2)

        RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: ATANX                                   *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              MARCH 1989.                          *
*                                                   *
*  EXTENDED PRECISION ARC TANGENT: R0 <= ATAN(R0).  *
*                                                   *
*  APPROXIMATE ACCURACY: 8 DECIMAL DIGITS.          *
*  INPUT RESTRICTIONS: NONE.                        *
*  REGISTERS FOR INPUT: R0.                         *
*  REGISTERS USED AND RESTORED: DP AND SP.          *
*  REGISTERS ALTERED: AR0, IR0, AND R0-7.           *
*  REGISTERS FOR OUTPUT: R0 (IN RADIANS).           *
*  ROUTINES NEEDED: FMULTX, AND FDIVX.              *
*  EXECUTION CYCLES (MIN, MAX): 210 (!ATANX!<=1), 332. *
*****************************************************
```

```
;       EXTERNAL PROGRAM NAMES

        .GLOBL  ATANX
        .GLOBL  FMULTX
        .GLOBL  FDIVX

;       INTERNAL CONSTANTS

        .DATA

;       SCALING COEFFS. FOR ATAN(X)

        .WORD   00000005DH      ; BOTTOM OF -PI/4
        .WORD   0FFB6F025H      ; TOP OF -PI/4
        .WORD   0000000A2H      ; BOTTOM OF PI/4
        .WORD   0FF490FDAH      ; TOP OF PI/4
        .WORD   000000000H      ; BOTTOM OF ZERO
        .WORD   080000000H      ; TOP OF ZERO

;       POLYNOMIAL COEFFS. FOR ATAN(X), -1 <= X <= 1.

C1      .WORD   000000000H      ; TOP OF C1 (1.0)
        .WORD   00000006EH      ; BOTTOM OF C3
        .WORD   0FED55594H      ; TOP OF C3
        .WORD   000000009H      ; BOTTOM OF C5
        .WORD   0FD4CB8E4H      ; TOP OF C5
        .WORD   0000000FFH      ; BOTTOM OF C7
        .WORD   0FDEE8038H      ; TOP OF C7
        .WORD   000000056H      ; BOTTOM OF C9
        .WORD   0FC5A3D83H      ; TOP OF C9
        .WORD   000000093H      ; BOTTOM OF C11
        .WORD   0FCE5CE8BH      ; TOP OF C11
        .WORD   0000000BFH      ; BOTTOM OF C13
        .WORD   0FB2FC1FDH      ; TOP OF C13
```

```
        .WORD   0FAFB91FEH      ; TOP OF C15
C17     .WORD   0F738D74AH      ; TOP OF C17

AC17    .WORD   C17

        .TEXT

;       START OF ATANX PROGRAM

ATANX:

;       SCALE VARIABLE X

        PUSH    DP              ; SAVE DP
        LDP     @AC17           ; LOAD DATA PAGE POINTER
        ABSF    R0,R2           ; R2 <= |X|
        SUBF    @C1,R2          ; R2 <= |X| - 1
        BLED    SKIP            ; IF |X| > 1 THEN SCALE (DELAYED)
        LDF     R0,R3           ; R3 <= X
        LDF     R0,R1           ; R1 <= X
        LDI     0,IR0           ; IR0 <= 0, POST SCALE INDEX

;       SCALE FOR |X| > 1

        PUSHF   R0              ; SAVE X
        ABSF    R0,R1           ; R1 <= |X|
        ADDF    @C1,R1          ; R1 <= |X| + 1
        LDF     R2,R0           ; R0 <= |X| - 1
        CALL    FDIVX           ; R0 <= (|X| - 1)/(|X| + 1)

;       TEST FOR X' < 0

        POPF    R4              ; GET ORIGINAL X
        BGED    SKIP            ; IF X < 0 THEN R0 <= -X' (DELAYED)
        LDF     R0,R3           ; R3 <= X'
        LDF     R0,R1           ; R1 <= X'
        SUBI    2,IR0           ; IR0 <= -2, (PI/4)

        NEGF    R0,R3           ; R3 <= -X'
        SUBI    2,IR0           ; IR0 <= -4, (-PI/4)

SKIP:   CALL    FMULTX          ; R0 <= X**2
        LDI     @AC17,AR0       ; AR0 -> COEFF. TABLE
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED (ODD) SERIES

        LDF     R0,R1           ; R1 <= X**2
        MPYF    *AR0--,R1,R0    ; R0 <= X**2*C17
        ADDF    *AR0--,R0       ; R0 <= C15 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C15 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C13
        OR      *AR0--,R2       ; OR IN BOTTOM OF C13
        ADDF    R2,R0           ; R0 <= C13 + R0

        MPYF    R1,R0           ; R0 <= X**2*(C13 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C11
        OR      *AR0--,R2       ; OR IN BOTTOM OF C11
        ADDF    R2,R0           ; R0 <= C11 + R0

        CALL    FMULTX          ; R0 <= X**2*(C11 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C9
        OR      *AR0--,R2       ; OR IN BOTTOM OF C9
        ADDF    R2,R0           ; R0 <= C9 + R0

        CALL    FMULTX          ; R0 <= X**2*(C9 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C7
        OR      *AR0--,R2       ; OR IN BOTTOM OF C7
        ADDF    R2,R0           ; R0 <= C7 + R0

        CALL    FMULTX          ; R0 <= X**2*(C7 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C5
        OR      *AR0--,R2       ; OR IN BOTTOM OF C5
        ADDF    R2,R0           ; R0 <= C5 + R0

        CALL    FMULTX          ; R0 <= X**2*(C5 + R0)
        LDF     *AR0--,R2       ; R2 <= TOP OF C3
        OR      *AR0--,R2       ; OR IN BOTTOM OF C3
        ADDF    R2,R0           ; R0 <= C3 + R0

        CALL    FMULTX          ; R0 <= X**2*(C3 + R0)

;       FINISH UP

        ADDF    *AR0--,R0,R1    ; R1 <= C1 + R0
        LDF     R3,R0           ; R0 <= X (SIGNED)
        CALL    FMULTX          ; R0 <= ATANX(X) = X*(1 + R0)
        NOP     *AR0++(IR0)     ; AR0 -> C (0.0, PI/4 OR -PI/4)

;       ADD IN POST SCALE VALUE C AND RETURN

        POP     R4              ; R4 <= RETURN ADDRESS
        BUD     R4              ; RETURN (DELAYED)
        LDF     *AR0--,R1       ; R1 <= TOP OF C
        OR      *AR0,R1         ; OR IN BOTTOM OF C
        ADDF    R1,R0           ; R0 <= ATAN(X) + C
```

```
*********************************************************
*   PROGRAM: SQRTI                                      *
*                                                       *
*   WRITTEN BY: GARY A. SITTON                          *
*               GAS LIGHT SOFTWARE                      *
*               HOUSTON, TEXAS                          *
*               MARCH 1989.                             *
*                                                       *
*   APPROXIMATE ACCURACY: 10 DECIMAL DIGITS.            *
*   INPUT RESTRICTIONS: R0 >= 0.0.                      *
*   REGISTERS FOR INPUT: R0.                            *
*   REGISTERS USED AND RESTORED: DP AND SP.             *
*   REGISTERS ALTERED: R0-7.                            *
*   REGISTERS FOR OUTPUT: R0.                           *
*   ROUTINES NEEDED: FMULTX.                            *
*   EXECUTION CYCLES (MIN, MAX): 138, 138.              *
*********************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  SQRTX
        .GLOBL  FMULTX

;       INTERNAL CONSTANTS

        .DATA

CNST1   .SET    0.5
CNST2   .SET    1.5
CNST3   .FLOAT  1.103553391     ; ADJUSTED 1.0
CNST4   .FLOAT  0.780330086     ; ADJUSTED SQRT(1/2)

SMSK    .WORD   0FF7FFFFFH

        .TEXT

;       START OF SQRTX PROGRAM.

SQRTX:

        LDF     R0,R3           ; TEST AND SAVE V
        RETSLE                  ; RETURN NOW IF V <= 0

;       GET APPROXIMATION TO 1/V.  FOR V = (1+M)*2**E
;       AND 0 <= M < 1, FOR E EVEN: X[0] = (1-M/2)*2**-E/2
;       AND FOR E ODD: X[0] = SQRT(1/2)*(1-M/2)*2**-E/2

        PUSH    DP              ; SAVE DP
        LDP     @SMSK           ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE V AS FLT. PT. V = (1+M)*2**E
        POP     R4              ; R4 <= V AS INTEGER
        XOR     @SMSK,R4        ; R4 <= COMPLEMENT ALL BUT SIGN
        LDI     R4,R1           ; R1 <= (1-M/2)*2**-E
        LDI     R4,R5           ; R5 <= R1
        LSH     8,R1            ; R1 <= R1 EXP. REMOVED
        ASH     -1,R4           ; R4 <= R4 WITH -E/2 EXP.
        PUSH    R4              ; SAVE R4 AS INTEGER
        POPF    R4              ; R4 <= FLT. PT.
        LDE     R4,R1           ; R1 <= (1-M/2)*2**-E/2
        LDF     @CNST3,R2       ; R2 <= 1.1... FOR ODD E
        LSH     7,R5            ; TEST LSB OF E (AS SIGN)
        LDFNN   @CNST4,R2       ; IF E EVEN R2 <= 0.78...
        MPYF    R2,R1           ; R1 <= CORRECTED ESTIMATE

;       GENERATE V/2 (USES MPYF).

        MPYF    CNST1,R0        ; R0 <= V/2 TRUNC.
        LDM     R3,R0           ; R0 <= V/2 FULL PREC.

;       NEWTON ITERATION FOR Y(X) = X - V**-2 = 0 ...

        MPYF    R1,R1,R2        ; R2 <= X[0]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[0]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[0]**2
        MPYF    R2,R1           ; R1 <= X[1] = X[0] * (1.5 - (V/2)*X[0]**2)
;
        MPYF    R1,R1,R2        ; R2 <= X[1]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[1]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[1]**2
        MPYF    R2,R1           ; R1 <= X[2] = X[1] * (1.5 - (V/2)*X[1]**2)
;
        MPYF    R1,R1,R2        ; R2 <= X[2]**2
        MPYF    R0,R2           ; R2 <= (V/2) * X[2]**2
        SUBRF   CNST2,R2        ; R2 <= 1.5 - (V/2) * X[2]**2
        MPYF    R2,R1           ; R1 <= X[3] = X[2] * (1.5 - (V/2)*X[2]**2)
;
        LDF     R0,R2           ; R2 <= V/2
        LDF     R1,R0           ; R0 <= X[3]
        CALL    FMULTX          ; R0 <= X[3]**2
        LDF     R1,R4           ; R4 <= X[3]
        LDF     R2,R1           ; R1 <= V/2
        LDF     R4,R2           ; R2 <= X[3]
        CALL    FMULTX          ; R0 <= (V/2) * X[3]**2
        SUBRF   CNST2,R0        ; R0 <= 1.5 - (V/2) * X[3]**2
        LDF     R2,R1           ; R1 <= X[3]
        CALL    FMULTX          ; R0 <= X[4] = X[3] * (1.5 - (V/2)*X[3]**2)

;       INVERT FINAL RESULT AND RETURN

        BRD     FMULTX          ; R0 = SQRT(V) = V*SQRT(1/V) (DELAYED)
        LDF     R3,R1           ; R1 <= V
        POP     DP              ; UNSAVE DP
        NOP                     ; DEAD CYCLE

;       RETURN OCCURS FROM FMULTX !
```

```
*****************************************************
*   PROGRAM: FPINVX                                 *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PREC. FLT. PT. INVERSE: R0 <= 1/R0.    *
*                                                   *
*   APPROXIMATE ACCURACY: 10 DECIMAL DIGITS.        *
*   INPUT RESTRICTIONS: R0 != 0.0.                  *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-1 AND R4-7.               *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: FMULTX.                        *
*   EXECUTION CYCLES (MIN, MAX): 76, 76.            *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FPINVX
        .GLOBL  FMULTX

;       INTERNAL CONSTANTS

        .DATA

ONE     .SET    1.0
TWO     .SET    2.0

MSK     .WORD   0FF7FFFFFH

        .TEXT

;       START OF FPINVX PROGRAM

FPINVX:

        LDF     R0,R0           ; TEST F
        RETSZ                   ; RETURN NOW IF F = 0

;       GET APPROXIMATION TO 1/F.  FOR F = (1+M) * 2**E
;       AND 0 <= M < 1, USE: X[0] = (1-M/2) * 2**-E

        PUSH    DP              ; SAVE DP
        LDP     @MSK            ; LOAD DATA PAGE POINTER
        PUSHF   R0              ; SAVE AS FLT. PT. F = (1+M) * 2**E
        POP     R1              ; FETCH BACK AS INTEGER
        XOR     @MSK,R1         ; COMPLEMENT E & M BUT NOT SIGN BIT
        PUSH    R1              ; SAVE AS INTEGER, AND BY MAGIC...
        POPF    R1              ; R1 <= X[0] = (1-M/2) * 2**-E.
        POP     DP              ; UNSAVE DP

;       NEWTON ITERATION FOR: Y(X) = X - 1/F = 0 ...

        MPYF    R1,R0,R4        ; R4 <= F * X[0]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[0]
        MPYF    R4,R1           ; R1 <= X[1] = X[0] * (2 - F * X[0])

        MPYF    R1,R0,R4        ; R4 <= F * X[1]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[1]
        MPYF    R4,R1           ; R1 <= X[2] = X[1] * (2 - F * X[1])

        MPYF    R1,R0,R4        ; R4 <= F * X[2]
        SUBRF   TWO,R4          ; R4 <= 2 - F * X[2]
        MPYF    R4,R1           ; R1 <= X[3] = X[2] * (2 - F * X[2])

;       FOR THE LAST ITERATION: X[4] = (X[3] * (1 - (F * X[3]))) + X[3]

        CALL    FMULTX          ; R0 <= F * X[3] = 1 + EPS
        SUBRF   ONE,R0          ; R0 <= 1 - F * X[3] = EPS
        CALL    FMULTX          ; R0 <= X[3] * EPS
        ADDF    R1,R0           ; R0 <= X[4] = (X[3]*(1 - (F*X[3]))) + X[3]

        RETS                    ; RETURN

        .END
```

```
*****************************************************
*   PROGRAM: FDIVX                                  *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PRECISION DIVIDE: R0 <= R0/R1.         *
*                                                   *
*   APPROXIMATE ACCURACY: 10 DECIMAL DIGITS.        *
*   INPUT RESTRICTIONS: R1 != 0.0.                  *
*   REGISTERS FOR INPUT: R0 (DIVIDEND) AND R1 (DIVISOR).*
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0-7.                        *
*   REGISTERS FOR OUTPUT: R0 (QUOTIENT).            *
*   ROUTINES NEEDED: FMULTX AND FPINVX.             *
*   EXECUTION CYCLES (MIN, MAX): 107, 107.          *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FDIVX
        .GLOBL  FPINVX
        .GLOBL  FMULTX

        .TEXT

;       START OF FDIVX PROGRAM

FDIVX:

        LDF     R0,R3           ; R3 <= X
        LDF     R1,R0           ; R1 <= Y
        CALL    FPINVX          ; R0 <= 1/Y
        LDF     R3,R1           ; R1 <= X
        BR      FMULTX          ; R0 <= X/Y

;       RETURN OCCURS FROM FMULTX !
```

```
*****************************************************
*   PROGRAM: FMULTX                                 *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   EXTENDED PRECISION MULTIPLY: R0 <= R0*R1.       *
*                                                   *
*   APPROXIMATE ACCURACY: 10 DECIMAL DIGITS.        *
*   INPUT RESTRICTIONS: NONE.                       *
*   REGISTERS FOR INPUT: R0.                        *
*   REGISTERS USED AND RESTORED: DP AND SP.         *
*   REGISTERS ALTERED: R0 AND R4-7.                 *
*   REGISTERS FOR OUTPUT: R0.                       *
*   ROUTINES NEEDED: NONE.                          *
*   EXECUTION CYCLES (MIN, MAX): 20, 20.            *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  FMULTX

        .TEXT

;       START OF FMULTX PROGRAM

FMULTX:

        ABSF    R0,R4           ; R4 <= |XA|
        XOR     R1,R0           ; R0 <= SIGN INFO.
        ABSF    R1,R7           ; R7 <= |XB|
        MPYF    R4,R7,R6        ; R6 <= A*B
        LDF     R4,R5           ; R5 <= |XA|
        ANDN    0FFH,R5         ; R5 <= A = XA - EA*2**-24
        SUBRF   R4,R5           ; R5 <= EA*2**-24
        MPYF    R7,R5           ; R5 <= B*EA*2**-24
        ADDF    R6,R5           ; R5 <= A*B + B*EA*2**-24
        LDF     R7,R6           ; R6 <= |XB|
        ANDN    0FFH,R6         ; R6 <= B = XB - EB*2**-24
        SUBRF   R7,R6           ; R6 <= EB*2**-24
        MPYF    R4,R6           ; R6 <= A*EB*2**-24
        ADDF    R6,R5           ; R5 <= |XA*XB| = A*B + (B*EA+A*EB)*2**-24
        NEGF    R5,R6           ; R6 <= - |XA*XB|

        TEST FOR XA*XB < 0 AND RETURN

        POP     R4              ; R4 <= RETURN ADDRESS
        BUD     R4              ; RETURN (DELAYED)
        LDF     R0,R0           ; TEST ORIGINAL (XA ^ XB)
        LDFN    R6,R5           ; IF XA*XB < 0 THEN R5 <= -|XA*XB|
        LDF     R5,R0           ; R0 <= XA*XB
```

```
*************************************************************************
*
*   PROGRAM: %MATHI.ASM
*
*   INTEGER (32-BIT) MATH ROUTINES
*
*   %MATHI.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*       ILOG2 - COMPUTES M = LOG2(N), N=< 2**M FOR USE WITH RADIX 2 FFT
*               PROGRAMS.
*
*       IMULT - COMPUTES A 64-BIT PRODUCT OF TWO 32-BIT NUMBERS.
*
*       IDIV  - COMPUTES THE QUOTIENT AND REMAINDER OF TWO 32-BIT NUMBERS.
*
*************************************************************************
```

```
*************************************************
*  PROGRAM: ILOG2                               *
*                                               *
*  WRITTEN BY: GARY A. SITTON                   *
*              GAS LIGHT SOFTWARE               *
*              HOUSTON, TEXAS                   *
*              MARCH 1989.                      *
*                                               *
*  INTEGER LOG BASE 2: R0 <= (INTEGER) LOG2(R0).*
*                                               *
*  INPUT RESTRICTIONS: R0 > 0.                  *
*  REGISTERS FOR INPUT: R0.                     *
*  REGISTERS USED AND RESTORED: SP.             *
*  REGISTERS ALTERED: IR0-1 AND R0.             *
*  REGISTERS FOR OUTPUT: R0.                    *
*  ROUTINES NEEDED: NONE.                       *
*************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  ILOG2

        .TEXT

;       START OF ILOG2 PROGRAM

ILOG2:

        LDI     1,IR0           ; IR0 <= I (INIT. 1)
        LDI     -1,IR1          ; IR1 <= M (INIT. -1)

        CMPI    IR0,R0          ; COMPARE I TO N
LOOP:   BGTD    LOOP            ; LOOP IF N > I (DELAYED)
        LSH     1,IR0           ; I <= 2*I
        ADDI    1,IR1           ; M = M + 1
        CMPI    IR0,R0          ; COMPARE I TO N

        LDI     IR1,R0          ; R0 <= LOG2(N)
        RETS                    ; RETURN
```

```
*********************************************************
*   PROGRAM: IMULT                                      *
*                                                       *
*   WRITTEN BY: GARY A. SITTON                          *
*               GAS LIGHT SOFTWARE                      *
*               HOUSTON, TEXAS                          *
*               MARCH 1989.                             *
*                                                       *
*   INTEGER 32 X 32 MULTIPLY: R1, R0 <= R0*R1.          *
*   RESULT IS THE 64 BIT PRODUCT OF TWO 32 BIT INPUTS.  *
*                                                       *
*   INPUT RESTRICTIONS: NONE.                           *
*   REGISTERS FOR INPUT: R0 AND R1.                     *
*   REGISTERS USED AND RESTORED: SP.                    *
*   REGISTERS ALTERED: AR0-1 AND R0-4.                  *
*   REGISTERS FOR OUTPUT: R1 (UPPER) AND R0 (LOWER).    *
*   ROUTINES NEEDED: NONE.                              *
*********************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  IMULT

        .TEXT

;       START OF IMULT PROGRAM

IMULT:

        XOR     R0,R1,AR0       ; AR0 <= SIGNUM (R0*R1)
        ABSI    R0              ; R0 <= |X|
        ABSI    R1              ; R1 <= |Y|

;       SEPARATE MULTIPLIER AND MULTIPLICAND IN TWO PARTS

        LDI     -16,AR1         ; AR1 <= -16 (FOR SHIFTS)
        LSH     AR1,R0,R2       ; R2 <= X1 = UPPER 16 BITS OF |X|
        AND     0FFFFH,R0       ; R0 <= X0 = LOWER 16 BITS OF |X|
        LSH     AR1,R1,R3       ; R3 <= Y1 = UPPER 16 BITS OF |Y|
        AND     0FFFFH,R1       ; R1 <= Y0 = LOWER 16 BITS OF |Y|

;       CARRY OUT THE MULTIPLICATION

        MPYI    R0,R1,R4        ; R4 <= X0*Y0 = P1
        MPYI    R3,R0           ; R0 <= X0*Y1 = P2
        MPYI    R2,R1           ; R1 <= X1*Y0 = P3
        ADDI    R0,R1           ; R1 <= P2+P3
        MPYI    R2,R3           ; R3 <= X1*Y1 = P4

;       PUT THE PRODUCTS TOGETHER

        LDI     R1,R2           ; R2 <= P2+P3
        LSH     16,R2           ; R2 <= LOWER 16 BITS OF P2+P3
        CMPI    0,AR0           ; CHECK THE SIGN OF THE PRODUCT
        BGED    DONE            ; IF >= 0 THEN DONE (DELAYED)
        LSH     AR1,R1          ; R1 <= UPPER 16 BITS OF P2+P3
        ADDI    R4,R2,R0        ; R0 <= W0 = LOWER WORD OF THE PRODUCT
        ADDC    R3,R1           ; R1 <= W1 = UPPER WORD OF THE PRODUCT

;       NEGATE THE PRODUCT IF NUMBERS WERE OF OPPOSITE SIGN

        SUBRI   0,R0            ; R0 <= -W0
        SUBRB   0,R1            ; R1 <= -W1 (WITH BORROW)

DONE:   RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: IDIV                                    *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              MARCH 1989.                          *
*                                                   *
*                                                   *
*  INTEGER 32 / 32 DIVIDE: R0, R1 <= R0/R1.         *
*  RESULT IS A 32 BIT QUOTIENT AND |REMAINDER|.     *
*                                                   *
*  INPUT RESTRICTIONS: R1 != 0.                     *
*  REGISTERS FOR INPUT: R0 (DIVIDEND) AND R1 (DIVISOR).*
*  REGISTERS USED AND RESTORED: SP.                 *
*  REGISTERS ALTERED: IR0-1 AND R0-3.               *
*  REGISTERS FOR OUTPUT: R0 (QUOTIENT) AND          *
*     R1 (|REMAINDER|).                             *
*  ROUTINES NEEDED: NONE.                           *
*****************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  IDIV

;       START OF IDIV PROGRAM

        .TEXT

IDIV:

;       DETERMINE SIGN OF RESULT.  GET ABSOLUTE VALUE OF OPERANDS.

        XOR     R0,R1,R2        ; R2 <= SIGNUM (R0/R1)
        ABSI    R0              ; R0 <= |X|
        ABSI    R1              ; R1 <= |Y|

;       TEST INPUT VALUES

        CMPI    R0,R1           ; COMPARE DIVISOR TO DIVIDEND
        BHID    ZERO            ; IF R1 > R0 THEN RETURN 0 (DELAYED)

;       NORMALIZE OPERANDS.  USE DIFFERENCE IN EXPONENTS AS
;       SHIFT COUNT FOR DIVISOR, AND AS REPEAT COUNT FOR SUBC.

        FLOAT   R0,R3           ; R3 <= NORMALIZED DIVIDEND
        PUSHF   R3              ; PUSH AS FLOAT
        POP     IR1             ; IR1 <= INTEGER
        LSH     -24,IR1         ; IR1 <= DIVIDEND EXPONENT

        FLOAT   R1,R3           ; R3 <= NORMALIZED DIVISOR
        PUSHF   R3              ; PUSH AS FLOAT
        POP     IR0             ; IR0 <= INTEGER
        LSH     -24,IR0         ; IR0 <= DIVISOR EXPONENT

        SUBI    IR0,IR1         ; IR1 <= DIFFERENCE IN EXPONENTS
        LSH     IR1,R1          ; R1 <= ALIGNED DIVISOR WITH DIVIDEND

;       DO IR1+1 SUBTRACT & SHIFTS.

        RPTS    IR1             ; REPEAT IR1+1 TIMES
        SUBC    R1,R0           ; R0 <= 2*(R0 - R1)

;       MASK OFF THE LOWER IR1+1 BITS OF R0

        LDI     R0,R1           ; R1 <= |REMAINDER, QUOTIENT|
        SUBRI   31,IR1          ; IR1 <= 32 - (IR1+1)
        LSH     IR1,R0          ; R0 <= R0 SHIFT LEFT IR1
        NEGI    IR1             ; IR1 <= -IR1
        LSH     IR1,R0          ; R0 <= |X|/|Y|
        SUBRI   -32,IR1         ; IR1 <=  -(IR1+1)
        LSH     IR1,R1          ; R1 <= |REMAINDER|

;       CHECK SIGN AND NEGATE RESULT IF NECESSARY.

        NEGI    R0,R3           ; R3 <= -|X|/|Y|
        ASH     -31,R2          ; TEST SIGN BIT
        LDINZ   R3,R0           ; IF SET R0 <= -R0
        CMPI    0,R0            ; SET STATUS FROM RESULT
        RETS                    ; RETURN

;       RETURN ZERO QUOTIENT.

ZERO:   LDI     R0,R1           ; R1 <= |REMAINDER|
        LDI     0,R0            ; R0 <= 0 QUOTIENT
        RETS                    ; RETURN

        .END
```

```
*****************************************************************
*
*    PROGRAM: $VECTOR.ASM
*
*    VECTOR UTILITIES
*
*    $VECTOR.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*         *CORMULT - IN-PLACE COMPUTATION OF THE COMPLEX VECTOR PRODUCT OF TWO
*                    COMPLEX ARRAYS USING THE COMPLEX CONJUGATE OF THE SECOND
*                    ARRAY.
*
*         *CONMULT - IN-PLACE COMPUTATION OF THE COMPLEX VECTOR PRODUCT OF TWO
*                    COMPLEX ARRAYS.
*
*         *CBITREV - IN-PLACE BIT REVERSE PERMUTATION ON A COMPLEX ARRAY WITH
*                    SEPARATE REAL AND IMAGINARY ARRAYS.
*
*         *FMIEEE  - IN-PLACE FAST CONVERSION OF AN IEEE ARRAY TO A TMS320C30
*                    ARRAY.
*
*         *TOIEEE  - IN-PLACE FAST CONVERSION OF A TMS320C30 ARRAY TO AN IEEE
*                    ARRAY.
*
*         *VECMULT - IN-PLACE MULTIPLIES A CONSTANT TIMES AN ARRAY.
*
*         *CONMOV  - MOVES (FILLS) A CONSTANT INTO AN ARRAY.
*
*         *VECMOV  - MOVES (COPIES) AN ARRAY INTO ANOTHER ARRAY.
*
*****************************************************************
```

```
*****************************************************
*   PROGRAM: *CORMULT                               *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               FEBRUARY 1989.                      *
*                                                   *
*   COMPLEX IN-PLACE FREQUENCY DOMAIN CORRELATION:  *
*   C1 (= C1 * CONJ(C2), C1 AND C2 ARE BOTH OF LENGTH
*   N, AND C1 = (X1 + I*Y1) AND CONJ(C2) = (X2 - I*Y2).
*                                                   *
*   MCORMULT ENTRY PROTOCOL:                        *
*      VARIABLES FOR INPUT:                         *
*         $IAD1 -> X1[0], $IAD2 -> Y1[0],           *
*         $SAD1 -> X2[0], $SAD2 -> Y2[0],           *
*         $N = N (LENGTH), $PARMS = DATA PAGE.      *
*      INPUT RESTRICTIONS: $N > 0.                  *
*      REGISTERS ALTERED: RC, DP, AR0-3 AND R0-3.   *
*                                                   *
*   RCORMULT ENTRY PROTOCOL:                        *
*      REGISTERS FOR INPUT:                         *
*         AR0 -> X1[0], AR1 -> Y1[0], AR2 -> X2[0], *
*         AR3 -> Y2[0], RC = N (LENGTH).            *
*      INPUT RESTRICTIONS: RC > 0.                  *
*      REGISTERS ALTERED: RC, AR0-3 AND R0-3.       *
*                                                   *
*   REGISTERS USED AND RESTORED: SP.                *
*   REGISTERS FOR OUTPUT: NONE.                     *
*   ROUTINES NEEDED: NONE.                          *
*****************************************************
```

```
;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X1
        .GLOBL  $IAD2       ; ADDRESS OF INPUT Y1
        .GLOBL  $SAD1       ; ADDRESS OF INPUT X2
        .GLOBL  $SAD2       ; ADDRESS OF INPUT Y2

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MCORMULT    ; MEMORY ENTRY FOR COMPLEX (CORR.) MULTIPLY
        .GLOBL  RCORMULT    ; REGISTER ENTRY FOR COMPLEX (CORR.) MULTIPLY

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY
```

```
MCONMULT:

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,RC          ; RC <= N
        LDI     @$IAD1,AR0      ; AR0 -> X1[0]
        LDI     @$IAD2,AR1      ; AR1 -> Y1[0]
        LDI     @$SAD1,AR2      ; AR2 -> X2[0]
        LDI     @$SAD2,AR3      ; AR3 -> Y2[0]

;       REGISTER BASED PARAMETER ENTRY

RCONMULT:

;       COMPLEX MULTIPLY (CORRELATION) LOOP

        SUBI    1,RC            ; RC <= N - 1

        RPTB    LOOP1           ; REPEAT BLOCK N TIMES
        MPYF    *AR0,*AR2,R1    ; R1 <= X1[I]*X2[I]
        MPYF    *AR1,*AR3,R3    ; R3 <= Y1[I]*Y2[I]
        MPYF    *AR2++,*AR1,R0  ; R0 <= Y1[I]*X2[I], INCR. AR2 AND...
||      ADDF    R1,R3,R2        ; R2 <= X1[I]*X2[I] + Y1[I]*Y2[I]
        MPYF    *AR0,*AR3++,R1  ; R1 <= X1[I]*Y2[I], INCR. AR3
        SUBF    R1,R0,R3        ; R3 <= Y1[I]*X2[I] - X1[I]*Y2[I]
LOOP1:  STF     R2,*AR0++       ; X1[I] <= R2, INCR. AR0 AND...
||      STF     R3,*AR1++       ; Y1[I] <= R3, INCR. AR1

        RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: *CONMULT                                *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              APRIL 1989.                          *
*                                                   *
*  COMPLEX IN-PLACE FREQUENCY DOMAIN CONVOLUTION:   *
*  C1 <= C1 * C2, C1 AND C2 ARE BOTH OF LENGTH      *
*  N, AND C1 = (X1 + I*Y1) AND C2 = (X2 + I*Y2).    *
*                                                   *
*  MCONMULT ENTRY PROTOCOL:                         *
*     VARIABLES FOR INPUT:                          *
*        $IAD1 -> X1[0], $IAD2 -> Y1[0],            *
*        $SAD1 -> X2[0], $SAD2 -> Y2[0],            *
*        $N = N (LENGTH), $PARMS = DATA PAGE.       *
*     INPUT RESTRICTIONS: $N > 0.                   *
*     REGISTERS ALTERED: RC, DP, AR0-3 AND R0-3.    *
*                                                   *
*  RCONMULT ENTRY PROTOCOL:                         *
*     REGISTERS FOR INPUT:                          *
*        AR0 -> X1[0], AR1 -> Y1[0], AR2 -> X2[0],  *
*        AR3 -> Y2[0], RC = N (LENGTH).             *
*     INPUT RESTRICTIONS: RC > 0.                   *
*     REGISTERS ALTERED: RC, AR0-3 AND R0-3.        *
*                                                   *
*  REGISTERS USED AND RESTORED: SP.                 *
*  REGISTERS FOR OUTPUT: NONE.                      *
*  ROUTINES NEEDED: NONE.                           *
*****************************************************

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X1
        .GLOBL  $IAD2       ; ADDRESS OF INPUT Y1
        .GLOBL  $SAD1       ; ADDRESS OF INPUT X2
        .GLOBL  $SAD2       ; ADDRESS OF INPUT Y2

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MCONMULT    ; MEMORY ENTRY FOR COMPLEX (CONV.) MULTIPLY
        .GLOBL  RCONMULT    ; REGISTER ENTRY FOR COMPLEX (CONV.) MULTIPLY

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY
```

```
MCONVULT:

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,RC          ; RC <= N
        LDI     @$IAD1,AR0      ; AR0 -> X1[0]
        LDI     @$IAD2,AR1      ; AR1 -> Y1[0]
        LDI     @$SAD1,AR2      ; AR2 -> X2[0]
        LDI     @$SAD2,AR3      ; AR3 -> Y2[0]

;       REGISTER BASED PARAMETER ENTRY

RCONVULT:

;       COMPLEX MULTIPLY (CONVOLUTION) LOOP

        SUBI    1,RC            ; RC <= N - 1

        RPTB    LOOP2           ; REPEAT BLOCK N TIMES
        MPYF    *AR0,*AR2,R1    ; R1 <= X1[I]*X2[I]
        MPYF    *AR1,*AR3,R3    ; R3 <= Y1[I]*Y2[I]
        MPYF    *AR2++,*AR1,R0  ; R0 <= Y1[I]*X2[I], INCR. AR2 AND...
||      SUBF    R3,R1,R2        ; R2 <= X1[I]*X2[I] - Y1[I]*Y2[I]
        MPYF    *AR0,*AR3++,R1  ; R1 <= X1[I]*Y2[I], INCR. AR3
        ADDF    R1,R0,R3        ; R3 <= Y1[I]*X2[I] + X1[I]*Y2[I]
LOOP2:  STF     R2,*AR0++       ; X1[I] <= R2, INCR. AR0 AND...
||      STF     R3,*AR1++       ; Y1[I] <= R3, INCR. AR1

        RETS                    ; RETURN
```

```
*****************************************************
*   PROGRAM: *CBITREV                               *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   BIT REVERSE INDEX MAP TWO REAL ARRAYS AS A SINGLE *
*   COMPLEX ARRAY WITH THE SWAPPING DONE IN-PLACE.  *
*   X[I], Y[I] <-> X[J], Y[J], WHERE J = BR(I).     *
*   LENGTH OF ARRAYS N >= 4 IS ABSOLUTELY REQUIRED. *
*                                                   *
*   MCBITREV ENTRY PROTOCOL:                        *
*      VARIABLES FOR INPUT:                         *
*         $IAD1 -> X[0], $IAD2 -> Y[0],             *
*         $N = N (LENGTH), $PARMS = DATA PAGE.      *
*      INPUT RESTRICTIONS: $N >= 4.                 *
*      REGISTERS ALTERED: RC, DP, IR0, AR0-3 AND R0-3. *
*                                                   *
*   RCBITREV ENTRY PROTOCOL:                        *
*      REGISTERS FOR INPUT:                         *
*         AR0 -> X[0], AR1 -> Y[0], RC = N (LENGTH). *
*      INPUT RESTRICTIONS: RC >= 4.                 *
*      REGISTERS ALTERED: RC, IR0, AR0-3 AND R0-3.  *
*                                                   *
*   REGISTERS USED AND RESTORED: SP.                *
*   REGISTERS FOR OUTPUT: NONE.                     *
*   ROUTINES NEEDED: NONE.                          *
*****************************************************

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X
        .GLOBL  $IAD2       ; ADDRESS OF INPUT Y

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MCBITREV    ; MEMORY ENTRY FOR COMPLEX BIT REVERSE
        .GLOBL  RCBITREV    ; REGISTER ENTRY FOR COMPLEX BIT REVERSE

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MCBITREV:
```

```
        LDP     @@PARMS     ; LOAD DATA PAGE POINTER
        LDI     @@N,RC      ; RC <= N
        LDI     @@IAD1,AR0  ; AR0 -> ARRAY X
        LDI     @@IAD2,AR1  ; AR1 -> ARRAY Y

;       REGISTER BASED PARAMETER ENTRY

RCBITREV:

        LDI     RC,IR0      ; IR0 <= N
        SUBI    3,RC        ; RC <= N - 3
        LSH     -1,IR0      ; IR0 <= N/2 FOR BIT REVERSE
        LDI     AR0,AR2     ; AR2 -> ARRAY X (BIT REV.)
        NOP     *AR2++(IR0)B    ; INCR. BR(AR2) (OUTSIDE LOOP)
        NOP     *AR0++      ; INCR. AR0 (OUTSIDE LOOP)
        LDI     AR1,AR3     ; AR3 -> ARRAY Y (BIT REV.)

;       DO BIT REVERSE SWAP ON BOTH ARRAYS
;       SKIPPING THE 0TH AND N-1ST ELEMENTS

        RPTB    LOOP3       ; REPEAT LOOP N-2 TIMES
        CMPI    AR2,AR0     ; COMPARE AR2 TO AR0
        BGED    LOOP3       ; IF AR0 >= AR2, LOOP (DELAYED)
        NOP     *AR1++      ; INCR. AR1
        NOP     *AR3++(IR0)B    ; INCR. BR(AR3)
        LDF     *AR0++,R0   ; R0 <= X[I], INCR. AR0

        LDF     *AR2,R2     ; R2 <= X[J]
        LDF     *AR1,R1     ; R1 <= Y[I]
||      LDF     *AR3,R3     ; R3 <= Y[J]
        STF     R0,*AR2     ; X[J] <= R0
||      STF     R2,*-AR0    ; X[I] <= R2
        STF     R1,*AR3     ; Y[J] <= R1
||      STF     R3,*AR1     ; Y[I] <= R3
LOOP3:  NOP     *AR2++(IR0)B    ; INCR. BR(AR2)

        RETS                ; RETURN
```

```
*****************************************************
*   PROGRAM: *FMIEEE                                *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               MARCH 1989.                         *
*                                                   *
*   CONVERT AN ARRAY OF IEEE FLOATING-POINT NUMBERS TO  *
*   TMS320C30 FLOATING-POINT FORMAT.  ASSUMES NO: INF., *
*   NAN, OR DENORMALIZED NUMBERS.                   *
*                                                   *
*   MFMIEEE ENTRY PROTOCOL:                         *
*       VARIABLES FOR INPUT:                        *
*           $IAD1 -> X[0], $N = N (LENGTH),         *
*           $PARMS = DATA PAGE.                     *
*       INPUT RESTRICTIONS: $N > 0.                 *
*       REGISTERS ALTERED: RC, DP, AR0-1 AND R0-1.  *
*                                                   *
*   RFMIEEE ENTRY PROTOCOL:                         *
*       REGISTERS FOR INPUT:                        *
*           AR0 -> X[0], RC = N (LENGTH).           *
*       INPUT RESTRICTIONS: RC > 0.                 *
*       REGISTERS ALTERED: RC, AR0-1 AND R0-1.      *
*                                                   *
*   REGISTERS USED AND RESTORED: SP.                *
*   REGISTERS FOR OUTPUT: NONE.                     *
*   ROUTINES NEEDED: NONE.                          *
*****************************************************

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  *PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MFMIEEE     ; MEMORY ENTRY FOR IEEE -> 'C30 CONVERSION
        .GLOBL  RFMIEEE     ; REGISTER ENTRY FOR IEEE -> 'C30 CONVERSION

;       CONSTANTS FOR BOTH CONVERSIONS

        .DATA

CTAB    .WORD       0FF800000H
        .WORD       0FF000000H
        .WORD       07F000000H
        .WORD       080000000H
        .WORD       081000000H
```

```
TABA    .WORD       CTAB

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MFMIEEE:

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,RC          ; RC <= N
        LDI     @$IAD1,AR0      ; AR0 -> IEEE ARRAY

;       REGISTER BASED PARAMETER ENTRY

RFMIEEE:

        SUBI    1,RC            ; RC <= N - 1
        LDP     @CTAB           ; LOAD DATA PAGE POINTER
        LDI     @TABA,AR1       ; AR1 -> CONSTANT TABLE

;       IEEE -> 'C30 CONVERSION LOOP

        RPTB    LOOP4           ; REPEAT LOOP N TIMES
        AND     *AR0,*AR1,R0    ; REPLACE FRACTION WITH 0
        ADDI    *AR0,R0         ; SHIFT SIGN AND EXPONENT INSERTING 0
        LDIZ    *++AR1(1),R0    ; IF ALL ZERO, LOAD 'C30 0.0
        LDI     *AR0,R1         ; TEST ORIGINAL NUMBER
        BGED    LOOP4           ; IF >= 0, STORE NUMBER (DELAYED)
        SUBI    *++AR1(2),R0    ; REMOVE EXPONENT BIAS (127)
        PUSH    R0              ; SAVE AS AN INTEGER
        POPF    R0              ; UNSAVE AS A FLT. PT. NUMBER

        NEGF    R0              ; NEGATE 'C30 NUMBER

LOOP4:  STF     R0,*AR0++       ; STORE 'C30 NUMBER, INCR. AR0

        RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: *TOIEEE                                 *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              APRIL 1989.                          *
*                                                   *
*  CONVERT AN ARRAY OF TMS320C30 FLOATING-POINT     *
*  NUMBERS TO IEEE FLOATING-POINT FORMAT.  ZERO     *
*  IS THE ONLY SPECIAL CASE.                        *
*                                                   *
*  MTOIEEE ENTRY PROTOCOL:                          *
*      VARIABLES FOR INPUT:                         *
*          $IAD1 -> X[0], $N = N (LENGTH),          *
*          $PARMS = DATA PAGE.                      *
*      INPUT RESTRICTIONS: $N > 0.                  *
*      REGISTERS ALTERED: RC, DP, AR0-1 AND R0-1.   *
*                                                   *
*  RTOIEEE ENTRY PROTOCOL:                          *
*      REGISTERS FOR INPUT:                         *
*          AR0 -> X[0], RC = N (LENGTH).            *
*      INPUT RESTRICTIONS: RC > 0.                  *
*      REGISTERS ALTERED: RC, AR0-1 AND R0-1.       *
*                                                   *
*  REGISTERS USED AND RESTORED: SP.                 *
*  REGISTERS FOR OUTPUT: NONE.                      *
*  ROUTINES NEEDED: NONE.                           *
*                                                   *
*  NOTE: *TOIEEE SHARES THE CTAB TABLE FROM *FMIEEE *
*****************************************************

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MTOIEEE     ; MEMORY ENTRY FOR 'C30 -> IEEE CONVERSION
        .GLOBL  RTOIEEE     ; REGISTER ENTRY FOR 'C30 -> IEEE CONVERSION

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MTOIEEE:
```

```
        LDP     @@PARMS         ; LOAD DATA PAGE POINTER
        LDI     @@N,RC          ; RC <= N
        LDI     @@IAD1,AR0      ; AR0 -> 'C30 ARRAY

;       REGISTER BASED PARAMETER ENTRY

RTOIEEE:

        SUBI    1,RC            ; RC <= N - 1
        LDP     @CTAB           ; LOAD DATA PAGE POINTER
        LDI     @TABA,AR1       ; AR1 -> CONSTANT TABLE

;       'C30 -> IEEE CONVERSION LOOP

        RPTB    LOOP5           ; REPEAT LOOP N TIMES
        ABSF    *AR0,R0         ; TEST |NUMBER|
        LDFZ    *++AR1(4),R0    ; IF == 0, LOAD FAKE 0.0
        LSH     1,R0            ; SHIFT OFF SIGN BIT
        PUSHF   R0              ; SAVE AS A FLT. PT.
        LDF     *AR0,R1         ; TEST ORIGINAL NUMBER
        BGED    LOOP5           ; IF >= 0, STORE NUMBER (DELAYED)
        POP     R0              ; UNSAVE AS AN INTEGER
        ADDI    *++AR1(2?),R0   ; ADD EXPONENT BIAS (127)
        LSH     -1,R0           ; ADJUST FOR SIGN BIT

        OR      *++AR1(3),R0    ; NEGATE IEEE NUMBER

LOOP5:  STI     R0,*AR0++       ; STORE IEEE NUMBER, INCR. AR0

        RETS                    ; RETURN
```

```
*************************************************************
*  PROGRAM: *+VECMULT                                       *
*                                                           *
*  WRITTEN BY: GARY A. SITTON                               *
*              GAS LIGHT SOFTWARE                           *
*              HOUSTON, TEXAS                               *
*              FEBRUARY 1989.                               *
*                                                           *
*  SCALAR - VECTOR MULTIPLY: X[I] <= X[I]*C, C IS A         *
*  CONSTANT AND THE ARRAY X IS OF LENGTH N >= 1.            *
*                                                           *
*  MVECMULT ENTRY PROTOCOL:                                 *
*     VARIABLES FOR INPUT:                                  *
*        $IAD1 -> X[0], $N = N (LENGTH),                    *
*        $CNST = C, $PARMS = DATA PAGE.                     *
*     INPUT RESTRICTIONS: $N > 0.                           *
*     REGISTERS ALTERED: RC, DP, AR0 AND R0-1.              *
*                                                           *
*  RVECMULT ENTRY PROTOCOL:                                 *
*     REGISTERS FOR INPUT:                                  *
*        AR0 -> X[0], R0 = C, RC = N (LENGTH).              *
*     INPUT RESTRICTIONS: RC > 0.                           *
*     REGISTERS ALTERED: RC, AR0 AND R1.                    *
*                                                           *
*  REGISTERS USED AND RESTORED: SP.                         *
*  REGISTERS FOR OUTPUT: NONE.                              *
*  ROUTINES NEEDED: NONE.                                   *
*************************************************************

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $CNST       ; ADDRESS OF CONSTANT C
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MVECMULT    ; MEMORY ENTRY FOR SCALAR - VECTOR MULTIPLY
        .GLOBL  RVECMULT    ; REGISTER ENTRY FOR SCALAR - VECTOR MULTIPLY

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MVECMULT:

        LDP     @@PARMS         ; LOAD DATA PAGE POINTER
        LDI     @@N,RC          ; RC <= N
```

```
        LDI     @$IAD1,AR0      ; AR0 -> X[0]
        LDF     @$CNST,R0       ; R0 <= C

;       REGISTER BASED PARAMETER ENTRY

RVECMULT:

        SUBI    2,RC            ; RC <= N - 2
        MPYF    R0,*AR0,R1      ; R1 <= C*X[0]
        CMPI    0,RC            ; COMPARE RC TO 0
        BLT     SKIP1           ; IF RC < 0 THEN SKIP LOOP

;       SCALAR - VECTOR MULTIPLY LOOP

        RPTS    RC              ; REPEAT INST. N-1 TIMES
        MPYF    R0,*++AR0,R1    ; R1 <= C*X[I+1]
||      STF     R1,*AR0         ; X[I] <= C*X[I]

SKIP1:  STF     R1,*AR0         ; X[N-1] <= C*X[N-1]

        RETS                    ; RETURN
```

```
*****************************************************
*   PROGRAM: *CONMOV                                *
*                                                   *
*   WRITTEN BY: GARY A. SITTON                      *
*               GAS LIGHT SOFTWARE                  *
*               HOUSTON, TEXAS                      *
*               FEBRUARY 1989.                      *
*                                                   *
*   SCALAR -> VECTOR MOVE: X[I] <= C, C IS A        *
*   CONSTANT AND THE ARRAY X IS OF LENGTH N.        *
*                                                   *
*   MCONMOV ENTRY PROTOCOL:                         *
*       VARIABLES FOR INPUT:                        *
*           $IAD1 -> X[0], $N = N (LENGTH),         *
*           $CNST = C, $PARMS = DATA PAGE.          *
*       INPUT RESTRICTIONS: $N > 0.                 *
*       REGISTERS ALTERED: RC, DP, AR0, AND R0.     *
*                                                   *
*   RCONMOV ENTRY PROTOCOL:                         *
*       REGISTERS FOR INPUT:                        *
*           AR0 -> X[0], R0 = C, RC = N (LENGTH).   *
*       INPUT RESTRICTIONS: RC > 0.                 *
*       REGISTERS ALTERED: RC, AR0.                 *
*                                                   *
*   REGISTERS USED AND RESTORED: SP.                *
*   REGISTERS FOR OUTPUT: NONE.                     *
*   ROUTINES NEEDED: NONE.                          *
*****************************************************
```

```
;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $CNST       ; ADDRESS OF CONSTANT C
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MCONMOV     ; MEMORY ENTRY FOR CONSTANT TO VECTOR MOVE
        .GLOBL  RCONMOV     ; REGISTER ENTRY FOR CONSTANT TO VECTOR MOVE

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MCONMOV:

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,RC          ; RC <= N
```

```
        LDI     @$IAD1,AR0      ; AR0 -> X[0]
        LDF     @$CNST,R0       ; R0 <= C

;       REGISTER BASED PARAMETER ENTRY

RCONMOV:

        SUBI    1,RC            ; RC <= N - 1

;       SCALAR TO VECTOR MOVE LOOP

        RPTS    RC              ; REPEAT INST. N TIMES
        STF     R0,*AR0++       ; X[I] <= C

        RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: *VECMOV                                 *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              FEBRUARY 1989.                       *
*                                                   *
*  VECTOR MOVE: Y[I] <= X[I], I = 0,...,N-1 (N >= 1).*
*                                                   *
*  MVECMOV ENTRY PROTOCOL:                          *
*      VARIABLES FOR INPUT:                         *
*          $IAD1 -> X[0], $IAD2 -> Y[0],            *
*          $N = N (LENGTH), $PARMS = DATA PAGE.     *
*      INPUT RESTRICTIONS: $N > 0.                  *
*      REGISTERS ALTERED: RC, DP, AR0-1, AND R0.    *
*                                                   *
*  RVECMOV ENTRY PROTOCOL:                          *
*      REGISTERS FOR INPUT:                         *
*          AR0 -> X[0], AR1 -> Y[0], RC = N (LENGTH).*
*      INPUT RESTRICTIONS: RC > 0.                  *
*      REGISTERS ALTERED: RC, AR0-1, AND R0.        *
*                                                   *
*  REGISTERS USED AND RESTORED: SP.                 *
*  REGISTERS FOR OUTPUT: NONE.                      *
*  ROUTINES NEEDED: NONE.                           *
*****************************************************
```

```
;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $PARMS      ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N          ; ARRAY LENGTH N
        .GLOBL  $IAD1       ; ADDRESS OF INPUT X
        .GLOBL  $IAD2       ; ADDRESS OF INPUT Y

;       EXTERNAL PROGRAM NAMES

        .GLOBL  MVECMOV     ; MEMORY ENTRY FOR VECTOR TO VECTOR MOVE
        .GLOBL  RVECMOV     ; REGISTER ENTRY FOR VECTOR TO VECTOR MOVE

;       START OF PROGRAM AREA

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MVECMOV:

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,RC          ; RC <= N
        LDI     @$IAD1,AR0      ; AR0 -> X[0]
```

```
        LDI     @@IAD2,AR1      ; AR1 -> Y[0]

;       REGISTER BASED PARAMETER ENTRY

RVECMOV:

        SUBI    2,RC            ; RC <= N - 2
        LDF     *AR0++,R0       ; R0 <= X[0]
        CMPI    0,RC            ; COMPARE RC TO 0
        BLT     SKIP2           ; IF RC < 0 THEN SKIP LOOP

;       VECTOR MOVE LOOP

        RPTS    RC              ; REPEAT INST. N-1 TIMES
        LDF     *AR0++,R0       ; R0 <= X[I+1]
;;      STF     R0,*AR1++       ; MOVE X[I] TO Y[I]

SKIP2:  STF     R0,*AR1         ; MOVE X[N-1] TO Y[N-1]

        RETS                    ; RETURN

        .END
```

```
*****************************************************************
*
*   PROGRAM: $FFT2.ASM
*
*   RADIX 2 FFT ROUTINES
*
*   $FFT2.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*       CFFFT2 - COMPLEX DIF FORWARD RADIX 2 FFT USING SEPARATE REAL AND
*                IMAGINARY ARRAYS AND 3/4 CYCLE SINE TABLE.
*
*       CIFFT2 - COMPLEX DIT INVERSE RADIX 2 FFT USING SEPARATE REAL AND
*                IMAGINARY ARRAYS AND 3/4 CYCLE SINE TABLE (DOES NOT INCLUDE
*                THE 1/N SCALE FACTOR.
*
*****************************************************************
```

```
*************************************************
*                                               *
RAM: CFFFT2                                     *
*                                               *
TEN BY: GARY A. SITTON                          *
        GAS LIGHT SOFTWARE                      *
        HOUSTON, TEXAS                          *
        MARCH 1989.                             *
*                                               *
IAL VERSION USES 3/4 SINE TABLE LOOKUP WITH     *
PARAMETERS PASSED IN PREDEFINED MEMORY LOCATIONS.*
LEX RADIX-2 DIF FORWARD FFT FOR THE TMS320C30.  *
 PROGRAM ASSUMES NORMAL ORDERED DATA AS INPUT,  *
LEAVES THE OUTPUT INDEXED IN BIT REVERSED ORDER.*
POINTERS ARE USED FOR SEPARATE REAL AND IMAGINARY*
YS.                                             *
*                                               *
ABLES FOR INPUT:                                *
$IAD1 -> REAL[0], $IAD2 -> IMAG[0],             *
$N = N (LENGTH), $M = M (LOG2(N)),              *
$SINE -> SINE TABLE, $PARMS = DATA PAGE.        *
T RESTRICTIONS: $N > 1.                         *
STERS ALTERED: RC, DP, IR0-1, AR0-7, AND R0-7.  *
STERS USED AND RESTORED: SP.                    *
STERS FOR OUTPUT: NONE.                         *
INES NEEDED: NONE.                              *
*************************************************

EXTERNAL PROGRAM NAMES

.GLOBL  CFFFT2          ; ENTRY POINT FOR EXECUTION

EXTERNAL MEMORY ADDRESSES

.GLOBL  $SINE           ; SINE TABLE ADDRESS
.GLOBL  $PARMS          ; PARAMETER PAGE ADDRESS

EXTERNAL VARIABLE ADDRESSES

.GLOBL  $N              ; FFT LENGTH, N = 2**M
.GLOBL  $M              ; M = LOG2(N) >= 2
.GLOBL  $IAD1           ; REAL INPUT ARRAY ADDRESS
.GLOBL  $IAD2           ; IMAGINARY INPUT ARRAY ADDRESS

.TEXT

START OF DIF FFFT PROGRAM



INITIALIZE LOOP VARIABLES

LDP     @$PARMS         ; LOAD DATA PAGE POINTER
LDI     @$N,IR0         ; IR0 <= N1 (INIT. N)
        LDI     IR0,IR1         ; IR1 <= N
        LSH     -2,IR1          ; IR1 <= N/4, OFFSET FOR COSINE
        LDI     0,AR6           ; AR6 <= K (INIT. 0)
        LDI     IR0,R7          ; R7 <= N1
        LSH     -1,R7           ; R7 <= N2 (INIT. N/2)
        LDI     1,R5            ; R5 <= IE (INIT. 1)

;       OUTER LOOP

FLOOP:  ADDI    1,AR6           ; K <= K + 1
        LDI     @$IAD1,AR0      ; AR0 -> X(0)
        ADDI    R7,AR0,AR1      ; AR1 -> X(L)
        LDI     @$IAD2,AR2      ; AR2 -> Y(0)
        ADDI    R7,AR2,AR3      ; AR3 -> Y(L)
        LDI     R5,RC           ; SETUP 1ST INNER LOOP REPEAT COUNTER.
        SUBI    1,RC            ; RC (ONE LESS THAN THE DESIRED #)

;       FIRST INNER LOOP (UNITY TWIDDLE FACTOR)

        RPTB    FBLK1           ; REPEAT BLOCK IE TIMES
        ADDF    *AR0,*AR1,R0    ; R0 <= X(I) + X(L)
        SUBF    *AR1,*AR0,R1    ; R1 <= X(I) - X(L)
        ADDF    *AR2,*AR3,R2    ; R2 <= Y(I) + Y(L) AND...
        SUBF    *AR3,*AR2,R3    ; R3 <= Y(I) - Y(L)
        STF     R0,*AR0++(IR0)  ; X(I) <= R0, INCR. AR0 AND...
;;      STF     R1,*AR1++(IR0)  ; X(L) <= R1, INCR. AR1
FBLK1:  STF     R2,*AR2++(IR0)  ; Y(I) <= R2, INCR. AR2 AND...
;;      STF     R3,*AR3++(IR0)  ; Y(L) <= R3, INCR. AR3

;       PROGRAM EXIT TEST

        CMPI    @$M,AR6         ; COMPARE M TO K
        RETSGE                  ; IF K >= M THEN RETURN

;       MAIN INNER LOOP

        LDI     2,AR7           ; J <= 2, (PRE-INCREMENTED)
        LDI     1,AR0           ; AR0 <= I (INIT. 1)
        LDI     1,AR2           ; AR2 <= I (INIT. 1)
        LDI     @$SINE,AR5      ; AR5 <= SINTAB[IA] (INIT. IA = 0)

FINLOP: ADDI    R5,AR5          ; AR5 -> SINTAB[IA <= IA + IE]
        LDF     *AR5,R6         ; R6 <= SIN(X), (X = (2*PI/N)*IA)
        ADDI    AR5,IR1,AR4     ; AR4 -> COS(X)
        ADDI    @$IAD1,AR0      ; AR0 -> X(I)
        ADDI    @$IAD2,AR2      ; AR2 -> Y(I)
        ADDI    R7,AR0,AR1      ; AR1 -> X(L)
        ADDI    R7,AR2,AR3      ; AR3 -> Y(L)
        LDI     R5,RC           ; SETUP 2ND INNER LOOP REPEAT COUNTER.
        SUBI    1,RC            ; RC (ONE LESS THAN THE DESIRED #)

;       SECOND INNER LOOP (DOES TWIDDLE ROTATION)

        RPTB    FBLK2           ; REPEAT BLOCK IE TIMES
```

```
        SUBF    *AR1,*AR0,R2    ; R2 <= XT = X(I) - X(L)
        SUBF    *AR3,*AR2,R1    ; R1 <= YT = Y(I) - Y(L)
        MPYF    R2,R6,R0        ; R0 <= XT*SIN AND...
||      ADDF    *AR2,*AR3,R3    ; R3 <= Y(I) + Y(L)
        MPYF    R1,*AR4,R3      ; R3 <= YT*COS AND...
||      STF     R3,*AR2++(IR0)  ; Y(I) <= Y(I) + Y(L), INCR. AR2
        SUBF    R0,R3,R4        ; R4 <= COS*YT - SIN*XT
        MPYF    R1,R6,R0        ; R0 <= SIN*YT AND...
||      ADDF    *AR0,*AR1,R3    ; R3 <= X(I) + X(L)
        MPYF    R2,*AR4,R3      ; R3 <= COS*XT AND...
||      STF     R3,*AR0++(IR0)  ; X(I) <= X(I) + X(L) , INCR. AR0
        ADDF    R0,R3           ; R3 <= COS*XT + SIN*YT
FBLK2:  STF     R3,*AR1++(IR0)  ; X(L) <= COS*XT + SIN*YT, INCR. AR1 AND...
||      STF     R4,*AR3++(IR0)  ; Y(L) <= COS*YT - SIN*XT, INCR. AR3

        CMPI    R7,AR7          ; COMPARE N2 TO J

        BLTD    FINLOP          ; IF J < N2 THEN LOOP (DELAYED)
        LDI     AR7,AR0         ; AR0 <= J
        LDI     AR7,AR2         ; AR2 <= J
        ADDI    1,AR7           ; J <= J + 1

        BRD     FLOOP           ; NEXT FFT STAGE (DELAYED)
        LSH     1,R5            ; IE <= 2*IE
        LDI     R7,IR0          ; N1 <= N2
        LSH     -1,R7           ; N2 <= N2/2

;       END OF OUTER LOOP
```

```
*****************************************************************
*                                                               *
*   PROGRAM: CIFFT2                                             *
*                                                               *
*   WRITTEN BY: GARY A. SITTON                                  *
*               GAS LIGHT SOFTWARE                              *
*               HOUSTON, TEXAS                                  *
*               MARCH 1989.                                     *
*                                                               *
*   SPECIAL VERSION USES 3/4 SINE TABLE LOOKUP WITH             *
*   THE PARAMETERS PASSED IN PREDEFINED MEMORY LOCATIONS.       *
*   COMPLEX RADIX-2 DIT INVERSE FFT FOR THE TMS320C30.          *
*   THIS PROGRAM ASSUMES BIT REVERSED ORDERED DATA AS           *
*   INPUT, BUT LEAVES THE OUTPUT INDEXED IN NORMAL ORDER.       *
*   TWO POINTERS ARE USED FOR SEPARATE REAL AND IMAGINARY       *
*   ARRAYS.                                                     *
*                                                               *
*   VARIABLES FOR INPUT:                                        *
*      $IAD1 -> REAL[0], $IAD2 -> IMAG[0],                      *
*      $N = N (LENGTH), $M = M (LOG2(N)),                       *
*      $SINE -> SINE TABLE, $PARMS = DATA PAGE.                 *
*   INPUT RESTRICTIONS: $N > 1.                                 *
*   REGISTERS ALTERED: RC, DP, IR0-1, AR0-7, AND R0-7.          *
*   REGISTERS USED AND RESTORED: SP.                            *
*   REGISTERS FOR OUTPUT: NONE.                                 *
*   ROUTINES NEEDED: NONE.                                      *
*****************************************************************

;       EXTERNAL PROGRAM NAMES

        .GLOBL  CIFFT2          ; ENTRY POINT FOR EXECUTION

;       EXTERNAL MEMORY ADDRESSES

        .GLOBL  $SINE           ; SINE TABLE ADDRESS
        .GLOBL  $PARMS          ; PARAMETER PAGE ADDRESS

;       EXTERNAL VARIABLE ADDRESSES

        .GLOBL  $N              ; FFT LENGTH, N = 2**M
        .GLOBL  $M              ; M = LOG2(N) >= 2
        .GLOBL  $IAD1           ; REAL INPUT ARRAY ADDRESS
        .GLOBL  $IAD2           ; IMAGINARY INPUT ARRAY ADDRESS

;       START OF DIT IFFT PROGRAM

        .TEXT

CIFFT2:

;       INITIALIZE LOOP VARIABLES

        LDP     @$PARMS         ; LOAD DATA PAGE POINTER
        LDI     @$N,IR0         ; IR0 <= N
```

```
        LDI     IR0,IR1         ; IR1 <= N
        LSH     -2,IR1          ; IR1 <= N/4, OFFSET FOR COSINE
        LDI     @M,AR6          ; AR6 <= K (INIT. M)
        LDI     1,R7            ; R7 <= N2 (INIT. 1)
        LDI     IR0,R5          ; R5 <= N
        LSH     -1,R5           ; R5 <= IE (INIT. N/2)
        LDI     2,IR0           ; IR0 <= N1 (INIT. 2)

;       OUTER LOOP

ILOOP:  LDI     @IAD1,AR0       ; AR0 -> X(0)
        ADDI    R7,AR0,AR1      ; AR1 -> X(L)
        LDI     @IAD2,AR2       ; AR2 -> Y(0)
        ADDI    R7,AR2,AR3      ; AR3 -> Y(L)
        LDI     R5,RC           ; SETUP 1ST INNER LOOP REPEAT COUNTER.
        SUBI    1,RC            ; RC (ONE LESS THAN THE DESIRED #)

;       FIRST INNER LOOP (UNITY TWIDDLE FACTOR)

        RPTB    IBLK1           ; REPEAT BLOCK IE TIMES
        ADDF    *AR0,*AR1,R0    ; R0 <= X(I) + X(L)
        SUBF    *AR1,*AR0,R1    ; R1 <= X(I) - X(L)
        ADDF    *AR2,*AR3,R2    ; R2 <= Y(I) + Y(L) AND...
        SUBF    *AR3,*AR2,R3    ; R3 <= Y(I) - Y(L)
        STF     R0,*AR0++(IR0)  ; X(I) <= R0, INCR. AR0 AND...
||      STF     R1,*AR1++(IR0)  ; X(L) <= R1, INCR. AR1
IBLK1:  STF     R2,*AR2++(IR0)  ; Y(I) <= R2, INCR. AR2 AND...
||      STF     R3,*AR3++(IR0)  ; Y(L) <= R3, INCR. AR3

        CMPI    @M,AR6          ; COMPARE M TO K
        BEQD    SKIP            ; IF K == M THEN SKIP TWIDDLED LOOP

;       MAIN INNER LOOP

        LDI     2,AR7           ; J <= 2, (PRE-INCREMENTED)
        LDI     1,AR0           ; AR0 <= I (INIT. 1)
        LDI     1,AR2           ; AR2 <= I (INIT. 1)
        LDI     @SINE,AR5       ; AR5 <= IA (INIT. 0)

IINLOP: ADDI    R5,AR5          ; AR5 -> SINTABLIA <= IA + IE]
        LDF     *AR5,R6         ; R6 <= SIN(X), (X = (2*PI/N)*IA)
        ADDI    AR5,IR1,AR4     ; AR4 -> COS(X)
        ADDI    @IAD1,AR0       ; AR0 -> X(I)
        ADDI    @IAD2,AR2       ; AR2 -> Y(I)
        ADDI    R7,AR0,AR1      ; AR1 -> X(L)
        ADDI    R7,AR2,AR3      ; AR3 -> Y(L)
        LDI     R5,RC           ; SETUP 2ND INNER LOOP REPEAT COUNTER.
        SUBI    1,RC            ; RC (ONE LESS THAN THE DESIRED #)

;       SECOND INNER LOOP (DOES TWIDDLE ROTATION)

        RPTB    IBLK2           ; REPEAT BLOCK IE TIMES
        MPYF    *AR4,*AR1,R4    ; R4 <= COS*X(L)
        MPYF    R6,*AR3,R3      ; R3 <= SIN*Y(L)
        MPYF    *AR4,*AR3,R0    ; R0 <= COS*Y(L), AND...
||      SUBF    R3,R4,R2        ; R2 <= XT = COS*X(L) - SIN*Y(L)
        MPYF    R6,*AR1,R1      ; R1 <= SIN*X(L), AND...
||      SUBF    R2,*AR0,R3      ; R3 <= X(I) - XT
        ADDF    R0,R1,R4        ; R4 <= YT = COS*Y(L) + SIN*X(L)
        SUBF    R4,*AR2,R3      ; R3 <= Y(I) - YT, AND...
||      STF     R3,*AR1++(IR0)  ; X(L) <= X(I) - XT, INCR. AR1
        ADDF    R2,*AR0,R3      ; R3 <= X(I) + XT, AND...
||      STF     R3,*AR3++(IR0)  ; Y(L) <= Y(I) - YT, INCR. AR3
        ADDF    R4,*AR2,R4      ; R4 <= Y(I) + YT
IBLK2:  STF     R3,*AR0++(IR0)  ; X(I) <= X(I) + XT, INCR. AR0 AND...
||      STF     R4,*AR2++(IR0)  ; Y(I) <= Y(I) + YT, INCR. AR2

        CMPI    R7,AR7          ; COMPARE N2 TO J

        BLTD    IINLOP          ; IF J < N2 THEN LOOP (DELAYED)
        LDI     AR7,AR0         ; AR0 <= J
        LDI     AR7,AR2         ; AR2 <= J
        ADDI    1,AR7           ; J <= J + 1

SKIP:   SUBI    1,AR6           ; K <= K - 1
        CMPI    0,AR6           ; COMPARE 0 TO K
        BGTD    ILOOP           ; IF K > 0 THEN LOOP (DELAYED)
        LSH     -1,R5           ; IE <= IE/2
        LDI     IR0,R7          ; N2 <= N1
        LSH     1,IR0           ; N1 <= 2*N1

;       PROGRAM EXIT POINT

        RETS                    ; RETURN

        .END
```

```
*************************************************************************
*
*    PROGRAM: $LINALG.ASM
*
*    LINEAR ALGEBRA ROUTINES
*
*    $LINALG.ASM CONSISTS OF THE FOLLOWING ROUTINES:
*
*         #SOLUTN  - SOLVES A WELL CONDITIONED SYSTEM OF LINEAR EQUATIONS WITH
*                    ANY NUMBER OF DEPENDENT VARIABLE SETS. USES NO (DIAGONAL)
*                    PIVOTING WITH NORMAL-PRECISION FLOATING-POINT MATH.
*
*         #SOLUTNX - SOLVES A WELL CONDITIONED SYSTEM OF LINEAR EQUATIONS WITH
*                    ANY NUMBER OF DEPENDENT VARIABLE SETS. USES NO (DIAGONAL)
*                    PIVOTING WITH EXTENDED-PRECISION FLOATING-POINT MATH.
*
*************************************************************************
```

```
*************************************************************
*  PROGRAM: #SOLUTN                                         *
*                                                            *
*  WRITTEN BY: GARY A. SITTON                                *
*              GAS LIGHT SOFTWARE                            *
*              HOUSTON, TEXAS                                *
*              MAY 1989.                                     *
*                                                            *
*  (NORMAL PRECISION VERSION)                                *
*                                                            *
*  SOLVES A SYSTEM OF LINEAR EQUATIONS A*X = Y IN THE        *
*  TABLEAU FORMAT B = A:-Y, AN M X N MATRIX.  THIS           *
*  MEANS THAT A IS AN M X M SQUARE MATRIX OF COEFFI-         *
*  CIENTS, AND -Y IS AN M X N-M RECTANGULAR MATRIX           *
*  OF N-M VECTORS EACH HAVING M ELEMENTS.  EACH DEPEN-       *
*  DENT VARIABLE COLUMN VECTOR IS NEGATED AND APPENDED       *
*  TO THE COEFFICIENT MATRIX A.  THE SET OF N-M INDE-        *
*  PENDENT SOLUTION VECTORS X WILL APPEAR IN PLACE OF        *
*  THE ORIGINAL APPENDED COLUMNS WHEN SOLUTN FINISHES.       *
*  ROW MAJOR MATRIX STORAGE FORMAT IS ASSUMED PLUS           *
*  THE PROGRAM ASSUMES N > M > 1 AND B[0, 0] != 0.0          *
*  SINCE THE METHOD USES DIAGONAL PIVOTING AND STARTS        *
*  WITH B[0, 0].  ANY PIVOT ELEMENT < 10**-8 IN ITS          *
*  ABSOLUTE VALUE WILL IMPLY AN "ILL CONDITIONED"            *
*  SYSTEM OF EQUATIONS, I. E. NOT HAVING SUFFICIENT          *
*  LINEAR INDEPENDENCE,  AND WILL RESULT IN AN INCOM-        *
*  PLETE SOLUTION.  AN INCOMPLETE SOLUTION WILL BE           *
*  INDICATED BY THE VALUE OF R3 == 0.0 ON EXIT, ELSE         *
*  R3 != 0.0 AND EQUALS THE LAST PIVOT ELEMENT VALUE.        *
*                                                            *
*  MSOLUTN ENTRY PROTOCOL:                                   *
*      VARIABLES FOR INPUT:                                  *
*          $IAD1 -> B[0, 0], $NROW = M,                      *
*          $NCOL = N, $PARMS = DATA PAGE.                    *
*      INPUT RESTRICTIONS: N > M > 1.                        *
*      REGISTERS ALTERED: RC, DP, AR0-7, IR0-1,              *
*                         AND R0-7.                          *
*                                                            *
*  RSOLUTN ENTRY PROTOCOL:                                   *
*      REGISTERS FOR INPUT:                                  *
*          AR0 -> B[0, 0], AR1 = M, AR2 = N.                 *
*      INPUT RESTRICTIONS: AR2 > AR1 > 1.                    *
*      REGISTERS ALTERED: RC, AR0-7, IR0-1, AND R0-7.        *
*                                                            *
*  REGISTERS USED AND RESTORED: SP.                          *
*  REGISTERS FOR OUTPUT: R3.                                 *
*  ROUTINES NEEDED: FPINV (SEE $MATH).                       *
*                                                            *
*  NOTE: COMMENTED OUT RND INSTRUCTIONS MAY BE ACTI-         *
*  VATED FOR ADDITIONAL ACCURACY WITH LOSS OF SPEED.         *
*************************************************************

:       EXTERNAL PROGRAM NAMES
```

```
        .GLOBL  MSOLUTN     ; MEMORY BASED ENTRY
        .GLOBL  RSOLUTN     ; REGISTER BASED ENTRY
        .GLOBL  FPINV       ; RECIPROCAL ROUTINE

;       EXTERNAL PARAMETER NAMES

        .GLOBL  $PARMS      ; PARAMETER SPACE ADDRESS
        .GLOBL  $IAD1       ; POINTER TO MATRIX B, ADDRESS OF B[0, 0]
        .GLOBL  $NROW       ; NUMBER OF ROWS IN B, VALUE OF M
        .GLOBL  $NCOL       ; NUMBER OF COLUMNS IN B, VALUE OF N

;       INTERNAL CONSTANTS

        .DATA

EPSN    .FLOAT  1.0E-8      ; SINGULARITY CRITERION
ZERO    .SET    0.0         ; SINGULARITY FLAG

;       START SOLUTN PROGRAM

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MSOLUTN:

        LDP     @$PARMS     ; LOAD DATA PAGE POINTER
        LDI     @$IAD1,AR0  ; AR0 -> B[0, 0]
        LDI     @$NROW,AR1  ; AR1 <= M
        LDI     @$NCOL,AR2  ; AR2 <= N

;       REGISTER BASED PARAMETER ENTRY

RSOLUTN:

;       SETUP LOOP REGISTERS

        LDP     @EPSN       ; LOAD DATA PAGE POINTER
        LDI     0,IR0       ; IR0 <= K (INIT. 0)
        LDI     AR0,AR3     ; AR3 -> B[0, 0]
        SUBI    1,AR1       ; AR1 <= M-1
        LDI     AR2,AR6     ; AR6 <= N
        SUBI    2,AR6       ; AR6 <= N-2

;       MAIN LOOP (K INDEX)

KLOOP:  LDF     *++AR3(IR0),R3  ; R3 <= B[K, K], NEXT PIVOT
        ABSF    R3,R0       ; R0 <= |R3|
        CMPF    @EPSN,R0    ; COMPARE |B[K, K]| TO EPS
        BLT     SING        ; IF |B[K, K]| < EPS THEN STOP

;       COMPUTE RECIPROCAL OF -PIVOT ELEMENT

        NEGF    R3,R0       ; R0 <= -B[K, K]
        CALL    FPINV       ; R0 <= -1/B[K, K]
        RND     R0          ; ROUND INVERSE

;       DIVIDE RIGHT PART OF PIVOT ROW BY -PIVOT ELEMENT

        ADDI    AR3,IR0,AR7 ; AR7 -> B[K, K]
        LDI     AR6,RC      ; RC <= N-K-2

        RPTB    DLOOP       ; REPEAT DIVIDE LOOP N-K-1 TIMES
        MPYF    R0,*++AR7,R2    ; R2 <= B[K, J]*(-1/B[K, K])
*       RND     R2          ; REMOVE "*" TO ROUND *
DLOOP:  STF     R2,*AR7     ; B[K, J] <= R2

;       START INNER LOOP (I INDEX)

        LDI     0,IR1       ; IR1 <= I (INIT. 0)
        LDI     AR0,AR4     ; AR4 -> B[0, 0]

        CMPI    IR0,IR1     ; COMPARE I TO K
ILOOP:  BEQ     SKIP        ; IF I == K THEN SKIP PIVOT ROW

;       COMPLETE PIVOTING OPERATION

        ADDI    AR4,IR0,AR5 ; AR5 -> B[I, K]
        LDF     *AR5,R0     ; R0 <= B[I, K]
        LDI     AR6,RC      ; RC <= N-K-2
        CMPI    1,RC        ; COMPARE RC TO 1
        BLTD    JUMP        ; IF RC < 1 THEN NO RPTB (DELAYED)

        SUBI    1,RC        ; RC <= N-K-3
        ADDI    AR3,IR0,AR7 ; AR7 -> B[K, J]
        MPYF    R0,*++AR7,R1    ; R1 <= B[K, K+1]*B[I, K]

;       START INNER-INNER LOOP (J INDEX)

        RPTB    JLOOP       ; REPEAT PIVOT LOOP N-K-2 TIMES
        MPYF    R0,*++AR7,R1    ; R1 <= B[K, J]*B[I, K]
;|     ADDF    R1,*++AR5,R2    ; R2 <= B[I, J] + R1
*       RND     R2          ; REMOVE "*" TO ROUND +
JLOOP:  STF     R2,*AR5     ; B[I, J] <= R2

;       END OF INNER-INNER LOOP (J INDEX)

JUMP:   ADDF    R1,*++AR5,R2    ; R2 <= B[I, N-1] + R1
*       RND     R2          ; REMOVE "*" TO ROUND +
        STF     R2,*AR5     ; B[I, N-1] <= R2

SKIP:   CMPI    AR1,IR1     ; COMPARE I TO M-1
        BLTD    ILOOP       ; IF I < M-1 THEN LOOP (DELAYED)

        ADDI    AR2,AR4     ; AR4 -> B[I+1, 0]
        ADDI    1,IR1       ; I <= I+1
        CMPI    IR0,IR1     ; COMPARE I TO K
```

```
;       END OF INNER LOOP (I INDEX)

        CMPI    AR1,IR0         ; COMPARE K TO M-1
        BLTD    KLOOP           ; IF K < M-1 THEN LOOP

        ADDI    AR2,AR3         ; AR3 -> B[K+1, 0]
        ADDI    1,IR0           ; K <= K+1
        SUBI    1,AR6           ; AR6 <= N-K-1

;       END OF OUTER LOOP (K INDEX)

        RETS                    ; RETURN

;       SINGULAR SYSTEM EXIT

SING:   LDF     ZERO,R3         ; SET "SINGULAR" FLAG

        RETS                    ; RETURN
```

```
*****************************************************
*  PROGRAM: *SOLUTNX                                *
*                                                   *
*  WRITTEN BY: GARY A. SITTON                       *
*              GAS LIGHT SOFTWARE                   *
*              HOUSTON, TEXAS                       *
*              MAY 1989.                            *
*                                                   *
*  (EXTENDED PRECISION VERSION)                     *
*                                                   *
*  SOLVES A SYSTEM OF LINEAR EQUATIONS A*X = Y IN THE *
*  TABLEAU FORMAT B = A|-Y, AN M X N MATRIX.  THIS   *
*  MEANS THAT A IS AN M X M SQUARE MATRIX OF COEFFI- *
*  CIENTS, AND -Y IS AN M X N-M RECTANGULAR MATRIX   *
*  OF N-M VECTORS EACH HAVING M ELEMENTS.  EACH DEPEN- *
*  DENT VARIABLE COLUMN VECTOR IS NEGATED AND APPENDED *
*  TO THE COEFFICIENT MATRIX A.  THE SET OF N-M INDE- *
*  PENDENT SOLUTION VECTORS X WILL APPEAR IN PLACE OF *
*  THE ORIGINAL APPENDED COLUMNS WHEN SOLUTNX FINISHES.*
*  ROW MAJOR MATRIX STORAGE FORMAT IS ASSUMED PLUS   *
*  THE PROGRAM ASSUMES N > M > 1 AND B[0, 0] != 0.0  *
*  SINCE THE METHOD USES DIAGONAL PIVOTING AND STARTS *
*  WITH B[0, 0].  ANY PIVOT ELEMENT < 10**-10 IN ITS *
*  ABSOLUTE VALUE WILL IMPLY AN "ILL CONDITIONED"    *
*  SYSTEM OF EQUATIONS, I. E. NOT HAVING SUFFICIENT  *
*  LINEAR INDEPENDENCE,  AND WILL RESULT IN AN INCOM- *
*  PLETE SOLUTION.  AN INCOMPLETE SOLUTION WILL BE   *
*  INDICATED BY THE VALUE OF R3 == 0.0 ON EXIT, ELSE *
*  R3 != 0.0 AND EQUALS THE LAST PIVOT ELEMENT VALUE. *
*                                                   *
*  MSOLUTNX ENTRY PROTOCOL:                         *
*     VARIABLES FOR INPUT:                          *
*        $IADI -> B[0, 0], $NROW = M,               *
*        $NCOL = N, $PARMS = DATA PAGE.             *
*     INPUT RESTRICTIONS: N > M > 1.                *
*     REGISTERS ALTERED: RC, DP, AR0-7, IR0-1,      *
*                        AND R0-7.                  *
*                                                   *
*  RSOLUTNX ENTRY PROTOCOL:                         *
*     REGISTERS FOR INPUT:                          *
*        AR0 -> B[0, 0], AR1 = M, AR2 = N.          *
*     INPUT RESTRICTIONS: AR2 > AR1 > 1.            *
*     REGISTERS ALTERED: RC, AR0-7, IR0-1, AND R0-7. *
*                                                   *
*  REGISTERS USED AND RESTORED: SP.                 *
*  REGISTERS FOR OUTPUT: R3.                        *
*  ROUTINES NEEDED: FPINVX AND FMULTX (SEE $MATHI).  *
*                                                   *
*  NOTE: THE RND INSTRUCTIONS MAY BE REMOVED WITH   *
*  SOME LOSS OF ACCURACY BUT INCREASE IN SPEED.     *
*****************************************************

;       EXTERNAL PROGRAM NAMES
```

```
        .GLOBL  MSOLUTNX    ; MEMORY BASED ENTRY
        .GLOBL  RSOLUTNX    ; REGISTER BASED ENTRY
        .GLOBL  FPINVX      ; RECIPROCAL ROUTINE
        .GLOBL  FMULTX      ; MULTIPLY ROUTINE

;       EXTERNAL PARAMETER NAMES

        .GLOBL  $PARMS      ; PARAMETER SPACE ADDRESS
        .GLOBL  $IAD1       ; POINTER TO MATRIX B, ADDRESS OF B[0, 0]
        .GLOBL  $NROW       ; NUMBER OF ROWS IN B, VALUE OF M
        .GLOBL  $NCOL       ; NUMBER OF COLUMNS IN B, VALUE OF N

;       INTERNAL CONSTANTS

        .DATA

EPSNX   .FLOAT  1.0E-10     ; SINGULARITY CRITERION
ZEROX   .SET    0.0         ; SINGULARITY FLAG

;       START SOLUTNX PROGRAM

        .TEXT

;       MEMORY BASED PARAMETER ENTRY

MSOLUTNX:

        LDP     @$PARMS     ; LOAD DATA PAGE POINTER
        LDI     @$IAD1,AR0  ; AR0 -> B[0, 0]
        LDI     @$NROW,AR1  ; AR1 <= M
        LDI     @$NCOL,AR2  ; AR2 <= N

;       REGISTER BASED PARAMETER ENTRY

RSOLUTNX:

;       SETUP LOOP REGISTERS

        LDP     @EPSNX      ; LOAD DATA PAGE POINTER
        LDI     0,IR0       ; IR0 <= K (INIT. 0)
        LDI     AR0,AR3     ; AR3 -> B[0, 0]
        SUBI    1,AR1       ; AR1 <= M-1
        LDI     AR2,AR6     ; AR6 <= N
        SUBI    2,AR6       ; AR6 <= N-2

;       MAIN LOOP (K INDEX)

KLOOPX: LDF     *++AR3(IR0),R3  ; R3 <= B[K, K], NEXT PIVOT
        ABSF    R3,R0       ; R0 <= |R3|
        CMPF    @EPSNX,R0   ; COMPARE |B[K, K]| TO EPS
        BLT     SINGX       ; IF |B[K, K]| < EPS THEN STOP

;       COMPUTE RECIPROCAL OF -PIVOT ELEMENT

        NEGF    R3,R0       ; R0 <= -B[K, K]
        CALL    FPINVX      ; R0 <= -1/B[K, K]
        LDF     R0,R1       ; R1 <= -1/B[K, K]

;       DIVIDE RIGHT PART OF PIVOT ROW BY -PIVOT ELEMENT

        ADDI    AR3,IR0,AR7 ; AR7 -> B[K, K]
        LDI     AR6,RC      ; RC <= N-K-2

        RPTB    DLOOPX      ; REPEAT DIVIDE LOOP N-K-1 TIMES
        LDF     *++AR7,R0   ; R0 <= B[K, J]
        CALL    FMULTX      ; R0 <= B[K, J]*(-1/B[K, K])
        RND     R0          ; ROUND *
DLOOPX: STF     R0,*AR7     ; B[K, J] <= R0

;       START INNER LOOP (I INDEX)

        LDI     0,IR1       ; IR1 <= I (INIT. 0)
        LDI     AR0,AR4     ; AR4 -> B[0, 0]

        CMPI    IR0,IR1     ; COMPARE I TO K
ILOOPX: BEQ     SKIPX       ; IF I == K THEN SKIP PIVOT ROW

;       COMPLETE PIVOTING OPERATION

        ADDI    AR4,IR0,AR5 ; AR5 -> B[I, K]
        LDF     *AR5,R0     ; R0 <= B[I, K]
        LDI     AR6,RC      ; RC <= N-K-2
        CMPI    1,RC        ; COMPARE RC TO 1
        BLTD    JUMPX       ; IF RC < 1 THEN NO RPTB (DELAYED)

        SUBI    1,RC        ; RC <= N-K-3
        ADDI    AR3,IR0,AR7 ; AR7 -> B[K, J]
        MPYF    R0,*++AR7,R1 ; R1 <= B[K, K+1]*B[I, K]

;       START INNER-INNER LOOP (J INDEX)

        RPTB    JLOOPX      ; REPEAT PIVOT LOOP N-K-2 TIMES
        MPYF    R0,*++AR7,R1 ; R1 <= B[K, J]*B[I, K]
;;      ADDF    R1,*++AR5,R2 ; R2 <= B[I, J] + R1
        RND     R2          ; ROUND +
JLOOPX: STF     R2,*AR5     ; B[I, J] <= R2

;       END OF INNER-INNER LOOP (J INDEX)

JUMPX:  ADDF    R1,*++AR5,R2 ; R2 <= B[I, N-1] + R1
        RND     R2          ; ROUND +
        STF     R2,*AR5     ; B[I, N-1] <= R2

SKIPX:  CMPI    AR1,IR1     ; COMPARE I TO M-1
        BLTD    ILOOPX      ; IF I < M-1 THEN LOOP (DELAYED)

        ADDI    AR2,AR4     ; AR4 -> B[I+1, 0]
        ADDI    1,IR1       ; I <= I+1
```

```
        CMPI    IR0,IR1         ; COMPARE I TO K

;       END OF INNER LOOP (I INDEX)

        CMPI    AR1,IR0         ; COMPARE K TO M-1
        BLTD    KLOOPX          ; IF K < M-1 THEN LOOP

        ADDI    AR2,AR3         ; AR3 -> B[K+1, 0]
        ADDI    1,IR0           ; K <= K+1
        SUBI    1,AR6           ; AR6 <= N-K-1

;       END OF OUTER LOOP (K INDEX)

        RETS                    ; RETURN

;       SINGULAR SYSTEM EXIT

SINGX:  LDF     ZEROX,R3        ; SET "SINGULAR" FLAG

        RETS                    ; RETURN

        .END
```