TABLE OF CONTENTS

FLOATING POINT PACKAGE

I. DESCRIPTION OF THE FLOATING POINT PACKAGE

A. Introduction

The NIC Floating Point Package (FPP) is a collection of subroutines which free the user from the need to program complex arithmetic operations. Each of the routines operates on a number in a floating point format, similar to that of scientific notation. Floating Point-1971 occupies locations 6000-7577 and consists of two parts: the Basic Arithmetic section, and the Extended Functions section. These routines assume the presence of the hardware multiply-divide circuitry now standard on all NIC-1080 computers.

B. Reasons for Using the Floating Point Routines

The NIC-1080 computer stores all information in 20-bit memory words, in which one can represent unsigned integers from 0 to $2^{20}-1$. This is equivalent to $0 - 1{,}048{,}575_{10}$ or $0 - 3777777_8$. If one chooses to operate on signed numbers, the range drops to -2^{19} to $2^{19}-1$, or $-524{,}288$ to $+524{,}287_{10}$. Note that these large numbers contain nearly six significant figures. However, if one is handling small integers such as 20 or 6 or 1, the number of significant figures drops off rapidly. Further, it is not possible to represent fractional numbers successfully within the limits of 20 bits without reducing the number of significant figures even more drastically.

One solution to this problem is to represent each number in two 20-bit words, or double precision, allowing one word to represent the integer part and the other word to represent the fractional part. However, the same problem arises in this format concerning very large and very small numbers. There is no effective way to represent numbers such as 10^{15}, or 10^{-12} and still maintain the same significance.

The Floating Point package overcomes these problems by utilizing an internal computer representation similar to scientific notation. In scientific notation, one represents all numbers in the form $n.nnnn \times 10^{nnn}$. By convention, all numbers are reduced to lie between 1 and 10, and are multiplied by an appropriate power of ten.

Similarly, the internal representation or floating point format requires that the number lie between 0.5 and 1.0 and that the exponent be adjusted appropriately. It is customary, although not altogether accurate, to refer to the number as the <u>mantissa</u>, and the power to which the base is raised to as the <u>exponent</u>. Since the 1080 is a binary machine, the exponent and mantissa are both binary (base 2) numbers.

C. Conversion to Binary Representation

Let us consider a simple example of this conversion procedure. The decimal number 5 is represented in binary as 101. This is scaled right to lie in the requisite range as follows:

$$101 \times 2^0$$
$$10.1 \times 2^1$$
$$1.01 \times 2^2$$
$$.101 \times 2^3$$

The last item in this list, $.101 \times 2^3$, is the representation used by the Floating Point package. Since this is binary notation, the period to the left of the mantissa is called the binary point.

Just as the decimal fraction .213 means

$$2 \times 10^{-1}$$
$$+1 \times 10^{-2}$$
$$+3 \times 10^{-3}$$

the binary fraction .101 means

$$1 \times 2^{-1}$$
$$+0 \times 2^{-2}$$
$$+1 \times 2^{-3}$$

Two more examples of conversions are given below.

$$50_{10} = 62_8 = 110\ 010_2$$

Shifting right, this equals $.110010 \times 2^6$

The representation of fractions in octal and binary is somewhat harder to grasp, but since the program takes care of all such conversions internally, it is not usually necessary to become too familiar with this technique. If we wished to represent 0.75 as a binary number, we need only recognize that $0.75 = 0.50 + 0.25$, and that this is equivalent to $2^{-1} + 2^{-2}$. Thus, $0.75 = 0.11 \times 2^0$.

D. Internal Representation of Binary Fractions

Each floating point number is taken to occupy two consecutive locations. These two 20-bit words are divided so that the exponent occupies 10 bits and the mantissa 30 bits. This division allows us to represent numbers having signed exponents in the range $\pm 2^9$ or ± 512. This is equivalent to roughly 10^{-150} to 10^{+150}. The 30-bit mantissa is used to represent signed numbers in the range $\pm 2^{29}$. This range is greater than ± 500,000,000 and thus implies an accuracy of better than eight decimal digits.

The two computer memory words are divided so that the sign of the exponent and mantissa are each represented by the sign bit (bit 19) for rapid access during calculations. Thus, the left hand ten bits of the first word constitute the exponent, with bit 19 the sign, and the second word represents the most significant 19 bits (plus sign) of the mantissa. The right hand ten bits of the first word, then, represent the least significant 10 bits of the mantissa. This is illustrated below.

First word	Sign	Exponent	Low Order Mantissa
	19	18 10	9 0

Second word	Sign	High Order Mantissa
	19	18 0

A negative mantissa or exponent is represented by its two's complement. Thus, if the first word begins with octal digit 2 or 3 (bit 19 = 1) the exponent is negative and if the second word begins with a 2 or 3, the mantissa is negative.

In summary, the principal advantages of using the floating point routines are

(1) All numbers are represented to the same number of significant figures.
(2) A much larger range of magnitudes can be represented.
(3) The programmer need not keep track of a binary point.
(4) Simplified programming of mathematical functions.

It should be pointed out that there are a few disadvantages to the use of these routines, the most important being:

(1) The execution time is much slower than similar integer routines.
(2) An entire page of memory is required for the subroutines, so that less space is available for programming.

II. PROGRAMMING USING THE FLOATING POINT PACKAGE

A. Pseudo-Registers

The FPP utilizes two pseudo-registers which behave as if they were actually hardware registers: the Floating Accumulator (FAC) and the Floating Argument (FAR). These are actually a set of memory locations, but their loading and operation is handled entirely by software internal to the FPP.

Just as all numerical operations appear to occur in the hardware accumulator (AC) all floating point operations appear to occur in the FAC. Whenever an operation requires two numbers, such as floating point addition, one number is loaded into each of these registers prior to calling the subroutine to perform the addition. The result is always held in the FAC. In all basic arithmetic operations, the FAR is destroyed by the computation.

Locations 7572 and 7573 constitute the FAC and have been given the names FACE and FACM, representing FAC-exponent and FAC-mantissa. During calculations internal to the FPP, the FAC is expanded into three locations: FACE, FACM and FACML, where FACML is location 7574. The FAR occupies 7575, 7576 and 7577. At the end of internal calculation, the calculated result is rounded to 30 significant bits and re-packed into the two-word format.

B. Offpage Subroutine Calls

Since the FPP resides on page 6000, all calls to these subroutines must be in the form of an indirect call. It is convenient to give the subroutine pointers the same names as the actual subroutines as a simple way of remembering the function of the subroutine. Each memory page which refers to the FPP must have its own set of pointers, however, and in the event that more than one page is assembled at a time, different names must be given to the pointers to avoid the DL (duplicate label) error message.

The convention that only one page is being assembled at a time will be adopted in this manual in order to simplify the examples. Thus, to call the FPP addition routine, one simply calls

```
            JMS @ FADD        /PERFORM FPP ADDITION
              .
              .
              .
FADD,       7245              /POINTER TO ADDITION SUBROUTINE
```

This causes a JMS to the subroutine located at address 7245_8, and the subsequent addition of the FAC and FAR. The subroutine returns to the calling program with the result of the addition left in the FAC. The FAR is destroyed.

C. Loading the FAC and FAR

Since all operations take place in the FAC and FAR, it is necessary that the programmer load these registers before calling the arithmetic routines. Subroutines to load these registers are part of the FPP. To load the FAC, one calls the subroutine GETAC with the address of the first of the two words to be transferred in the location following the call. The subroutine GETAC looks at this location and takes its contents as the address of the first word to be moved into the FAC. This is illustrated by the following example, in which the floating point constant FPNUM is loaded into the floating accumulator.

Address	Contents		Mnemonic	
2000	3000003		JMS @ GETAC	/LOAD THE FAC
2001	2004		FPNUM	/ADDRESS OF THE FIRST WORD
2002	5220		STOP	/HALT THE PROCESSOR
2003	7062	GETAC,	7062	/POINTER TO GETAC SUBROUTINE
2004	xxxx	FPNUM,	xxxx	/FP CONSTANT
2005	xxxx		xxxx	

The above routine transfers the contents of locations 2004 and 2005 into the FAC and then halts. The first instruction, JMS @ GETAC jumps to the subroutine pointed to by GETAC, location 2003, causing an effective JMS to 7062. The subroutine at 7062 examines the location following the JMS call, location 2001, and finds the number 2004. It takes this number as the address of the first word to be moved to the FAC. It then increments this address internally and transfers the contents of 2005 to the second word of the FAC. Following this second transfer it exits to the second location following the subroutine call, location 2002, where the computer halts. At this point locations 7572 and 7573 (the actual FAC addresses) contain the same numbers as 2004 and 2005. Neither the actual number nor the pointer to it are changed by this operation. The previous contents of the FAC are destroyed.

A similar subroutine, GETAR, loads the floating argument in an exactly analogous way. A third routine, FACFAR, transfers the contents of the FAC to the FAR directly. FACFAR requires no calling addresses since its action is entirely internal to the FPP. There is no analogous routine to transfer from the FAR to the FAC, since the FAR is generally destroyed by the arithmetic operation, leaving valid information only in the FAC.

Depositing of floating point numbers in memory following such calculations is accomplished in an entirely analogous manner, with the address of the first word given in the location following the call to PUTAC. Thus one can store the contents of the FAC in the two word location TEMP by simply calling:

```
        JMS @ PUTAC    /CALL THE PUTAC ROUTINE
        TEMP           /ADDRESS OF LOCATION 1 OF TEMP
        STOP
TEMP,   Ø              /ACTUAL LOCATION TEMP
        Ø
PUTAC,  7074
```

It should be emphasized at this point that the surest way to programming disaster is to neglect to specify two locations for each floating point constant to be used by the program.

A simple routine for adding the contents of floating point numbers ANUM and BNUM is given below. The result is stored in ANUM following the operation.

```
            JMS @ GETAC     /LOAD THE FAC WITH ANUM
            ANUM            /ADDRESS OF ANUM
            JMS @ GETAR     /LOAD THE FAR WITH BNUM
            BNUM            /ADDRESS OF BNUM
            JMS @ FADD      /PERFORM FP ADDITION
            JMS @ PUTAC     /STORE IN ANUM
            ANUM
            STOP            /AND STOP
GETAC,      7062            /POINTER TO GETAC
GETAR,      7050            /POINTER TO GETAR
ANUM,       xxxx            /ACTUAL FP NUMBER ANUM
            xxxx
BNUM,       xxxx            /ACTUAL FP NUMBER BNUM
            xxxx
PUTAC,      7074
```

An additional temporary storage register TEM is available for programmer use. The subroutines FACTEM and TEMFAC move FAC to TEM and vice-versa. This register must be used with care, since the floating point input routine, FLIP, and all of the extended functions utilize TEM. The basic arithmetic routines, however, do not utilize TEM.

All of the data moving routines are summarized below:

GETAC	7062
GETAR	7050
FACFAR	7026
PUTAC	7074
FACTEM	7034
TEMFAC	7042

D. Basic Arithmetic Routines

The following subroutines accomplish basic arithmetic functions:

Subroutine Name	Function	Subroutine Location
FADD	FAC + FAR → FAC	7245
FSUB	FAC - FAR → FAC	7314
FMULT	FAC × FAR → FAC	7416
FDIV	FAC / FAR → FAC	7461
FNEG	- FAC → FAC	7320

In each case, the result is contained in the FAC. Negation is accomplished by taking the two's complement of the mantissa. Addition and subtraction are accomplished by shifting the FAR or FAC right until the exponents are aligned and then adding, or negating and adding. Multiplication is accomplished by multiplying the mantissas together and adding the exponents, and division by dividing the mantissas and subtracting the exponents. A complete discussion of the algorithms used is given in Part V.

The FAR is destroyed by all basic arithmetic functions except negation. It should be noted that the FAR is subtracted from the FAC during subtraction and that the FAR is the divisor during division. If the exponent becomes too large during addition or subtraction or if division by zero is attempted, the error flag is set. Exponent overflow or underflow does not, however, cause a "wrap-around" which would allow 10^{-150} to ever become 10^{+150} or vice-versa.

E. The Error Flag

Location 7555, called ERRF, is a general purpose error flag. It is set to zero on the FPP binary tape, and is set to one by various error conditions. The error flag is never cleared, once set by any routine. It is thus up to the user to zero it at the beginning of routines and check it at the end of routines. Since ERRF is never re-zeroed, it is only necessary to check its state occasionally, such as at the end of each major section of your program rather than after each individual step. The error flag is set by such conditions as division by zero, exponent overflow, square root or logarithm of a negative number, and number out of range.

F. Floating Point Input and Output

In order to allow the programmer to utilize the FPP most efficiently, a pair of subroutines have been designed to accept data from and output data to the Teletype. The Floating Point Input routine FLIP will accept data in virtually any format and the output routine FLOP outputs data in scientific notation, with a variable number of digits, and with a character counter allowing column justification.

1. FLIP

This routine is called by the call JMS @ FLIP, where FLIP actually points to the input routine, located at 6736. The Teletype is then active, and will accept and echo all characters typed, until an illegal character is entered. Exit then occurs, with the terminating character in the AC, and the converted number stored in the FAC. A carriage return-line feed is not typed by FLIP and must be provided externally. The FAR is destroyed.

Since the Teletype does not have superscript capability, exponents are represented by typing E followed by the desired power of ten. Thus, 5E1 represents 50. Should an exponent larger than 150 be entered, the error flag ERRF is set = 1 upon exit.

Virtually any format is legal for input, except that spaces may not be embedded in the number. A space is detected as an illegal character and causes immediate exit from FLIP. Thus, 5.03E-6 is legal input, but 5.03 E-6 is not. Other examples of legal input include:

```
50
050
50.0
0.005E5
500E-1
+50E+0
etc.
```

Other conditions that cause FLIP to exit include:

(a) a sign anywhere except immediately before the mantissa or exponent,
(b) two decimal points in the mantissa or one in the exponent,
(c) a second E.

The above conditions are not interpreted as errors, however, and the numbers typed before they occur are converted and stored in the FAC. However, the terminating character is always in the AC upon exit from FLIP and can be examined by the calling program. One might require, for example, that the terminating character be a Return.

A mistake during entry of data to FLIP can be corrected by typing a Rubout. The typing of a Rubout causes FLIP to echo with a backslash (\) and zeroes the FAC. The entire number can then be re-entered. FLIP automatically types a backslash and zeroes the FAC if more than 11 significant figures were entered. The value can then be re-entered.

Following an FP input, the valid data flag VFLAG (7004) can be checked for validity of the input. VFLAG is always set to Ø on entry to FLIP and to 1 if valid data was typed. This enables one to wait for a correct input before going on to the next program step. For instance if upon entrance to FLIP, only Q were typed, there would be an immediate exit from FLIP with 321 (ASCII Q) in the AC and VFLAG = Ø.

VFLAG can be used in combination with the terminating character to determine the nature of the data entered. For example, one might wish to read in a pre-punched paper tape containing floating point numbers with a variable number of terminating characters between them, such as spaces, carriage returns, line feeds, etc. The following program will read in 10 numbers from paper tape, store them, and then halt. All illegal characters will be ignored.

```
         MEMA PNTSET    /SET BEGINNING OF STORAGE LIST INTO POINTER
         ACCM DPNT
         MEMA (12       /SET COUNTER TO 10 (BASE 8)
         ACCM COUNT
FAGN,    JMS @ FLIP     /ENTER FLOATING INPUT ROUTINE
         MMOZ @ VFLAG   /LEGAL INPUT?
         JMP FAGN       /NO, RE-ENTER SUBROUTINE
         JMS @ FPUT     /YES, STORE THE RESULT
DPNT,    Ø              /IN LOCATIONS POINTED TO BY THIS POINTER
         MPOM DPNT      /INCREMENT POINTER TWICE
         MPOM DPNT      /FOR NEXT FP NUMBER
         MMOMZ COUNT    /10 DONE YET?
         JMP FAGN       /NO, GET ANOTHER
         STOP           /YES, HALT PROCESSOR
PNTSET,  100000
COUNT,   Ø
FLIP,    6736
VFLAG,   7004
```

2. FLOP

The floating point output routine FLOP types out the value of the FAC in scientific notation on the Teletype. If the FAC contained

```
7572/0002000
7573/1000000
```

calling JMS @ FLOP

```
        .
        .
        .
FLOP, 6513
```

would produce on the Teletype: 1.00000E0. Both the FAC and FAR are destroyed by FLOP. FLOP does not type a carriage return or line feed.

The number of significant figures typed out by FLOP is controlled by the contents of location 6554. In the form the tape is provided, FLOP types out six significant figures, and location 6554 contains 70006, equivalent to MNGA (6. To change the number of figures to 3, for example, this location would be changed to 70003, or MNGA (3.

Since the total number of characters typed by FLOP will vary with the sign of the exponent and the size of the exponent, a character counter has been included in the print routine. Each time any part of the FPP prints a character using the internal subroutine PCHAR, the counter CARCNT

(7021) is incremented. It is up to the user to set and check this counter. In designing programs using this feature, it should be kept in mind that the maximum number of characters which could be produced by FLOP with six significant figures is 13_{10}: -n.nnnnnE-nnn.

The following example causes each output from FLOP to be exactly 14 characters wide. The number of significant figures is set to 4.

```
PFLOP,   Ø                 /SUBROUTINE ENTRY WITH FAC LOADED
         MEMA (16          /14 BASE TEN
         ANGM @ CARCNT     /SET CHARACTER COUNTER TO -14
         JMS @ FLOP
PAGN,    MEMZ @ CARCNT     /IS CHARACTER COUNTER = Ø?
         ZERZ              /NO, SKIP EXIT INSTRUCTION
         JMP @ PFLOP       /YES, EXIT FROM SUBROUTINE
         MEMA (240         /PRINT SPACES TO FILL TO 14
         JMS @ PCHAR       /PRINT ROUTINE IN FPP INCLUDES CARCNT
                           /INCREMENT
         JMP PAGN          /LOOP UNTIL 14 CHARACTERS TYPED
FLOP,    6510
CARCNT,  7021
PCHAR,   7013
         *6554
         MNGA (4           /SETS 4 SIGNIFICANT FIGURE OUTPUT
```

G. Conversion to Floating Point Format

The subroutine FLOAT converts a fixed point integer in the FAC to floating point format, leaving the converted result in the FAC. FLOAT operates on signed integers or signed fractions with a fixed binary point. It considers the two locations FACM and FACML (7573 and 7574) of the expanded FAC to be a 40-bit number with the binary point located between the two words. The contents of the floating exponent word FACE (7572) are unimportant on entry. On exit, the result is left in the FAC in standard floating point format.

Thus, to float a standard 20-bit integer, such as might be found in signal averaged data, one must be sure to zero FACML before calling FLOAT. The following subroutine accomplishes this flotation, assuming the integer is in the AC on entry:

```
FLOTIT,  Ø                 /SUBROUTINE TO FLOAT 20-BIT INTEGERS
         ACCM @ FACM       /AC CONTAINED INTEGER ON ENTRY, STORE IN FACM
         ZERM @ FACML      /ZERO LOW ORDER FAC
         JMS @ FLOAT       /PERFORM THE FLOAT
         JMP @ FLOTIT      /AND EXIT FROM THE SUBROUTINE
FACM,    7573
FACML,   7574
FLOAT,   7534
```

H. Conversion of Floating Point Numbers to Fixed Point

The subroutine FIX converts the floating point number found in FAC to a fixed point number whose binary point lies between FACM and FACML. Since converting to integer format requires that the exponent be decremented until it reaches zero, FACE will be zero upon exit if the FIX was successful. If the FP number was too large to FIX, FACE will be non-zero. If the number was too small to FIX, FACM-FACML will be all zeroes if the sign was positive and all ones if the sign was negative.

III. THE EXTENDED FUNCTIONS

A. Summary of the Extended Functions

The Floating Point package may be logically divided into two sections: the basic arithmetic section, and the extended functions. While the extended functions utilize the basic arithmetic section, the basic arithmetic section stands by itself. In fact, if additional program storage space is needed, and the extended functions are not used by that program, one can overwrite the extended functions section, from 6000 - 6507.

The complete list of extended functions is given below:

Subroutine Name	Function	Location
FSIN	sin(FAC) ⟶ FAC	6001
FCOS	cos(FAC) ⟶ FAC	6113
FARCTN	arctan(FAC) ⟶ FAC	6121
FRIP	1/FAC ⟶ FAC	6170
FSQRT	$(FAC)^{1/2}$ ⟶ FAC	6176
FLOG	log(FAC) ⟶ FAC	6322
FLN	ln(FAC) ⟶ FAC	6330
FSQAR	$(FAC)^2$ ⟶ FAC	6352
FEXP	10^{FAC} ⟶ FAC	6370
FEXPN	e^{FAC} ⟶ FAC	6376

In each case, the result of the calculation is placed in the FAC. If the calculation is not possible, the error flag ERRF is set, and the result is meaningless.

B. Square, Square Root and Reciprocal Functions

FSQAR, FSQRT and FRIP all maintain 30 bits of accuracy. If the squaring of a number causes exponential overflow, the error flag will be set. If FAC is negative, the error flag will be set, and the square root is taken of the absolute value of the FAC. Any attempt to take the reciprocal of zero will also set the error flag. In this last case the FAC will be meaningless.

C. Sine, Cosine and Arctangent

FSIN, FCOS and FARCTN all maintain at least 26-bit accuracy. The decrease in accuracy is a result of the successive approximation methods employed. There are no error conditions.

The argument presented to FSIN and FCOS must be in units of $\pi/2$ radians. This is a convenient unit to work with since four such units make a circle. One represents an angle such as 45° by 0.5, for example. Similarly, FARCTN produces a result in units of $\pi/2$ radians.

D. FLOG, FLN, FEXP and FEXPN

FLOG, FLN, FEXP and FEXPN all maintain at least 26 bits of accuracy. An attempt to exponentiate too large a number will cause the error flag to be set. This number is about 150 for FEXP and about 350 for FEXPN. Any attempt to compute the logarithm of a negative number or of zero will cause the error flag to be set. No operation is performed on FAC in that case.

E. Extended Functions Programming Example

The extreme ease with which the extended functions can be used to perform complex calculations is shown by the following example which calculates $\exp(1/x^2)$.

```
JMS @ GETAC      /GET X FROM MEMORY
X
JMS @ FSQAR      /X**2
JMS @ FRIP       /1/X**2
JMS @ FEXPN      /EXP(1/X**2)
STOP
```

IV. ADVANCED PROGRAMMING CONCEPTS

A. Testing the Terminating Character of FLIP

The first illegal character encountered by the floating point input routine causes an immediate exit, with that ASCII character remaining in the AC. This feature can be used to determine how the input is to be converted. In the following example, FLIP is used to accept numbers assumed to be in the units of $\pi/2$ radians. The terminating character is then either S or C, which implies that the program is to compute the sine or cosine of the entered number, and print it on the Teletype.

```
START,     JMS @ FLIP      /ENTER FLOATING INPUT ROUTINE, GET ARG IN
                           /PI/2 UNITS
           A-MZ (323       /WAS TERMINATING CHARACTER "S"?
           JMP CTEST       /NO, TEST FOR C
           JMS @ FSIN      /YES, CONVERT TO SINE
OUTPUT,    MEMA (275       /PRINT EQUALS SIGN
           JMS @ PCHAR     /USING FFP PRINT ROUTINE
           JMS @ FLOP      /PRINT CONVERTED SINE OR COSINE
           JMS CRLF        /PRINT CARRIAGE RETURN-LINE FEED; ROUTINE
                           /NOT SHOWN
           JMP START       /AND GET NEW INPUT VALUE
CTEST,     A-MZ (303       /WAS CHARACTER "C"?
           STOP            /NO, ERROR, HALT PROCESSOR
           JMS @ FCOS      /YES, CONVERT TO COSINE
           JMP OUTPUT      /AND PRINT VALUE ON TTY
FLIP,      6736            /POINTERS TO FLOATING POINT SUBROUTINES
FLOP,      6510
FSIN,      6001
FCOS,      6113
PCHAR,     7013
```

B. Reading and Printing Characters

The subroutines RCHAR and PCHAR read and print characters on the Teletype. RCHAR reads a character from the Teletype and then calls PCHAR to print it. It is therefore not necessary, in general, to write one's own read and print subroutines. Should the user decide to write similar routines for other memory pages, it is necessary that they have the same timing structure as those in the FPP. RCHAR and PCHAR are both structured in the sense: wait for the flag and skip, jump back, then listen or print:

```
T1,  TTYRF     P1,  TTYPF
     JMP T1         JMP P1
     RDTTY          PRTTY
```

It is possible, of course, to write routines in the order:

```
     PRTTY
T2,  TTYPF
     JMP T2
```

so that the program waits for the flag to go up before continuing. This second method can not be used with the FPP, since it would cause timing errors between the user's subroutine and the FPP subroutine.

C. Multiplication and Division by Two

In fairly long calculations, it becomes apparent that Floating Point calculations are significantly slower than integer calculations. It is therefore desirable to avoid the slower method whenever a faster one is available. Multiplication by 2 can be relatively time consuming if carried out in floating point, for example, but is easily accomplished in fixed point. Since the exponent of a floating point number is a power of two, simply incrementing the exponent by 1 will accomplish this multiplication. However, the exponent occupies the left hand ten bits of an FP word, and therefore the addition must be done to the exponent alone, by adding 2000_8 to the first word. The following sequence of code multiplies the FAC by 2:

```
MPOA (1777      /GET THE CONSTANT 2000 INTO THE AC
A+MM @ FACE     /AND ADD INTO THE EXPONENT
```

Similarly, division by 2 can be accomplished by subtracting 2000_8 from FACE:

```
MPOA (1777
M-AM @ FACE     /SUBTRACT 2000 FROM FACE
```

D. Testing the FAC for Zero

After any operation, one can test the FAC to see if it has become zero by examining FACM. While the exponent may still have some non-zero value, the mantissa will be zero if and only if the FP number is zero. One can therefore test for a zero input from FLIP by the following code:

```
FLOOP,  JMS @ FLIP       /FLOATING INPUT ROUTINE
        MEMZ @ FACM      /ZERO INPUT?
        ZERZ             /NO, INPUT OK
        JMP FLOOP        /YES, GET NEW INPUT
```

It should be emphasized however, that it is poor programming practice, just as in high-level languages, to assume that any two floating point numbers will ever become <u>exactly</u> equal. If one wishes to find out whether a number has reached a value of 1.9, he cannot assume that subtraction of 1.9 from that number will produce exactly zero. The actual result of such a subtraction may well be 10^{-9} or so, but will be finite and non-zero. This is simply because the internal representation of some numbers is not exactly the same in base 2 as in base 10. It also could be because a calculated number may be somewhat different than a floated integer. The usual procedure in this case is to subtract the two numbers and determine whether their difference is less than some tolerance, such as 10^{-6}.

E. Skipping on Positive or Negative FAC

Since the sign bit of FAC is readily available in bit 19 of FACM, it is quite possible to perform a simple calculation and then allow the program to branch depending on the sign of the result. If this procedure is to be carried out a number of times in a program, it is advantageous to use a branching subroutine like that shown below. The subroutine is entered with FAC and FAR loaded with the two numbers to be compared. It will produce a skip if the result after subtraction is positive.

```
SKIP+,  Ø               /ENTER WITH FAC AND FAR LOADED
        JMS @ FSUB      /PERFORM THE SUBTRACTION
        MEMA @ FACM     /TEST SIGN OF MANTISSA
        EXCT AC19       /IS THE SIGN NEGATIVE?
        JMP @ SKIP+     /YES, EXIT WITHOUT SKIPPING
        MPOM SKIP+      /NO, INCREMENT EXIT POINTER
        JMP @ SKIP+     /AND EXIT WITH SKIP OF NEXT INSTRUCTION
```

F. Determining of Floating Point Constants

The following constants have been converted into packed two word floating point format for general use:

Constant	Octal Value	Decimal Value
π	0005526 1444176	3.1415926
$\pi/2$	0003526 1444176	1.5707963
e	0005212 1267702	2.7182818
10.0	0010000 1200000	
1.0	0002000 1000000	

One can easily generate constants for one's own use by utilizing Nicobug to call the FLIP or FLOP routines. The following simple code, entered near the end of page Ø allows one to call subroutines from Nicobug.

Address	Contents	Mnemonic
1770	3001772	JMS @ 1772
1771	5220	STOP
1772	6736	Address of subroutine to be called.

On starting Nicobug, enter these numbers as shown. Then enter a breakpoint in the location following the subroutine call, at location 1771, by typing 1771B. To run the subroutine, type 1770G. This will immediately enter FLIP at 6736. Type the decimal number to be converted, followed by any terminating character. Upon receipt of the terminating character, FLIP will exit to 1771, which now contains the breakpoint jump back to Nicobug. Nicobug will type out the contents of the AC, and then allow you to examine the FAC, at locations 7572 and 7573.

G. Roundoff and Overflow

There has been a great deal of discussion among programmers about roundoff problems. The magnitude of the problem is illustrated by the following experiment. Ask several computers in several languages to add pairs of numbers, like .1 and 1.9, and take the integer part. The answers will undoubtedly differ somewhat from machine to machine.

This problem arises partly because computers are binary machines. A number that is simple to represent in decimal, such as 0.1, is impossible to represent exactly in binary. The internal representation of 0.1 may be high (or low) by an amount not greater than one part in one billion in the case of the FPP. The most accurate decimal representation of the internal representation may well be .099999999. Knowing this makes it easy to see how the integer part of (1.9 + 0.1) can be one.

The appearance of a number like .0999999 is something of a surprise when one expects 0.1. The FPP solves this problem of aesthetics by adding approximately one part per billion to a number before printing it. The effect on the sixth digit of the mantissa is almost always invisible. The feature can be removed or the amount of roundoff changed by varying the roundoff constant shown in the listing.

As mentioned earlier, all arithmetic operations produce 40-bit mantissas which are truncated to 30 bits. Before truncation these 10 bits are examined. If the most significant bit is a one, the 30 bit final mantissa is incremented. If this were not done, all arithmetic operations would produce answers systematically too small by an amount averaging .5 parts per billion.

TABLE I

POINTERS TO FLOATING POINT, 1971 NIC-80/S-7118B

FADD	fac + far → fac	7245		
FSUB	fac - far → fac	7314		
FNEG	- fac → fac	7320		
FMULT	fac × far → fac	7416		
FDIV	fac / far → fac	7461	ERRF	= 7555
FLOP	floating output	6510	CARCNT	= 7021
FLIP	floating input	6736	VFLAG	= 7004
PCHAR	prints character	7013	FACE	= 7572
RCHAR	reads & prints char.	7005	FACM	= 7573
GETAC	x → fac	7062	FARE	= 7575
GETAR	x → far	7050	FARM	= 7576
FACFAR	fac → far	7026		
PUTAC	fac → x	7074		
FACTEM	fac → tem	7034		
TEMFAC	tem → fac	7042		
FLOAT	floats facm-facml	7534		
FIX	fixes fac	7541		
FSIN	sin(fac) → fac	6001		
FCOS	cos(fac) → fac	6113		
FARCTN	arctan(fac) → fac	6121		
FRIP	1/fac → fac	6170		
FSQRT	$fac^{1/2}$ → fac	6176		
FLOG	log(fac) → fac	6322		
FLN	ln(fac) → fac	6330		
FEXP	10^{fac} → fac	6370		
FEXPN	e^{fac} → fac	6376		
FSQAR	fac^2 — fac	6352		

TABLE II

FLOATING POINT PACKAGE SUMMARY

Operation	Mnemonic	Units	Accuracy	Speed*	Error Conditions	Registers Destroyed
Multiplication & Division	FMULT, FDIV		30 bits	.8 ms	Exponent overflow, Zero division	FAR
Addition & Subtraction	FADD, FSUB		30 bits	.8 ms	None	FAR
Square	FSQAR		30 bits	.8 ms	Exponent overflow	FAR
Square root	FSQRT		30 bits	6 ms	Negative argument	FAR TEM
Reciprocal	FRIP		30 bits	.8 ms	Zero argument	FAR
Sine	FSIN	$\pi/2$ radians (input)	26 bits	12 ms	None	FAR TEM
Cosine	FCOS	$\pi/2$ radians (input)	26 bits	12 ms	None	FAR TEM
Arctangent	FARCTN	$\pi/2$ radians (output)	26 bits	31 ms	None	FAR TEM
Logarithm, base ten	FLOG		26 bits	12 ms	Negative or zero argument	FAR TEM
Logarithm, base e	FLN		26 bits	12 ms	Negative or zero argument	FAR TEM
Exponentiate, base ten	FEXP		26 bits	20 ms	Zero argument Argument > 150	FAR TEM
Exponentiate, base e	FEXPN		26 bits	20 ms	Zero argument Argument > 350	FAR TEM

*Highly data dependent; as rough guide only

V. ALGORITHMS USED IN THE EXTENDED FUNCTIONS

A. Introduction

This is a description of the algorithms used to compute the extended functions. Most of the functions are computed by methods described by Cecil Hastings in his book Approximations for Digital Computers (Princeton University Press, 1955).

B. Square Root

First a guess is made by dividing the exponent of the argument by 2. Then the guess is refined by setting it equal to:

$$\text{New Guess} = \left(\frac{\text{Old Guess} + \text{Argument}}{2^{*}\ \text{Old Guess}}\right)$$

Then the process is repeated 5 times.

C. Sine

First the argument is "rotated" into the first quadrant by adding or subtracting ones. The following identities are used:

sine (-x) = -sin(x)
sine (1+x) = sine (1-x)

Then the following Taylor series polynomial is evaluated:

$$\sin x = \sum_{i=0}^{n} C_{2i+1} x^{2i+1}$$

where n = 4. The values for C are

$C_1 = .157080 \times 10^1$
$C_3 = -.645964 \times 10^0$
$C_5 = .796897 \times 10^{-1}$
$C_7 = -.467377 \times 10^{-2}$
$C_9 = .151484 \times 10^{-3}$

D. Cosine

The cosine function is evaluated with the sine subroutine with the aid of the identity:

cos(x) = sin (1+x)

E. Arctangent

If the argument is ≤ 1, then

$$\text{arctangent } x = \sum_{i=0}^{n} C_{2i+1} x^{2i+1}$$

Otherwise:

$$\text{arctangent } x = 1 - \sum_{i=0}^{n} C_{2i+1} \left(\frac{1}{x}\right)^{2i+1}$$

where n = 7.

The values of C are

$C_1 = 0.636619347$
$C_3 = -0.212184453$
$C_5 = 0.126983591$
$C_7 = -0.08854474$
$C_9 = 0.061382906$
$C_{11} = -0.035503338$
$C_{13} = 0.013917289$
$C_{15} = -0.002580893$

F. Logarithm Base ten and Base e

The logarithm to the base 2 is computed and the final result is determined from the fact that

$\log_e x = (\log_2 x)(\log_e 2)$ for base e

$\log_{10} x = (\log_2 x)(\log_{10} 2)$ for base ten

The log to base 2 is calculated as follows:

(1) If the argument is ≤ 0, the error flag is set and the logarithm subroutine exits.
(2) If the argument is < 1, the end result is negated.
(3) If the argument is ≥ 1, the reciprocal of the argument is taken.
(4) The original exponent is saved and the exponent of FAC is set to 1. Thus $1 < FAC \leq 2$.
(5) Z is computed from the equation

$$Z = \frac{x - \sqrt{2}}{x + \sqrt{2}}$$

(6) Then

$$\text{Log}_2 x = -1/2 + \sum_{i=0}^{n} C_{2i+1} Z^{2i+1}$$

is computed for n = 2. The values of C are:

$C_1 = .288539 \times 10^1$
$C_3 = .961471 \times 10^0$
$C_5 = .598979 \times 10^0$

(7) The exponent is retrieved, converted to a floating number, and added to FAC.

(8) FAC is negated if necessary and multiplied by the proper constant.

G. Exponentiation, Base ten and Base e

FAC is multiplied by a constant so that the internal base 2 exponentiation subroutine can be used.

$$e^x = 2^x \log_2 e \quad \text{for base e}$$

$$10^x = 2^x \log_2 10 \quad \text{for base ten}$$

If FAC is negative the absolute value is taken and the final answer is the reciprocal.

Then FAC is separated into a fractional part and an integer part by subtracting ones. The fractional part, F, is evaluated.

$$2^F \times 1 + \cfrac{2F}{A - F + BF^2 - \cfrac{C}{D + F2}}$$

where the constants are:

A = +9.95459578
B = +0.03465735903
C = +617.97226053
D = +87.417497202

Now the integer part is converted into a fixed point number and added to the exponent of FAC.

Section VI, Listing of Basic Arithmetic and Section VII, Listing of Extended Functions are included as a part of the 1080 Instruction Manual and are available upon request.

VI. LISTING OF BASIC ARITHMETIC

```
/FLOATING POINT PACKAGE 1971
/PART 1 OF 2

/NIC-8Ø/S-7118-L
/COPYRIGHT 1971, NICOLET INSTRUMENT CORPORATION, MADISON, WIS.

/FLOATING OUTPUT

*651Ø

   651Ø        Ø  FLOP, Ø
   6511 2Ø25562  ONEM TEM2
   6512 2165557  ZERM DEXP
   6513 2ØØ12Ø6  JMS EXFAC
   6514 2111573  MEMA FACM
   6515    51Ø4  SKIP AC19
   6516     522  JMP FLOP1
   6517 2ØØ1236  JMS N3FAC
   652Ø  11Ø255  MEMA (255
   6521  162ØØØ  ZERZ
   6522  11Ø24Ø  FLOP1, MEMA (24Ø
   6523 2ØØ1Ø13  JMS PCHAR  /PRINT SPACE OR - SIGN
/ADD ROUNDING
   6524 2ØØ1364  JMS RSHFAC
   6525  1114ØØ  MEMA (14ØØ
   6526 25Ø7574  A+MMZ FACML
   6527 2715573  MMOMA FACM
   653Ø 2135573  MPOMA FACM
   6531  40516Ø  EXCT ZAC  /TEST Ø
   6532     567  JMP OML1
   6533 2ØØ11Ø6  JMS FNOR3
   6534     537  JMP FGO2

/ADJUST FAC
   6535 2ØØ1461  FGO3, JMS FDIV
   6536 2125557  MPOM DEXP
   6537 2ØØ1Ø5Ø  FGO2, JMS GETAR
   654Ø    6661  KTEN
   6541 2111572  MEMA FACE
   6542    5Ø32  RASH 12
   6543  4Ø5124  SKIP AC19 ZAC
   6544     535  JMP FGO3  /FACE TOO LARGE
   6545  51ØØØ4  A+MA (4
   6546  4Ø5124  SKIP AC19  ZAC
   6547     553  JMP FGO4  /-1 <FACE<-4
   655Ø 2ØØ1416  JMS FMULT  /MULTIPLY BY 1Ø
   6551 27Ø5557  MMOM DEXP
   6552     537  JMP FGO2

   6553 2ØØ12Ø6  FGO4, JMS EXFAC
   6554 216756Ø  ZERMZ MANLZS
   6555 2ØØ1364  FGO5, JMS RSHFAC
```

```
   6556     5144  EXCT AC19
   6557      555  JMP FG05
/NUMBER OF DIGITS IN MANTISSA CAN BE CHANGED
   6560    70006  RESTART, MNGA (6  /OUTPUT MANTISSA
   6561 2405563  ACCM CHRCNT
   6562 2000634  OMLOOP, JMS TTEN  /MULTIPLY FAC BY TEN
   6563 2111564  MEMA DDIGIT
   6564 2507560  A+MMZ MANLZS  /1ST DIGIT 0?
   6565      571  JMP CONT1
   6566 2705557  MMOM DEXP
   6567 2025560  OML1, ONEM MANLZS  /FIX 0 CASE
   6570      560  JMP RESTART

   6571 2001022  CONT1, JMS PDECN
   6572 2707562  MMOMZ TEM2  /FIRST CHAR?
   6573      576  JMP FLOP10
   6574   110256  MEMA (256  /YES
   6575 2001013  JMS PCHAR
   6576 2127563  FLOP10, MPOMZ CHRCNT  /LAST CHAR.?
   6577      562  JMP OMLOOP
   6600   110305  MEMA (305  /OUTPUT EXPONENT
   6601 2001013  JMS PCHAR  /PRINT "E"
   6602 2165564  ZERM DDIGIT
/PRINT EXPONENT
   6603 2715557  MMOAM DEXP
   6604     5104  SKIP AC19
   6605      611  JMP SUBHUN
   6606 2065557  MNGM DEXP
   6607   110255  MEMA (255
   6610 2001013  JMS PCHAR  /PRINT SIGN
   6611   110144  SUBHUN, MEMA (144
   6612 2331557  M-AA DEXP
   6613     5144  EXCT AC19
   6614      620  JMP SUBTEN
   6615 2405557  ACCM DEXP
   6616   110001  MEMA (1
   6617 2001022  JMS PDECN
   6620   110012  SUBTEN, MEMA (12
   6621 2335557  M-AAM DEXP
   6622 2125564  MPOM DDIGIT
   6623     5104  SKIP AC19
   6624      620  JMP SUBTEN
   6625   510012  A+MA (12
   6626 2405557  ACCM DEXP
   6627 2713564  MMOAZ DDIGIT
   6630 2001022  JMS PDECN
   6631 2111557  MEMA DEXP
   6632 2001022  JMS PDECN
   6633 1000510  JMP @FLOP

   6634        0  TTEN, 0  /MULTIPLY FAC BY TEN, INTEGER
   6635 2165564  ZERM DDIGIT
   6636 2001026  JMS FACFAR
```

```
6637 2111574  MEMA FACML
6640 2405577  ACCM FARML
6641   70011  MNGA (11
6642 2405565  ACCM CNTR
6643    5210  MPTENL, CLL
6644 2111577  MEMA FARML
6645 2515574  A+MAM FACML
6646 2111576  MEMA FARM
6647    5141  EXCT L
6650  430000  APOA
6651 2511573  A+MA FACM
6652    5144  EXCT AC19
6653 2125564  MPOM DDIGIT
6654    5212  CLL TLAC
6655 2405573  ACCM FACM
6656 2127565  MPOMZ CNTR
6657     643  JMP MPTENL
6660 1000634  JMP @TTEN

6661   10000  KTEN, 10000  /10
6662 1200000   1200000

6663       0  RSINT, 0  /INPUT SIGNED DECIMAL INTEGER
6664 2165563  ZERM CHRCNT
6665 2165573  ZERM FACM
6666 2165574  ZERM FACML
6667 2165554  ZERM SIGNF
6670 2001005  JMS RCHAR
6671 2405005  ACCM RCHAR  /SAVE TERMINATING CHAR.
6672  462255  A-MZ (255  /"-"?
6673     676  JMP FETCP
6674 2025554  ONEM SIGNF
6675     733  JMP RSINT2
6676  462253  FETCP, A-MZ (253  /+
6677  162000  ZERZ
6700     733  JMP RSINT2
6701 2103566  FETCH2, MEMZ PERSW  /ILLEGAL . CAUSES EXIT
6702  462256  A-MZ (256  /.?
6703     706  JMP FETCH3
6704 2165566  ZERM PERSW
6705     733  JMP RSINT2

6706  462377  FETCH3, A-MZ (377  /RUBOUT?
6707  162000  ZERZ
6710    1000  JMP BSLANT
6711  470260  A-MA (260  /DECIMAL NUMBER
6712 2405561  ACCM TEMP
6713    5144  EXCT AC19
6714 1000663  JMP @RSINT  /NO
6715  470012  A-MA (12
6716    5104  SKIP AC19
6717 1000663  JMP @RSINT  /NO
6720 2025004  ONEM VFLAG  /DECIMAL NUMBER DETECTED
```

```
    6721 2703566  MMOZ PERSW
    6722 2125563  MPOM CHRCNT
    6723 2000634  JMS TTEN  /MULTIPLY FAC BY TEN, INTEGER
    6724 2003003  ANDZ MK
    6725    1000  JMP BSLANT
    6726 2111561  MEMA TEMP
    6727    5210  CLL
    6730 2515574  A+MAM FACML
    6731    5141  EXCT L
    6732 2125573  MPOM FACM
    6733 2001005  RSINT2, JMS RCHAR
    6734 2405005  ACCM RCHAR  /SAVE TERM. CHAR.
    6735     701  JMP FETCH2

/FLOATING INPUT

    6736       0  FLIP, 0
    6737 2025566  RSTART, ONEM PERSW
    6740 2165004  ZERM VFLAG
    6741 2000663  JMS RSINT
    6742  110047  MEMA (47
    6743 2405572  ACCM FACE
    6744 2001117  JMS NORCON
    6745 2001034  JMS FACTEM  /SAVE FAC
    6746 2111563  MEMA CHRCNT
    6747 2225557  ANGM DEXP
    6750 2111005  MEMA RCHAR
    6751  462305  A-MZ (305  /TERMINATING CHAR. E?
    6752     761  JMP COMPEN
    6753 2165566  ZERM PERSW  /MAKE . IN EXP ILLEGAL
    6754 2000663  JMS RSINT
    6755 2111574  MEMA FACML
    6756 2103554  MEMZ SIGNF
    6757  230000  ANGA
    6760 2505557  A+MM DEXP
/COMPENSATE FOR DECIMAL EXPONENT
/MULT. OR DIVIDE BY 10 AS REQUIRED.
    6761 2001042  COMPEN, JMS TEMFAC
    6762 2001050  COMP1, JMS GETAR
    6763    6661  KTEN
    6764 2113557  MEMAZ DEXP
    6765     770  JMP COMP2
    6766 2111005  MEMA RCHAR
    6767 1000736  JMP @FLIP

    6770    5104  COMP2, SKIP AC19
    6771     775  JMP COMP3
    6772 2001461  JMS FDIV
    6773 2125557  MPOM DEXP
    6774     762  JMP COMP1

    6775 2001416  COMP3, JMS FMULT
    6776 2705557  MMOM DEXP
```

```
   6777     762  JMP COMP1

   7000  110334  BSLANT, MEMA (334
   7001 2001013  JMS PCHAR  /PRINT \ AND START OVER
   7002     737  JMP RSTART

   7003 3600000  MK, 3600000
   7004       0  VFLAG, 0  /VALID INPUT FLAG. SET TO 1 IF FLIP FINDS
/A VALID NUMBER. OTHERWISE SET TO 0.

   7005       0  RCHAR, 0  /READ CHARACTER FROM TTY
   7006    6454  RCHAR1, TTYRF
   7007    1006  JMP RCHAR1
   7010   44453  RDTTY
   7011 2001013  JMS PCHAR
   7012 1001005  JMP @RCHAR

   7013       0  PCHAR, 0  /PRINT CHARACTER
   7014    6444  WAIT, TTYPF
   7015    1014  JMP WAIT
   7016    4443  PRTTY
   7017 2125021  MPOM CARCNT  /COUNT CHARACTERS PRINTED
   7020 1001013  JMP @PCHAR

/CHARACTER COUNTER MAY BE USED TO JUSTIFY COLUMNS OF NUMBERS.
/COUNTER MUST BE SET AND EXAMINED EXTERNALLY.
   7021       0  CARCNT, 0

   7022       0  PDECN, 0  /PRINT DECIMAL DIGIT
   7023  510260  A+MA (260
   7024 2001013  JMS PCHAR
   7025 1001022  JMP @PDECN

/MOVE FAC TO FAR
   7026       0  FACFAR, 0
   7027 2111572   MEMA FACE
   7030 2405575  ACCM FARE
   7031 2111573  MEMA FACM
   7032 2405576  ACCM FARM
   7033 1001026  JMP @FACFAR

/MOVE AC TO TEM
   7034       0  FACTEM, 0
   7035 2111572  MEMA FACE
   7036 2405567  ACCM TEMPEX
   7037 2111573  MEMA FACM
   7040 2405570  ACCM TEMPM
   7041 1001034  JMP @FACTEM

/MOVE TEM TO FAC

   7042       0  TEMFAC, 0
   7043 2111567  MEMA TEMPEX
```

```
    7Ø44 24Ø5572  ACCM FACE
    7Ø45 211157Ø  MEMA TEMPM
    7Ø46 24Ø5573  ACCM FACM
    7Ø47 1ØØ1Ø42  JMP @TEMFAC

/THE POINTER TO THE DATA IS FOUND IN THE LOCATION AFTER THE CALL.
/GET 2 WORDS FOR FAR

    7Ø5Ø       Ø  GETAR, Ø
    7Ø51 31115Ø5Ø MEMA @GETAR
    7Ø52 2125Ø5Ø  MPOM GETAR
    7Ø53 24Ø5556  ACCM TEM1
    7Ø54 3111556  MEMA @TEM1
    7Ø55 24Ø5575  ACCM FARE
    7Ø56 2125556  MPOM TEM1
    7Ø57 3111556  MEMA @TEM1
    7Ø6Ø 24Ø5576  ACCM FARM
    7Ø61 1ØØ1Ø5Ø  JMP @GETAR

/GET 2 WORDS FOR FAC

    7Ø62       Ø  GETAC, Ø
    7Ø63 31110Ø62 MEMA @GETAC
    7Ø64 2125Ø62  MPOM GETAC
    7Ø65 24Ø5556  ACCM TEM1
    7Ø66 3111556  MEMA @TEM1
    7Ø67 24Ø5572  ACCM FACE
    7Ø7Ø 2125556  MPOM TEM1
    7Ø71 3111556  MEMA @TEM1
    7Ø72 24Ø5573  ACCM FACM
    7Ø73 1ØØ1Ø62  JMP @GETAC

/"PUT" FAC

    7Ø74       Ø  PUTAC, Ø
    7Ø75 31110Ø74 MEMA @PUTAC
    7Ø76 2125Ø74  MPOM PUTAC
    7Ø77 24Ø5556  ACCM TEM1
    71ØØ 2111572  MEMA FACE
    71Ø1 34Ø5556  ACCM @TEM1
    71Ø2 2125556  MPOM TEM1
    71Ø3 2111573  MEMA FACM
    71Ø4 34Ø5556  ACCM @TEM1
    71Ø5 1ØØ1Ø74  JMP @PUTAC

/SUBROUTINE TO CONVERT 3 WORD, SIGNED, UNNORMALIZED, FAC TO
/TO 2 WORD, NORMALIZED, ROUNDED, SIGNED FAC.

    71Ø6       Ø  FNOR3, Ø
    71Ø7 2111573  MEMA FACM
    711Ø    51Ø4  SKIP AC19
    7111 2167554  ZERMZ SIGNF  /PASS SIGN TO NORCON
    7112 2145554  MONM SIGNF
```

```
    7113     5144  EXCT AC19
    7114 2001236  JMS N3FAC
    7115 2001117  JMS NORCON
    7116 1001106  JMP @FNOR3

/SUBROUTINE TO;
/NORMALIZE FAC
/ROUND TO 30 BITS
/NEGATE IF SIGNF=-1, NO NEGATION IF 0
/CHECK FOR EXPONENT OVERFLOW
/CONTRACT FAC TO 2 WORDS
/ASSUMES POSITIVE INPUT

    7117        0  NORCON, 0
    7120 2113573  MEMAZ FACM  /ZERO MANTISSA?
    7121     1127  JMP NOR2
    7122 2103574  MEMZ FACML
    7123     1127  JMP NOR2
    7124     1166  JMP NOR7

    7125 2001170  NOR2A, JMS LSHFAC
    7126 2111573  MEMA FACM
    7127     5001  NOR2, LASH 1
    7130     5104  SKIP AC19  /NORMALIZED?
    7131     1125  JMP NOR2A
/ROUND SECTION
    7132   111000  MEMA (1000
    7133     5210  CLL
    7134 2515574  A+MAM FACML
    7135    51777  MCPA (1777
    7136 2005574  ANDM FACML  /MASK LSB'S
    7137 2111573  MEMA FACM
    7140     5141  EXCT L
    7141 2135573  MPOAM FACM
    7142     5104  SKIP AC19  /MANTISSA OVERFLOW?
    7143     1147  JMP NOR3  /NO
    7144   405021  RISH 1  /MANTISSA WAS 1777777,3777NNN
    7145 2405573  ACCM FACM  /DIVIDE FAC BY 2
    7146 2125572  MPOM FACE
    7147 2103554  NOR3, MEMZ SIGNF
    7150 2001236  JMS N3FAC
    7151 2111572  MEMA FACE  /CHECK EXPONENT OVERFLOW
    7152     5144  EXCT AC19
    7153   230000  ANGA
    7154   471000  A-MA  (1000
    7155     5144  EXCT AC19
    7156     1166  JMP NOR7
/EXPONENT OVERFLOW RECOVERY
    7157 2025555  ONEM ERRF
    7160 2111572  MEMA FACE
    7161     5104  SKIP AC19  /FAKE AN EXPONENT
    7162   172000  ZERAZ
    7163    30000  ONEA
```

```
    7164  510777  A+MA (777
    7165 2405572  ACCM FACE
    7166 2001226  NOR7, JMS CONFAC
    7167 1001117  JMP @NORCON

/LEFT SHIFT 3 WORD FAC, COMPENSATE EXPONENT

    7170       0  LSHFAC, 0
    7171 2111574  MEMA FACML
    7172    5210  CLL
    7173 2515574  A+MAM FACML
    7174 2111573  MEMA FACM
    7175 2505573  A+MM FACM
    7176    5141  EXCT L
    7177 2125573  MPOM FACM
    7200 2715572  MMOMA FACE
    7201 1001170  JMP @LSHFAC

    7202       0  0  /SPARES
    7203       0  0
    7204       0  0
    7205       0  0

/EXPAND FAC TO 3 WORDS

    7206       0  EXFAC, 0
    7207 2111572  MEMA FACE
    7210    5012  LASH 12
    7211 2405574  ACCM FACML
    7212 2111572  MEMA FACE
    7213    5032  RASH 12
    7214 2405572  ACCM FACE
    7215 1001206  JMP @EXFAC

/EXPAND FAR

    7216       0  EXFAR, 0
    7217 2111575  MEMA FARE
    7220    5012  LASH 12
    7221 2405577  ACCM FARML
    7222 2111575  MEMA FARE
    7223    5032  RASH 12
    7224 2405575  ACCM FARE
    7225 1001216  JMP @EXFAR

/CONTRACT 3 WORD FAC

    7226       0  CONFAC, 0
    7227 2111572  MEMA FACE
    7230    5012  LASH 12
    7231 2405572  ACCM FACE
    7232 2111574  MEMA FACML
    7233  405032  RISH 12
```

```
    7234 2505572  A+MM FACE
    7235 1001226  JMP @CONFAC

 /NEGATE 3 WORD FAC

    7236       0  N3FAC, 0
    7237 2075574  MNGAM FACML
    7240 2045573  MCPM FACM
    7241  405160  EXCT ZAC
    7242 2135573  MPOMA FACM
    7243 1001236  JMP @N3FAC
    7244       0  0  /SPARE

 /FLOATING POINT ADDITION

    7245       0  FADD, 0
    7246 2001216  JMS EXFAR
    7247 2103576  EADD, MEMZ FARM
    7250  162000  ZERZ
    7251 1001245  JMP @FADD
    7252 2001206  JMS EXFAC
    7253 2001364  JMS RSHFAC
    7254 2001401  JMS RSHFAR  /PREVENT OVERFLOWS
    7255 2103573  MEMZ FACM  /FORCE ALIGNMENT IF FAC=0
    7256  162000  ZERZ
    7257 2405572  ACCM FACE
    7260 2333572  M-AAZ FACE
    7261  162000  ZERZ
    7262    1275  JMP FADD1  /MANTISSAS ALIGNED
    7263 2405556  ACCM TEM1
    7264    5104  SKIP AC19  /WHICH IS LARGER?
    7265    1272  JMP FADD2  /FACE IS
    7266 2001364  FADD3, JMS RSHFAC
    7267 2127556  MPOMZ TEM1
    7270    1266  JMP FADD3
    7271    1275  JMP FADD1  /MANTISSAS ALIGNED

    7272 2001401  FADD2, JMS RSHFAR
    7273 2707556  MMOMZ TEM1
    7274    1272  JMP FADD2
/DO ACTUAL ADDITION
    7275    5210  FADD1, CLL
    7276 2111577  MEMA FARML
    7277 2515574  A+MAM FACML
    7300 2111576  MEMA FARM
    7301 2505573  A+MM FACM
    7302    5141  EXCT L
    7303 2135573  MPOMA FACM
/FORM ABSOLUTE VALUE OF FAC
    7304 2001106  JMS FNOR3
    7305 1001245  JMP @FADD

    7306       0  0  /SPARE
```

```
   7307 2111314  SUB1, MEMA FSUB
   7310 2405245  ACCM FADD
   7311 2001216  JMS EXFAR
   7312 2001325  JMS N3FAR
   7313    1247  JMP EADD

/FLOATING POINT SUBTRACTION

   7314       0  FSUB, 0
   7315    1307  JMP SUB1
   7316       0  0  /SPARES
   7317       0  0

/NEGATE FAC

   7320       0  FNEG, 0
   7321 2001206  JMS EXFAC
   7322 2001236  JMS N3FAC
   7323 2001226  JMS CONFAC
   7324 1001320  JMP @FNEG

/NEGATE 3 WORD FAR

   7325       0  N3FAR, 0
   7326 2075577  MNGAM FARML
   7327 2045576  MCPM FARM
   7330  405160  EXCT ZAC
   7331 2135576  MPOMA FARM
   7332 1001325  JMP @N3FAR
   7333       0  0  /SPARES
   7334       0  0
   7335       0  0
   7336       0  0
   7337       0  0
   7340       0  0
   7341       0  0
   7342       0  0
   7343       0  0

/PREPARATION FOR MULTIPLICATION OR DIVISION
/DETERMINE SIGN OF ANSWER
/SET FAC AND FAR TO ABSOLUTE VALUE
/EXPAND FAC AND FAR

   7344       0  PREPAR, 0
   7345  635212  635212  /SET AC TO 2000000
   7346 2011573  ANDA FACM
   7347 2511576  A+MA FARM
   7350    5104  SKIP AC19  /SIGNS DIFFERENT
   7351 2167554  ZERMZ SIGNF  /NO, PASS SIGN TO NORCON
   7352 2145554  MONM SIGNF
/FORM ABSOLUTE VALUES
   7353 2001206  JMS EXFAC
```

```
    7354 2001216  JMS EXFAR
    7355 2111573  MEMA FACM
    7356    5144  EXCT AC19
    7357 2001236  JMS N3FAC
    7360 2111576  MEMA FARM
    7361    5144  EXCT AC19
    7362 2001325  JMS N3FAR
    7363 1001344  JMP @PREPAR

/RIGHT SHIFT 3 WORD FAC
/CONPENSATES EXPONENT

    7364       0  RSHFAC, 0
    7365    5210  CLL
    7366 2111573  MEMA FACM
    7367    5150  EXCT AC0
    7370    5204  STL
    7371    5021  RASH 1
    7372 2405573  ACCM FACM
    7373 2111574  MEMA FACML
    7374    5021  RASH 1
    7375    5202  TLAC
    7376 2405574  ACCM FACML
    7377 2135572  MPOMA FACE
    7400 1001364  JMP @RSHFAC

/RIGHT SHIFT 3 WORD FAR
/COMPENSATES EXPONENT

    7401       0  RSHFAR, 0
    7402    5210  CLL
    7403 2111576  MEMA FARM
    7404    5150  EXCT AC0
    7405    5204  STL
    7406    5021  RASH 1
    7407 2405576  ACCM FARM
    7410 2111577  MEMA FARML
    7411    5021  RASH 1
    7412    5202  TLAC
    7413 2405577  ACCM FARML
    7414 2135575  MPOMA FARE
    7415 1001401  JMP @RSHFAR

/FLOATING POINT MULTIPLY

    7416       0  FMULT, 0
    7417 2001344  JMS PREPAR
    7420 2111575  MEMA FARE
    7421 2525572  AMPM FACE
    7422 2111573  MEMA FACM
    7423 2405433  ACCM MPL1
    7424 2405442  ACCM MPL2
    7425 2111574  MEMA FACML
```

```
   7426 2405556  ACCM TEM1  /SAVE FACML
   7427 2111576  MEMA FARM
   7430 2405452  ACCM MPL3
   7431    4354  TACMQ
   7432  505320  MULT
   7433       0  MPL1, 0
   7434 2405573  ACCM FACM
   7435    4343  TMQAC
   7436 2405574  ACCM FACML
/COMPUTE LOW ORDER TERMS
   7437 2111577  MEMA FARML
   7440    4354  TACMQ
   7441  505320  MULT
   7442       0  MPL2, 0
   7443    5210  CLL
   7444 2515574  A+MAM FACML
   7445    5141  EXCT L
   7446 2125573  MPOM FACM
   7447 2111556  MEMA TEM1
   7450    4354  TACMQ
   7451  505320  MULT
   7452       0  MPL3, 0
   7453    5210  CLL
   7454 2515574  A+MAM FACML
   7455    5141  EXCT L
   7456 2125573  MPOM FACM
   7457 2001117  JMS NORCON
   7460 1001416  JMP @FMULT

/FLOATING POINT DIVIDE

/(FACM+FACML)/(FARM+FARML) IS APPROXIMATELY = TO
/(FACM+FACML)/FARM-((FACM+FACML)/FARM)*FARML/FARM
/ERROR ALWAYS INVISIBLE AFTER ROUNDING

   7461       0  FDIV, 0
   7462 2001344  JMS PREPAR
   7463 2001364  JMS RSHFAC  /PREVENT OVERFLOWS
   7464 2001364  JMS RSHFAC
   7465 2111575  MEMA FARE
   7466 2325572  M-AM FACE
   7467 2111576  MEMA FARM
   7470 2405502  ACCM DIV1
   7471 2405511  ACCM DIV2
   7472 2407523  ACCMZ DIV3
   7473    1476  JMP FDIV2
   7474 2025555  ONEM ERRF
   7475 1001461  JMP @FDIV

   7476 2111574  FDIV2, MEMA FACML
   7477    4354  TACMQ
   7500 2111573  MEMA FACM
/COMPUT (FACM+FACML)/FARM
```

```
   7501  465300  DIVD
   7502       0  DIV1, 0
   7503 2405556  ACCM TEM1
   7504    4343  TMQAC
   7505 2405573  ACCM FACM
   7506   44354  ZRAM
   7507 2111556  MEMA TEM1
   7510  465300  DIVD
   7511       0  DIV2, 0
   7512    4343  TMQAC
   7513 2405574  ACCM FACML
/COMPUTE CORRECTION TERM
   7514 2111573  MEMA FACM
   7515 2405521  ACCM FDIV1
   7516 2111577  MEMA FARML
   7517    4354  TACMQ
   7520  505320  MULT
   7521       0  FDIV1, 0
   7522  465300  DIVD
   7523       0  DIV3, 0
   7524    4343  TMQAC
   7525    5001  LASH 1
   7526    5210  CLL
   7527 2335574  M-AAM FACML
   7530    5101  SKIP L
   7531 2705573  MMOM FACM
   7532 2001117  JMS NORCON
   7533 1001461  JMP @FDIV

/FIXED POINT TO FLOATING POINT CONVERSION
/INPUT AND OUTPUT IN FAC
/BINARY POINT BETWEEN FACM AND FACML

   7534       0  FLOAT, 0
   7535  110023  MEMA (23
   7536 2405572  ACCM FACE
   7537 2001106  JMS FNOR3
   7540 1001534  JMP @FLOAT

/REVERSES EFFECT OF FLOAT
/MAY BE USED TO GET INTEGER PART OR FRACTIONAL PART OF FLOATING NUMBERS.
/EXITS WITH FRACTIONAL PART IN ACC AND FACML
/AND INTEGER PART IN FACM
/"FIXABLE" NUMBERS EXIT WITH 0 FACE. FACE CAN BE USED AS ERROR FLAG.

   7541       0  FIX, 0
   7542 2001206  JMS EXFAC
   7543  110023  MEMA (23
   7544 2335572  M-AAM FACE
   7545    5144  FIX2, EXCT AC19
   7546    1551  JMP FIX1
   7547 2111574  MEMA FACML
   7550 1001541  JMP @FIX
```

```
7551 2001364  FIX1, JMS RSHFAC
7552    1545  JMP FIX2

7553       0  0  /SPARE

7554       0  SIGNF, 0  /SIGN FLAG FOR NORCON
7555       0  ERRF, 0  /ERROR FLAG
7556       0  TEM1, 0  /GENERAL PURPOSE GARBAGE STORAGE

/SHARED VARIABLES. USED BY FLIP, FLOP AND EXTENDED FUNCTIONS.

7557       0  DEXP, 0
7560       0  MANLZS, 0
7561       0  TEMP, 0
7562       0  TEM2, 0
7563       0  CHRCNT, 0
7564       0  DDIGIT, 0
7565       0  CNTR, 0
7566       0  PERSW, 0

/TEMPORARY FLOATING STORAGE

7567       0  TEMPEX, 0
7570       0  TEMPM, 0
7571       0  TEMPML, 0

/FLOATING ACCUMULATOR

7572       0  FACE, 0
7573       0  FACM, 0
7574       0  FACML, 0

/FLOATING ARGUMENT

7575       0  FARE, 0
7576       0  FARM, 0
7577       0  FARML, 0
```

VII. LISTING OF EXTENDED FUNCTIONS

```
/FLOATING POINT PACKAGE 1971
/EXTENDED FUNCTIONS, PART 2 OF 2

/NIC-80/S-7118-L
/COPYRIGHT 1971, NICOLET INSTRUMENT CORPORATION, MADISON, WIS.

/SUBROUTINE NAMES

FACFAR=2001026  /=JMS FACFAR
FACTEM=2001034  /=JMS FACTEM
EXFAC=2001206  /ETC
CONFAC=2001226
TEMFAC=2001042
GETAC=2001062
GETAR=2001050
PUTAC=2001074
FADD=2001245
FSUB=2001314
FNEG=2001320
FMULT=2001416
FDIV=2001461
RSHFAC=2001364
LSHFAC=2001170
N3FAC=2001236
FNOR3=2001106
FLOAT=2001534

*6000

  6000    6000  PAGE, PAGE
/MUST BE AT BEGINNING OF PAGE IF PROGRAM IS RELOCATED.

/FLOATING POINT SINE
/ARGUMENT IN RADIANS/(PI/2)

  6001       0  FSIN, 0
  6002 2001206  EXFAC  /SEPARATE FRACTION AND INTEGER
  6003 2001364  RSHFAC
  6004 2713572  SIN1, MMOAZ FACE
  6005    5144  EXCT AC19
  6006      11  JMP SIN2
  6007 2001170  LSHFAC
  6010       4  JMP SIN1

  6011 2111573  SIN2, MEMA FACM  /. BETWEEN 17 AND 18
  6012    5201  TACL
  6013    5001  LASH 1
  6014    5021  RASH 1  /SIGN FAC
  6015 2405573  ACCM FACM
  6016    5141  EXCT L
  6017  210000  ACPA
  6020    5144  EXCT AC19  /NEGATE IF QUADRANT -2,-1,2,3,6,7---
```

```
   6021 2001236  N3FAC
   6022 2001106  FNOR3
/BEGIN SERIES APPROXIMATION
   6023  110356  MEMA (K1
   6024 2404100  ACCM KPOINT
   6025  110005  MEMA (5
   6026 2000042  JMS POX
   6027 1000001  JMP @FSIN

   6030    2000  ONE, 2000
   6031 1000000  KP5, 1000000

/CONSTANTS FOR EXPONENTIATION

   6032   10306  AX, 10306 //9.95459578
   6033 1175060  1175060
   6034 3771776  BX, 3771776  /.03465735903
   6035 1067646  1067646
   6036   24320  CX, 24320  /-617.97226053
   6037 2626016  2626016
   6040   16105  DX, 16105  /87.417497202
   6041 1273256  1273256

/SERIES APPROXIMATOR
/COMPUTE SUM OF K(2I+1)X↑(2I+1),I=0,1,2,3,4,N
/N FOUND IN CNTR

   6042       0  POX, 0
   6043 2405563  ACCM CNTR
   6044 2110000  MEMA PAGE
   6045 2504100  A+MM KPOINT  /SET UP CONSTANT POINTER FOR KMULT
   6046 2001074  PUTAC
   6047    7557  ARG
   6050 2001034  FACTEM
   6051 2000352  JMS FSQAR
   6052 2001074  PUTAC  /ARGS=FAC**2
   6053    7561  ARGS
   6054 2001042  TEMFAC
   6055 2000076  JMS KMULT
   6056 2001034  POX1, FACTEM  /FINISH FIRST ITERATION
   6057 2707563  MMOMZ CNTR  /DONE?
   6060  162000  ZERZ
   6061 1000042  JMP @POX
   6062 2001062  GETAC  /PREPARE NEXT ITERATION
   6063    7561  ARGS
   6064 2001050  GETAR
   6065    7557  ARG
   6066 2001416  FMULT
   6067 2001074  PUTAC  /ARG=ARG*ARGS
   6070    7557  ARG
   6071 2000076  JMS KMULT
   6072 2001026  FACFAR
   6073 2001042  TEMFAC
```

```
   6Ø74 2ØØ1245  FADD
   6Ø75      56  JMP POX1

/MULTIPLY BY SUCCESIVE CONSTANTS

   6Ø76       Ø  KMULT, Ø
   6Ø77 2ØØ1Ø5Ø  GETAR
   61ØØ       Ø  KPOINT, Ø
   61Ø1 2ØØ1416  FMULT
   61Ø2 21241ØØ  MPOM KPOINT  /MOVE DOWN CONSTANT LIST FOR NEXT CALL
   61Ø3 21241ØØ  MPOM KPOINT
   61Ø4 1ØØØØ76  JMP @KMULT

   61Ø5       Ø  Ø  /SPARES
   61Ø6       Ø  Ø
   61Ø7       Ø  Ø
   611Ø       Ø  Ø
   6111       Ø  Ø
   6112       Ø  Ø

/FLOATING POINT COSINE
/ARGUMENT IN RADIANS/(PI/2)

   6113       Ø  FCOS, Ø
   6114 2ØØ1Ø5Ø  GETAR
   6115    6Ø3Ø  ONE
   6116 2ØØ1245  FADD
   6117 2ØØØØØ1  JMS FSIN
   612Ø 1ØØØ113  JMP @FCOS

/FLOATING POINT ARCTANGENT
/OUTPUT IN RADIANS/(PI/2)

   6121       Ø  FARCTN, Ø
   6122 2111572  MEMA FACE
   6123    5Ø32  RASH 12
   6124  4Ø5164  EXCT ZAC AC19  /FAC=>1?
   6125     13Ø  JMP FARC2
   6126 2ØØØ17Ø  JMS FRIP  /YES
   6127 2167565  ZERMZ OFLAG
   613Ø 2Ø25565  FARC2, ONEM OFLAG  /REMEMBER TO SUBTRACT FROM 1
   6131  11Ø147  MEMA (AK1
   6132 24Ø41ØØ  ACCM KPOINT
   6133  11ØØ1Ø  MEMA (1Ø
   6134 2ØØØØ42  JMS POX
   6135 21Ø3565  MEMZ OFLAG  /WAS FAC=<1?
   6136 1ØØØ121  JMP @FARCTN  /NO
   6137 2ØØ1Ø26  FACFAR
   614Ø 2ØØ1Ø62  GETAC
   6141    6Ø3Ø  ONE
   6142 2111576  MEMA FARM
   6143    5144  EXCT AC19
   6144 2ØØ132Ø  FNEG
```

```
   6145 2001314  FSUB
   6146 1000121  JMP @FARCTN

/CONSTANTS FOR ARCTAN

   6147    1612  AK1, 1612  /.636619347
   6150 1213713  1213713
   6151 3775715  AK3, 3775715  /-.212184453
   6152 2232710  2232710
   6153 3775622  AK5, 3775622  /.126983591
   6154 1010077  1010077
   6155 3773073  AK7, 3773073  /-.088544474
   6156 2452511  2452511
   6157 3770213  AK9, 3770213  /.061382906
   6160 1755545  1755545
   6161 3770702  AK11, 3770702  /-.035593338
   6162 2670655  2670655
   6163 3765350  AK13, 3765350  /.013917289
   6164 1620052  1620052
   6165 3760636  AK15, 3760636  /-.002580893
   6166 2533336  2533336

   6167       0  0  /SPARE

/FLOATING RECIPROCAL

   6170       0  FRIP, 0
   6171 2001026  FACFAR
   6172 2001062  GETAC
   6173    6030  ONE
   6174 2001461  FDIV
   6175 1000170  JMP @FRIP

/FLOATING SQUARE ROOT. NEWTONS METHOD USED.
/NEW GUESS=((OLD GUESS+ARG)/OLD GUESS)/2

   6176       0  FSQRT, 0
   6177 2111573  MEMA FACM
   6200  405160  EXCT ZAC
   6201 1000176  JMP @FSQRT  /IGNOR 0 FAC
   6202    5104  SKIP AC19  /-FAC
   6203     206  JMP FSQ1
   6204 2001320  FNEG  /YES, TAKE ABSOLUTE VALUE
   6205 2025555  ONEM ERRF  /SET ERROR LAG
   6206 2001034  FSQ1, FACTEM
   6207 2111572  MEMA FACE
   6210    5021  RASH 1  /VERY CRUDE GUESS
   6211 2405572  ACCM FACE
   6212  110005  MEMA (5
   6213 2405563  ACCM CNTR
   6214 2001026  FSQ2, FACFAR  /REFINE GUESS LOOP
   6215 2001074  PUTAC
   6216    7557  ARG
```

```
   6217 2001042  TEMFAC
   6220 2001461  FDIV
   6221 2001050  GETAR
   6222    7557  ARG
   6223 2001245  FADD
   6224  131777  MPOA (1777
   6225 2325572  M-AM FACE
   6226 2707563  MMOMZ CNTR
   6227     214  JMP FSQ2
   6230 1000176  JMP @FSQRT  /EXIT AFTER 5 ITERATIONS

/FLOATING POINT BASE 2 LOG

   6231       0  FLOGB2, 0
   6232 2111573  MEMA FACM
   6233  405124  SKIP AC19 ZAC  /0 OR -FAC IS A NO-NO
   6234     237  JMP FLOG1
   6235 2025555  ONEM ERRF
   6236 1000231  JMP @FLOGB2
/REDUCE ARG.
   6237 2111572  FLOG1, MEMA FACE
   6240    5032  RASH 12
   6241  405124  SKIP AC19 ZAC  /FAC<1?
   6242     245  JMP FLOG2
   6243 2000170  JMS FRIP  /LOG(1/X)=-LOG(X)
   6244 2167564  ZERMZ FLAG
   6245 2025564  FLOG2, ONEM FLAG
   6246 2001206  EXFAC
   6247 2405565  ACCM OFLAG  /SAVE EXP.
   6250 2025572  ONEM FACE  /SET EXP TO 1
   6251 2001226  CONFAC
   6252 2001034  FACTEM
/Z=(X-2↑.5)/(X+2↑.5)
   6253 2001050  GETAR
   6254    6342  SQRT2
   6255 2001245  FADD
   6256 2001074  PUTAC
   6257    7557  ARG
   6260 2001050  GETAR
   6261    6342  SQRT2
   6262 2001042  TEMFAC
   6263 2001314  FSUB
   6264 2001050  GETAR
   6265    7557  ARG
   6266 2001461  FDIV
/COMPUTE SERIES APPROXIMATION, Z IS ARG.
   6267  110344  MEMA (KLOG1
   6270 2404100  ACCM KPOINT
   6271  110003  MEMA (3
   6272 2000042  JMS POX
   6273 2001026  FACFAR
   6274  635212  635212  /SET AC=2000000
   6275 2405574  ACCM FACML  /ADD .5
```

```
    6276 2111565  MEMA OFLAG  /RETRIEVE INTEGER
    6277 2545573  AMOM FACM  /SUBTRACT 1
    6300 2001534  FLOAT
    6301 2001245  FADD
    6302 2703564  MMOZ FLAG  /NEGATE?
    6303 2001320  FNEG
    6304 1000231  JMP @FLOGB2

    6305       0  0  /SPARES
    6306       0  0
    6307       0  0
    6310       0  0
    6311       0  0
    6312       0  0
    6313       0  0
    6314       0  0
    6315       0  0
    6316       0  0
    6317       0  0
    6320       0  0
    6321       0  0

/FLOATING POINT BASE TEN LOG
/LOG(X)=LOGBASE2(X)*LOG(2)

    6322       0  FLOG, 0
    6323 2000231  JMS FLOGB2
    6324 2001050  GETAR
    6325    6336  KLB10
    6326 2001416  FMULT
    6327 1000322  JMP @FLOG

/FLOATING POINT BASE E LOG
/LN(X)=LOGBASE2(X)*LN(2)

    6330       0  FLN, 0
    6331 2000231  JMS FLOGB2
    6332 2001050  GETAR
    6333    6340  KLBE
    6334 2001416  FMULT
    6335 1000330  JMP @FLN

    6336 3777516  KLB10, 3777516  /.30102999267
    6337 1150404  1150404
    6340    1376  KLBE, 1376  /.6931471806
    6341 1305620  1305620
    6342    3145  SQRT2, 3145  /1.41421356237
    6343 1324047  1324047

/CONSTANTS FOR SERIES APPROXIMATION OF LOG

    6344    4010  KLOG1, 4010  /2.8853913
    6345 1342522  1342522
```

```
   6346     1016  KLOG3, 1016  /.96147063
   6347 1730427  1730427
   6350      506  KLOG5, 506  /.59897865
   6351 1145265  1145265

/FLOATING POINT SQUARE

   6352        0  FSQAR, 0
   6353 2001026  FACFAR
   6354 2001416  FMULT
   6355 1000352  JMP @FSQAR

/CONSTANTS FOR SINE

   6356     3522  K1, 3522  /1.570796318
   6357 1444176  1444176
   6360     1751  K3, 1751  /-.645963711
   6361 2552420  2552420
   6362 3773367  K5, 3773367  /.07968967928
   6363 1214642  1214642
   6364 3763633  K7, 3763633  /-.00467376557
   6365 2633314  2633314
   6366 3751511  K9, 3751511  /.00015148419
   6367 1173275  1173275

/BASE 10 EXPONENTIATION
/10↑X=2↑X/LOG(2)

   6370        0  FEXP, 0
   6371 2110370  MEMA FEXP
   6372 2404376  ACCM FEXPN
   6373 2001050  GETAR
   6374     6336  KLB10
   6375      401  JMP FEXP2

/BASE E EXPONENTIATION
/E↑X=2↑X/LN(2)

   6376        0  FEXPN, 0
   6377 2001050  GETAR
   6400     6340  KLBE
/EXPONENTIATE BASE 2
   6401 2001461  FEXP2, FDIV
   6402 2145564  MONM FLAG
   6403 2111573  MEMA FACM
   6404     5104  SKIP AC19  /REMEMBER SIGN
   6405 2167564  ZERMZ FLAG
   6406 2001320  FNEG  /[FAC]
/SEPARATE INTEGER AND FRACTIONAL PART
   6407 2165565  ZERM OFLAG  /OFLAG HOLDS INTEGER PART
   6410 2001206  EXFAC
   6411 2113572  EXP1, MEMAZ FACE
   6412     5144  EXCT AC19
```

```
   6413      424  JMP EXP2
   6414 2001170  LSHFAC
   6415 2111573  MEMA FACM
   6416     5201  TACL
   6417 2111565  MEMA OFLAG
   6420     5202  TLAC
   6421     5041  LLSH 1
   6422 2405565  ACCM OFLAG
   6423      411  JMP EXP1

   6424 2111573  EXP2, MEMA FACM  /RESTORE SIGN
   6425     5212  CLL TLAC
   6426 2405573  ACCM FACM
   6427 2001106  FNOR3
/2↑F=1+2*F/((A-F+B*F↑D-C)/(D+F↑2))
   6430 2001074  PUTAC
   6431     7557  ARG
   6432 2000352  JMS FSQAR
   6433 2001074  PUTAC
   6434     7561  ARGS
/SOLVE DNOMINATOR
   6435 2001050  GETAR
   6436     6040  DX
   6437 2001245  FADD
   6440 2001026  FACFAR
   6441 2001062  GETAC
   6442     6036  CX
   6443 2001461  FDIV
   6444 2001050  GETAR
   6445     6032  AX
   6446 2001245  FADD
   6447 2001050  GETAR
   6450     7557  ARG
   6451 2001314  FSUB
   6452 2001034  FACTEM
   6453 2001050  GETAR
   6454     7561  ARGS
   6455 2001062  GETAC
   6456     6034  BX
   6457 2001416  FMULT
   6460 2001026  FACFAR
   6461 2001042  TEMFAC
   6462 2001245  FADD
/SOLVE NUMERATOR, DIVIDE, AND ADD 1
   6463 2001026  FACFAR
   6464 2001062  GETAC
   6465     7557  ARG
   6466 2001461  FDIV
   6467   131777  MPOA (1777  /MULTIPLY BY 2
   6470 2505572  A+MM FACE
   6471 2001050  GETAR
   6472     6030  ONE
   6473 2001245  FADD
```

```
/COMBINE FRACTIONAL AND INTEGER PARTS
   6474 2111565  MEMA OFLAG
   6475    5012  LASH 12
   6476 2515572  A+MAM FACE
   6477    5144  EXCT AC19  /FACE OVEFLOW?
   6500 2025555  ONEM ERRF
   6501 2103564  MEMZ FLAG
   6502 2000170  JMS FRIP  /2**-X=1/2**X
   6503 1000376  JMP @FEXPN

*7555

   7555       0  ERRF, 0  /ERROR FLAG.  USED BY EXTENDED FUNCTIONS
/AND BASIC ARITH. SET TO 1 IF ERROR OCCURS. IS NEVER CLEARED.

*7557  /FOLLOWING LOCATIONS ARE SHARED WITH FLIP AND FLOP

   7557       0  ARG, 0
   7560       0  0
   7561       0  ARGS, 0
   7562       0  0

   7563       0  CNTR, 0
   7564       0  FLAG, 0  /GENERAL PURPOSE FLAGS
   7565       0  OFLAG, 0

*7572  /FLOATING AC AND ARGUMENT

   7572       0  FACE, 0
   7573       0  FACM, 0
   7574       0  FACML, 0
   7575       0  FARE, 0
   7576       0  FARM, 0
   7577       0  FARML, 0
```