

A Floating Point Package for the HPC

National Semiconductor
Application Note 486
Ashok Krishnamurthy
April 1987



INTRODUCTION

This report describes the implementation of a Single Precision Floating Point Arithmetic package for the National Semiconductor HPC microcontroller. The package is based upon the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985). However, the package is not a conforming implementation of the standard. The differences between the HPC implementation and the standard are described later in this report.

The following single precision (SP) operations have been implemented in the package.

- (1) **FADD.** Addition of two SP floating point (FLP) numbers.
- (2) **FSUB.** Subtraction of two SP FLP numbers.
- (3) **FMULT.** Multiplication of two SP FLP numbers.
- (4) **FDIV.** Division of two SP FLP numbers.
- (5) **ATOF.** Convert an ASCII string representing a decimal FLP number to a binary SP FLP number.
- (6) **FTOA.** Convert a binary SP FLP number to a decimal FLP number and output the decimal FLP number as an ASCII string.

The report is organized as follows. The next section discusses the representation of FLP numbers. Then, the differences between the HPC implementation and the IEEE/ANSI standard are described. This is followed by a description of the algorithms used in the computations. Appendix A is a User's Manual for the package, Appendix B describes the test data for the package and Appendix C is a listing of the code.

Note that this report assumes that the reader is familiar with the IEEE/ANSI Binary Floating-Point Standard. Please refer to this document for an explanation of the terms used here.

REPRESENTATION OF FLOATING POINT NUMBERS

The specification of a binary floating point number involves two parts: a mantissa and an exponent. The mantissa is a signed fixed point number and the exponent is a signed integer. The IEEE/ANSI standard specifies that a SP FLP number shall be represented in 32 bits as shown in *Figure 1*.

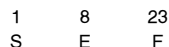


FIGURE 1

The significance of each of these fields is as follows:

1. **S**—this 1-bit field is the sign of the mantissa. $S = 0$ means that the number is positive, while $S = 1$ means that it is negative.
2. **E**—this is the 8-bit exponent field. The exponent is represented as a biased value with a bias of 127-decimal.
3. **F**—this is the 23-bit mantissa field. For normalized FLP numbers (see below), a MSB of 1 is assumed and not represented. Thus, for normalized numbers, the value of the mantissa is $1.F$. This provides an effective precision of 24 bits for the mantissa.

Normalized FLP number: A binary FLP number is said to be normalized if the value of the MSB of the mantissa is 1. Normalization is important and useful because it provides maximum precision in the representation of the number. If we deal with normalized numbers only (as the HPC imple-

mentation does) then since the MSB of the mantissa is always 1, it need not be explicitly represented. This is as specified in the IEEE/ANSI standard.

Given the values of S , E and F , the value of the SP FLP number is obtained as follows.

If $0 < E < 255$, then the FLP number is $(-1)^S \times 1.F \times 2^{(E-127)}$.

If $E = 0$, then the value of the FLP number is 0.

If $E = 255$, then the FLP number is not a valid number (NaN).

The above format for binary SP FLP numbers provides for the representation of numbers in the range -3.4×10^{38} to -1.75×10^{-38} , 0, and 1.75×10^{-38} to 3.4×10^{38} . The accuracy is between 7 and 8 decimal digits.

DIFFERENCES BETWEEN THE IMPLEMENTATION AND THE IEEE/ANSI STANDARD

The IEEE/ANSI standard specifies a comprehensive list of operations and representations for FLP numbers. Since an implementation that fully conforms to this standard would lead to an excessive amount of overhead, a number of the features in the standard were dropped. This section describes the differences between the implemented package and the standard.

1. Omission of -0 . The IEEE/ANSI standard requires that both $+$ and $-$ zero be represented, and arithmetic carried out using both. The implementation does not represent -0 . Only $+0$ is represented and arithmetic is carried out with $+0$ only.
2. Omission of Infinity Arithmetic. The IEEE/ANSI standard provides for the representation of plus and minus Infinity, and requires that valid arithmetic operations be carried out on Infinity. The HPC implementation does not support this.
3. Omission of Quiet NaN. The IEEE/ANSI standard provides for both quiet and signalling NaNs. The HPC implementation provides for signalling NaNs only. A signalling NaN can be produced as the result of overflow during an arithmetic operation. If the NaN is passed as input to further floating point routines, then these routines will produce another NaN as output. The routines will also set the Invalid Operation flag, and call the user floating point error trap routine at address FPTRAP.
4. Omission of denormalized numbers. Denormalized numbers are FLP numbers with a biased exponent, E of zero and a non zero mantissa F . Such denormalized numbers are useful in providing gradual underflow to zero. Denormalized numbers are not represented or used in the HPC implementation. Instead, if the result of a computation cannot be represented as a normalized number within the allowable exponent range, then an underflow is signaled, the result is set to zero, and the user floating point error trap routine at address FPTRAP is called.
5. Omission of the Inexact Result exception. The IEEE/ANSI standard requires that an Inexact Result exception be signaled when the rounded result of an operation is not exact, or it overflows without an overflow trap. This feature is not provided in the HPC implementation.

6. Biased Rounding to Nearest. The IEEE/ANSI standard requires that rounding to nearest be provided as the default rounding mode. Further, the rounding is required to be unbiased. The HPC implementation provides biased rounding to nearest only. An example will help clarify this.

Suppose the result of an operation is .b1b2b3XXX and needs to be rounded to 3 binary digits. Then if XXX is 0YY, the round to nearest result is .b1b2b3. If XXX is 1YY, with at least one of the Y's being 1, then the result is .b1b2b3 + 0.001. Finally if XXX is 100, it is a tie situation. In such a case, the IEEE/ANSI standard requires that the rounded result be such that its LSB is 0. The HPC implementation, on the other hand, will round the result in such a case to .b1b2b3 + 0.001.

DESCRIPTION OF ALGORITHMS

1. **General Considerations.** The HPC implementation of the SP floating point package consists of a series of subroutines. The subroutines have been designed to be compatible with the CCHPC C Cross Compiler. They have, however, not been tested with the CCHPC Cross Compiler.

The Arithmetic subroutines that compute F1 op F2 (where op is +, -, * or /) expect that F1 and F2 are input in the IEEE format. Each of F1 and F2 consists of two 16-bit words organized as follows.

Fn-HI: S EXP 7 MS bits of F

Fn-LO: 16 LS bits of F

In the above, S is the sign of the mantissa, EXP is the biased exponent, and F is the mantissa.

On input it is assumed that F1-HI is in register K, F1-LO is in the accumulator A, and F2-HI and F2-LO are on the stack just below the return address i.e., F2-HI is at W(SP-4) and F2-LO is at W(SP-6). The result, C, is also returned in IEEE format with C-HI in register K and C-LO in the accumulator A.

The two Format Conversion routines, ATOF and FTOA expect that on entry, register B contains the address of the start of the ASCII byte string representing the decimal FLP number. ATOF reads the byte string starting from this address. Note that the string must be terminated with a null byte. The binary floating point number is returned in registers K and A. FTOA, on the other hand, writes the decimal FLP string starting from the address in register B on entry. A terminating null byte is also output. Also, FTOA expects that the binary FLP number to be converted is in registers K and A on entry.

Most of the storage required by the subroutines is obtained from the stack. Two additional words of storage in the base page are also used. The first is W(0), and is referenced in the subroutines as W(TMP1). The second word of storage can be anywhere in the base page and is used to store the sticky flags used to signal floating point exceptions. This is referenced in the subroutines as W(FPERWD). Thus any user program that uses the floating point package needs to have the symbols TMP1 and FPERWD defined appropriately.

2. **Exception Handling.** The following types of exception can occur during the course of a computation.

- (i) Invalid Operand. This exception occurs if one of the input operands is a NaN.
- (ii) Exponent Overflow. This occurs if the result of a computation is such that its exponent has a biased value of 255 or more.

(iii) Exponent Underflow. This occurs if the result of a computation is such that its exponent is 0 or less.

(iv) Divide-by-zero. This exception occurs if the FDIV routine is called with F2 being zero.

The package signals exceptions in two ways. First a word at address FPERWD is maintained that records the history of these exception conditions. Bits 0-3 of this word are used for this purpose.

Bit 0—Set on Exponent Overflow.

Bit 1—Set on Exponent Underflow.

Bit 2—Set on Illegal Operand.

Bit 3—Set on Divide-by-zero.

These bits are never cleared by the floating point package, and can be examined by the user software to determine the exception conditions that occurred during the course of a computation. It is the responsibility of the user software to initialize this word before calling any of the floating point routines.

The second method that the package uses to signal exceptions is to call a user floating point exception handler subroutine whenever an exception occurs. The corresponding exception bit in FPERWD is set before calling the handler. The starting address of the handler should be defined by the symbol FPTRAP.

3. **Unpacked Floating Point Format.** The IEEE/ANSI standard floating point format described earlier is very cumbersome to deal with during computation. This is primarily because of the splitting of the mantissa between the two words. The subroutines in the package unpack the input FLP numbers into an internal representation, do the computations using this representation, and finally pack the result into the IEEE format before return to the calling program. The unpacking is done by the subroutine FUNPAK and the packing by the subroutine FPAK. The unpacked format consists of 3 words and is organized as follows.

Fn-EXP.Fn-SIGN	8 bits biased	sign (extended to
	exponent	8 bits)
Fn-HI	MS 16 bits of mantissa	
	(implicit 1 is present as MSB)	
Fn-LO	LS 8 bits of	Eight
	mantissa	Zeros

Since all computations are carried out in this format, note that the result is actually known to 32 bits. This 32-bit mantissa is rounded to 24 bits before being packed to the IEEE format.

4. **Algorithm Description.** All the arithmetic algorithms first check for the easy cases when either F1 or F2 is zero or a NaN. The result in these cases is immediately available. The description of the algorithms below is for those cases when neither F1 nor F2 is zero or a NaN. Also, in order to keep the algorithm description simple, the check for underflow/overflow at the various stages is not shown. The documentation in the program, the descriptions given below, and the theory as described in the references should allow these programs to be easily maintained.

(i) FADD.

The processing steps are as follows:

1. Compare F1-EXP and F2-EXP. Let the difference be D. Shift right the mantissa (Fn-HI.Fn-LO, n = 1 or 2) of the FLP number with the smaller exponent D times. Let the numbers after this step be F1-EXP.F1-SIGN, F1-HI, F1-LO and F2-EXP.F2-SIGN,

F2-HI and F2-LO. This step equalizes the two exponents.

2. Take the XOR of F1-SIGN and F2-SIGN. If this is 0, then go to step 4, else go to step 3.

3. Do a true subtract of F2-LO from F1-LO. (A true subtract is when the SUBC instruction is preceded by a SET C instruction.) Then do a 1's complement subtract of F2-HI from F1-HI. If the last subtract resulted in C = 1, then go to step 3.2, else go to step 3.1.

3.1. Get here means that F2 is larger than F1, and the computed result is negative. Take the 2's complement of the result to make it positive. Set the sign of the result to be the sign of F2. Go to step 3.3.

3.2. Get here means F1 is larger than F2, and the result of the mantissa subtract is positive. Set the sign of the result to be the sign of F1. Go to step 3.3.

3.3. The result after a subtract need not be normalized. Shift left the result mantissa until its MSB is 1. Decrement the exponent of the result by 1 for each such left shift. Go to step 5.

4. Add F2-LO to F1-LO. Next add with any carry from the previous add, F2-HI to F1-HI. If this last add results in C = 1, then go to step 4.1, else go to step 5.

4.1. Rotate Right with carry C-HI. Next load C-LO in and rotate it right with carry. Increase the exponent of the result, C by 1. Go to step 5.

5. Round the result. Go to step 6.

6. Pack the result and return.

(ii) FSUB.

The processing steps are as follows:

1. Copy F2 to the stack and change its sign. Go to step 2.

2. Call FADD.

3. Remove the copy of -F2 from the stack and return.

(iii) FMULT.

The processing steps are as follows.

1. Add F1-EXP and F2-EXP to get C1-EXP. Subtract from C1-EXP 127-decimal which is the IEEE bias, to get C-EXP. Go to step 2.

2. Take the XOR of F1-SIGN and F2-SIGN to get C-SIGN. Go to step 3.

3. Compute F1-HI*F2-HI. Let the upper half of the product be C1-HI and the lower half C1-LO. Go to step 4.

4. Compute F1-HI*F2-LO. Let the upper half of this product be C2-HI. Add C2-HI to C1-LO to give C11-LO. If this last add results in C = 1, then increment C1-HI. Go to step 5.

5. Compute F1-LO*F2-HI. Let the upper half of this product be C3-HI. Add C3-HI to C11-LO to get C12-LO. If this last add results in C = 1, then increment C1-HI. Go to step 6.

6. Mantissa normalization. If the MSB of C1-HI is 1, then increment C-EXP, else shift left C1-HI.C12-LO. Go to step 7.

7. Round C1-HI.C12-LO to get C-HI.C-LO. Go to step 8.

8. Pack C-EXP.C-SIGN, C-HI and C-LO and return as the answer.

(iv) FDIV.

The processing steps are as follows:

1. Compare F1-HI and F2-HI. If F2-HI is greater than F1-HI then go to Step 3, else go to step 2.

2. Shift right F1-HI.F1-LO. Increase F1-EXP by 1.

3. Subtract F2-EXP from F1-EXP. Add to the result 127-decimal to get C1-EXP. Go to step 4.

4. Take the XOR of F1-SIGN and F2-SIGN to get C-SIGN. Go to step 5.

5. Compute F1-HI*F2-LO. Let the result be M1-HI.M1-LO. Go to step 6.

6. Divide M1-HI.M1-LO by F2-HI. Let the quotient be M2-HI. Go to step 7.

7. Do a true subtract of M2-HI from F1-LO. Let the result be M3-LO. If C = 1 as a result of this subtract, then go to step 8, else decrement F1-HI and go to step 8.

8. Divide F1-HI.M3-LO by F2-HI. Let the quotient be C1-HI and the remainder R1. Go to step 9.

9. Divide R1 .0000 by F2-HI. Let the quotient be C1-LO. Go to step 10.

10. If the MSB of C1-HI is 1 then go to step 11, else shift left C1-HI.C1-LO, decrease C1-EXP by 1 and go to step 11.

11. Round C1-HI.C1-LO to get C-HI.C-LO. go to step 12.

12. Pack C1-EXP.C-SIGN, C-HI and C-LO and return as the result.

(v) ATOF.

The processing steps in this case are as follows.

1. Set M-SIGN, the mantissa sign to 0.

Set M10-EXP, the implicit decimal exponent to 0.

Set HI-INT to 0.

Set LO-INT to 0.

Go to step 2.

2. Get a character from the input string. Let the character be C.

If C is a '+', then go to the start of step 2.

If C is a '-', then set M-SIGN to FF and go to start of step 2.

If C is a '.', then go to step 5.

If C is none of the above, then go to step 3.

3. Subtract 30 from C to get its integer value. Let this be I. Check and see if (HI-INT.LO-INT)*10 + 9 can fit in 32 bits. If it can, then go to step 3.1, else go to step 3.2.

3.1. Multiply HI-INT.LO-INT by 10 and add I to the product. Store this sum back in HI-INT.LO-INT. Go to step 4.

3.2. Increase M10-EXP by 1 and go to step 4.

4. Get a character from the input string. Let the character be C.

If C is a '.', then go to step 5.

If C is a 'E', then go to step 7.

If C is the space character, then go to the start of step 4.

If C is none of the above, then go to step 3.

5. Get a character from the input string. Let the character be C.
If C is a 'E', then go to step 7.
If C is the space character, then go to the start of step 5.
If C is none of the above, then go to step 6.
 6. Subtract 30 from C to get its integer value. Let this be I. Check and see if (HI-INT.LO-INT)*10 + 9 can fit in 32 bits. If it can, then go to step 6.1, else go to step 5.
6.1. Multiply HI-INT.LO-INT by 10 and add I to the product. Store this sum back in HI-INT.LO-INT. Decrement M10-EXP by 1. Go to step 5.
 7. Set SEXP, the exponent sign to be 0. Go to step 8.
 8. Get a character from the input string. Let the character be C.
If C is a '+', then go to start of step 8.
If C is a '-', then set SEXP to be FF and go to the start of step 8.
If C is none of the above, then go to step 9.
 9. Set M20-EXP, the explicit decimal exponent to 0. Go to step 10.
 10. Subtract 30 from C to get its integer value. Let this be I. Multiply M20-EXP by 10 and add I to the product. Store this sum back in M20-EXP. Go to step 11.
 11. Get a character from the input string. Let this be C. If C is the null character, then go to step 12, else go to step 10.
 12. Add M10-EXP and M20-EXP (with the proper sign as determined by SEXP) to get the 10's exponent M-EXP. Save in M-EXP the magnitude of the sum and in SEXP the sign of the sum. Go to step 13.
 13. Check and see if HI-INT.LO-INT is 0. If it is, then set the resulting floating point number, C, to zero and return. If it is not then go to step 14.
 14. Normalize HI-INT.LO-INT by left shifts such that the MSB is 1. Let the number of left shifts needed to do this be L. Set B1-EXP to 32-decimal - L. Go to step 15.
 15. If SEXP is 0, then set P-HI.P-LO to the binary representation of 0.625, else set P-HI.P-LO to the binary representation of 0.8. Go to step 16.
 16. Multiply HI-INT.LO-INT by P-HI.P-LO M-EXP times. After each multiplication, normalize the partial product if needed by left shifting. Accumulate the number of left shifts needed in B2-EXP. Let the final product be C-HI.C-LO. Go to step 17.
 17. Subtract B2-EXP from B1-EXP. Let the result be B-EXP. Go to step 18.
 18. If SEXP is 0, then multiply M-EXP by 4, else multiply M-EXP by -3. Let the result be B3-EXP. Go to step 19.
 19. Add B-EXP and B3-EXP. Let the result be C1-EXP. Add 126 to C1-EXP to restore the IEEE bias, getting C-EXP. Go to step 20.
 20. Round C-HI.C-LO. Go to step 21.
 21. Pack C-EXP.M-SIGN, C-HI and C-LO and return.
- (vi) FTOA.
- The processing steps are as follows.

1. Unpack the input FLP number. Let the unpacked number be represented by C-EXP.C-SIGN, C-HI and C-LO. Go to step 2.
2. Subtract 126-decimal from C-EXP to remove the IEEE bias. Let the result be C1-EXP. Go to step 3.
3. Multiply C1-EXP by the binary representation of log(2). Let the product be U-HI.U-LO. Go to step 4.
4. Subtract 8 from U-HI.U-LO. Let the magnitude of the integer part of the result be V and its sign VSIGN. Go to step 5.
5. If VSIGN is 0, then set P-HI.P-LO to the binary representation of 0.8, else set P-HI.P-LO to the binary representation of 0.625. Go to step 6.
6. Multiply C-HI.C-LO by P-HI.P-LO V times. Normalize the partial product after each multiplication, if needed, by left shifting. Accumulate any left shifts needed in B1-EXP. Let the final product be HI-INT.LO-INT. Go to step 7.
7. Subtract B1-EXP from C1-EXP. Let the result be B2-EXP. Go to step 8.
8. If VSIGN is 0, then multiply V by -3, else multiply it by 4. Let the result be B3-EXP. Go to step 9.
9. Add B2-EXP and B3-EXP. Let the result be B4-EXP. Go to step 10.
10. If B4-EXP is more than 32-decimal, then increase V and go to step 6, else go to step 11.
11. If B4-EXP is less than 28-decimal, then decrease V and go to step 6, else go to step 12.
12. Subtract B4-EXP from 32. Let the result be B5-EXP. Go to step 13.
13. Shift HI-INT.LO-INT right B5-EXP number of times. Go to step 14.
14. Add 16-decimal to the address of the start of the decimal string. Output a null byte there. Go to step 15.
15. Divide V by 10-decimal. Let the quotient be Q and the remainder R. Add 30 to R and output it to the decimal string. Next add 30 to Q and output it to the decimal string. Go to step 16.
16. If VSIGN is 0, then output '+' to the output string, else output '-' to the output string. Go to step 17.
17. Output 'E' to the output string. Output '.' to the output string. Go to step 18.
18. Divide C-HI.C-LO by 10-decimal 10 times. Let the remainder in each division be R. Add 30 to each R and output it to the output string. Go to step 19.
19. If C-SIGN is 0, then output the space character to the output string, else output '-' to the output string. Then return to the calling program.

REFERENCES

1. ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, Aug. 12, 1985.
2. J.T. Coonen, "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic," IEEE Computer, Jan. 1980, pp. 68-79.
3. K. Hwang, *Computer Arithmetic*, John-Wiley and Sons, 1979.
4. M. M. Mano, *Computer System Design*, Prentice-Hall, 1980.

APPENDIX A

A USER'S MANUAL FOR THE HPC FLOATING POINT PACKAGE

The Single Precision Floating Point Package for the HPC implements the following functions.

ARITHMETIC FUNCTIONS

1. FADD—Add two floating point numbers.
2. FSUB—Subtract two floating point numbers.
3. FMULT—Multiply two floating point numbers.
4. FDIV—Divide two floating point numbers.

FORMAT CONVERSION FUNCTIONS

5. ATOF—Convert an ASCII string representing a decimal floating point number to a single precision floating point number.
6. FTOA—Convert a single precision floating point number to an ASCII string that represents the decimal floating point value of the number.

The entire package is in the form of a collection of subroutines and is contained in the following files.

1. FERR.MAC
2. FNACHK.MAC
3. FZCHK.MAC
4. FUNPAK.MAC
5. FPAK.MAC
6. FPTRAP.MAC
7. ROUND.MAC
8. BFMUL.MAC
9. ISIOK.MAC
10. MUL10.MAC
11. ATOF.MAC
12. FTOA.MAC
13. FADD.MAC
14. FMULT.MAC
15. FDIV.MAC

The first 7 files are general utility routines that are used by all the Arithmetic and Format Conversion subroutines. The next 3 files, BFMUL.MAC, ISIOK.MAC and MUL10.MAC are used only by the Format Conversion subroutines, ATOF and FTOA. Depending on the functions being used in the user program, only the necessary files need be included.

INTERFACE WITH USER PROGRAMS

1. All the Arithmetic routines expect the input to be in the IEEE Single Precision format. This format requires 2 words for the storage of each floating point number. If the required arithmetic operation is FlopF2, where op is +, −, * or /, then the routines expect that F1 is available in registers K and A on entry, with the high half in K. Also, the two words of F2 are expected to be on the stack. If SP is the stack pointer on entry into one of the Arithmetic function subroutines, then the high word of F2 should be at W(SP-4) and the low word at W(SP-6). The result of the Arithmetic operation is returned in IEEE format in registers K and A, with the high word in K.

2. The Format Conversion subroutine ATOF expects that on entry, B contains the address of the ASCII string representing the decimal floating point number. This string must be of the form

Siiii.fffffEsNND

where

S is an optional sign for the mantissa. Thus S can be '+', '−' or not present at all.

iiii is the optional integer part of the mantissa. If it is present, it can be of any length, must contain only the characters '0' through '9' and must not contain any embedded blanks.

. is the optional decimal point. It need not be present if the number has no fractional part.

fffff is the optional fractional part of the mantissa. fffff, if it is present must consist of a sequence of digits '0' through '9'. It can be of any length. Note that either iiiii, the integer part or .fffff the fractional part must be present.

E is the required exponent start symbol.

s is the optional sign of the exponent. If it is present, it must be '+' or '−'.

NN is the exponent and consists of at most two decimal digits. It is required to be present.

D is the null byte <00> and must be present to terminate the string.

The floating point number represented by the above string is returned by ATOF in IEEE format in registers K and A.

3. The format conversion routine FTOA expects the floating point number input to be in registers K and A in the IEEE format. Register B is expected to contain the starting address of a 17 byte portion of memory where the output string will be stored.
4. Three global symbols need to be defined in the user program before assembling the user program and any included floating point package files. These symbols are:
 - (i) TMP1 which must be set to 0. The package uses W(TMP1) for temporary storage.
 - (ii) FPERWD which must be set to an address in the base page. The package signals floating point exceptions using W(FPERWD). This is described below.
 - (iii) FPTRAP which must be set to the address of the start of a user floating point exception handler. Again this is described below.

FLOATING POINT EXCEPTS

The package maintains a history of floating point exceptions in the 4 least significant bits of the word W(FPERWD). The value of the symbol FPERWD should be defined by the user program, and should be an address in the base page. This word should also be cleared by the user program before calling any floating point routine. The word is never cleared by the floating point package, and the user program can examine this word to determine the type of exceptions that may have occurred during the course of a computation.

The following 4 types of error can occur in the course of a floating point computation.

1. Invalid Operand. This happens if one of the input numbers for an Arithmetic routine or the input for FTOA is not a valid floating point number. An invalid floating point number (or NaN) can be created either by an overflow in a previous computation step, or if the ASCII decimal floating point number input to ATOF is too large to be represented in the IEEE format. The result, if one of the inputs is a NaN is always set to a NaN.
2. Overflow. This happens if the result of a computation is too large to be represented within the exponent range available. Overflow can occur in any of the arithmetic routines or ATOF. On overflow, the result is set to a representation called NaN. An NaN is considered an illegal operand in all successive steps.
3. Underflow. This occurs if the result of a computation is too small to be represented with the precision and expo-

nent range available. On underflow, the result is set to zero.

4. Divide-by-zero. This error occurs if F2 is zero when computing $F1/F2$. The result is set to an NaN.

Each of the above errors results in a bit being set in W(FPERWD). This is done as follows:

Bit 0—Set on Overflow.

Bit 1—Set on Underflow.

Bit 2—Set on Illegal Operand.

Bit 3—Set on Divide-by-zero.

One further action is taken when a floating point exception occurs. After the result has been set to the appropriate value, and the corresponding bit in W(FPERWD) set, the package does a subroutine call to address FPTRAP. The user can provide any exception handler at this address. The file FPTRAP.MAC contains the simplest possible user exception handler. It does nothing, but merely returns back to the calling program.

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FLP

PAGE: 1

1			.TITLE FLP
2		LISTER:	
3	0071		.LIST 071
4	F000		. = 0F000
5	0002		FPERWD = W(2)
6	0000		TMP1 = W(0)

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FLP
THE FLP ROUTINES

PAGE: 2

7			.FORM 'THE FLP ROUTINES'
---	--	--	--------------------------

The code listed in this App Note is available on Dial-A-Helper.

Dial-A-Helper is a service provided by the Microcontroller Applications Group. The Dial-A-Helper system provides access to an automated information storage and retrieval system that may be accessed over standard dial-up telephone lines 24 hours a day. The system capabilities include a MESSAGE SECTION (electronic mail) for communicating to and from the Microcontroller Applications Group and a FILE SECTION mode that can be used to search out and retrieve application data about NSC Microcontrollers. The minimum system requirement is a dumb terminal, 300 or 1200 baud modem, and a telephone.

With a communications package and a PC, the code detailed in this App Note can be down loaded from the FILE SECTION to disk for later use. The Dial-A-Helper telephone lines are:

Modem (408) 739-1162
Voice (408) 721-5582

For Additional Information, Please Contact Factory

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FLP
FERR.MAC

PAGE: 3

```

8          .FORM 'FERR.MAC'
9          .INCLD FERR.MAC
1         ; EXCEPTION HANDLING.
2         ; DIVIDE BY ZERO.
3         DIVBYO:
4 F000 820802FA          OR FPERWD, 08      ; SET THE DIVIDE BY 0 BIT.
5 F004 00                CLR A
6 F005 B17F80            LD K, 07F80
7 F008 3093              JSR FPTRAP
8 F00R 3FCC              POP B
9 F00C 3FCE              POP X
10 F00E 3C               RET
11         ; ILLEGAL OPERAND - ONE OF F1 OR F2 IS A NAN.
12         FNAN:
13 F00F 820402FA          OR FPERWD, 04      ; SET THE ILLEGAL OPERAND BIT.
14 F013 00                CLR A
15 F014 B17F80            LD K, 07F80      ; RETURN NAN IN K AND A.
16 F017 3084              JSR FPTRAP      ; GO TO USER TRAP ROUTINE.
17 F019 3FCC              POP B
18 F01B 3FCE              POP X
19 F01D 3C               RET
20         ; EXPONENT UNDERFLOW.
21         UNDFL:
22 F01E 820202FA          OR FPERWD, 02      ; SET THE EXPONENT UNDERFLOW BIT.
23 F022 00                CLR A
24 F023 ACC8CA            LD K, A
25 F026 3075              JSR FPTRAP
26 F028 3FC4              POP SP
27 F02A 3FCC              POP B
28 F02C 3FCE              POP X
29 F02E 3C               RET
30         ; EXPONENT OVERFLOW.
31         OVRFL:
32 F02F 820102FA          OR FPERWD, 01      ; SET THE EXPONENT OVERFLOW BIT.
33 F033 00                CLR A
34 F034 B17F80            LD K, 07F80
35 F037 3064              JSR FPTRAP
36 F039 3FC4              POP SP
37 F03B 3FCC              POP B
38 F03D 3FCE              POP X
39 F03F 3C               RET
40         ;
41         .END

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FLP
FNACHK.MAC

PAGE: 4

```
10          .FORM 'FNACHK.MAC'
11          .INCLD FNACHK.MAC
12          .TITLE FNACHK
13          .LOCAL
14          ;
15          ; SUBROUTINE TO CHECK IF A SP FLOATING POINT NUMBER STORED IN THE
16          ; IEEE FLOATING POINT FORMAT IN REGS. K AND A IS NAN.
17          ;
18          ; RETURNS 0 IN C IF THE NUMBER IS NOT A NAN.
19          ; RETURNS 1 IN C IF THE NUMBER IS A NAN.
20          ;
21          ; PRESERVES REGS. K, A, X AND B. DESTROYS C.
22          ;
23          FNACHK:
24              X A, K
25              SHL A
26              IFGT A,0FEFF
27              JP $ISNAN
28              RRC A
29              RESET C
30              X A, K
31              RET
32          $ISNAN:
33              RRC A
34              SET C
35              X A, K
36              RET
37          ;
38          .END
```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FNACHK
FZCHK.MAC

PAGE: 5

```
12          .FORM 'FZCHK.MAC'
13          .INCLD FZCHK.MAC
14          .TITLE FZCHK
15          .LOCAL
16
17          ;
18          ; SUBROUTINE THAT CHECKS IF A SP FLOATING POINT NUMBER STORED
19          ; IN THE IEEE FORMAT IN REGS K AND A IS ZERO.
20          ;
21          ; RETURNS 0 IN C IF THE NUMBER IS NOT ZERO.
22          ; RETURNS 1 IN C IF THE NUMBER IS ZERO.
23          ; SAVES REGS. K, A, X, AND B BUT DESTROYS C.
24          ;
25          FZCHK:
26          X A, K
27          SHL A
28          IFGT A,OFF
29          JP $ANOTO
30          RRC A
31          SET C
32          X A, K
33          RET
34          $ANOTO:
35          RRC A
36          RESET C
37          X A, K
38          RET
39          ;
40          .END
```

11

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FUNPAK
FPAK.MAC

PAGE: 7

```

16          .FORM 'FPAK.MAC'
17          .INCLD FPAK.MAC
1          .TITLE FPAK
2          .LOCAL
3
4          ;
5          ; SUBROUTINE TO PACK A SP FLOATING POINT NUMBER STORED IN THE
6          ; 3 WORD FEXP-FSIGN/FHI/FLO FORMAT INTO THE IEEE FORMAT IN REGS.
7          ; K AND A.
8          ;
9          ; ON ENTRY TO THE SUBROUTINE, X POINTS TO FLO. ON EXIT, X POINTS
10         ; TO THE WORD AFTER FSIGN.
11         ;
12         ; REGS. K, A AND B ARE DESTROYED.
13         ;
14         FPAK:
15         X A, M(X+)      ; GET RID OF ZERO LOW BYTE OF FLO.
16         X A, M(X+)      ; GET HIGH BYTE OF FLO.
17         ST A, K          ; STORE IT IN K.
18         X A, M(X+)      ; GET LOW BYTE OF FHI.
19         SWAP A
20         SWAP A
21         AND A, OFF00     ; SHIFT LEFT 8 TIMES.
22         OR K, A          ; LOW WORD OF RESULT IS NOW IN K.
23         ;
24         X A, M(X+)      ; GET HIGH BYTE OF FHI.
25         RESET A.7       ; ZERO IMPLIED MSB 1 IN MANT.
26         ST A,B          ; SAVE IN REG. B.
27         LD A, M(X)      ; GET SIGN BYTE FROM ASIGN.
28         SHR A           ; MOVE 1 SIGN BIT INTO CARRY.
29         LD A, W(X+)      ; GET FEXP-FSIGN.
30         AND A, OFF00     ; ZERO SIGN.
31         RRC A           ; MOVE RIGHT 1 BIT. SIGN BIT FROM C
32         ; ENTERS INTO THE MSB.
33         OR A, B          ; GET MANT BITS IN FROM B.
34         X A, K          ; SWAP A AND K
35         RET
36         ;
37         .END

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FPAK
FPTRAP.MAC

PAGE: 8

```

18          .FORM 'FPTRAP.MAC'
19          .INCLD FPTRAP.MAC
1          .TITLE FPTRAP
2          ; USER SUPPLIED FP TRAP ROUTINE.
3          FPTRAP:
4          F09D 3C        RET
5          ;
6          .END

```

```

20          .FORM 'ROUND.MAC'
21          .INCLD ROUND.MAC
22          .TITLE SROUND
23          .LOCAL
24
25          ;
26          ; THIS SUBROUTINE IS USED TO ROUND THE 32 BIT MANTISSA OBTAINED
27          ; IN THE FLOATING POINT CALCULATIONS TO 24 BITS.
28          ;
29          ; THE UNPACKED FLOATING POINT NUMBER SHOULD BE STORED IN
30          ; CONSECUTIVE WORDS OF MEMORY. ON ENTRY, X SHOULD CONTAIN
31          ; THE ADDRESS OF C-HI. C-EXP.C-SIGN IS AT W(X+2) AND
32          ; C-LO IS AT W(X-2).
33          ;
34          ; ON EXIT X HAS THE ADDRESS OF C-EXP.C-SIGN.
35          SROUND:
36              LD A, W(X-)      ; REMEMBER X POINTS TO C-HI.
37              LDA, W(X)        ; LOAD C-LO.
38              IF A.7           ; IF BIT 25 OF MANTISSA IS 1,
39              JP $RNDUP        ; THEN NEED TO INCREASE MANTISSA.
40              LD A, W(X+)
41              LD A, W(X+)      ; X NOW POINTS TO C-EXP.C-SIGN.
42              JP $EXIT         ; DONE, SO GET OUT.
43          ; INCREASE MANTISSA.
44          $RNDUP:
45              ADD A, 0100
46              X A, W(X+)      ; INCREASE LOW BYTE BY 1.
47              IF C            ; IF THERE IS A CARRY,
48              JP $HIUP        ; THEN NEED TO INCREASE C-HI.
49              LD A, W(X+)      ; X NOW POINTS TO C-EXP.C-SIGN.
50              JP $EXIT         ; DONE, SO GET OUT.
51          ; MANTISSA INCREASE PROPAGATING TO HIGH WORD.
52          $HIUP:
53              LD A, W(X)
54              .BYTE 0B8,00,01 ; DO ADD A, 01 BUT WITH WORD CARRY!!
55              IF C            ; IF THERE IS A CARRY,
56              JP $EXIN2        ; THEN NEED TO INCREASE EXPONENT.
57              X A, W(X+)
58              JP $EXIT         ; GET OUT.
59          ; ROUND UP LEADS TO EXPONENT INCREASE.
60          $EXIN2:
61              RRC A            ; CARRY->MSB, LSB->CARRY.
62              X A, W(X-)
63              LD A, W(X)        ; LOW WORD IS NOW IN A.
64              RRC A
65              X A, W(X+)
66              LD A, W(X+)      ; X NOW POINTS TO C-EXP.CSIGN.
67              LD A, W(X)
68              ADD A, 0100
69              IF C

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
SROUND
ROUND.MAC

PAGE: 10

```
48 FOC2 BAFF00          OR A, OFF00      ; MAKE IT A NAN.
49 FOC5 F6              ST A, W(X)
50                      ;
51                      $EXIT:
52 FOC6 3C              RET
53                      ;
54                      .END
```

```

22          .FORM 'BFMUL.MAC'
23          .INCLD BFMUL.MAC
1           .TITLE BFMUL
2
3           ;
4           ; THIS SUBROUTINE IS USED TO MULTIPLY TWO 32 BIT FIXED POINT FRACTIONS.
5           ; THE ASSUMED BINARY POINT IS TO THE IMMEDIATE LEFT OF THE MSB.
6           ;
7           ; THE FIRST FRACTION IS STORED IN REGS K AND A, WITH THE MORE
8           ; SIGNIFICANT WORD BEING IN K.
9           ;
10          ; THE SECOND FRACTION IS STORED ON THE STACK. THE MORE SIGNIFICANT
11          ; WORD IS AT W(SP-4) AND THE LOWER SIGNIFICANT WORD
12          ; IS IN THE WORD BELOW IT.
13          ;
14          ; THE 32 BIT PRODUCT IS LEFT IN REGS. K AND A, WITH THE MORE
15          ; SIGNIFICANT WORD BEING IN K.
16          ;
17          ; IMPORTANT NOTE : THE FRACTIONS ARE ASSUMED TO BE UNSIGNED.
18          ;
19          ; REGS. B AND X ARE UNCHANGED.
20          ;
21          BFMUL:
22          FOC7 AFCE          PUSH X          ; SAVE X.
23          FOC9 AFC8          PUSH A          ; SAVE F1-LO
24          FOCB AFCA          PUSH K          ; SAVE F1-HI.
25          FOD4 3FCA          LD A, K          ; MOVE F1-HI TO A.
26          FOD6 AFCE          MULT A, W(SP-OA) ; MULTIPLY F1-HI BY F2-HI.
27          FOD8 AFC8          POP K           ; GET F1-HI.
28          FODA A8CA          PUSH X          ; SAVE PR-HI.
29          FODC A6FFF2C4FE    PUSH A          ; SAVE PR-LO.
30          FOE1 3FC8          LD A, K          ; MOVE F1-HI TO A.
31          FOE3 3FCA          MULT A, W(SP-OE) ; MULTIPLY F1-HI BY F2-LO.
32          FOE5 96CEF8        POP A           ; GET PR-LO SAVED. NOTE THAT THE
33          FOE8 07            ; LO WORD OF THIS PRODUCT IS DISCARDED.
34          FOE9 A9CA          POP K           ; GET PR-HI SAVED.
35          FOEB 3FCE          ADD A, X         ; ADD TO PR-LO THE HI WORD OF THIS PRODUCT.
36          FOED AFCA          IF C            ; ON CARRY,
37          FOEF AFC8          INC K            ; PROPAGATE THRU TO PR-HI.
38          FOF1 A8CE          POP X           ; GET F1-LO.
39          FOF3 A6FFF6C4FE    PUSH K          ; SAVE PR-HI.
40          FOF8 3FC8          PUSH A          ; SAVE PR-LO.
41          FOF9 96CEF8        LD A, X         ; MOVE F1-LO TO A.
42          FOFB 3FCE          MULT A, W(SP-OA) ; MULTIPLY BY F2-HI.
43          FOFD 3FC8          POP A           ; GET PR-LO SAVED.
44          FOF8 3FC8          POP K           ; GET PR-HI SAVED.
45          FOF9 96CEF8        ADD A, X         ; ADD TO PR-LO THE HI-WORD OF THIS PRODUCT.
46          FOFB 3FCE          IF C            ; PROPAGATE ANY CARRY TO PR-HI.
47          FOFD 3FC8          INC K            ; RESTORE X.
48          FOF8 3FC8          POP X
49          FOF9 96CEF8        RET

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
BFMUL
BFMUL.MAC

PAGE: 12

50 .END

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
BFMUL
ISIOK.MAC

PAGE: 13

```
24 .FORM 'ISIOK.MAC'
25 .INCLD ISIOK.MAC
1 .TITLE ISIOK
2 .LOCAL
3 ;
4 ; THIS SUBROUTINE IS USED TO DETERMINE IF ANOTHER DECIMAL DIGIT CAN
5 ; BE ACCUMULATED IN THE 32 BIT INTEGER STORED IN REGS. K AND A.
6 ; THE MORE SIGNIFICANT WORD IS IN K.
7 ; SETS THE CARRY TO 1 IF IT CAN BE ACCUMULATED; RESETS THE CARRY
8 ; OTHERWISE. PRESERVES ALL REGS.
9 ;
10 ISIOK:
11 F105 02 SET C
12 F106 861999CAFC IFEQ K, 01999
13 F10B 47 JP $CHKOT
14 F10C 861999CAFD IFGT K, 01999
15 F111 03 RESET C
16 F112 3C RET
17 F113 BD9998 $CHKOT: IFGT A, 09998
18 F116 03 RESET C
19 F117 3C RET
20 ;
21 .END
```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
ISIOK
MUL10.MAC

PAGE: 14

```

26          .FORM 'MUL10.MAC'
27          .INCLD MUL10.MAC
1           .TITLE MUL10
2           .LOCAL
3
4           ;
5           ; THIS SUBROUTINE MULTIPLIES THE 32 BIT INTEGER STORED IN REGS K AND A
6           ; BY 10-DECIMAL AND ADDS TO IT THE INTEGER STORED IN X.
7           ; THE RESULT IS RETURNED IN K AND A.
8           ; REGS. B AND X ARE NOT CHANGED.
9           ;
10          MUL10:
11          PUSH X          ; SAVE INTEGER.
12          PUSH A          ; SAVE LONG INT-LO.
13          LD A, K
14          MULT A, OA      ; MULT LONG INT-HI BY 10.
15          PUSH A          ; SAVE LOW WORD OF PRODUCT.
16          LD A, W(SP-4)   ; GET LONG INT-LO.
17          MULT A, OA
18          POP K           ; GET LO WORD OF LAST PRODUCT.
19          ADD K, X        ; ADD TO IT HI WORD OF THIS PRODUCT.
20          POP X           ; GET RID OF GARBAGE.
21          POP X           ; GET INTEGER TO BE ADDED.
22          ADD A, X
23          IF C
24          INC K
25          RET
26          ;
27          .END
28          ;

```

```

29          .FORM 'ATOF.MAC'
30          .INCLD ATOF.MAC
1           .TITLE ATOF
2           .LOCAL
3
4           ;
5           ; THIS SUBROUTINE CONVERTS A DECIMAL FLOATING POINT STRING TO
6           ; AN IEEE FORMAT SINGLE PRECISION FLOATING POINT NUMBER. THE
7           ; INPUT DECIMAL STRING IS ASSUMED TO BE OF THE FORM
8           ; SMMMMMM.FFFFFEDNN
9           ; WHERE S IS THE SIGN OF THE DECIMAL MANTISSA,
10          ; M...M IS THE INTEGER PART OF THE MANTISSA,
11          ; F...F IS THE FRACTIONAL PART OF THE MANTISSA,
12          ; D IS THE SIGN OF THE DECIMAL EXPONENT,
13          ; AND NNN IS THE DECIMAL EXPONENT.
14
15          ; ON ENTRY, B SHOULD POINT TO THE ADDRESS OF THE ASCII
16          ; STRING HOLDING THE DECIMAL FLOATING POINT NUMBER. THIS STRING
17          ; MUST BE TERMINATED BY A NULL BYTE.
18
19          ; THE BINARY FLOATING POINT NUMBER IS RETURNED IN
20          ; REGS. K AND A.
21
22          ; REGS. B AND X ARE LEFT UNCHANGED.
23
24          ;
25          ;
26          ;
27          ATOF:
28 F13A AFCE          PUSH X
29 F13C AFCC          PUSH B
30 F13E 00            CLR A          ; ZERO A.
31 F13F AFC8          PUSH A          ; STORAGE FOR MANTISSA SIGN.
32 F141 AFC8          PUSH A          ; STORAGE FOR IMPLICIT 10'S EXPONENT.
33 F143 AFC8          PUSH A          ; STORAGE FOR HI-INT.
34 F145 AFC8          PUSH A          ; STORAGE FOR LO-INT.
35
36          ;
37          ; DECIMAL STRING MUST START WITH A '+', '-', '.' OR A DIGIT.
38          ; RESULTS ARE UNPREDICTABLE IF IT DOES NOT.
39          ; THE '+' MEANS THAT THE MANTISSA IS POSITIVE. IT CAN BE OMITTED.
40          ; THE '-' MEANS THAT THE MANTISSA IS NEGATIVE.
41          ; THE '.' MEANS THAT THE MANTISSA HAS NO INTEGER PART.
42          ;
43          $LOOP1:
44 F147 C0            LDS A, M(B+)
45 F148 40            NOP          ; GET THE CHARACTER.
46 F149 9C2B          IFEQ A, '+'   ; IF IT IS A '+',
47 F14B 64            JP $LOOP1    ; DO NOTHING, BUT GET 1 MORE.
48 F14C 9C2D          IFEQ A, '-'   ; IF IT IS A '-',
49 F14E 45            JP $MSIGN     ; GO AND CHANGE THE MANTISSA SIGN.
50 F14F 9C2E          IFEQ A, '.'   ; IF IT IS A '.',

```

```

50 F151 9438          JMP $FRCOL      ; GO AND COLLECT THE FRACTION PART.
51                      ; GET HERE MEANS IT IS A DIGIT.
52 F153 48            JP $INCOL       ; SO GO AND COLLECT THE INTEGER PART.
53          $MSIGN:
54 F154 90FF          LD A, OFF
55 F156 A6FFF8C4AB     ST A, W(SP-08) ; CHANGE MANTISSA SIGN TO NEG.
56 F15B 74            JP $LOOP1      ; GO BACK FOR SOME MORE.
57                      ;
58          $INCOL:
59                      ; GET HERE MEANS COLLECTING INTEGER PART OF MANTISSA.
60                      ;
61 F15C 02            SET C
62 F15D 8230C8EB       SUBC A, '0'    ; CONVERT DIGIT FROM ASCII TO INTEGER.
63 F161 ACC8CE         LD X, A        ; MOVE INTEGER TO X.
64 F164 3FCA          POP K          ; GET HI-INT COLLECTED SO FAR.
65 F166 3FC8          POP A          ; GET LO-INT COLLECTED SO FAR.
66 F168 3463          JSR ISI0K      ; CHECK IF THE DIGIT CAN BE ACCUMULATED.
67 F16A 07            IF C           ; LOOK AT C.
68 F16B 4B            JP $ACCM       ; YES, IT CAN BE SO GO DO IT.
69                      ; GET HERE MEANS CAN ACCUMULATE ANY MORE.
70                      ; SO INCREASE THE IMPLICIT 10'S EXPONENT.
71 F16C 3FCE          POP X          ; GET IMPLICIT 10'S EXPONENT COLLECTED
72 F16E A9CE          INC X          ; SO FAR AND INCREMENT IT.
73 F170 AFCE          PUSH X         ; SAVE IT BACK.
74 F172 AFC8          PUSH A         ; SAVE LO-INT.
75 F174 AFCA          PUSH K         ; SAVE HI-INT.
76 F176 46            JP $ISNXT
77                      ;
78          $ACCM:
79                      ; GET HERE MEANS THE PRESENT DIGIT CAN BE ACCUMULATED.
80 F177 345F          JSR MUL10      ; MULTIPLY BY 10 AND ADD DIGIT.
81 F179 AFC8          PUSH A         ; SAVE LO-INT.
82 F17B AFCA          PUSH K         ; SAVE HI-INT.
83          $ISNXT:
84                      ; PROCESS THE NEXT CHARACTER.
85 F17D C0            LDS A, M(B+)
86 F17E 40            NOP
87 F17F 9C2E          IFEQ A, '.'    ; IF IT IS A '.'
88 F181 49            JP $FRCOL      ; GO COLLECT FRACTION PART.
89 F182 9C45          IFEQ A, 'E'    ; IF IT IS 'E',
90 F184 9434          JMP $EXCOL     ; GO COLLECT EXPONENT PART.
91 F186 9C20          IFEQ A, ' '    ; IF IT IS A SPACE,
92 F188 6B            JP $ISNXT      ; GO GET SOME MORE.
93                      ; GET HERE MEANS IT IS A DIGIT.
94 F189 952D          JMP $INCOL
95                      ;
96          $FRCOL:
97                      ; GET HERE MEANS COLLECT THE FRACTIONAL PART OF THE MANTISSA.
98                      ;
99 F18B C0            LDS A, M(B+)
100 F18C 40            NOP

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
ATOF
ATOF.MAC

PAGE: 17

```

101 F18D 9C45          IFEQ A, 'E'      ; IF IT IS A 'E',
102 F18F 9429          JMP $EXCOL      ; GO COLLECT EXPONENT.
103 F191 9C20          IFEQ A, ' '      ; IF IT IS SPACE,
104 F193 68            JP $FRCOL        ; GO GET SOME MORE.
105                    ; GET HERE MEANS IT IS A DIGIT.
106 F194 D2            SET C
107 F195 8230C8EB      SUBC A, '0'      ; GET INTEGER FROM DIGIT.
108 F199 ACC8CE        LD X, A          ; SAVE IT IN A.
109 F19C 3FCA          POP K            ; GET HI-INT.
110 F19E EFC8          POP A            ; GET LO-INT.
111 F1A0 349B          JSR ISIOK        ; CHECK IF IT CAN BE ACCUMULATED.
112 F1A2 07            IF C
113 F1A3 45            JP $ACCF         ; YES, SO GO DO IT.
114                    ; GET HERE MEANS CAN'T COLLECT MORE DIGITS.
115 F1A4 AFC8          PUSH A
116 F1A6 AFCA          PUSH K
117 F1A8 7D            JP $FRCOL        ; SO JUST IGNORE IT.
118                    ;
119                    $ACCF:
120                    ; ACCUMULATE THE FRACTIONAL DIGIT.
121 F1A9 3491          JSR MULIO        ; MULTIPLY BY 10 AND ADD DIGIT.
122 F1AB 3FCE          POP X            ; GET IMPLICIT 10'S EXPONENT COLLECTED SO FAR,
123 F1AD 86FFFFCEFB      ADD X, OFFF    ; AND DECREMENT IT BY 1.
124 F1B2 AFCE          PUSH X           ; SAVE IT BACK.
125 F1B4 AFC8          PUSH A           ; SAVE LO-INT.
126 F1B6 AFCA          PUSH K           ; SAVE HI-INT.
127 F1B8 952D          JMP $FRCOL        ; GO GET SOME MORE.
128                    ;
128                    $EXCOL:
130                    ; GET HERE MEANS THE EXPLICIT 10'S EXPONENT NEEDS TO BE
131                    ; COLLECTED FROM THE STRING.
132 F1BA 03            RESET C          ; MAKE EXPONENT SIGN POST.
133                    $EXCHR:
134 F1BB C0            LDS A, M(B+)
135 F1BC 40            MOP
136 F1BD 9C2B          IFEQ A, '+'      ; IF IT IS A '+',
137 F1BF 64            JP $EXCHR        ; GET SOME MORE.
138 F1C0 9C2D          IFEQ A, '-',    ; IF IT IS A '-',
139 F1C2 44            JP $ESIGN        ; GO FIX EXPONENT SIGN.
140 F1C3 9C20          IFEQ A, ' '      ; IF IT IS SPACE,
141 F1C5 6A            JP $EXCHR        ; GO GET SOME MORE.
142                    ; GET HERE MEANS IT IS A DIGIT.
143 F1C6 42            JP $EXACC        ; SO GO COLLECT THE EXPONENT.
144                    $ESIGN:
145 F1C7 02            SET C
146 F1C8 6D            JP $EXCHR
147                    ;
148                    $EXACC:
149                    ; ACCUMULATE THE EXPLICIT 10'S EXPONENT.
150 F1C9 9100          LD K, 0
151 F1CB 07            IF C

```

```

152 F1CC 91FF          LD K, OFF          ; GET SIGN BITS SET.
153 F1CE AFCA          PUSH K             ; SAVE EXPLICIT EXPONENTS SIGN.
154 F1D0 9300          LD X, 0
155 F1D2 AFCE          PUSH X             ; ZERO EXPLICIT EXPONENT COLLECTED SO FAR.
156                   $EXCLP:
157 F1D4 02            SET C
158 F1D5 8230C8EB      SUBC A, '0'        ; GET INTEGER FROM ASCII DIGIT.
159 F1D9 ACC8CE          LD X, A
160 F1DC 3FC8           POP A             ; GET EXPLICIT EXPONENT COLLECTED SO FAR.
161 F1DE A0C8CEF8       ADD X, A          ; X CONTAINS DIGIT + EXP.
162 F1E2 A0C8CEF8       ADD X, A          ; X CONTAINS DIGIT + 2*EXP.
163 F1E6 E7            SHL A
164 F1E7 E7            SHL A
165 F1E8 E7            SHL A             ; A CONTAINS 8*EXP.
166 F1E9 96CEF8        ADD A, X           ; A CONTAINS DIGIT + 10*EXP.
167 F1EC AFC8          PUSH A             ; SAVE BACK ON STACK.
168 F1EE C0            LDS A, M(B+)
169 F1EF 40            NOP               ; GET NEXT CHAR.
170 F1F0 9C00          IFEQ A, 0          ; IS IT A NULL ?
171 F1F2 41            JP $A10EX          ; YES SO ADD EXPLICIT AND IMPLICIT
172                   ; 10'S EXPONENT.
173                   ; GET HERE MEANS IT IS A DIGIT,
174 F1F3 7F            JP $EXCLP          ; SO GO BACK AND ACCUMULATE IT.
175                   ;
176                   $A10EX:
177                   ; DONE COLLECTING DIGITS. ADD THE EXPLICIT AND IMPLICIT
178                   ; 10'S EXPONENT COLLECTED SO FAR.
179 F1F4 3FC8          POP A             ; GET EXPLICIT 10'S EXPONENT.
180 F1F6 3FCA          POP K             ; GET ITS SIGN.
181 F1F8 8200CAFC       IFEQ K, 0         ; IS IT POSITIVE ?
182 F1FC 42            JP $ADDEX          ; YES SO ADD 'EM.
183 F1FD 01            COMP A
184 F1FE 04            INC A             ; CHANGE TO 2'S COM.
185                   $ADDEX:
186 F1FF AGFFFAC4F8     ADD A, W(SF-06)   ; ADD IMPLICIT EXPONENT.
187 F204 BD7FFF        IFGT A, 07FFF     ; IS IT NEGATIVE ?
188 F207 43            JP $NEG10         ; YES, CHANGE IT.
189 F208 9300          LD X, 0           ; LOAD POST. SIGN IN X.
190 F20A 44            JP $ESAVE
191                   $NEG10:
192 F20B 93FF          LD X, OFF          ; LOAD NEG. SIGN IN X.
193 F20D 01            COMP A
194 F20E 04            INC A             ; MAKE IT POSITIVE.
195                   $ESAVE:
196 F20F ACC8CC          LD B, A          ; SAVE 10'S EXPONENT IN A.
197 F212 3FC8          POP A             ; GET HI-INT.
198 F214 3FCA          POP K             ; GET LO-INT.
199 F216 AFCE          PUSH X             ; SAVE SIGN OF 10'S EXPONENT.
200 F218 AFCC          PUSH B             ; AND ITS VALUE.
201                   ;
202                   ; NOW CONVERT HI-INT, LO-INT TO A NORMALIZED FLOATING POINT

```

```

203                ; NUMBER. THE BINARY EXPONENT IS COLLECTED IN B.
204                ;
205 F21A 9000      IFGT A, 0          ; IF HI-INT IS NOT 0,
206 F21C 58        JP $NORM2         ; NEED TO SHIFT K AND A.
207 F21D 8200CAFD  IFGT K, 0          ; IF HI-INT IS 0, BUT NOT LO-INT,
208 F221 4E        JP $NORM1         ; NEED TO SHIFT ONLY K.
209                ; GET HERE MEANS MANTISSA IS 0.
210 F222 00        CLR A
211 F223 ACC8CA    LD K, A
212 F226 02        SET C
213 F227 8208C4EB  SUB SP, 08        ; ADJUST SP. DONE!!!
214 F22B 3FCC      POP B
215 F22D 3FCE      POP X
216 F22F 3C        RET
217                ;
218                $NORM1:
219                ; HI-INT IS 0, SO WORK WITH LO-INT ONLY.
220 F230 AECA      X A, K
221 F232 9210      LD B, 010         ; LOAD 16 INTO EXPONENT COUNTER.
222 F234 42        JP $NRLUP
223                ;
224                $NORM2:
225                ; HI-INT IS NOT 0, SO NEED TO HANDLE BOTH.
226 F235 9220      LD B, 020         ; LOAD 32 INTO LOOP COUNTER.
227                $NRLUP:
228 F237 E7        SHL A
229 F238 07        IF C              ; DID A 1 COME OUT ?
230 F239 4D        JP $NRDUN         ; YES IT IS NORMALIZED NOW.
231 F23A AECA      X A, K
232 F23C E7        SHL A
233 F23D 07        IF C
234 F23E 96CA08    SET K.0
235 F241 AECA      X A, K
236 F243 AACC      DECSZ B
237 F245 40        NOP              ; SHOULD NEVER BE SKIPPED!!
238 F246 6F        JP $NRLUP
239                $NRDUN:
240 F247 D7        RRC A              ; RESTORE SHIFTED 1.
241 F248 AB00      ST A, TMP1        ; STORE IN W(0).
242 F24A 3FCE      POP X              ; GET 10'S EXPONENT.
243 F24C 3FC8      POP A              ; GET 10'S EXPONENT SIGN.
244 F24E AE00      X A, TMP1        ; A IS HI-INT ONCE MORE.
245 F250 AFCC      PUSH B             ; SAVE BINARY EXPONENT.
246 F252 AFCE      PUSH X             ; SAVE 10'S EXPONENT.
247 F254 AECA      X A, K             ; HI-INT TO K, LO-INT TO A.
248 F256 960010    IF TMP1.0         ; IS 10'S EXPONENT NEGATIVE ?
249 F259 58        JP $DIV10         ; YES, GO TO DIVIDE BY 10.
250                ; GET HERE MEANS 10'S EXPONENT IS POSITIVE, SO MULTIPLY BY 10.
251                ; ACTUALLY, WHAT IS USED IS
252                ;      10^N = (0.625*(2^4)^N
253                ; SO MULTIPLY BY 0.625 NOW AND TAKE CARE OF 2^(4*N) LATER.

```

```

254 F25A A4F26ACCAB      LD B, W($MTLO)
255 F25F AFCC             PUSH B           ; SAVE LO WORD OF 0.625 ON STACK.
256 F261 A4F26CCCAB      LD B, W($MTHI)
257 F266 AFCC             PUSH B           ; SAVE HI WORD OF 0.625 ON STACK.
258 F268 57               JP $JAMIT          ; GO TO ROUTINE THAT JAMS 0.625*N
259                       ; BY REPEATED MULTIPLICATION INTO HI-INT.LO-INT.
260                       ;
261                       ; DEFINE SOME CONSTANTS.
262 F269 40               . EVEN          ; FORCE EVEN ADDRESS.
263 F26A 0000             $MTLO: .WORD 0
264 F26C 00A0             $MTHI: .WORD 0A000
265 F26E CDCC             $DTLO: .WORD OCCCD
266 F270 CCCC             $DTHI: .WORD OCCCC
267                       ;
268 $DIV10:
269 ; GET HERE MEANS 10'S EXPONENT IS NEGATIVE, SO DIVIDE BY 10.
270 ; ACTUALLY WHAT IS DONE IS
271 ;  $10^{(-N)} = ((2^3)/(0.8))^{(-N)} = ((0.8)^N) * (2^{(-3*N)})$ 
272 ; SO MULTIPLY BY 0.8 NOW AND TAKE CARE OF  $2^{(-3*N)}$  LATER.
273 F272 A4F26ECCAB      LD B, W($DTLO)
274 F277 AFCC             PUSH B           ; SAVE LO WORD OF .8
275 F279 A4F270CCAB      LD B, W($DTHI)
276 F27E AFCC             PUSH B           ; SAVE HI WORD OF .8
277                       ;
278 $JAMIT:
279 ; JAM IN THE MULTIPLICATION PART NEEDED TO HANDLE THE 10'S EXP.
280 F280 9200             LD B, 0           ; B IS USED TO TRACK ANY BINARY POWERS
281                       ; THAT COME UP DURING NORMALIZATION.
282 F282 8200CEFC         IFEQ X, 0         ; IS 10'S EXPONENT 0 ?
283 F286 57               JP $JAMDN        ; YES, DONE ALREADY.
284 $JAMLP:
285 F287 35C0             JSR BFMUL         ; MULTIPLY USING 32 BIT UNSIGNED.
286 F289 AECA             X A, K           ; SWAP HI AND LO WORDS.
287 F28B E7               SHL A
288 F28C 07               IF C             ; IS THERE A CARRY ?
289 F28D 4A               JP $ISNED        ; YES, SO IT IS ALREADY NORMALIZED.
290 F28E A9CC             INC B            ; NEED TO SHIFT LEFT TO NORMALIZE, SO
291                       ; INCREASE B BY 1.
292 F290 AECA             X A, K
293 F292 E7               SHL A
294 F293 07               IS C
295 F294 96CA08           SET K.0
296 F297 43               JP $OVR1
297 $ISNED:
298 F298 D7               RRC A
299 F299 AECA             X A, K
300 $OVR1:
301 F29B AACE             DECSZ X           ; DONE YET ?
302 F29D 76               JP $JAMLP        ; NO SO DO IT ONCE MORE.
303 ; GET HERE MEANS MULTIPLICATIONS HAVE BEEN DONE. NOW TAKE
304 ; CARE OF THE EXPONENTS.

```

```

305          $JAMDN:
306 F29E 3FCE          POP X
307 F2A0 3FCE          POP X          ; GET 0.625 OR 0.8 OFF THE STACK.
308 F2A2 3FCE          POP X          ; GET THE 10'S EXPONENT.
309 F2A4 AFC8          PUSH A          ; SAVE LO WORD OF FLP NUMBER.
310 F2A6 AFCA          PUSH K          ; SAVE HI WORD OF FLP NUMBER.
311 F2A8 A6FFFAC4A8    LD A, W(SP-6)  ; GET THE BINARY EXPONENT THAT WAS SAVED.
312 F2AD 02            SET C
313 F2AE 96CCEB        SUBC A, B      ; SUBTRACT FROM IT BINARY EXPONENT COLLECTED
314                                ; DURING THE JAMMING.
315 F2B1 ACC8CC        LD B, A        ; SAVE IT IN B.
316 F2B4 A8CE          LD A, X        ; MOVE THE 10'S EXPONENT TO A.
317 F2B6 960010        IF TMP1.0      ; IS THE 10'S EXPONENT NEGATIVE ?
318 F2B9 49            JP $NAGAS      ; YES, SO GOT TO SUBTRACT.
319                                ; GET HERE MEANS 10'S EXPONENT IS
320                                ; POSITIVE, SO MUL IT BY 4.
321 F2BA E7            SHL A          ; MULTIPLY BY 2.
322 F2BB E7            SHL A          ; MULTIPLY BY 2 AGAIN.
323 F2BC 96CCF8        ADD A, B        ; GET THE BINARY EXPONENT IN ALSO.
324 F2BF B8007E        ADD A, 07E     ; AND THE IEEE BIAS.
325 F2C2 4C            JP $EXCPT      ; GO CHECK FOR OVER/UNDERFLOW.
326          ;
327          $NAGAS:
328          ; GET HERE MEANS 10'S EXPONENT IS NEGATIVE, SO GOT TO MULTIPLY
329          ; IT BY -3.
330 F2C3 E7            SHL A          ; MULTIPLY BY 2.
331 F2C4 96CEF8        ADD A, X        ; ADD TO GIVE MULTIPLY BY 3.
332 F2C7 01            COMP A
333 F2C8 04            INC A          ; MAKE IT NEGATIVE.
334 F2C9 96CCF8        ADD A, B        ; GET IN THE BINARY EXPONENT.
335 F2CC B8007E        ADD A, 07E     ; AND THE IEEE BIAS.
336          $EXCPT:
337          ; CHECK FOR OVERFLOW/UNDERFLOW.
338 F2CF ACC4CE        LD X, SP        ; FIRST DO SOME JUGGLING
339 F2D2 02            SET C          ; TO BE COMPATIBLE WITH EXCEPTION
340 F2D3 820ACEEB      SUBC X, 0A      ; HANDLING IN OTHER ROUTINES.
341 F2D7 AFCE          PUSH X
342 F2D9 BD7FFF        IFGT A, 07FFF  ; IS BIASED EXPONENT NEGATIVE ?
343 F2DC B4FD3F        JMWL UNDFL
344 F2DF 9C00          IFEQ A, 0       ; IS IT 0 ?
345 F2E1 B4FD3A        JMWL UNDFL      ; YES IT IS STILL UNDERFLOW.
346 F2E4 9DFE          IFGT A, 0FE    ; IS IT GT THAN 254 ?
347 F2E6 B4FD46        JMWL OVRFL
348          ; GET HERE MEANS VALID SP FLP NUMBER.
349 F2E9 3FCE          POP X          ; X POINTS TO MANTISSA SIGN.
350 F2EB E7            SHL A
351 F2EC E7            SHL A
352 F2ED E7            SHL A
353 F2EE E7            SHL A
354 F2EF E7            SHL A
355 F2FD E7            SHL A

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
ATOF
ATOF.MAC

PAGE: 22

```

356 F2F1 E7          SHL A
357 F2F2 E7          SHL A          ; MOVE EXPONENT TO HIGH BYTE.
358 F2F3 8FFA        OR A, W(X)     ; GET THE MANTISSA SIGN IN.
359 F2F5 AB00        ST A, TMP1      ; SAVE IT IN TMP1.
360 F2F7 3FCA        POP K          ; FI-HI TO K.
361 F2F9 3FC8        POP A          ; FI-LO TO A.
362 F2FB F1          X A, W(X+)     ; SAVE FI-LO.
363 F2FC A8CA        LD A, K
364 F2FE F1          X A, W(X+)     ; SAVE FI-HI.
365 F2FF A800        LD A, TMP1
366 F301 F3          X A, W(X-)     ; SAVE FI-EXP.FI-SIGN, X POINTS TO FI-HI.
367 F302 3664        JSRL SROUND    ; ROUND THE RESULT.
368 F304 F2          LD A, W(X-)    ; X POINTS TO FI-HI.
369 F305 F2          LD A, W(X-)    ; X POINTS TO FI-LO.
370 F306 AFCE        PUSH X
371 F308 368C        JSR FPAK       ; PACK IT INTO IEEE FORMAT.
372 F30A 3FC4        POP SP
373 F30C 3FCC        POP B
374 F30E 3FCE        POP X
375 F310 3C          RET
376                  ;
377                  .END

```

```

31          .FORM 'FTOA.MAC'
32          .INCLD FTOA.MAC
1           .TITLE FTOA
2           .LOCAL
3
4           ;
5           ; THIS SUBROUTINE CONVERTS A SINGLE PRECISION, BINARY FLOATING
6           ; POINT NUMBER IN THE IEEE FORMAT TO A DECIMAL FLOATING POINT
7           ; STRING. THE DECIMAL FLOATING POINT STRING IS OBTAINED TO A
8           ; PRECISION OF 9 DECIMAL DIGITS.
9           ;
10          ; THE ALGORITHM USED IS BASED ON:
11          ; J.T. COONEN, 'AN IMPLEMENTATION GUIDE TO A PROPOSED STANDARD
12          ; FOR FLOATING POINT ARITHMETIC,' IEEE COMPUTER, JAN. 1980, PP 68-79.
13          ;
14          ; ON INPUT, THE BINARY SP FLP NUMBER IS IN REGS. K AND A.
15          ; B CONTAINS THE ADDRESS OF THE LOCATION WHERE THE DECIMAL FLOATING
16          ; POINT STRING IS TO START. NOTE THAT AT LEAST 17 BYTES ARE NEEDED
17          ; FOR THE STORAGE OF THE STRING. THE LAST BYTE IS ALWAYS NULL.
18          ;
19          ; ALL REGISTERS ARE PRESERVED BY THIS SUBROUTINE.
20          ;
21          FTOA:
22          PUSH X          ; SAVE X ON THE STACK.
23          PUSH B          ; SAVE B ON THE STACK.
24          ; CHECK AND SEE IF F1 IS A NAN.
25          JSR FNACHK
26          IF C
27          JNPL $NAN        ; YET IT IS, SO GET OUT.
28          ; CHECK AND SEE IF F1 IS ZERO.
29          JSR FZCHK
30          IF C
31          JNPL $ZERO       ; YES IT IS, SO GET OUT.
32          ; GET HERE MEANS F1 IS A NON-ZERO, NON-NAN FLP NUMBER.
33          LD X, SP
34          ADD SP, 06        ; ADJUST SP.
35          JSR FUNPAK        ; UNPACK THE NUMBER.
36          ; X POINTS ONE WORD PAST F1-EXP.F1-SIGN
37          ; ON RETURN.
38          ;
39          ; COMPUTE THE EXPONENT OF 10 FOR DECIMAL FLP NO.
40          ; THIS IS DONE AS FOLLOWS:
41          ; SUPPOSE F1 = FM * (2^M)
42          ; LET U = M*LOG(2)      NOTE: LOG IS TO BASE 10.
43          ; THEN V = INT(U+1-9)
44          ; IS USED AS THE 10'S EXPONENT.
45          ; NOTE: INT REFERS TO INTEGER PART.
46          LD A, M(X-)      ; X POINTS TO F1-EXP.
47          LD A, M(X-)      ; LOAD F1-EXP. X POINTS TO F1-SIGN.
48          LD TMP1, 0        ; FIRST GUESS POSITIVE SIGN FOR EXP.
49          ADD A, 0FF82      ; REMOVE IEEE BIAS FROM F1-EXP.

```

```

50 F333 AFC8          PUSH A          ; SAVE IT ON THE STACK.
51 F335 AFCE          PUSH X          ; SAVE F1-SIGN ADDRESS ALSO.
52 F337 07            IF C            ; WAS THERE A CARRY ON THE LAST ADD ?
53 F338 46            JP $MLOG2       ; YES, SO 2'S EXP IS POSITIVE.
54 F339 B700FF00      LD TMPL, OFF    ; 2'S EXPONENT IS NEGATIVE.
55 F33D 01            COMP A
56 F33E 04            INC A          ; MAKE IT POSITIVE.
57                    $MLOG2:
58                    ; MULTIPLY M BY LOG(2).
59 F33F BE4D10        MULT A, 04D10   ; LOG(2) IS 0.0100110100010000 TO 16 BITS.
60                    ; X CONTAINS INTEGER PART, AND A FRACT. PART.
61 F342 AECE          X A, X          ; SWAP THE TWO.
62 F344 960010        IF TMPL.0       ; WAS THE 2'S EXPONENT NEGATIVE ?
63 F347 41            JP $CSIGN       ; YES, SO MAKE U NEGATIVE.
64 F348 4B            JP $REMV9       ; NO, SO GO DO V = U + 1 - 9.
65                    $CSIGN:
66 F349 01            COMP A          ; COMP INTEGER PART.
67 F34A AECE          X A, X
68 F34C 01            COMP A          ; FRACTION PART.
69 F34D B80001        ADD A, 01
70 F350 AECE          X A, X
71 F352 07            IF C
72 F353 04            INC A
73                    $REMV9:
74 F354 04            INC A          ; INCREASE FRACTION PART.
75 F355 B8FFF7        ADD A, OFFF7    ; SUBTRACT 9.
76 F358 BD7FFF        IFGT A, 07FFF   ; IS IT NEGATIVE ?
77 F35B 45            JP $CHNGS       ; YES, SO CHANGE ITS SIGN.
78 F35C B7000000      LD TMPL, 0      ; REMEMBER POSITIVE SIGN.
79 F36D 4F            JP $DIV10
80                    $CHNGS:
81 F361 B700FF00      LD TMPL, OFF    ; REMEMBER NEGATIVE SIGN.
82 F365 01            COMP A          ; MAKE V POSITIVE.
83 F366 AECE          X A, X
84 F368 01            COMP A
85 F369 B80001        ADD A, 01
86 F36C AECE          X A, X
87 F36E 07            IF C
88 F36F 04            INC A
89                    $DIV10:
90                    ;
91                    ; V = INT (U+1-9) HAS BEEN COMPUTED AND IS IN A.
92                    ; NOW COMPUTE W = F1/(10^V). W SHOULD BE AN INTEGER, AND IT IS
93                    ; COMPUTED TO A 32 BIT PRECISION.
94                    ; THIS COMPUTATION IS DONE AS FOLLOWS:
95                    ; IF V > 0, THEN F1/(10^V) = F1*(0.8^V)*(2^(-3V)).
96                    ; IF V < 0, THEN F1/(10^V) = F1*(0.625^U)*(2^(4V)).
97                    ; SO FIRST MULTIPLY THE MANTISSA OF F1 V TIMES BY 0.8 (OR 0.625)
98                    ; AND THEN ADJUST THE EXPONENT OF F1. NOTE THAT THE PARTIAL PRODUCTS
99                    ; IN MULTIPLYING BY 0.8 (OR 0.625) ARE KEPT NORMALIZED. THIS IS
100                   ; ESSENTIAL TO PRESERVE 32 BIT ACCURACY IN THE FINAL RESULT.

```

```

101          ; SINCE THE MANTISSA OF F1 IS NORMALIZED, AND 0.8 (OR 0.625 IS ALSO
102          ; NORMALIZED, EACH PRODUCT NEEDS AT MOST 1 LEFT SHIFT FOR
103          ; RENORMALIZATION. THE SHIFTS ACCUMULATED DURING RENORMALIZATION ARE
104          ; TRACKED AND ACCOUNTED FOR IN THE CALCULATION.
105 F370 3FCE          POP X          ; X NOW POINTS TO F1-SIGN.
106 F372 AFC8          PUSH A         ; SAVE U ON THE STACK.
107 F374 ACC8CC        LD B, A         ; MOVE V TO B ALSO.
108 F377 F2            LD A, W(X-)     ; X POINTS TO F1-HI.
109 F378 F2            LD A, W(X-)     ; LOAD F1-HI. X POINTS TO F1-LO.
110 F379 ACC8CA        LD K, A
111 F37C F4            LD A, W(X)      ; LOAD F1-LO.
112 F370 960010        IF TMP1.0      ; IS V NEGATIVE?
113 F380 57            JP $MUL10      ; YES, SO MULTIPLY V TIMES BY .625.
114                      ; GET HERE MEANS MULTIPLY V TIMES BY .8.
115 F381 A4F390CEAB    LD X, W($DTLO)
116 F386 AFCE          PUSH X         ; LO WORD OF 0.8 TO STACK.
117 F388 A4F392CEAB    LD X, W($DTHI)
118 F38D AFCE          PUSH X         ; HI WORD OF 0.8 TO STACK.
119 F38F 56            JP $JAMIT      ; GO DO MULTIPLICATION.
120          ;
121          .EVEN                  ; FORCE EVEN ADDRESS.
122 F390 CDCC          $DTLO: .WORD 0CCCC
123 F392 CCCC          $DTHI: .WORD 0CCCC
124 F394 0000          $MILO: .WORD 0
125 F396 00A0          $MTHI: .WORD 0A000
126          ;
127          $MUL10:
128 F398 A4F394CEAB    LD X, W($MTLO)
129 F39D AFCE          PUSH X         ; LO WORD OF 0.625 TO STACK.
130 F39F A4F396CEAB    LD X, W($MTHI)
131 F3A4 AFCE          PUSH X         ; HI WORD OF 0.625 TO STACK.
132          ;
133          $JAMIT:
134 F3A6 9300          LD X, 0         ; INIT X TO TRACK ANY POWERS OF
135                      ; 2 GENERATED DURING NORMALIZATION
136                      ; OF PARTIAL PRODUCTS.
137 F3A8 8200CCFC      IFEQ B, 0      ; IS B ALREADY 0 ?
138 F3AC 57            JP $JAMON      ; YES, SO SKIP MULTIPLY LOOP.
139          $JAMLP:
140 F3AD 36E6          JSR BFMUL      ; MULTIPLY.
141 F3AF AECA          X A, K         ; SWAP HI AND LO WORDS OF PART. PROD.
142 F3B1 E7            SHL A
143 F3B2 07            IF C          ; IS THERE A CARRY ?
144 F3B3 4A            JP $ISNED     ; YES, SO SKIP OVER RENORMALIZATION.
145                      ; GET HERE MEANS NEED TO RENORM.
146 F3B4 A9CE          INC X         ; UPDATE RENORM COUNT.
147 F3B6 AECA          X A, K         ; SWAP HI AND LO PART. PROD.
148 F3B8 E7            SHL A
149 F3B9 07            IF C
150 F3BA 96CA08        SET K.0       ; SET BIT SHIFTED OUT FROM LO WORD.
151 F3BD 43            JP $OVR1

```

```

152          $ISNED:
153 F3BE D7          RRC A          ; PUT BACK SHIFTED BIT.
154 F3EF AECA          X A, K
155          $OVR1:
156 F3C1 AACCC        DECSZ B          ; IS B 0 YET ?
157 F3C3 76          JP $JAMLP        ; NO, SO DO IT AGAIN.
158          $JAMON:
159          ; GET HERE MEANS MULTIPLICATION HAS BEEN DONE, SO TAKE CARE
160          ; OF EXPONENT.
161 F3C4 3FCC          POP B
162 F3C6 3FCC          POP B          ; GET RID OF 0.8 (OR 0.625) FROM STACK.
163 F3C8 AFC8          PUSH A          ; SAVE LO WORD OF PROD.
164 F3CA AFCA          PUSH K          ; SAVE HI WORD OF PRODUCT.
165 F3CC A6FFF8C4A8    LD A, W(SP-08) ; GET F1'S BINARY EXPONENT.
166 F3D1 02          SET C
167 F3D2 96CEEB        SUBC A, X      ; SUBTRACT FROM IT ANYTHING COLLECTED
168          ; DURING RENORM.
169 F3D5 ACC8CE        LD X, A          ; AND SAVE IT IN X.
170 F3D8 A6FFFAC4A8    LD A, W(SP-06) ; GET V FROM THE STACK.
171 F3DD 960010        IF TMPL.0      ; IS V NEGATIVE ?
172 F3E0 49          JP $ML4          ; YES, SO MULTIPLY V BY 4.
173          ; GET HERE MEANS MULTIPLY V BY -3.
174 F3E1 E7          SHL A          ; NOW A CONTAINS 2*V.
175 F3E2 A6FFFAC4F8    ADD A, W(SP-06) ; NOW A CONTAINS 3*V.
176 F3E7 01          COMP A
177 F3E8 04          INC A          ; NOW A CONTAINS -3*V.
178 F3E9 42          JP $ADEM        ; GO FIGURE FINAL EXPONENT.
179          $ML4:
180 F3EA E7          SHL A
181 F3EB E7          SHL A          ; NOW A CONTAINS 4*V.
182          $ADEM:
183 F3EC 96CEF8        ADD A, X      ; A SHOULD NOW BE A POSITIVE INTEGER
184          ; IN THE RANGE 0 TO 32.
185          ; NOW CHECK AND SEE IF A HAS ENOUGH PRECISION.
186 F3EF 9020          IFGT A, 020    ; NEED MORE THAN 32 BITS ?
187 F3F1 5A          JP $INCRV        ; YES, SO GO INCREASE V.
188 F3F2 9D1B          IFGT A, 01B    ; NEED AT LEAST 28 BITS ?
189 F3F4 9435          JMP $GOON      ; YES, SO ALL IS OK. GO ON.
190          ; GET HERE MEANS NEED MORE
191          ; PRECISION, SO DECREASE V.
192 F3F6 3FC8          POP A          ; GET HI-PROD OFF STACK.
193 F3F8 3FC8          POP A          ; GET LO WORD OFF STACK.
194 F3FA 3FC8          POP A          ; GET MAGN. OF V.
195 F3FC 96D010        IF TMPL.0      ; IS V NEG. ?
196 F3FF 56          JP $VUP          ; YES, SO GO INCR. MAGN. OF V.
197          ; GET HERE MEANS V IS POSITIVE,
198          ; AND NEED TO DECREMENT IT.
199 F400 B8FFFF        ADD A, OFFF    ; SUBTRACT 1 FROM A.
200 F403 07          IF C          ; GOT A CARRY ?
201 F404 5A          JP $GOBAK        ; THEN OK.
202 F405 01          COMP A

```

```

203 F406 04          INC A
204 F407 B700FF00    LD TMP1, OFF      ; U CHANGES SIGN.
205 F40B 53          JP $GOBAK
206
207 F40C 3FC8        $INCRV: POP A          ; GET HI PROD. OFF STACK.
208 F40E 3FC8        POP A          ; GET LO PROD. OFF STACK.
209 F410 3FC8        POP A          ; GET MAGN. OF V.
210 F412 960010      IF TMP1,0       ; IS V NEGATIVE ?
211 F415 42          JP $VDOWN       ; YES.
212
213 F416 04          $VUP:  INC A
214 F417 47          JP $GOBAK
215
216 F418 AAC8        $VDOWN: DECSZ A
217 F41A 44          JP $GOBAK
218 F41B B7000000    LD TMP1, 0      ; V CHANGES SIGN.
219
220 F41F ACC4CE      $GOBAK: LD X, SP
221 F422 02          SET C
222 F423 8204CEEB    SUBC X, 04
223 F427 AFCE        PUSH X
224 F429 95B9        JMP $DIV10
225
226 F42B 01          $GOON:  COMP A
227 F42C 04          INC A          ; NEGATE A.
228 F42D B80020      ADD A, 020      ; SUBTRACT IT FROM 32.
229 F430 ACC8CE      LD X, A        ; AND MOVE IT TO X.
230 F433 3FCA        POP X          ; GET HI WORD OF PRODUCT.
231 F435 3FC8        POP A          ; GET LO WORD OF PROD.
232 F437 8200CEFC    IFEQ X, 0      ; IS X 0 ?
233 F43B 49          JP $INDUN      ; YES, SO ALREADY A 32 BIT INTEGER.
234
235                  $INTFY:
236                  ; NOW ADJUST THE PRODUCT TO FORM A 32 BIT INTEGER.
237 F43C AECA        X A, K          ; SWAP HI AND LO WORDS.
238 F43E C7          SHR A
239 F43F AECA        X A, K
240 F441 D7          RRC A          ; SHIFT IT RIGHT ONCE.
241 F442 AACE        DECSZ X        ; X 0 YET ?
242 F444 68          JP $INTFY      ; NO SO GO DO SOME MORE.
243
244                  $INDUN:
245                  ; GET HERE MEANS K.A CONTAIN THE 32 BIT INTEGER THAT IS THE
246                  ; MANTISSA OF THE DECIMAL FLP NUMBER.
247 F445 AFC8        PUSH A          ; SAVE LO-INT.
248 F447 AFCA        PUSH K          ; SAVE HI INT.
249 F449 A6FFF0C4A8  LD A, W(SP-010) ; GET STARTING ADDRESS OF DECIMAL STRING.
250 F44E B80010      ADD A, 010      ; ADD 16 TO IT.
251 F451 ACC8CC      LD B, A        ; AND MOVE IT B.
252 F454 00          CLR A
253 F455 C3          XS A, M(B-)     ; OUTPUT TERMINATING NULL BYTE.
254 F456 40          NOP
255 F457 A6FFFAC4A8  LD A, W(SP-06) ; GET V.

```

```

254 F45C 9FOA          DIV A, OA      ; DIVIDE IT BY 10. QUOT. IN A,
255                                ; REM. IN X.
256 F45E AECE          X A, X          ; REM TO A.
257 F460 B80030        ADD A, 030      ; MAKE IT INTO ASCII BYTE.
258 F463 C3            XS A, M(B-)     ; OUTPUT IT.
259 F464 40            NOP
260 F465 A8CE          LD A, X
261 F467 B80030        ADD A, 030
262 F46A C3            XS A, M(B-)
263 F46B 40            NOP              ; FINISHED OUTPUTING EXPONENT.
264 F46C 902B          LD A, 028      ; SAY EXP SIGN IS '+'.
265 F46E 960010        IF TMPl.0
266 F471 902D          LD A, 02D      ; NOPE, IT IS '-'.
267 F473 C3            XS A, M(B-)     ; OUTPUT IT.
268 F474 40            NOP
269 F475 9045          LD A, 045
270 F477 C3            XS A, M(B-)     ; OUTPUT 'E'.
271 F478 40            NOP
272 F479 902E          LD A, 02E
273 F47B C3            XS A, M(B-)     ; OUTPUT '.'.
274 F47C 40            NOP
275                                ; NOW NEED TO OUTPUT 10 DECIMAL DIGITS.
276 F47D B7000A00      LD TMPl, OA     ; LOAD 10 INTO TMPl AS LOOP COUNTER.
277 $DOLUP:
278 F481 3FC8          POP A           ; A CONTAINS HI INT.
279 F483 9FOA          DIV A, OA      ; DIVIDE IT BY 10. QUOT. IN A,
280                                ; REM. IN X.
281 F485 ACC8CA        LD K, A
282 F488 3FC8          POP A           ; A CONTAINS LO INT.
283 F48A AFCC          PUSH B          ; SAVE DEC. STR. ADDR.
284 F48C ACCACC        LD B, K         ; B CONTAINS HI-QUOT.
285 F48F 82            .BYTE 082,0A,0C8,0EF
      F490 OA
      F491 C8
      F492 EF
286                                ; THE ABOVE 4 BYTES REPRESENT THE INSTRUCTION DIVD A, OA.
287                                ; BECAUSE THE ASSEMBLER DOES NOT KNOW ABOUT IT YET, WE HAVE TO
288                                ; KLUDGE IT THIS WAY.
289                                ; AFTER THE DIVD, A CONTAINS THE LO-QUOT. AND X THE REM.
290 F493 ACCCCA        LD K, B         ; MOVE HI-QUOT TO K.
291 F496 3FCC          POP B           ; B CONTAINS DEC. STR. ADDR.
292 F498 AFC8          PUSH A          ; SAVE LO INT.
293 F49A AFCA          PUSH K          ; SAVE HI INT.
294 F49C A8CE          LD A, X         ; MOVE REM TO A.
295 F49E B80030        ADD A, 030      ; ASCII-FY IT.
296 F4A1 C3            XS A, M(B-)     ; AND OUTPUT IT.
297 F4A2 40            NOP
298 F4A3 AA00          DECSZ TMPl      ; IS TMPl 0 YET ?
299 F4A5 9524          JMP $DOLUP      ; NO, GO GET SOME MORE.
300                                ; GET HERE MEANS DONE WITH OUTPUTING MANTISSA.
301 F4A7 3FC8          POP A

```

```

302 F4A9 3FC8      POP A
303 F4AB 3FC8      POP A
304 F4AD 3FC8      POP A      ; GET SOME GARBAGE OFF THE STACK.
305 F4AF 3FCA      POP K      ; GET F1-EXP.F1-SIGN TO K.
306 F4B1 9020      LD A, 02D   ; LOAD SF INTO A.
307 F4B3 96CA10    IF K.0
308 F4B6 902D      LD A, 02D   ; IF MAINT. IS NEG. LOAD '-'.
309 F4B8 C6        ST A, M(B)  ; OUTPUT SIGN.
310 F4B9 AFCA      PUSH K     ; F1-EXP.F1-SIGN BACK ON STACK.
311 F4BB ACC4CE    LD X, SP
312 F4BE 02        SET C
313 F4BF 8206CEEB  SUBC X, 06   ; X POINTS TO F1-L0.
314 F4C3 AFCE      PUSH X
315 F4C5 B5FBB4    JSR FPAK    ; PACK IT, SO RESTORING K AND A.
316 F4C8 3FC4      POP SP
317 F4CA 3FCC      POP B      ; RESTORE B.
318 F4CC 3FCE      POP X      ; RESTORE X.
319 F4CE 3C        RET
320                ;
321                $NAN:
322                ; GET HERE MEANS F1 IS A NAN.
323 F4CF AFC8      PUSH A
324 F4D1 ACCCCE    LD X, B
325 F4D4 8210CEF8  ADD X, 010
326 F4D8 00        CLR A
327 F4D9 03        X A, M(X-)
328 F4DA 9210      LD B, 010
329                $NANLP:
330 F4DC 90FF      LD A, OFF
331 F4DE D3        X A, M(X-)
332 F4DF AACC      DECSZ B
333 F4E1 65        JP $NANLP
334 F4E2 3FC8      POP A
335 F4E4 3FCC      POP B
336 F4E6 3FCE      POP X
337 F4E8 3C        RET
338                ;
339                $ZERO:
340                ; GET HERE MEANS F1 IS ZERO.
341 F4E9 AFC8      PUSH A
342 F4EB ACCCCE    LD X, B      ; X CONTAINS DECIMAL STRING ADDR.
343 F4EE 8210CEF8  ADD X, 010
344 F4F2 00        CLR A
345 F4F3 D3        X A, M(X-)  ; OUTPUT TERMINATING NULL BYTE.
346 F4F4 9030      LD A, 030   ; LOAD 0 INTO A.
347 F4F6 D3        X A, M(X-)
348 F4F7 9030      LD A, 030
349 F4F9 D3        X A, M(X-)  ; OUTPUT 00 FOR EXPONENT.
350 F4FA 902B      LD A, 02B   ; LOAD '+' SIGN.
351 F4FC D3        X A, M(X-)
352 F4FD 9045      LD A, 045   ; LOAD 'E'

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FTOA
FTOA.MAC

PAGE: 30

```
353 F4FF D3          X A, M(X-)
354 F500 902E        LD A, 02E      ; LOAD '.'.
355 F502 D3          X A, M(X-)
356 F503 920A        LD B, 0A
357                  $ZERLP:
358 F505 9030        LD A, 030
359 F507 D3          X A, M(X-)
360 F508 AACC        DECSZ B
361 F50A 65          JP $ZERLP
362 F508 9020        LD A, 020      ; LOAD SP.
363 F50D D5          X A, M(X)
364 F50E 3FC8        POP A
365 F510 3FCC        POP B
366 F512 3FCE        POP X
367 F514 3C          RET
368                  ;
369                  .END
```

```

33          .FORM, 'FADD.MAX'
34          .INCLD FADD.MAC
1           .TITLE FADD
2           .LOCAL
3
4           ;
5           ; SUBROUTINE TO ADD/SUBTRACT TWO SP FLOATING POINT NUMBERS.
6           ; C = F1 + F2 OR C = F1 - F2
7           ;
8           ; F1 IS STORED IN THE IEEE FORMAT IN REGS K AND A.
9           ; THE HIGH WORD OF F1 WILL BE REFERRED AS F1-R1 AND IS IN K.
10          ; THE LOW WORD OF F1 WILL BE REFERRED TO AS F1-RO AND IS IN A.
11          ;
12          ; F2 IS STORED IN THE IEEE FORMAT ON THE STACK. IF SP IS THE
13          ; STACK POINTER ON ENTRY, THEN
14          ; THE HIGH WORD OF F2, REFERRED TO AS F2-R1 IS AT SP - 4 AND
15          ; THE LOW WORD OF F2, REFERRED TO AS F2-RO IS AT SP - 6.
16          ;
17          ; C IS RETURNED IN THE IEEE FORMAT IN REGS K AND A.
18          ;
19          FSUB:
20          ST A, TMP1
21          LD A, W(SP-06) ; LOAD F2-RO.
22          PUSH A ; AND SAVE ON STACK.
23          LD A, W(SP-06) ; LOAD F2-R1.
24          XOR A, 08000 ; CHANGE THE SIGN.
25          PUSH A ; AND SAVE ON THE STACK.
26          LD A, TMP1 ; RESTORE A.
27          JSR FADD ; CALL THE ADD ROUTINE.
28          ST A, TMP1 ; SAVE A.
29          POP A ; GET RID OF JUNK
30          POP A ; FROM THE STACK.
31          LD A, TMP1 ; RESTORE A.
32          RET
33          ;
34          FADD:
35          ; SAVE ADDRESS OF F2-RO IN TMP1.
36          PUSH X ; SAVE X ON ENTRY.
37          PUSH B ; AND B ON ENTRY.
38          LD X, SP
39          ADD X, OFFF6 ; SUBTRACT 10.
40          LD TMP1, X ; AND SAVE IN TMP1.
41          ; CHECK AND SEE IF F1 IS A NAN.
42          JSR FNACHK
43          IF C
44          JNPL FNAN ; F1 IS A NAN.
45          ; CHECK AND SEE IF F2 IS A NAN.
46          LD B, K
47          LD X, A
48          LD A, W(TMP1+2)
49          LD K, A
50          X A, X

```

```

50 F55A B5FAE3      +      JSR FNACHK
51 F55D 07          IF C
52 F55E B4FAAE      JMPL FNAN      ; F2 IS NAN.
53                  ; CHECK AND SEE IF F2 IS ZERO.
54 F561 B5FAED      +      JSR FZCHK
55 F564 06          IFN C
56 F565 48          JP $F1CHK      ; F2 IS NOT ZERO. CHECK F1.
57 F566 ACCCCA      LD K, B      ; F2 IS ZERO, SO ANSWER IS F1.
58 F569 3FCC        POP B
59 F56B 3FCE        POP X
60 F56D 3C          RET
61                  ; CHECK AND SEE IF F1 IS ZERO.
62                  $F1CHK:
63 F56E ACCCCA      LD K, B      ; RESTORE F1-R1 FROM B.
64 F571 B5FADD      +      JSR FZCHK
65 F574 06          IFN C
66 F575 4F          JP $NTZERO     ; JUMP SINCE F1 IS ALSO NOT ZERO.
67 F567 A20200AB    LD A, W(TMP1+2) ; GET HERE MEANS F1 IS ZERO,
68 F57A ACC8CA      LD K, A      ; SO ANSWER IS F2.
69 F57D AD00A8      LD A, W(TMP1)
70 F580 3FCC        POP B
71 F582 3FCE        POP X
72 F584 3C          RET
73                  ; GET HERE MEANS NORMAL ADDITION.
74                  ; UNPACK F1 AND F2.
75                  $NTZERO:
76 F585 ACC4CE      LD X, SP      ; X POINTS TO F1-L0.
77 F588 8210C4F8    ADD SP, 010   ; MOVE SP PAST LOCAL STORAGE.
78 F58C AFCE        PUSH X      ; SAVE SP ON STACK FOR QUICK RETURN.
79 F58E B5FAD0      +      JSR FUNPAK ; UNPACK F1.
80 F591 AC00CC      LD B, TMP1    ; B NOW POINTS TO F2-R0.
81 F594 ACCE00      LD TMP1, X    ; TMP1 NOW POINTS TO F2-L0.
82 F597 E0          LDS A, W(B+)  ; LOAD F2-R0 INTO A.
83 F598 40          NOP
84 F599 AECA        X A, K
85 F59B E4          LD A, W(B)
86 F59C AECA        X A, K      ; LOAD F2-R1 INTO K.
87 F59E B5FAC0      +      JSR FUNPAK ; UNPACK F2.
88                  ; SET X TO POINT TO F2-SIGN AND B TO POINT TO F1-SIGN.
89 F5A1 F2          LD A, W(X-)
90 F5A2 AC00CC      LD B, TMP1
91 F5A5 E2          LDS A, W(B-)
92 F5A6 40          NOP
93                  ; COMPARE F1-EXP AND F2-EXP.
94 F5A7 F2          LD A, W(X-)  ; LOAD F2-EXP.F2-SIGN INTO A.
95 F5A8 B9FF00      AND A, OFF00  ; MASK OUT SIGN.
96 F5AB A6FFFCC4AB  ST A, W(SP-4) ; SAVE IN C-SIGN.C-EXP.
97 F5B0 E2          LDS A, W(B-)
98 F5B1 40          NOP      ; LOAD F1-EXP.F1-SIGN INTO A.
99 F5B2 B9FF00      AND A, OFF00  ; CHANGE TO F1-EXP.00000000.
100 F5B5 02         SET C

```

```

101 F5B6 A6FFFCC4EB          SUBC A, W(SP-4) ; SUBTRACT F2-EXP.00000000.
102 F5BB 06                  IFN C
103 F5BC 942D                JMP $F2GTR          ; F2-EXP IS BIGGER THAN F1-EXP.
104                          ; GET HERE MEANS F1-EXP IS BIGGER THAN F2-EXP.
105 F5BE 80C9CAAB            LD K, H(A)          ; SAVE DIFF. IN K TO BE USED AS LOOP COUNTER.
106 F5C2 A6FFFCC4FB          ADD A, W(SP-4)
107 F5C7 A6FFFCC4AB          ST A, W(SP-4)        ; RESTORE F1-EXP AND STORE IN C-SIGN.
108 F5CC 8217CAFD            IFGT K, 017
109 F5D0 51                  JP $ZROF2           ; K GT 23-DEC MEANS F2 GETS ZEROED IN SHIFTS.
110                          ; LOOP TO SHIFT F2 INTO ALIGNMENT.
111 F5D1 8200CAFC            IFEQ K, 0
112 F5D5 943B                JMP $ADDMN          ; K = 0 MEANS DONE SHIFTING.
113                          $LOOP2:
114 F5D7 F4                  LD A, W(X)
115 F5D8 C7                  SHR A
116 F5D9 F3                  X A, W(X-)
117 F5DA F4                  LD A, W(X)
118 F5DB D7                  RRC A
119 F5DC F1                  X A, W(X+)
120 F5DD AACA                DECSZ K
121 F5DF 68                  JP $LOOP2
122 F5E0 9430                JMP $ADDMN
123                          $ZROF2:
124                          ; SET F2 MANTISSA TO 0.
125 F5E2 F0                  LD A, W(X+)          ; X POINTS TO F2-EXP.F2-SIGN.
126 F5E3 00                  CLR A
127 F5E4 F3                  X A, W(X-)          ; AND STORE IT BACK.
128 F5E5 00                  CLR A
129 F5E6 F3                  X A, W(X-)
130 F5E7 00                  CLR A
131 F5E8 F1                  X A, W(X+)
132 F5E9 9427                JMP $ADDMN
133                          ; F2 EXPONENT IS GREATER THAN F1 EXPONENT.
134                          $F26TR:
135 F5EB 01                  COMP A
136 F5EC 04                  INC A                ; CHANGE DIFF IN EXP TO POSITIVE.
137 F5ED 80C9CAAB            LD K, H(A)          ; LOAD K WITH LOOP COUNTER.
138 F5F1 8217CAFD            IFGT K, 017
139 F5F5 51                  JP $ZROF1           ; F1 MANT. REDUCED TO 0 IN SHIFTS.
140                          ; LOOP TO SHIFT F1 MANT INTO ALIGNMENT.
141 F5F6 8200CAFC            IFEQ K, 0
142 F5FA 57                  JP $ADDMN          ; K=0 MEANS DONE SHIFTING.
143                          $LOOP1:
144 F5FB E4                  LD A, W(B)
145 F5FC C7                  SHR A
146 F5FD E3                  XS A, W(B-)
147 F5FE 40                  NOP
148 F5FF E4                  LD A, W(B)
149 F600 D7                  RRC A
150 F601 E1                  XS A, W(B+)
151 F602 40                  NOP

```

```

152 F603 AACA          DECSZ K          ;
153 F605 6A           JP $LOOP1
154 F606 4B           JP $ADDMN
155                   $ZROF1:             ; SET F1 MANT TO 0.
156 F607 E0           LOS A, W(B+)       ; B POINTS TO F1-EXP.F1-SIGN.
157 F608 40           NOP
158 F609 00           CLR A
159 F60A E3           XS A, W(B-)       ; STORE IT BACK.
160 F60B 40           NOP
161 F60C 00           CLR A
162 F60D E3           XS A, W(B-)
163 F60E 40           NOP
164 F60F 00           CLR A
165 F610 E1           XS A, W(B+)
166 F611 40           NOP
167                   ; DETERMINE IF MANTISSAS ARE TO BE ADDED OR SUBTRACTED.
168                   $ADDMN:             ; B POINTS TO F1-HI, X TO F2-HI.
169 F612 E0           LDS A, W(B+)
170 F613 40           NOP
171 F614 F0           LD A, W(X+)
172 F615 D4           LD A, M(X)         ; LOAD F2-SIGN.
173 F616 D8           XOR A, M(B)       ; XOR WITH F1-SIGN.
174 F617 9C00         IFEQ A, 0
175 F619 9451         JMP $TRADD         ; SAME SIGN SO GO TO ADD MANTISSA.
176                   ; GET HERE MEANS TRUE SUBTRACT OF MANTISSA.
177 F61B F2           LD A, W(X-)
178 F61C F2           LD A, W(X-)       ; X POINTS TO F2-L0.
179 F61D E2           LDS A, W(B-)
180 F61E 40           NOP
181 F61F E2           LDS A, W(B-)
182 F620 40           NOP               ; B NOW POINTS TO F1-L0.
183 F621 E0           LDS A, W(B+)
184 F622 40           NOP               ; A NOW CONTAINS F1-L0.
185 F623 02           SET C
186 F624 8FEB         SUBC A, W(X)       ; SUBTRACT F2-L0.
187 F626 F1           X A, W(X+)
188 F627 E0           LDS A, W(B+)       ; A CONTAINS F1-HI.
189 F628 40           NOP
190 F629 8FEB         SUBC A, W(X)       ; SUBTRACT F2-HI.
191 F62B F1           X A, W(X+)
192 F62C 07           IF C
193 F62D 55           JP $F1SIN         ; F1 GE F2, SO SIGN IS F1-SIGN.
194                   ; GET HERE MEANS F1 LT F2, SO SIGN IS F2-SIGN.
195 F62E A6FFFCC4A8   LD A, W(SP-4)
196 F633 8FDA         OR A, M(X)
197 F635 F3           X A, W(X-)       ; C-EXP.C-SIGN HAS BEEN DETERMINED.
198 F636 F4           LD A, W(X)
199 F637 D1           COMP A
200 F638 F3           X A, W(X-)
201 F639 F4           LD A, W(X)
202 F63A 01           COMP A

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FADD
FADD.MAC

PAGE: 35

```

203 F63B B80001      ADD A, 01
204 F63E F1          X A, W(X+)
205 F63F 07          IF C
206 F640 8FA9        INC W(X)
207 F642 47          JP $ANORM
208                  ; GET HERE MEANS F1 GE F2.
209                  $F1SIN:
210 F643 A6FFFC4A8    LD A, W(SP-4)
211 F648 DA          OR A, M(B)
212 F649 F3          X A, W(X-)
213                  ; NORMALIZE THE MANTISSA.
214                  $ANORM:
215 F64A ACCECC        LD B, X          ; B POINTS TO C-HI.
216 F64D E0          LDS A, W(B+)
217 F64E 40          NOP
218 F64F C0          LDS A, M(B+)
219 F650 40          NOP          ; B NOW POINTS TO C-EXP BYTE.
220 F651 9118        LD K, 018        ; SET UP LOOP LIMIT OF 24-DEC IN K.
221                  $NLOOP:
222 F653 F4          LD A, W(X)
223 F654 E7          SHL A
224 F655 07          IF C          ; CARRY MEANS NORMALIZED.
225 F656 9448        JMP $ROUND      ; SO JUMP TO ROUNDING CODE.
226 F658 F3          X A, W(X-)
227 F659 F4          LD A, W(X)
228 F65A E7          SHL A
229 F65B F1          X A, W(X+)
230 F65C 07          IF C
231 F65D 8F08        SET W(X).0
232 F65F ADCC8A      DECSZ M(B)      ; ADJUST EXPONENT.
233 F662 43          JP $OV1
234 F663 B4F9B8      JMPL UNDFL      ; C-EXP ZERO MEANS UNDERFLOW.
235                  $OV1:
236 F666 AACA        DECSZ K          ; DECREMENT LOOP COUNTER.
237 F668 75          JP $NLOOP        ; GO BACK TO LOOP.
238 F669 B4F9B2      JMPL UNDFL      ; UNDERFLOW
239                  ;GET HERE MEANS TRUE ADDITION OF MANTISSA.
240                  $TRADD:
241 F66C E2          LDS A, W(B-)
242 F66D 40          NOP
243 F66E E2          LDS A, W(B-)
244 F66F 40          NOP          ; B NOW POINTS TO F1-HI.
245 F670 F2          LD A, W(X-)
246 F671 F2          LD A, W(X-)
247 F672 F4          LD A, W(X)      ; LOAD F2-LO INTO A.
248 F673 F8          ADD A, W(B)      ; ADD F1-LO.
249 F674 F1          X A, W(X+)      ; STORE IN F2-LO.
250 F675 E0          LDS A, W(B+)
251 F676 40          NOP          ; B NOW POINTS TO F1-HI.
252 F677 F4          LD A, W(X)      ; LOAD F2-HI INTO A.
253 F678 E8          ADC A, W(B)      ; ADD F1-HI WITH CARRY FROM LO ADD.

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FADD
FADD.MAC

PAGE: 36

```

254 F679 07          IF C
255 F67A 4A          JP $ADJEX      ; IF CARRY, NEED TO INCREASE EXP.
256 F67B F1          X A, W(X+)    ; STORE RESULT IN F2-HI.
257 F67C A6FFFCC4A8  LD A, W(SP-4) ; GET C-EXP.00000000.
258 F681 8FDA        OR A, M(X)    ; INTRODUCE SIGN.
259 F683 F3          K A, W(X-)    ; STORE IN F2-EXP.F2-SIGN.
260 F684 5B          JP $ROUND
261                  ; GET HERE MEANS NEED TO INCREASE EXP BY 1.
262 $ADJEX:
263 F685 D7          RRC A
264 F686 F3          X A, W(X-)
265 F687 F4          LD A, W(X)
266 F688 D7          RRC A
267 F689 F1          X A, W(X+)
268 F68A F0          LD A, W(X+)    ; X NOW POINTS TO F2-EXP.F2-SIGN.
269 F68B A6FFFCC4A8  LD A, W(SP-4) ; GET C-EXP.00000000.
270 F690 B80100      ADD A, 0100    ; INCREASE EXP BY 1.
271 F693 07          IF C
272 F694 B4F998      JMPL OVRFL
273 F697 BDFEFF      IFGT A, OFEFF  ; IS BIASED EXPONENT 255-DEC ?
274 F69A B4F992      JMPL OVRFL
275 F69D 8FDA        OR A, M(X)
276 F69F F3          X A, W(X-)
277                  ; NEED TO ROUND THE RESULT. X POINTS TO C-HI.
278 $ROUND:
279 F6A0 B5F9FB      JSRL SROUND
280                  ; FINAL CHECK OF EXPONENT.
281 F6A3 D0          LD A, M(X+)    ; X NOW POINTS TO C-EXP.
282 F6A4 D2          LD A, M(X-)
283 F6A5 9C00        IFEQ A, 0
284 F6A7 B4F974      JMPL UNDFL
285 F6AA 90FE        IFGT A, OFE
286 F6AC B4F980      JMPL OVRFL
287 F6AF F2          LD A, W(X-)
288 F6B0 F2          LD A, W(X-)    ; X NOW POINTS TO C-LO.
289 F6B1 B5F9C8      JSR FPAK      ; PACK C.
290 F6B4 3FC4        POP SP        ; SET UP SP FOR RETURN.
291 F6B6 3FCC        POP B
292 F6B8 3FCE        POP X
293 F6BA 3C          RET
294                  ;
295                  .END

```

```

35          .FORM 'FMULT.MAC'
36          .INCLD FMULT.MAC
1           .TITLE FMULT
2           .LOCAL
3
4           ;
5           ; SUBROUTINE TO MULTIPLY TWO SP FLOATING POINT NUMBERS.
6           ; C = F1*F2
7           ;
8           ; F1 IS STORED IN THE IEEE FORMAT IN REGS K AND A.
9           ; THE HIGH WORD OF F1 WILL BE REFERRED AS F1-R1 AND IS IN K.
10          ; THE LOW WORD OF F1 WILL BE REFERRED TO AS F1-RO AND IS IN A.
11          ;
12          ; F2 IS STORED IN THE IEEE FORMAT ON THE STACK. IF SP IS THE
13          ; STACK POINTER ON ENTRY, THEN
14          ; THE HIGH WORD OF F2, REFERRED TO AS F2-R1 IS AT SP - 4 AND
15          ; THE LOW WORD OF F2, REFERRED TO AS F2-RO IS AT SP - 6.
16          ;
17          ; C IS RETURNED IN THE IEEE FORMAT IN REGS K AND A.
18          ; REGS. X AND B ARE PRESERVED.
19          ;
20          FMULT:
21          PUSH X          ; SAVE X ON ENTRY.
22          PUSH B          ; SAVE B ON ENTRY.
23          ; SAVE ADDRESS OF F2-RO IN TMP1.
24          LD X, SP
25          ADD X, OFFF6     ; SUBTRACT 10.
26          LD TMP1, X      ; SAVE IN TMP1.
27          ; CHECK AND SEE IF F1 IS A NAN.
28          JSR FNACHK
29          IF C
30          JNPL FNAN        ; F1 IS A NAN.
31          ; CHECK AND SEE IF F2 IS A NAN.
32          LD B, K
33          LD X, A
34          LD A, W(TMP1+2)
35          LD K, A
36          X A, X
37          JSR FNACHK
38          IF C
39          JNPL FNAN        ; F2 IS NAN.
40          ; CHECK AND SEE IF F2 IS ZERO.
41          JSR FZCHK
42          IF C
43          JMP $CZERO       ; F2 IS ZERO.
44          ; CHECK AN SEE IF F1 IS ZERO.
45          LD K, B          ; RESTORE F1-R1 FROM B.
46          JSR FZCHK
47          IF C
48          JMP $CZERO       ; F1 IS ZERO.
49          ; GET HERE MEANS NORMAL MULTIPLICATION.
50          UNPACK F1 AND F2.

```

```

50 F6F6 ACC4CE          LD X, SP          ; X POINTS TO F1-L0.
51 F6F9 8210C4F8        ADD SP, 010       ; MOVE SP PAST LOCAL STORAGE.
52 F6FD AFCE            PUSH X           ; SAVE SP ON STACK FOR QUICK RETURN.
53 F6FF B5F95F          + JSR FUNPAK      ; UNPACK F1.
54 F702 AC00CC          LD B, TMP1       ; B NOW POINTS TO F2-R0.
55 F705 ACCE00          LD TMP1, X       ; TMP1 NOW POINTS TO F2-L0.
56 F708 E0             LOS A, W(B+)     ; LOAD F2-R0 INTO A.
57 F709 40             NOP
58 F70A AECA           X A, K
59 F70C E4             LD A, W(B)
60 F70D AECA           X A, K           ; LOAD F2-R1 INTO K.
61 F70F B5F94F          + JSR FUNPAK      ; UNPAK F2.
62                   ; SET X TO POINT TO F2-SIGN AND B TO POINT TO F1-SIGN.
63 F712 F2             LD A, W(X-)
64 F713 AC00CC          LD B, TMP1
65 F716 E2             LDS A, W(B-)
66 F717 40             NOP
67                   ; COMPUTE C-EXP AND C-SIGN AND STORE IN F2-EXP AND F2-SIGN.
68 F718 F4             LD A, W(X)       ; A IS (EEEEEEEE-F2).(SSSSSSSS-F2)
69 F719 C7             SHR A           ; SHR SINCE SUM OF EXPS CAN BE 9 BITS.
70 F71A ACC8CA          LD K, A         ; K IS (DEEEEEEEE-F2).(SSSSSSSS-F2)
71 F71D E4             LD A, W(B)       ; A IS (EEEEEEEE-F1).(SSSSSSSS-F1)
72 F71E B9FF00          AND A, OFF00    ; MASK OUT SIGN BITS.
73 F721 C7             SHR A           ; A IS (DEEEEEEEE-F1).(0000000)
74 F722 96CAF8          ADD A, K        ; A IS (EEEEEEEEEE-C).(SSSSSSSS-F2)
75 F725 F6             ST A, W(X)       ; STORE IN F2-SIGN.
76 F726 E2             LOS A, W(B-)     ; A IS (EEEEEEEE-F1).(SSSSSSSS-F1)
77 F727 40             NOP
78 F728 99FF          AND A, OFF       ; MASK OUT EXP BITS.
79 F72A C7             SHR A           ; A IS (00000000SSSSSSSS-F1)
80 F72B 8FFB          XOR A, W(X)       ; A IS (EEEEEEEESSSSSSSS-C)
81 F72D B8C080          ADD A, 0C080    ; REMOVE EXCESS BIAS OF 127-DEC FROM EXP.
82 F730 07             IF C
83 F731 46             JP $EXCH2        ; IF CARRY, THEN NO UNDERFLOW NOW
84                   ; CHECK TO SEE IF EXP IS ZERO. IF NOT, UNDERFLOW FOR SURE.
85 F732 E7             SHL A
86 F733 07             IF C
87 F734 B4F8E7          JMWL UNDFL      ; UNDERFLOW, SO JUMP.
88 F737 C7             SHR A           ; RESTORE BIT SHIFTED OUT (0).
89                   ; CHECK FOR EXPONENT OVERFLOW.
90 $EXCH2:
91 F738 E7             SHL A
92 F739 07             IF C           ; IF C IS 1,
93 F73A B4F8F2          JMWL OVRFL      ; THEN OVERFLOW FOR SURE.
94 F73D 96C817          IF A.7
95 F740 96C808          SET A.0        ; RESTORE LAST BIT OF SIGN.
96 F743 F3             X A, W(X-)     ; STORE C-EXP. C-SIGN IN F2-EXP.F2-SIGN.
97                   ;
98                   ; MULTIPLY THE MANTISSA.
99                   ; FIRST COMPUTE F1-HI*F2-HI.
100 F744 F2            LD A, W(X-)

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FMULT
FMULT.MAC

PAGE: 39

```

101 F745 ACCE00          LD TMP1, X          ; TMP1 NOW POINTS TO F2-L0.
102 F748 FE              MULT A, W(B)
103 F749 A6FFFAC4AB      ST A, W(SP-6)      ; STORE LOW WORD OF PRODUCT ON STACK.
104 F74E AECE            X A, X
105 F750 A6FFFC4AB       ST A, W(SP-4)      ; STORE HIGH WORD OF PRODUCT ON STACK.
106                      ; NOW COMPUTE F1-HI*F2-L0.
107 F755 AC00CE          LD X, TMP1
108 F758 FO              LD A, W(X+)
109 F759 ACCE00          LD TMP1, X          ; TMP1 NOW POINTS TO F2-HI.
110 F75C FE              MULT A, W(B)
111 F75D AECE            X A, X
112 F75F A6FFFAC4F8      ADD A, W(SP-6)      ; ADD LOW WORD OF LAST PROD. TO HIGH WORD.
113 F764 A6FFFAC4AB      ST A, W(SP-6)
114 F769 07              IF C
115 F76A A6FFFC4A9       INC W(SP-4)        ; IF CARRY, INCREASE HIGH WORD BY 1.
116                      ; FINALLY COMPUTE F1-L0*F2-HI.
117 F76F E2              LDS A, W(B-)        ; ADJUST B TO POINT TO F1-L0.
118 F770 40              NOP
119 F771 AC00CE          LD X, TMP1
120 F774 F4              LD A, W(X)
121 F775 FE              MULT A, W(B)
122 F776 AECE            X A, X
123 F778 A6FFFAC4F8      ADD A, W(SP-6)      ; ADD LOW WORD ACCUMULATED SO FAR.
124 F77D A6FFFAC4AB      ST A, W(SP-6)
125 F782 A6FFFC4AB       LD A, W(SP-4)      ; A CONTAINS HIGH WORD OF PRODUCT.
126 F787 07              IF C                ; IF CARRY ON LAST LOW WORD ADD,
127 F788 04              INC A                ; THEN INCREASE HIGH WORD.
128                      ;
129                      ; MANTISSA MULTIPLICATION DONE. NOW CHECK FOR NORMALIZATION.
130 F789 AC00CE          LD X, TMP1
131 F78C BD7FFF          IFGT A, 07FFF       ; IS MSB OF PRODUCT 1 ?
132 F78F 4D              JP $EXINC           ; YES, INCREASE MANTISSA.
133                      ; NEED TO SHIFT MANTISSA LEFT BY 1 BIT.
134 F790 E7              SHL A
135 F791 F3              X A, W(X-)
136 F792 A6FFFAC4AB      LD A, W(SP-6)
137 F797 E7              SHL A
138 F798 F1              X A, W(X+)
139 F799 07              IF C                ; DID SHIFT OF LOW WORD PUSH OUT A 1 ?
140 F79A 8F08            SET W(X).0          ; YES SO SET LSB OF HIGH WORD.
141 F79C 51              JP $ROUND           ; GO TO ROUNDING CODE.
142                      $EXINC:
143                      ; NEED TO INCREASE EXPONENT BY 1. REMEMBER X POINTS TO F2-HI.
144 F79D F3              X A, W(X-)          ; A CONTAINS HIGH WORD, X POINTS TO F2-L0.
145 F79E A6FFFAC4A8      LD A, W(SP-6)      ; GET LOW WORD.
146 F7A3 F1              X A, W(X+)          ; STORE LOW WORD.
147 F7A4 FO              LD A, W(X+)
148 F7A5 F4              LD A, W(X)          ; GET C-EXP.C-SIGN
149 F7A6 B80100          ADD A, 0100         ; INCREASE C-EXP.
150 F7A9 07              IF C
151 F7AA B4F882          JMWL OVRFL         ; EXPONENT OVERFLOW.

```

```

152 F7AD F3          X A, W(X-)      ; NO OVERFLOW, SO SAVE C-EXP.C-SIGN.
153                  ; ROUNDING CODE.
154                  $ROUND:
155 F7AE B5F8ED      JSRL SROUND
156                  ; FINAL CHECK OF EXPONENT.
157 F7B1 D0          LD A, M(X+)      ; X NOW POINTS TO C-EXP.
158 F7B2 D2          LD A, M(X-)
159 F7B3 9C00        IFEQ A, 0
160 F7B5 B4F866      JMPL UNDFL
161 F7B8 9DFE        IFGT A, OFE
162 F7BA B4F872      JMPL OVRFL
163 F7BD F2          LD A, W(X-)
164 F7BE F2          LD A, W(X-)      ; X NOW POINTS TO C-LO.
165 F7BF B5F8BA      + JSR FPAK      ; PACK C.
166 F7C2 3FC4        POP SP          ; SET UP SP FOR RETURN.
167 F7C4 3FCC        POP B
168 F7C6 3FCE        POP X
169 F7C8 3C          RET
170                  ; EXCEPTION HANDLING.
171                  ; C IS ZERO B'COS ONE OF F1 OR F2 IS ZERO.
172                  $CZERO:
173 F7C9 00          CLR A
174 F7CA ACC8CA      LD K, A
175 F7CD 3FCC        POP B
176 F7CF 3FCE        POP X
177 F7D1 3C          RET
178                  ;
179                  .END

```

```

37          .FORM 'FDIV.MAC'
38          .INCLD FDIV.MAC
1           .TITLE FDIV
2           .LOCAL
3
4           ;
5           ; SUBROUTINE TO DIVIDE TWO SP FLOATING POINT NUMBERS.
6           ;   C = F1/F2
7           ;
8           ; F1 IS STORED IN THE IEEE FORMAT IN REGS K AND A.
9           ; THE HIGH WORD OF F1 WILL BE REFERRED AS F1-R1 AND IS IN K.
10          ; THE LOW WORD OF F1 WILL BE REFERRED TO AS F1-RO AND IS IN A.
11          ;
12          ; F2 IS STORED IN THE IEEE FORMAT ON THE STACK. IF SP IS THE
13          ; STACK POINTER ON ENTRY, THEN
14          ; THE HIGH WORD OF F2, REFERRED TO AS F2-R1 IS AT SP - 4 AND
15          ; THE LOW WORD OF F2, REFERRED TO AS F2-RO IS AT SP - 6.
16          ;
17          ; C IS RETURNED IN THE IEEE FORMAT IN REGS K AND A.
18          ;
19          FDIV:
20          PUSH X
21          PUSH B
22          ; SAVE ADDRESS OF F2-RO IN TMP1.
23          LD X, SP
24          ADD X, OFFF6      ; SUBTRACT 10.
25          LD TMP1, X        ; AND SAVE IN TMP1.
26          ; CHECK AND SEE IF F1 IS A NAN.
27          JSR FNACHK
28          IF C
29          JNPL FNAN          ; F1 IS A NAN.
30          ; CHECK AND SEE IF F2 IS A NAN.
31          LD B, K
32          LD X, A
33          LD A, W(TMP1+2)
34          LD K, A
35          X A. X
36          JSR FNACHK
37          IF C
38          JNPL FNAN          ; F2 IS NAN.
39          ; CHECK AND SEE IF F2 IS ZERO.
40          JSR FZCHK
41          IF C
42          JNPL DIVBYO        ; F2 IS ZERO.
43          ; CHECK AND SEE IF F1 IS ZERO.
44          LD K, B            ; RESTORE F1-R1 FROM B.
45          JSR FZCHK
46          IF C
47          JMP $CZERO          ; F1 IS ZERO.
48          ; GET HERE MEANS NORMAL DIVISION.
49          ; UNPACK F1 AND F2.
          LD X, SP            ; X POINTS TO F1-LO.

```

```

50 F811 8210C4F8          ADD SP, 010      ; MOVE SP PAST LOCAL STORAGE.
51 F815 AFCE              PUSH X          ; SAVE SP ON STACK FOR QUICK RETURN.
52 F817 B5F847          + JSR FUNPAK      ; UNPACK F1.
53 F81A AC00CC          LD B, TMP1        ; B NOW POINTS TO F2-R0
54 F81D ACCE00          LD TMP1, X        ; TMP1 NOW POINTS TO F2-L0.
55 F820 E0              LDS A, W(B+)      ; LOAD F2-R0 INTO A.
56 F821 40              NOP
57 F822 AECA            X A, K
58 F824 E4              LD A, W(B)
59 F825 AECA            X A, K            ; LOAD F2-R1 INTO K.
60 F827 B5F837          + JSR FUNPAK      ; UNPAK F2.
61                      ;
62                      ; ENSURE THAT F1-HI IS LESS THAN F2-HI.
63                      ;
64 F82A F2              LD A, W(X-)        ; X POINTS TO F2-EXP.F2-SIGN.
65 F82B F2              LD A, W(X-)        ; X POINTS TO F2-HI.
66 F82C AC00CC          LD B, TMP1        ; B POINTS TO F2-L0.
67 F82F E2              LDS A, W(B-)      ; B POINTS TO F1-EXP.F1-SIGN.
68 F830 40              NOP
69 F831 E2              LDS A, W(B-)      ; LOAD F1-EXP.F1-SIGN.
70 F832 40              NOP              ; B POINTS TO F1-HI.
71 F833 ACC8CA          LD K, A            ; SAVE F1-EXP.F1-SIGN IN K.
72 F836 F4              LD A, W(X)        ; LOAD F2-HI.
73 F837 FD              IFGT A, W(B)      ; IS F2-HI > F1-HI ?
74 F838 51              JP $FEYSN         ; YES, SO ALL IS WELL.
75                      ; GET HERE MEANS NEED TO SHR F1,
76                      ; AND INCREASE ITS EXPONENT.
77 F839 E0              LOS A, W(B+)      ; GET F1-HI.
78 F83A 40              NOP              ; B POINTS TO F1-EXP.F1-SIGN.
79 F83B AECA            X A, K            ; SWAP F1-EXP.F1-SIGN AND F1-HI.
80 F83D B80100          ADD A, 0100       ; INCREASE F1-EXP BY 1.
81 F840 E3              XS A, W(B-)      ; STORE BACK IN F1-EXP.F1-SIGN.
82 F841 40              NOP              ; B POINTS TO F1-HI.
83 F842 E4              LD A, W(B)        ; LOAD F1-HI.
84 F843 C7              SHR A
85 F844 E3              XS A, W(B-)      ; STORE BACK IN F1-HI.
86 F845 40              NOP              ; B POINTS TO F1-L0.
87 F846 E4              LD A, W(B)        ; LOAD F1-L0.
88 F847 D7              RRC A
89 F848 E1              XS A, W(B+)      ; PUT IT BACK IN F1-L0.
90 F849 40              NOP              ; B POINTS TO F1-HI.
91                      ;
92                      $FEYSN:
93                      ; DETERMINE C-EXP AND C-SIGN.
94 F84A F0              LD A, W(X+)        ; X POINTS TO F2-EXP.F2-SIGN.
95 F84B E0              LDS A, W(B+)      ; B POINTS TO F1-EXP.F1-SIGN.
96 F84C 40              NOP
97 F84D F4              LD A, W(X)        ; LOAD F2-EXP.F2-SIGN.
98 F84E B9FF00          AND A, OFF00     ; MASK OUT THE SIGN.
99 F851 C7              SHR A            ; ALLOW 9 BITS FOR EXP CALCULATIONS.
100 F852 ACC8CA          LD K, A          ; SAVE IT IN K.

```

```

101 F855 E4          LD A, W(B)      ; LOAD F1-EXP.F1-SIGN.
102 F856 B9FF00      AND A, OFF00    ; MASK OUT SIGN.
103 F859 C7          SHR A
104 F85A 02          SET C
105 F85B 96CAEB      SUBC A, K        ; SUBTRACT THE EXPONENTS.
106                  ; NOTE THAT NOW THE MS 9 BITS
107                  ; OF A CONTAIN A 2'S COMP. INTEGER.
108 F85E B7000000    LD TMP1, 0
109 F862 E7          SHL A
110 F863 07          IF C
111 F864 B700FF00    LD TMP1, OFF
112 F868 D7          RRC A            ; SAVE SIGN OF NUMBER IN TMP1.
113 F869 B83F00      ADD A, 03F00     ; RESTORE IEEE BIAS.
114 F86C E7          SHL A            ; MAKE EXPONENT 8 BITS.
115 F86D 06          IFN C            ; NO CARRY ?
116 F86E 49          JP $FSIGN        ; THEN ALL IS WELL.
117 F86F 960010      IF TMP1.0        ; WAS EXP NEGATIVE BEFORE ?
118 F872 B4F7A9      JMPL UNDFL       ; YES, SO UNDERFLOW.
119 F875 B4F7B7      JMPL OVRFL       ; OTHERWISE OVERFLOW.
120                  ;
121                  $FSIGN:
122                  ; C-EXP HAS BEEN COMPUTED. NOW FIND C-SIGN.
123                  ; BUT FIRST TAKE CARE OF SPECIAL OVER/UNDERFLOW CASES.
124 F878 BCFF00      IFEQ A, OFF00
125 F87B B4F7B1      JMPL OVRFL
126 F87E 9C00        IFEQ A, 0
127 F880 B4F79B      JMPL UNDFL
128 F883 AB00        ST A, TMP1        ; SAVE C-EXP.00000000 IN TMP1.
129 F885 F4          LD A, W(X)        ; LOAD F2-EXP.F2-SIGN.
130 F886 99FF        AND A, OFF        ; MASK OUT F2-EXP.
131 F888 FB          XOR A, W(B)        ; A NOW HAS F1-EXP.C-SIGN.
132 F889 99FF        AND A, OFF        ; MASK OUT F1-EXP.
133 F88B 9600FA      OR A, TMP1        ; BRING IN C-EXP.
134 F88E F3          X A, W(X-)        ; STORE IN F2-EXP.F2-SIGN.
135                  ; X POINTS TO F2-HI.
136 F88F F2          LD A, W(X-)        ; X POINTS TO F2-LO.
137 F890 E2          LDS A, W(B-)      ; B POINTS TO F1-HI.
138 F891 40          NOP
139                  ;
140                  ; NOW DO THE MANTISSA DIVISION.
141                  ;
142 F892 F0          LD A, W(X+)        ; LOAD F2-LO. X POINTS TO F2-HI.
143 F893 ACCE00      LD TMP1, X        ; SAVE ADDRESS OF F2-HI IN TMP1.
144 F896 FE          MULT A, W(B)       ; COMPUTE F2-LO*F1-HI.
145                  ; X CONTAINS MS WORD AND A IS LS WORD.
146                  ;
147 F897 AECC        X A, B            ; A POINTS TO F1-HI, B CONTAINS LS WORD.
148 F899 AE00        X A, TMP1         ; A POINTS TO F2-HI, TMP1 POINTS TO F1-HI.
149 F89B AECC        X A, B            ; A CONTAINS LS WORD, B POINTS TO F2-HI.
150                  ;
151 F890 EF          .BYTE OEF         ; DIVD A, W(B) - KLUDGED !!

```

```

152          ;
153 F89E ACC8CA      LD K, A          ; SAVE QUOTIENT IN K.
154 F8A1 A8CC        LD A, B          ; A POINTS TO F2-HI.
155 F8A3 AE00        X A, TMP1       ; A POINTS TO F1-HI, TMP1 POINTS TO F2-HI.
156 F8A5 AECC        X A, B          ; B POINTS TO F1-HI.
157 F8A7 E2          LDS A, W(B-)    ; B POINTS TO F1-LO.
158 F8A8 40          NOP
159 F8A9 E0          LOS A, W(B+)    ; LOAD F1-LO.
160 F8AA 40          NOP             ; B POINTS TO F1-HI.
161 F8AB 02          SET C
162 F8AC 96CAEB      SUBC A, K        ; SUBTRACT QUOTIENT SAVED IN K.
163 F8AF ACC8CE      LD X, A          ; AND SAVE IN X.
164 F8B2 E4          LD A, W(B)      ; LOAD F1-HI.
165 F8B3 06          IFN C           ; IF C WAS NOT SET IN THE LAST SUBTRACT,
166 F8B4 05          DEC A            ; ADJUST THE BORROW.
167 F8B5 AECE        X A, X
168 F8B7 AC00CC      LD B, TMP1      ; B POINTS TO F2-HI.
169          ;
170 F8BA EF          .BYTE OEF       ; DIVD A, W(B) - KLUDGED AGAIN !
171          ;                     ; QUOTIENT IN A, REM IN X.
172          ;
173 F8BB AB00        ST A, TMP1      ; SAVE QUOTIENT IN TMP1.
174 F8BD 00          CLR A            ; ZERO A.
175          ;
176 F8BE EF          .BYTE OEF       ; DIVD A, W(B) - KLUDGED YET AGAIN !
177          ;
178 F8BF AE00        X A, TMP1      ; SWAP OLD AND NEW QUOTIENTS.
179          ;
180          ; CHECK FOR NORMALIZATION. CAN BE OFF BY AT MOST 1 BIT.
181 F8C1 E7          SHL A
182 F8C2 07          IF C
183 F8C3 56          JP $NMED         ; IT IS NORMALIZED.
184          ;                     ; GET HERE MEANS NEED TO SHIFT LEFT ONCE.
185 F8C4 AE00        X A, TMP1      ; SWAP HI AND LO WORDS.
186 F8C6 E7          SHL A
187 F8C7 AE00        X A, TMP1      ; HI WORD IS IN A, LO WORD IN TMP1.
188 F8C9 07          IF C           ; WAS 1 SHIFTED OUT OF LO WORD?
189 F8CA 96C808      SET A.0         ; YES, THEN SET LSB OF HI WORD.
190 F8CD ABCA        ST A, K         ; SAVE HI WORD IN K.
191 F8CF E1          XS A, W(B+)
192 F8D0 40          NOP             ; B POINTS TO F2-EXP.F2-SIGN.
193 F8D1 E4          LD A, W(B)      ; LOAD F2-EXP.F2-SIGN.
194 F8D2 B8FF00      ADD A, OFF00    ; SUBTRACT 1 FROM EXPONENT.
195 F8D5 E3          XS A, W(B-)    ; STORE BACK IN F2-EXP.F2-SIGN.
196 F8D6 40          NOP             ; B POINTS TO F2-HI.
197 F8D7 A8CA        LD A, K         ; HI WORD TO A.
198 F8D9 E7          SHL A
199          $NMED:
200 F8DA D7          RRC A            ; RESTORE BIT OUT.
201 F8DB E3          XS A, W(B-)    ; SAVE HI-WORD IN F2-HI.
202 F8DC 40          NOP             ; B POINTS TO F2-LO.

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
FDIV
FDIV.MAC

PAGE: 45

```

203 F8DD A800          LD A, TMP1
204 F8DF E1            XS A, W(B+)      ; SAVE C-LO.
205 F8E0 40            NOP              ; B POINTS TO F2-HI.
206 F8E1 ACCCCE        LD X, B          ; MOVE ADDRESS OF F2-HI TO X.
207                    ;
208                    ; ROUNDING CODE.
209 F8E4 B5F7B7        JSRL SROUND
210                    ; FINAL CHECK OF EXPONENT.
211 F8E7 D0            LD A, M(X+)      ; X NOW POINTS TO C-EXP.
212 F8E8 D2            LD A, M(X-)
213 F8E9 9C00          IFEQ A, 0
214 F8EB B4F730        JMPL UNDFL
215 F8EE 9DFE          IFGT A, OFE
216 F8F0 B4F73C        JMPL OVRFL
217 F8F3 F2            LD A, W(X-)
218 F8F4 F2            LD A, W(X-)      ; X NOW POINTS TO C-LO.
219 F8F5 B5F784        JSR FPAK         ; PACK C.
220 F8F8 3FC4          POP SP           ; SET UP SP FOR RETURN.
221 F8FA 3FCC          POP B
222 F8FC 3FCE          POP X
223 F8FE 3C            RET
224                    ; C IS ZERO B'COS F1 IS ZERO.
225                    $CZERO:
226 F8FF 00            CLR A
227 F900 ACC8CA        LD K, A
228 F903 3FCC          POP B
229 F905 3FCE          POP X
230 F907 3C            RET
231                    ;
232                    .END

```

```

39          .FORM 'FSINX.MAC'
40          .INCLD FSINX.MAC
1           ;
2           .TITLE SINX
3           .LOCAL
4           ; A VERY DIRTY APPROXIMATION TO SIN(X).
5           ; X SHOULD BE IN RADIANS.
6           ;
7           ; ON INPUT X SHOULD BE IN IEEE FLP FORMAT IN REGS. K AND A.
8           ; ON RETURN SIN(X) IS IN IEEE FLP FORMAT IN REGS. K AND A.
9           ;
10          SINX:
11 F908 AFCE          PUSH X          ; SAVE X.
12 F90A AFC8          PUSH A
13 F90C AFCA          PUSH K          ; X TO THE STACK.
14 F90E 3653          - JSRL FMULT      ; COMPUTE X^2.
15 F910 AFC8          PUSH A
16 F912 AFCA          PUSH K          ; X^2 TO THE STACK.
17 F914 B6F994A8      LD A, W($A5LO)
18 F918 A4F996CAAB    LD K, W($A5HI)  ; LOAD A5.
19 F91D 3662          - JSRL FMULT      ; COMPUTE A5*X^2.
20 F91F AFC8          PUSH A
21 F921 AFCA          PUSH K
22 F923 B6F998A8      LD A, W($A4LO)
23 F927 A4F99ACAAB    LD K, W($A4HI)  ; LOAD A4.
24 F92C B5FEE6        JSRL FSUB        ; COMPUTE A4-A5*X^2.
25 F92F 3FCE          POP X
26 F931 3FCE          POP X
27 F933 3678          - JSRL FMULT      ; COMPUTE
28                                     ; X^2(A4 - A5*X^2).
29 F935 AFC8          PUSH A
30 F937 AFCA          PUSH K
31 F939 B6F99CA8      LD A, W($A3LO)
32 F93D A4F99ECAAB    LD K, W($A3HI)  ; LOAD A3.
33 F942 B5FBD0        JSRL FSUB        ; COMPUTE
34                                     ; A3 - X^2(A4 - A5*X^2).
35 F945 3FCE          POP X
36 F947 3FCE          POP X
37 F949 368E          - JSRL FMULT      ; COMPUTE
38                                     ; X^2(A3 - X^2(A4 - A5*X^2)).
39 F94B AFC8          PUSH A
40 F94D AFCA          PUSH K
41 F94F B6F9A0A8      LD A, W($A2LO)
42 F953 A4F9A2CAAB    LD K, W($A2HI)  ; LOAD A2.
43 F958 B5FBBA        JSRL FSUB        ; COMPUTE
44                                     ; A2 - X^2(A3 - X^2(A4 - A5*X^2)).
45 F95B 3FCE          POP X
46 F95D 3FCE          POP X
47 F95F 36A4          - JSRL FMULT      ; COMPUTE
48                                     ; X^2(A2 - X^2(A3 - X^2(A4 - A5*X^2))).
49 F961 AFC8          PUSH A

```

```

50 F963 AFCA          PUSH K
51 F965 B6F9A4AB      LD A, W($A1L0)
52 F969 A4F9A6CAAB    LD K, W($A1HI) ; LOAD A1.
53 F96E B5FBA4        JSRL FSUB      ; COMPUTE
54                      ; A1 - X^2(A2 - X^2(A3 - X^2(A4 - A5*X^2))).
55 F971 3FCE          POP X
56 F973 3FCE          POP X
57 F975 36BA          - JSRL FMULT    ; COMPUTE
58                      ; X^2(A1 - X^2(A2 - X^2(A3 - X^2(A4 - A5*X^2))).
59 F977 AFC8          PUSH A
60 F979 AFCA          PUSH K
61 F97B B13F80        LD K, 03F80
62 F97E 00            CLR A          ; LOAD 1.0 INTO K-A.
63 F97F B5FB93        JSRL FSUB      ; COMPUTE
64                      ; 1 - ALL THE JUNK ABOVE.
65 F982 3FCE          POP X
66 F984 3FCE          POP X
67 F986 3FCE          POP X
68 F988 3FCE          POP X          ; NOW X IS AT THE TOP OF STACK.
69 F98A 36CF          - JSRL FMULT    ; COMPUTE
70                      ; X(1 - X^2(A1 - X^2(A2 - X^2(A3 - X^2(A4 - A5*X^2)))).
71 F98C 3FCE          POP X
72 F98E 3FCE          POP X
73 F990 3FCE          POP X
74 F992 3C            RET
75                      ;
76 F993 40            .EVEN
77                      ;
78 F994 2B32          $A5L0: .WORD 0322B
79 F996 D732          $A5HI: .WORD 032D7
80 F998 1DEF          $A4L0: .WORD 0EF1D
81 F99A 3836          $A4HI: .WORD 03638
82 F99C 010D          $A3L0: .WORD 00D01
83 F99E 5039          $A3HI: .WORD 03950
84 F9A0 8988          $A2L0: .WORD 08889
85 F9A2 083C          $A2HI: .WORD 03C08
86 F9A4 ADAA          $A1L0: .WORD 0AAAD
87 F9A6 2A3E          $A1HI: .WORD 03E2A
88                      ;
89                      ; A DIRTY APPROXIMATION TO COS(X) USING SIN(X).
90                      ;
91 COSX:
92 F9A8 AFCE          PUSH X
93 F9AA ACC8CE        LD X, A
94 F9AD B6F9C8A8      LD A, W($PI2L0)
95 F9B1 AFC8          PUSH A
96 F9B3 B6F9CAA8      LD A, W($PI2HI)
97 F9B7 AFC8          PUSH A
98 F9B9 A8CE          LD A, X
99 F9BB B5FB77        JSRL FADD      ; COMPUTE X + PI/2.
100 F9BE 3FCE         POP X

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
SINX
FSINX.MAC

PAGE: 48

```

101 F9C0 3FCE          POP X
102 F9C2 34BA          -   JSRL SINX          ; COMPUTE SIN(X+PI/2).
103 F9C4 3FCE          POP X
104 F9C6 3C            RET
105                    ;
106 F9C7 40            .EVEN
107 F9C8 DB0F          $PI2LO: .WORD 00FDB
108 F9CA C93F          $PI2HI: .WORD 03FC9
109                    ;
110                    ; A DIRTY APPROXIMATION TO TAN(X) USING SINX AND COSX.
111                    ;
112                    TANX:
113 F9CC AFCE          PUSH X
114 F9CE AFCC          PUSH B
115 F9D0 AFC8          PUSH A
116 F9D2 AFCA          PUSH K
117 F9D4 342C          JSR COSX          ; COMPUTE COS(X)
118 F9D6 ACC8CE        LD X, A
119 F9D9 ACCACC        LD B, K
120 F9DC 3FCA          POP K
121 F9DE 3FC8          POP A
122 F9E0 AFCE          PUSH X
123 F9E2 AFCC          PUSH B
124 F9E4 34DC          JSR SINX          ; COMPUTE SIN(X).
125 F9E6 3614          JSR FDIV          ; COMPUTE TAN(X) = SIN(X)/COS(X).
126 F9E8 3FCC          POP B
127 F9EA 3FCC          POP B
128 F9EC 3FCC          POP B
129 F9EE 3FCE          POP X
130 F9F0 3C            RET
131                    ;
132                    .END
41                    ;
42                    ;
43 FFFE 00F0          .END LISTER

```

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
SINX
SYMBOL TABLE

PAGE: 49

A	00C8 W	ATOF	F13A *	B	00CC W	BFMUL	F0C7
COSX	F9A8	DIVBYO	F000	FADD	F535	FDIV	F7D2
FMULT	F6BB	FNACHK	F040	FNAN	F00F	FPAC	F07C
PPERWD	0002 W	FPTRAP	F09D	FSUB	F515	FTOA	F311 *
FUNPAK	F061	FZCHK	F051	ISIOK	F105	K	00CA W
LISTER	F000	MUL10	F118	OVRFL	F02F	PC	00C6 W
SINX	F908	SP	00C4 W	SROUND	F09E	TANX	F9CC *
TMP1	0000 W	UNDFL	F01E	X	00CE W	\$A10EX	F1F4
\$A1HI	F9A6	\$A1LO	F9A4	\$A2HI	F9A2	\$A2LO	F9A0
\$A3HI	F99E	\$A3LO	F99C	\$A4HI	F99A	\$A4LO	F998
\$A5HI	F996	\$A5LO	F994	\$ACCF	F1A9	\$ACCM	F177
\$ADDEX	F1FF	\$ADDMN	F612	\$ADEM	F3EC	\$ADJEX	F685
\$ANORM	F64A	\$ANOTO	F05C	\$CHKOT	F113	\$CHNGS	F361
\$CSIGN	F349	\$CZERO	F7C9	\$CZERO	F8FF	\$DIV10	F272
\$DIV10	F370	\$DOLUP	F481	\$DTHI	F270	\$DTHI	F392
\$DTLO	F26E	\$DTLO	F390	\$ESAVE	F20F	\$ESIGN	F1C7
\$EXACC	F1C9	\$EXCH2	F738	\$EXCHR	F1BB	\$EXCLP	F1D4
\$EXCOL	F1BA	\$EXCPT	F2CF	\$EXIN2	F0B7	\$EXINC	F79D
\$EXIT	F0C6	\$F1CHK	F56E	\$F1SIN	F643	\$F2GTR	F5EB
\$TEXSN	F84A	\$FRCOL	F18B	\$FSIGN	F878	\$GOBAK	F41F
\$GOON	F42B	\$HIUP	F0AF	\$INCOL	F15C	\$INCRV	F40C
\$INDUN	F445	\$INTFY	F43C	\$ISNAN	F04C	\$ISNED	F298
\$ISNED	F3BE	\$ISNXT	F17D	\$JAMDN	F29E	\$JAMDN	F3C4
\$JAMIT	F280	\$JAMIT	F3A6	\$JAMLP	F287	\$JAMLP	F3AD
\$LOOP1	F147	\$LOOP1	F5FB	\$LOOP2	F5D7	\$ML4	F3EA
\$MLOG2	F33F	\$MSIGN	F154	\$MTHI	F26C	\$MTHI	F396
\$MTLO	F26A	\$MTLO	F394	\$MUL10	F398	\$NAGAS	F2C3
\$NAN	F4CF	\$NANLP	F4DC	\$NEG10	F20B	\$NLOOP	F653
\$NMED	F8DA	\$NORM1	F230	\$NORM2	F235	\$NRDUN	F247
\$NRLUP	F237	\$NTZER	F585	\$OV1	F666	\$OVR1	F29B
\$OVR1	F3C1	\$PI2HI	F9CA	\$PI2LO	F9C8	\$REMV9	F354
\$RNDUP	F0A7	\$ROUND	F6A0	\$ROUND	F7AE	\$TRADD	F66C
\$VDOWN	F418	\$VUP	F416	\$ZERLP	F505	\$ZERO	F4E9
\$ZROF1	F607	\$ZROF2	F5E2				

NATIONAL SEMICONDUCTOR CORPORATION
HPC CROSS ASSEMBLER,REV:C,30 JUL 86
SINK
MACRO TABLE

PAGE: 50

NO WARNING LINES

NO ERROR LINES

2547 ROM BYTES USED

SOURCE CHECKSUM = A31F
OBJECT CHECKSUM = 2AC3

INPUT FILE C:LISTER.MAC
LISTING FILE C:LISTER.PRN
OBJECT FILE C:LISTER.LM

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

 National Semiconductor Corporation 1111 West Bardin Road Arlington, TX 76017 Tel: 1(800) 272-9959 Fax: 1(800) 737-7018	National Semiconductor Europe Fax: (+49) 0-180-530 85 86 Email: cnjwge@tevm2.nsc.com Deutsch Tel: (+49) 0-180-530 85 85 English Tel: (+49) 0-180-532 78 32 Français Tel: (+49) 0-180-532 93 58 Italiano Tel: (+49) 0-180-534 16 80	National Semiconductor Hong Kong Ltd. 19th Floor, Straight Block, Ocean Centre, 5 Canton Rd. Tsimshatsui, Kowloon Hong Kong Tel: (852) 2737-1600 Fax: (852) 2736-9960	National Semiconductor Japan Ltd. Tel: 81-043-299-2309 Fax: 81-043-299-2408
---	---	--	--

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.