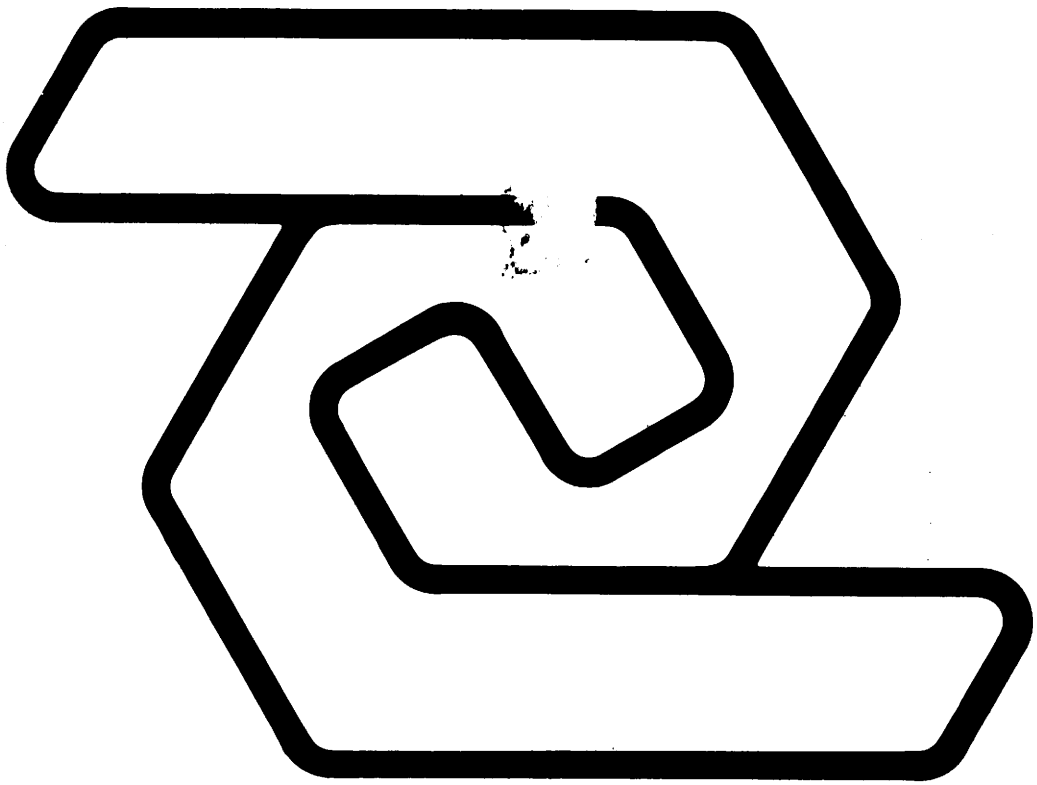

U S Software

FPAC

FPAC

Floating Point Library User's Guide

INTRODUCTION

The FPAC Floating Point Library provides its users with floating point computation, conversion, and utility procedures, based on the proposed KCS IEEE Standard single precision floating point format. By using the FPAC library, U S Software's customers can create their products more quickly, with less debugging and product life costs, in conformance to the proposed standard.

The FPAC library consists of the basic floating point operations (ADD, SUBTRACT, MULTIPLY, DIVIDE), transcendental functions (SIN, COS, TAN, ATN, LN, LOG, EXP, SQR), data conversion routines (ASCII to/from Floating Point, Floating Point to/from word integers), and various floating point utility procedures.

The FPAC library was designed to emphasize accuracy, source code clarity, code size efficiency, and execution speed. Wherever IEEE single precision proposed standards exist, and when they are feasible to implement with respect to the scope and purpose of this package, the FPAC libary adheres to them. All procedures have been thoroughly tested to assure proper operation.

Proposed K-C-S IEEE Single Precision Floating Point Standard

The format on which the FPAC Library operates is the proposed IEEE single precision standard. Its representation in bit form is:

S EEE EEEE	E MMM MMMM	MMMM MMMM	MMMM MMMM
byte 3	byte 2	byte 1	byte 0

"S" is the sign bit (1 if negative, 0 if positive)

The "E" field is the two's exponent. It is a two's complement value biased by 127 (decimal).

The "M" field is the 23-bit normalized mantissa. The most significant bit is always assumed to be 1, and so is not explicitly stored. This yields an effective precision of 24 bits.

The value of the floating point number described above is obtained by multiplying 2 raised to the power of the unbiased exponent, by the binary mantissa. The assumed bit of the binary mantissa (the most significant bit) has a value of 1.0, with the remaining bits providing a fractional value (i.e., the value of the mantissa is greater than or equal to 1.0 and less than 2.0).

Note that the four bytes of the floating point number are stored in lexicographic order. (In the case of LO/HI machines such as the Z-80 and 8085, this means the sign/exponent byte is stored at a higher memory address than the mantissa bytes.)

The dynamic range of the K-C-S single precision floating point format is $\pm$ 1.175494E−38 to 3.402823E+30.

Description and Use of FPAC Library Routines

A. Condensed and Working Representations

The FPAC Library operates on IEEE Standard format numbers. For the sake of execution speed, these numbers must be expanded from their 4 byte (compressed or packed) format into 5 byte (expanded or unpacked) format prior to being operated upon.

The expanded format is:

SSSS SSSS EEEE EEEE MMMM MMMM MMMM MMMM MMMM MMMM

The sign bit is replicated to fill a byte, the exponent is collected into a single byte, and the assumed bit of the mantissa is explicitly included.

When floating point operations have been completed, the results may be compressed into the 4 byte representation.

B. Pseudo Register and Floating Point Operation

The FPAC Library uses several dedicated memory locations when performing operations. The routines which ADD/SUBTRACT, MULTIPLY, and DIVIDE have operands residing in the 5 byte pseudo registers TOSBASE and NOSBASE, and produce the result in a special extended precision register RSLTBAS (the contents of TOSBASE and NOSBASE are not changed by these operations). The transcendental functions expect their arguments in NOSBASE and return their result in TOSBASE. Most of the other FPAC routines allow arbitrarily-located 5 byte values, but generally TOSBASE or NOSBASE are used.

FPAC internally allocates space for these pseudo registers. Their names, and associated subfields, are entry points so that the user may directly operate upon them. See the list of globals in the appendix for further information.

C. Data Translation and Transfer Routines

Translation and transfer routines exist to move values among the various formats mentioned above.

FROM	TO: Compressed	TO: Expanded
Compressed	—	UNPACK
Expanded	PACK	XPNCPY
RSLTBAS	RSLTPAC	RSLTCPY

1. 4 Byte (Compressed) Format to/from 5 Byte (Expanded) Format

The routines to perform these operations are named PACK and UNPACK for MICRO language calls, and PACKA and UNPACKA for Assembly language calls. The examples below assume CMPFPN is a 4 byte floating point number and XPNFPN is a 5 byte expanded format floating point number.

8085 Assembly Language	MICRO Language
LXI H,CMPFPN LXI D,XPNFPN CALL UNPACKA --- XPNFPN now holds the expanded form of CMPFPN ---	CALL UNPACK(CMPFPN, XPNFPN)
LXI H,XPNFPN LXI D,CMPFPN CALL PACKA --- CMPFPN now holds the compressed form of XPNFPN ---	CALL PACK(XPNFPN, CMPFPN)

The Assembly language routines (UNPACKA and PACKA) alter all registers. The MICRO language callable routines are, of course, compatible with the call and return sequences that MICRO expects.

2. Expanded Format Copy Routines

It is often necessary to shuttle expanded (5 byte) quantities around. The FPAC Library contains Assembly language callable routines to facilitate this (MICRO has inherent language constructions that perform these operations). Additionally, since it is often desirable to move quantities into TOSBASE and NOSBASE for floating point operations, routines to do this are provided in the FPAC Library.

8085 Assembly Language

```
LXI   H,XPNFPN1        ; Copy XPNFPN1 into XPNFPN2
LXI   D,XPNFPN2
CALL XPNCPY

LXI   H,XPNFPN1        ; Copy XPNFPN1 into TOSBASE
CALL XPNCPYT

LXI   H,CONST2         ; Copy CONST2 into NOSBASE
CALL XPNCPYN

LXI   H,NOSBASE        ; Save the operand in NOSBASE
LXI   D,XPNFPN2
CALL XPNCPY
```

3. Convert the Extended Precision Result into an Expanded Format

Several routines which re-format the contents of the extended precision result register into expanded (5 byte) format are available to the user. The user may copy the value in the extended precision result register into any expanded format (5 byte) variable (usually, this is to save an intermediate result for later use) or into one of the operand registers to be used in the next floating point operation.

8085 Assembly Language

```
LXI   D,IMMRSLT        ; Save an intermediate result
CALL RSLTCPY

CALL RSLT2T            ; Move the result into TOSBASE

CALL RSLT2N            ; Move the result into NOSBASE

LXI   D,TOSBASE        ; Functional equivalent of RSLT2T
CALL RSLTCPY
```

4. Convert the Extended Precision Result into a Compressed Format

The user may wish to directly move a value from the result register into a compressed format (4 byte) variable. The FPAC Library contains two routines, one Assembly language callable and one MICRO language callable, to do this.

8085 Assembly Language	MICRO Language
LXI D,CMPFPN CALL RSLTPAA	CALL RSLTPAC(CMPFPN)

Basic Floating Point Operations

A. Addition and Subtraction

The values in TOSBASE and NOSBASE are added to produce a sum in the extended precision result register. Subtraction is accomplished by complementing the sign of the value to be subtracted prior to calling the addition routine.

The examples below illustrate, first, the addition of two compressed format values, the result stored in a compressed format variable; second, the subtraction of the same values (by complementing the expanded sign byte then adding), the result again stored in a compressed format variable.

8085 Assembly Language	MICRO Language
LXI H,VAL1	CALL UNPACK(VAL1, TOSBASE)
LXI D,TOSBASE	CALL UNPACK(VAL2, NOSBASE)
CALL UNPACKA	CALL FPADD
LXI H,VAL2	CALL RSLTPAC(SUM)
LXI D,NOSBASE	
CALL UNPACKA	
CALL FPADD	
LXI D,SUM	
CALL RSLTPAA	
LDA TOSSGN	TOSBASE. SGN = NOT TOSBASE. SGN
CMA	CALL FPADD
STA TOSSGN	CALL RSLTPAC(DIF)
CALL FPADD	
LXI D,DIF	
CALL RSLTPAA	

B. Multiplication

The expanded format values in TOSBASE and NOSBASE are multiplied, the product being left in the extended precision result register. The example below illustrates a compressed format value being squared, then stored for use as an intermediate result. Be sure to read the note following the examples.

8085 Assembly Language	MICRO Language
LXI H,X	CALL UNPACK(X, TOSBASE)
LXI D,TOSBASE	NOSBASE = TOSBASE
CALL UNPACKA	CALL FPMUL
LXI H,TOSBASE	CALL RSLTPAC(IMMVAL)
CALL XPNCPYN	
CALL FPMUL	
LXI D,IMMRSLT	
CALL RSLTCPY	

--- NOTE: The Assembly code segment stores the product as an expanded format value, while the MICRO segment stores the product in compressed format ---

C. Division

The expanded format value in NOSBASE is divided by the expanded format value in TOSBASE, with the quotient placed in the extended precision result register. The examples below illustrate a compressed format value being divided by the intermediate result in the format produced by the respective segment in the multiply example.

8085 Assembly Language	MICRO Language
LXI H,INVAL	CALL UNPACK(INVAL, NOSBASE)
LXI D,NOSBASE	CALL UNPACK(IMMVAL, TOSBASE)
CALL UNPACKA	CALL FPDIV
LXI H,IMMRSLT	CALL RSLTPAC(ANSWER)
CALL XPNCPYT	
CALL FPDIV	
LXI D,ANSWER	
CALL RSLTPAA	

--- NOTE: The intermediate values used here are the same as produced in the multiply example. Thus, the Assembly language intermediate (IMMRSLT) is an expanded format while the MICRO language intermediate (IMMVAL) is a compressed format value ---

D. Conversion from Word Integer to Expanded Format Floating Point Value

A word size integer (16 bits) is converted into an expanded format floating point value by two routines, one callable from Assembly language and one callable from MICRO.

8085 Assembly Language	MICRO Language
LHLD VALUE	CALL FLOAT(VALUE, XPNFPN1)
LXI D,XPNFPN1	
CALL FLOATA	

--- NOTE: The Assembly callable routine expects the integer value to be in HL (not the address of the value as in previously described routines). The MICRO callable routine assumes the user has taken the necessary steps to insure that the first parameter given is a word (two byte) integer ---

E. Conversion from Expanded Format Floating Point Value to Word Integer

Two different routines exist to convert from an expanded floating point value to a word integer (both are callable, in slightly different forms, from Assembly language and MICRO language). The difference between the two routines, FIX and INT, is illustrated in the table below:

F.P. Value	FIX(val)	INT(val)
3.5	3	3
−3.5	−3	−4

The result of a FIX operation is the argument value stripped of its fractional part. The result of an INT operation is the largest integer such that it is less than or equal to the argument value.

The examples below illustrate the use of these routines.

8085 Assembly Language

```
LXI   H,XPNFPN1
CALL FIXA
XCHG
SHLD WRDFIX
```

MICRO Language

```
CALL FIX(XPNFPN1, WRDFIX)
```

8085 Assembly Language

```
LXI   H,XPNFPN1
CALL INTA
XCHG
SHLD WRDINT
```

MICRO Language

```
CALL INT(XPNFPN1, WRDINT)
```

--- NOTE: FIXA and INTA return values in DE ---

F. Floating Point INT Function

For some applications, it is necessary to perform an INT-style operation in floating point form. If the operation is performed in the floating point domain, a significant increase in the dynamic range is possible; the largest word integer is 32,767 while the largest floating point number that may have a fractional part is 8,388,607. Two routines are provided to do this.

8085 Assembly Language

```
LXI   H,XPNFPN1
LXI   D,AINTFPN
CALL AINTA
```

MICRO Language

```
CALL AINT(XPNFPN1, AINTFPN)
```

--- NOTE: This operation may be done "in place", i.e., the source and destination operand may be the same. Both operands, however, must be expanded format floating point values ---

ASCII Interpretation to/from Floating Point Value

A. ASCII Literal to Compressed Format Floating Point Value

These utilities convert an ASCII literal into a compressed format floating point value. The first parameter is a pointer to the first character of the literal to be interpreted. This first character of the literal must be either a minus sign (indicating a negative number), a decimal point, or a digit; these routines will not skip preceding blanks.

The first character of the literal field plus subsequent characters must form a valid decimal number, optionally followed immediately by "E" (signifying scientific notation). If an "E" is found, it may be followed by a plus or minus sign, then one or two digits (the sign and digits indicating the power of ten scaling).

The routine will process characters until an improper character is found. This "improper" character may be a blank, comma, zero byte, etc. When two digits have been found after the "E", the next character is automatically improper.

An error flag is returned by the routine (non-zero indicates an error). Note that reaching an improper character is NOT an error condition; it can indicate the correct termination of the ASCII literal. Errors include no digits in the literal, no digits preceeding the "E", no digits following the "E", or a literal that, when interpreted, is too large to represent.

Examples of ASCII Literals and Results of the Conversion Routine

3.567,	Value is 3.567	No error	Pointer at ","
−.5	Value is −0.5	No error	Pointer at byte after "5"
5E4	Value is 50,000	No error	Pointer at byte after "4"
−.E4	Value is 0	Error	Pointer at byte after "4"
3.1.2	Value is 3.1	No error	Pointer at second "."
4.2E 13	Value is 4.2	Error	Pointer at blank following "E"
−6E98	Value is −INF	Error	Pointer at byte after "8"
	Value is 0	Error	Pointer at " " (unchanged)

^ initial pointer position

The example that follows illustrate using the conversion routine:

8085 Assembly Language	MICRO Language
LHLD PNTR	PNTR = ^INLINE
LXI D,CMPFPN1	CALL ASCBIN(PNTR, CMPFPN1, ERRFLG)
CALL ASCBINA	IF ERRFLG THEN...
ANA A	
JNZ INPERR	
SHLD PNTR	

--- NOTE: The Assembly callable routine expects HL to point to the first byte of the ASCII literal and DE to point to the compressed format floating point value. It returns the error flag in A. The MICRO callable routine expects the first parameter to be a two byte pointer variable, the second parameter to be a compressed floating point variable, and the third parameter to be a single byte variable ---

B. Expanded Format Floating Point Value to ASCII Literal

Two routines are provided to convert the value in the expanded format floating point variable NOSBASE into a zero byte terminated string of ASCII characters at the location REALLIT (see list of globals in appendix A). The first character is either a minus sign (for a negative value) or a blank, depending on the sign of the number being converted. Based on the value being converted, one of the following formats is selected:

Form	Value Range		
0.n	0.1	to 0.9999999	
N.n	1.0	to 9.999999	
NN.n	10.0	to 99.99999	
NNN.n	100.0	to 999.9999	
NNNN.n	1000.0	to 9999.999	
NNNNN.n	10000.0	to 99999.99	
NNNNNN.n	100000.0	to 999999.9	
NNNNNNN.	1000000.0	to 9999999.	
N.nE#dd	other valid number		
0.	underflow		(error condition, see appendix E)
+ INF	positive infinity		(error condition, see appendix E)
− INF	negative infinity		(error condition, see appendix E)
NaN	not a number		(error condition, see appendix E)

Where N is a digit, n is the fractional part with trailing zeroes suppressed, "." is a decimal point, "#" is either "+" or "−", and "dd" is a two digit ten's exponent.

The following examples illustrate the use of BINASC.

8085 Assembly Language	MICRO Language
LXI H,XPNFPN1	NOSBASE = XPNFPN1
CALL XPNCPYN	CALL BINASC
CALL BINASC	

The symbolic constant NDIG specifies the number of digits to produce. As illustrated above, the release value is seven.

Transcendental Functions

The transcendental functions provided in the FPAC Library expect their arguments to be placed in expanded form in NOSBASE. Their results are returned, in expanded form, in TOSBASE. The results, with one exception, are within 1 to 2 bits (of the mantissa) of the exact value (the exception is the tangent function when the magnitude of the argument is very, very close to PI/2).

FPAC Functions	
FPATN	Arctangent, range is $-PI/2$ to $+PI/2$
FPCOS	Cosine (note: limited domain, see appendix E)
FPEXP	e raised to the power
FPLN	Natural logarithm
FPLOG	Common logarithm
FPSIN	Sine (note: limited domain, see appendix E)
FPSQR	Square root
FPTAN	Tangent (note: limited domain, see appendix E)

Error Conditions

The K-C-S IEEE Single Precision Floating Point standard defines several special representations. Signed infinity is represented as a sign bit, an exponent field of all ones, and a mantissa field of all zeroes. The result of an invalid operation is called Not-a-Number (NaN) and is represented as an exponent field of all ones and a non-zero mantissa field (the sign bit is insignificant).

Whenever an error condition occurs, the global byte variable FPERROR is set to one of the following values:

FPERROR	Description	Result Value
1	Underflow	zero
2	Overflow	+ INF or – INF
3	Invalid Operation	NaN

(Note: FPERROR is set to zero if no error occurred)

1. FPADD

Contents of NOSBASE	Contents of TOSBASE 0	number	+ INF	– INF	NaN
0	0	number	+ INF	– INF	NaN
number	number	**	+ INF	– INF	NaN
+ INF	+ INF	+ INF	+ INF	NaN	NaN
– INF	– INF	– INF	NaN	– INF	NaN
NaN	NaN	NaN	NaN	NaN	NaN

**: result could be 0 (possibly an underflow), a number, + INF, or – INF

2. FPMUL

Contents of NOSBASE	Contents of TOSBASE 0	number	+ INF	– INF	NaN
0	0	0	NaN	NaN	NaN
number	0	**	+ / – INF	– / + INF	NaN
+ INF	NaN	+ / – INF	+ INF	– INF	NaN
– INF	NaN	– / + INF	– INF	+ INF	NaN
NaN	NaN	NaN	NaN	NaN	NaN

**: result could be 0 (an underflow), a number, + INF, or – INF

3. FPDIV

Contents of NOSBASE	Contents of TOSBASE 0	number	+ INF	– INF	NaN
0	NaN	0	0	0	NaN
number	+ / – INF	**	0*	0*	NaN
+ INF	+ INF	+ / – INF	NaN	NaN	NaN
– INF	– INF	– / + INF	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN

*: underflow
**: result could be 0 (an underflow), a number, + INF, or – INF

4. Floating Point Functions

Contents of NOSBASE	Function ATN	EXP	LN/LOG	SQR	Trig
– INF	– PI/2	0*	NaN	NaN	NaN
– number	number	**	NaN	NaN	***
0	0	1	– INF	0	0 or 1
+ number	number	****	number	number	***
+ INF	PI/2	+ INF	+ INF	+ INF	NaN
NaN	NaN	NaN	NaN	NaN	NaN

*: underflow
**: result could be 0 (an underflow) or a number (less than 1)
***: result could be 0, a number, + INF, – INF, or (if the magnitude of the number is not less than 65536) NaN
****: result could be a number (greater than 1) or + INF

5. Special Conversion Routines

The INT and FIX routines treat values out of word integer range (and the NaN value) as a maximum magnitude integer of the same sign.

The AINT routine returns the parameter given it if the value of the parameter is not between + 2^23 and – 2^23.

FPAC Global Names

AINT

MICRO callable fractional truncation routine.

```
Ex:     CALL AINT(INXPN, OUTXPN)
```

AINTA

Assembly callable fractional truncation routine.

```
Ex:     LD    HL,INXPN        LXI   H,INXPN
        LD    DE,OUTXPN       LXI   D,OUTXPN
        CALL AINTA            CALL AINTA
```

ASCBIN

MICRO callable ASCII literal to compressed f.p. translation routine.

```
Ex:     CALL ASCBIN(INPNTR, OUTCMPR, ERRFLAG)
```

ASCBINA

Assembly callable ASCII literal to compressed f.p. translation routine.

```
Ex:     LD    HL,(INPNTR)     LHLD INPNTR
        LD    DE,OUTCMPR      LXI   D,OUTCMPR
        CALL ASCBINA          CALL ASCBINA
        LD    (ERRFLAG),A     STA   ERRFLG
        LD    (INPNTR),HL     SHLD INPNTR
```

BINASC

MICRO/Assembly callable expanded f.p. to ASCII translation routine. The expanded f.p. must be in NOSBASE.

```
Ex:     CALL UNPACK(INCMPR, NOSBASE)
        CALL BINASC
        !* Result in global string -REALLIT-
```

FIX

MICRO callable expanded f.p. to word integer (by fractional truncation) translation routine.

```
Ex:     CALL FIX(INXPN, WRDINT)
```

FIXA

Assembly callable expanded f.p. to word integer (by fractional truncation) translation routine.

```
Ex:     LD    HL,INXPN        LXI   H,INXPN
        CALL FIXA             CALL FIXA
        EX    DE,HL           XCHG
        LD    (WRDINT),HL     SHLD WRDINT
```

FLOAT

MICRO callable word integer to expanded f.p. translation routine.

```
Ex:     CALL FLOAT(WRDINT, OUTXPN)
```

FLOATA

Assembly callable word integer to expanded f.p. translation routine.

```
Ex:     LD    HL,(WRDINT)     LHLD WRDINT
        LD    DE,OUTXPN       LXI   D,OUTXPN
        CALL FLOATA           CALL FLOATA
```

FPADD

MICRO/Assembly callable routine to add TOSBASE to NOSBASE placing the result in the extended precision result register.

FPATN

MICRO/Assembly callable routine to place the value of the arc whose tangent is in NOSBASE into TOSBASE.

FPCOS

MICRO/Assembly callable routine to place the value of the cosine of the radian value in NOSBASE into TOSBASE.

FPDIV

MICRO/Assembly callable routine to divide NOSBASE by TOSBASE placing the result in the extended precision result register.

FPEXP

MICRO/Assembly callable routine to place the value of e raised to the power in NOSBASE into TOSBASE.

FPLN

MICRO/Assembly callable routine to place the value of the natural logarithm of the value in NOSBASE into TOSBASE.

FPLOG

MICRO/Assembly callable routine to place the value of the common logarithm of the value in NOSBASE into TOSBASE.

FPMUL — MICRO/Assembly callable routine to multiply TOSBASE by NOSBASE placing the result in the extended precision result register.

FPSIN — MICRO/Assembly callable routine to place the value of the sine of the radian value in NOSBASE into TOSBASE.

FPSQR — MICRO/Assembly callable routine to place the value of the square root of the value in NOSBASE into TOSBASE.

FPTAN — MICRO/Assembly callable routine to place the value of the tangent of the radian value in NOSBASE into TOSBASE.

INT — MICRO callable routine to produce the largest word integer value that is less than or equal an expanded f.p.

```
Ex:     CALL INT(INXPN, WRDINT)
```

INTA — Assembly callable routine to produce the largest word integer value that is less than or equal to an expanded f.p.

```
Ex:     LD    HL,INXPN        LXI   H,INXPN
        CALL INTA             CALL INTA
        EX    DE,HL           XCHG
        LD    (WRDINT),HL     SHLD WRDINT
```

NOSBASE — Starting address of 5 byte expanded f.p. used by FPAC routines.

NOSEXP — Single byte field (within NOSBASE) containing an exponent value.

NOSMAN — Three byte field (within NOSBASE) containing a mantissa value.

NOSSGN — Single byte field (within NOSBASE) containing a sign value.

PACK — MICRO callable expanded f.p. to compressed f.p. translation routine.

```
Ex:     CALL PACK(INXPN, OUTCMPR)
```

PACKA — Assembly callable expanded f.p. to compressed f.p. translation routine.

```
Ex:     LD    HL,INXPN        LXI   H,INXPN
        LD    DE,OUTCMPR      LXI   D,OUTCMPR
        CALL PACKA            CALL PACKA
```

REALLIT — Zero byte terminated ASCII literal string produced by BINASC. Maximum length is 14 characters (plus a zero byte).

RSLT2N — MICRO/Assembly callable routine to copy the value in the result register into NOSBASE.

RSLT2T — MICRO/Assembly callable routine to copy the value in the result register into TOSBASE.

RSLTBAS — Starting address of 7 byte extended precision result register used by FPAC routines.

RSLTCPY — Assembly callable routine to copy the value in the result register into an expanded f.p. variable.

```
Ex:     LD    DE,OUTXPN       LXI   D,OUTXPN
        CALL RSLTCPY          CALL RSLTCPY
```

RSLTEXP — Two byte field (within RSLTBAS) containing an extended exponent value.

RSLTMAN — Four byte field (within RSLTBAS) containing an extended mantissa value.

RSLTPAA — Assembly callable routine to pack the value in the extended precision result register into a compressed f.p. variable.

```
Ex:     LD    DE,OUTCMPR      LXI   D,OUTCMPR
        CALL RSLTPAA          CALL RSLTPAA
```

RSLTPAC — MICRO callable routine to pack the value in the extended precision result register into a compressed f.p. variable.

```
Ex:     CALL RSLTPAC(OUTCMPR)
```

RSLTSGN — Single byte field (within RSLTBAS) containing a sign value.

TOSBASE — Starting address of 5 byte expanded f.p. used by FPAC routines.

TOSEXP — Single byte field (within TOSBASE) containing an exponent value.

TOSMAN — Three byte field (within TOSBASE) containing a mantissa value.

TOSSGN — Single byte field (within TOSBASE) containing a sign value.

UNPACK — MICRO callable expanded f.p. to compressed f.p. translation routine.

```
Ex:     CALL UNPACK(INCMPR, OUTEXP)
```

UNPACKA — Assembly callable expanded f.p. to compressed f.p. translation routine.

```
Ex:     LD    HL,INCMPR       LXI   H,INCMPR
        LD    DE,OUTEXP       LXI   D,OUTEXP
        CALL UNPACKA          CALL UNPACKA
```

Notes About 8085 FPAC Routines

In general, the Assembly callable routines alter the contents of all registers.

The Assembly callable routines usually expect the source parameter to either be in HL or pointed to by HL. The destination parameter is either pointed to by DE on entry or placed in DE upon exit. If a flag is returned by a routine, it is placed in A. All of these conditions, of course, depend on the particular routine involved.

The MICRO callable routines, if they require a source parameter, expect the source parameter to be given first. If the routine returns a flag, it is the last parameter given. The destination parameter, when required, follows the source parameter (if any) and preceeds the flag parameter (again, if any).

Routine Space and Size Statistics

The sizes shown below are in bytes of ROM space. The execution time is in CPU cycles. The instances when the phrase "not meaningful" is used indicates that the time varies greatly and is extremely data dependent, with no useful way to relate execution time to given data.

A. Data Translation and Transfer Routines

Name	Size	Execution Time	Comments
UNPACKA	20H	174	
UNPACK	3H	36	In addition to UNPACKA
PACKA	19H	147	
PACK	3H	36	In addition to PACKA
RSLTPAA	1AH	135	
RSLTPAC	3H	32	In addition to RSLTPAA
RSLTCPY	19H	152	
RSLT2N	3H	10	In addition to RSLTCPY
RSLT2T	6H	20	In addition to RSLTCPY

B. Basic Floating Point Operations

Name	Size	Execution Time	Comments
FPADD	197H	Add: 1000 typ 1470 max Sub: 1200 typ 1600 max	About 50H bytes of exception handling code.
FPMUL	148H	3700 typ 4400 max	About 40H bytes of exception handling code.
FPDIV	101H	4100 typ 5300 max	About 30H bytes of exception handling code.
Exception	86H		Common exception handling code.
AINTA	9DH	500 to 2600	Max time when 2.0 > val >= 1.0
AINT	3H	36	In addition to AINTA
FLOATA	48H	170 to 190	Max time when ABS(val) is small
FLOAT	3H	36	In addition to FLOATA
FIXA	53H	140 to 660	Max time when small, negative number
FIX	CH	98	In addition to FIXA
INTA	AH	26 to 58	In addition to FIXA
INT	CH	106	In addition to INTA

C. ASCII Interpretation to/from Floating Point Representation

Name	Size	Execution Time	Comments
ASCBINA	18AH	(not meaningful)	Requires FPADD, FPDIV, FPMUL, FLOATA, PACKA, and tens constants
ASCBIN	15H	(not meaningful)	In addition to ASCBINA
BINASC	232H	(not meaningful)	Requires FPMUL, FPDIV, and tens constants
Tens cons.	1EH		List of powers of ten in expanded form.

D. Transcendental Functions

Name	Size	Execution Time		Comments
FPATN	D6H	90 000	(typ)	
FPEXP	D1H	65 000	(typ)	Requires power series evaluator
FPLN	E8H	100 000	(typ)	
FPLOG	5H	5 000		In addition to FPLN
FPSQR	6DH	33 000		
trig	1ECH	---		Requires power series evaluator
FPSIN & FPCOS		55 000	(typ)	
FPTAN		70 000	(typ)	
series	43H	---		Power series evaluator
misc	6FH	---		Misc support routines

In general, all functions require virtually the whole of the basic operations set.

Notes About Z-80 FPAC Routines

In general, the Assembly callable routines alter the contents of A, H, L, D, E, B, and C (but not the alternate register set or the index registers).

The Assembly callable routines usually expect the source parameter to either be in HL or pointed to by HL. The destination parameter is either pointed to by DE on entry or placed in DE upon exit. If a flag is returned by a routine, it is placed in A. All of these conditions, of course, depend on the particular routine involved.

The MICRO callable routines, if they require a source parameter, expect the source parameter to be given first. If the routine returns a flag, it is the last parameter given. The destination parameter, when required, follows the source parameter (if any) and preceeds the flag parameter (again, if any).

Routine Space and Size Statistics

The sizes shown below are in bytes of ROM space. The execution time is in CPU cycles. The instances when the phrase "not meaningful" is used indicates that the time varies greatly and is extremely data dependent, with no useful way to relate execution time to given data.

A. Data Translation and Transfer Routines

Name	Size	Execution Time	Comments
UNPACKA	1CH	152	
UNPACK	3H	49	In addition to UNPACKA
PACKA	14H	127	
PACK	3H	49	In addition to PACKA
RSLTPAA	1AH	135	
RSLTPAC	3H	31	In addition to RSLTPAA
RSLTCPY	12H	136	
RSLT2N	3H	10	In addition to RSLTCPY
RSLT2T	5H	22	In addition to RSLTCPY

B. Basic Floating Point Operations

Name	Size	Execution Time	Comments
FPADD	16DH	Add: 900 typ 1250 max Sub: 1050 typ 1500 max	About 50H bytes of exception handling code.
FPMUL	121H	3600 typ 4300 max	About 40H bytes of exception handling code.
FPDIV	E6H	4000 typ 5200 max	About 30H bytes of exception handling code.
Exception	84H		Common exception handling code.
AINTA	84H	500 to 2500	Max time when 2.0 > val >= 1.0
AINT	3H	39	In addition to AINTA
FLOATA	41H	180 to 510	Max time when ABS(val) is small
FLOAT	3H	39	In addition to FLOATA
FIXA	4AH	150 to 600	Max time when small, negative number
FIX	CH	119	In addition to FIXA
INTA	AH	34 to 57	In addition to FIXA
INT	CH	109	In addition to INTA

C. ASCII Interpretation to/from Floating Point Representation

Name	Size	Execution Time	Comments
ASCBINA	16EH	(not meaningful)	Requires FPADD, FPDIV, FPMUL, FLOATA, PACKA, and tens constants
ASCBIN	14H	(not meaningful)	In addition to ASCBINA
BINASC	200H	(not meaningful)	Requires FPMUL, FPDIV, and tens constants
Tens cons.	1EH		List of powers of ten in expanded form.

D. Transcendental Functions

Name	Size	Execution Time		Comments
FPATN	D2H	80 000	(typ)	
FPEXP	CAH	60 000	(typ)	Requires power series evaluator
FPLN	E6H	95 000	(typ)	
FPLOG	4H	5 000		In addition to FPLN
FPSQR	69H	30 000		
trig	1E3H	---		Requires power series evaluator
FPSIN & FPCOS		50 000	(typ)	
FPTAN		65 000	(typ)	
series	41H	---		Power series evaluator
misc	65H	---		Misc support routines

In general, all functions require virtually the whole of the basic operations set.

Example Evaluation Sequence

Below are two code segments to calculate one root of a quadratic equation (assuming the equation has real roots). The first segment is in Z-80 Assembly language.

```
            RSEG  VARS
AO          DS    4           ;  Coefficients: AO*X^2 + BO*X + CO
BO          DS    4
CO          DS    4
;
ANSWER      DS    4           ;  Set to (-BO + SQR(BO*BO - 4*AO*CO))/(2*AO)
;
TMP1        DS    5           ;  Intermediate result for BO*BO
;
            GLBL  UNPACKA, PACKA, RSLTPAA, FPMUL, FPADD, FPDIV, FLOATA, FPSQR
            GLBL  XPNCPYN, RSLT2N, RSLT2T, RSLTCPY
            GLBL  NOSBASE, TOSBASE, NOSSGN
            RSEG  CODE
;
            LD    HL, BO        ;  Calc BO*BO
            LD    DE, TOSBASE
            CALL  UNPACKA
            LD    HL,TOSBASE    ;  Copy expanded BO from TOSBASE into NOSBASE
            CALL  XPNCPYN
            CALL  FPMUL
            LD    DE,TMP1       ;  Copy expanded result into TMP1
            CALL  RSLTCPY
;
            LD    HL,-4         ;  Create floating point -4.0
            LD    DE,TOSBASE
            CALL  FLOATA
            LD    HL,AO         ;  Times AO
            LD    DE,NOSBASE
            CALL  UNPACKA
            CALL  FPMUL
            CALL  RSLT2T        ;  -4.0 * AO into TOSBASE
            LD    HL,CO         ;  Times CO
            LD    DE,NOSBASE
            CALL  UNPACKA
            CALL  FPMUL
            CALL  RSLT2T        ;  -4.0 * AO * CO into TOSBASE
;
            LD    HL,TMP1       ;  BO * BO into NOSBASE
            CALL  XPNCPYN
            CALL  RPADD
            CALL  RSLT2N        ;  BO * BO + (-4.0 * AO * CO) into NOSBASE
            CALL  FPSQR         ;  Square root into TOSBASE
;
            LD    HL,BO         ;  Unpack BO into NOSBASE
            LD    DE,NOSBASE
            CALL  UNPACKA
            LD    A,(NOSSGN)    ;  Negate NOSBASE (to get -BO)
            CPL
            LD    (NOSSGN),A
            CALL  FPADD
            CALL  RSLT2N        ;  -BO + SQR(BO*BO + (-4.0*AO*CO)) into NOSBASE
```

```
;
        LD      HL,2            ;  2.0 into TOSBASE
        LD      DE,TOSBASE
        CALL    FLOATA
        CALL    FPDIV           ;  Divide by 2.0
        CALL    RSLT2N          ;  Result into NOSBASE
        LD      HL,AO
        LD      DE,TOSBASE
        CALL    UNPACKA
        CALL    FPDIV           ;  Divide by AO
;
        LD      DE,ANSWER       ;  Pack result into destination var
        CALL    RSLTPAA
```

MICRO code segment:

```
REAL*4:     AO, BO, CO, ANSWER, TMP1

EXTERNAL SUBROUTINE:    UNPACK, PACK, RSLTPAC,
                        RSLT2N, RSLT2T,
                        FPMUL, FPADD, FPDIV, FPSQR, FLOAT

TYPE EXPND = RECORD
             POINTER*1:     MAN(3), EXP, SGN
             ENDRECORD

EXTERNAL EXPND:     TOSBASE, NOSBASE

CALL UNPACK(BO, TOSBASE)     NOSBASE = TOSBASE        !* Calc BO*BO
CALL FPMUL
CALL RSLTPAC(TMP1)                                    !* Stash in TMP1

CALL FLOAT(INTEGER*2(-4), TOSBASE)     CALL UNPACK(AO, NOSBASE)
CALL FPMUL
CALL RSLT2T                                 !* -4.0 * AO in TOSBASE
CALL UNPACK(CO, NOSBASE)
CALL FPMUL
CALL RSLT2T                                 !* -4.0*AO*CO in TOSBASE

CALL UNPACK(TMP1, NOSBASE)
CALL FPADD
CALL RSLT2N                                 !* BO*BO - 4.0*AO*CO in NOSBASE

CALL FPSQR                                  !* Square root of NOSBASE to TOSBASE

CALL UNPACK(BO, NOSBASE)                    !* Unpack BO, flip its sign
NOSBASE.SGN = NOT NOSBASE.SGN
CALL FPADD
CALL RSLT2N                                 !* -BO + SQR(...)

CALL FLOAT(INTEGER*2(2), TOSBASE)           !* Divide by two
CALL FPDIV
CALL RSLT2N
CALL UNPACK(AO, TOSBASE)                    !* Divide by AO
CALL FPDIV

CALL RSLTPAC(ANSWER)                        !* Result packed into ANSWER
```