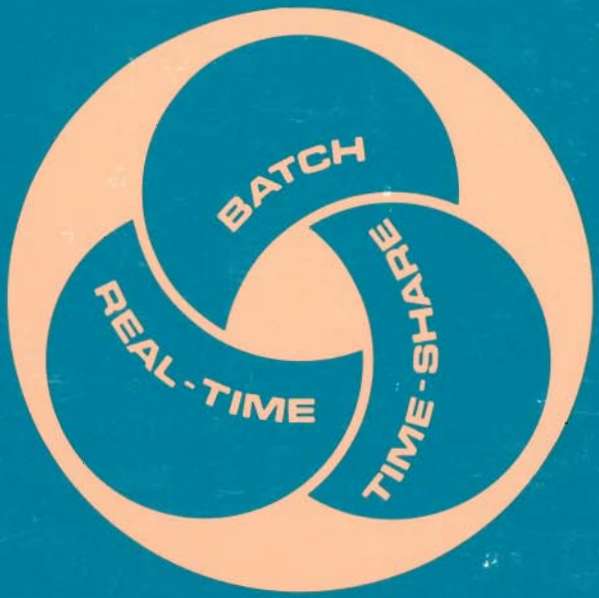
HP 3000

COMPILER LIBRARY



HP 3000

COMPILER LIBRARY

03000-90009A November 1972

List of Effective Pages

Pages	Effective Date
Title	Nov. 1972
Copyright	Nov. 1972
iii	Nov. 1972
v to xvii	Nov. 1972
1-1 to 1-57	Nov. 1972
2-1 to 2-48	Nov. 1972
3-1 to 3-35	Nov. 1972
4-1 to 4-6	Nov. 1972

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

Part No.	Date	Update Package	Date
03000-90009A	Nov. 1972		

PREFACE

HP 3000 Compiler Library is the programmer's reference to input/output formatting and mathematical and utility procedures available to users of Hewlett-Packard HP 3000 software.

The reader should have a working knowledge of the language(s) to be used and the MPE/3000 Operating System or have access to the appropriate reference manuals listed below.

HP 3000 Multiprogramming Executive Operating System (HP 03000-90005)

HP 3000 Systems Programming Language (HP 03000-90002)

HP 3000 FORTRAN (HP 03000-90007)

HP 3000 supporting software, including the procedures in the Compiler Library, are written in Hewlett-Packard's Systems Programming Language (SPL/3000). Thus, all procedures in this manual are SPL/3000 procedures.

The Introduction to this manual defines the purpose of the Library and its relationship to the MPE/3000 Operating System and defines the structural elements of this book. Section I is a detailed analysis of the Formatter, Section II contains mathematical procedures, Section III describes utility procedures, and Section IV outlines the special utility procedure for Library Errors.

CONTENTS

TABLES

Introduction

HP 3000 Compiler Library routines perform input/output, internal data conversion, mathematical, and error-reporting functions for user programs. The HP 3000 Multiprogramming Executive (MPE/3000) links each user program to the Compiler Library routines needed.

ORGANIZATION OF THIS BOOK

This book contains a function directory and four sections:

Section I: The Formatter

Section II: Mathematical Procedures

Section III: Utility Procedures

Section IV: Library Errors

Format for Procedures

Most of the procedures in Sections II and III are described in a standard format. The following items are included in that format, when applicable:

NAME (at top of page)	The procedure identifier.
FUNCTION:	Purpose of the procedure.
Declaration:	The parts of the procedure declaration that define the requirements for actual parameters (arguments) included in a procedure call or calling sequence. Procedure declarations are defined in the manual *HP 3000 Systems Programming Language (HP 03000-90002).*
Method:	A comment on the algorithm for the procedure.

Accuracy: A description of the procedure accuracy, using the following notation:

x = true value of the argument(s)

y = computed value of the argument(s)

f = true value of the result

g = computed value of the result

$|x-y|$ = absolute error in the argument(s)

$\frac{|x-y|}{|x|}$ = relative error in the argument(s)

$|f-g|$ = absolute error in the result(s)

$\frac{|f-g|}{|f|}$ = relative error in the result(s)

ATTRIBUTES:

Parameters: The type(s) and range[1] of value(s) allowed by the procedure. The FORTRAN type double precision is identical to the SPL/3000 type LONG (real).

Result(s): The type(s), and range of value(s).

FORTRAN: The procedure is an intrinsic or basic external function (each followed by the implicit call format) or is not callable by FORTRAN/3000.

Error(s): A brief description of the error conditions.

COMMENTS: (When needed, special comments.)

Text Conventions

The following conventions are used throughout the manual:

1. All numbers are decimal unless otherwise noted.
2. In all examples, a blank space is represented by a delta Δ.
3. All appearances of the initials TOS refer to the top-of-stack, as defined in the manual *HP 3000 Systems Programming Language (HP 03000-90002).*
4. The notation := means "is replaced by."

[1] The range and form of internal representations are summarized in text that follows.

5. Mathematical notation in the text includes the following definitions:

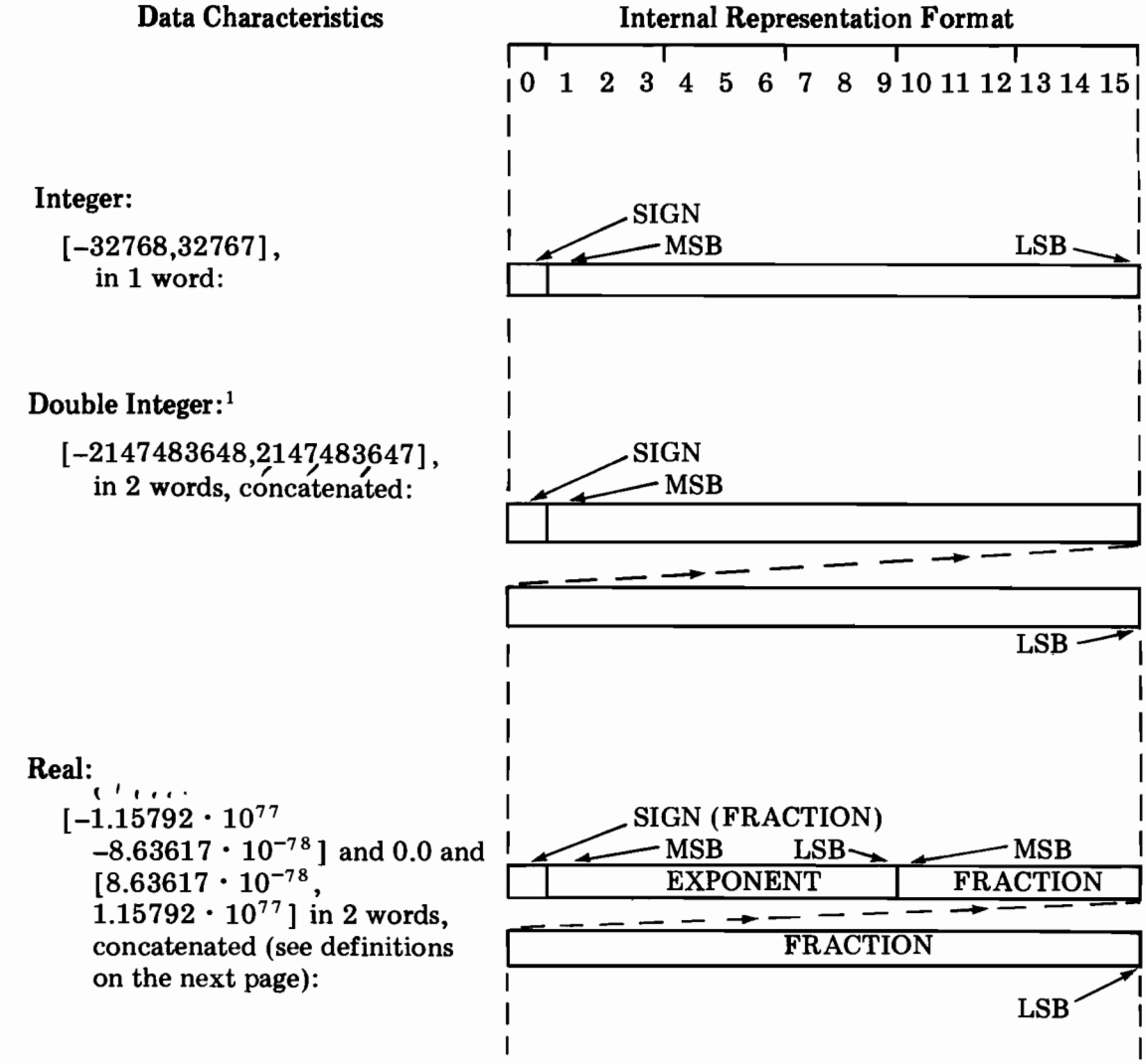
   A value x in the range (a,b) means $a < x < b$

   A value x in the range $[a,b]$ means $a \leqslant x \leqslant b$

   A value x in the range $(a,b]$ means $a < x \leqslant b$

   A value x in the range $[a,b)$ means $a \leqslant x < b$

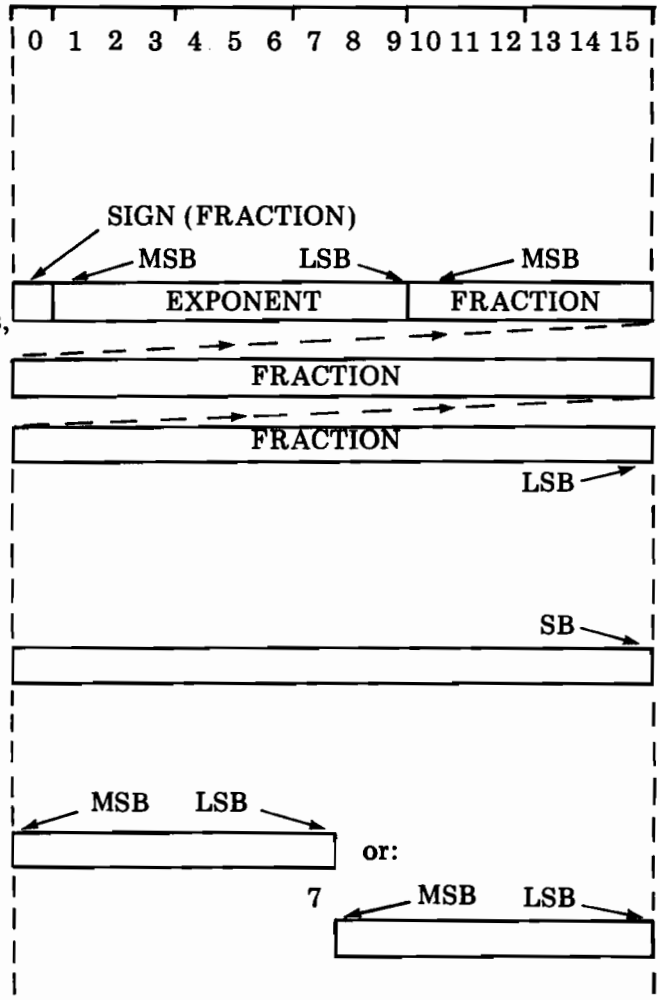
6. All references to internal representations are based on the following (unless otherwise noted):



where MSB is the most significant bit and LSB is the least significant bit.

[1] Not recognized in FORTRAN.

Data Characteristics | **Internal Representation Format**



Double Precision:[1]

[$-1.1579208924 \cdot 10^{77}$, $-8.6361685551 \cdot 10^{-78}$] and 0.0 and [$8.6361685551 \cdot 10^{-78}$, $1.1579208924 \cdot 10^{77}$] in 3 words, concatenated (see definitions below):

Logical (Boolean):

[True (odd), False (even)], in 1 word:

Byte (ASCII character code):

[any alphameric], in ½ word:

Definitions:

MSB = most significant bit

LSB = least significant bit

SB = significant bit (all others may be 0 or 1 for other uses)

SIGN = S = one bit for the sign of FRACTION, 0 for positive, 1 for negative

EXPONENT = E = $[0,777_8]$ = $[0,511_{10}]$

FRACTION = F = $[0,2^{22}-1]$ or $[0,2^{38}-1]$

Real and double precision numbers are stored in normalized form with an implied "1." to the left of the FRACTION MSB. Thus, DECIMAL VALUE =

$(-1)^S * 2^{E-256} * (1.+F*2^{-22})$ REAL

$(-1)^S * 2^{E-256} * (1.+F*2^{-38})$ DOUBLE PRECISION[1]

The exception occurs when S = E = F = 0; the decimal value is 0.0.

NOTE: When $E = 511_{10}$, $2^{E-256} = 2^{255}$.

[1] In SPL/3000, type LONG real.

Function Directory

The following list identifies each function provided in the *HP 3000 Compiler Library* and points to the descriptions in this manual. The functions have been grouped by general categories of functions. The grouping does *not* reflect the organization of the manual.

[1] In SPL/3000, type LONG real.

[1] In SPL/3000, type LONG real.
[2] In SPL/3000, a two-element real array.

[1] In SPL/3000, type LONG real.
[2] In SPL/3000, a two-element real array.

[1] In SPL/3000, type LONG real.
[2] In SPL/3000, a two-element real array.

[1] In SPL/3000, type LONG real.
[2] In SPL/3000, a two-element real array.
[3] In SPL/3000, type LONG real; equivalent to FORTRAN/3000 type double precision.

[1] In SPL/300, a two-element real array.

SECTION I
The Formatter

The Formatter is a subroutine called by FORTRAN compiler-generated code or by SPL/3000 user programs. The FORTRAN/3000 compiler interprets READ or WRITE statements of a FORTRAN program to generate the calls to the Formatter; an SPL/3000 user must generate the calls himself. The Formatter can perform the following functions:

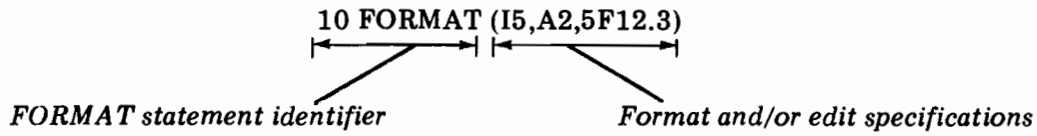
1. Convert between external ASCII numeric and/or character records and an internally represented list of variables. Formatting proceeds according to implicit parameters derived from a FORTRAN program's FORMAT statements or explicit parameters written into an SPL/3000 program.
2. Convert free-field external ASCII records to an internally represented list of variables according to format and/or edit control characters imbedded in the input records.
3. Convert an internally represented list of variables to external ASCII records which are free-field input-compatible.
4. Convert between an internally represented list of variables and a user-defined ASCII buffer storage area (core-to-core).
5. Transfer (unformatted and without conversion) between an internally represented list of variables and external files on disc or tape.

READ and WRITE statements in a FORTRAN program must meet the syntactic requirements of that language. The Formatter derives format and edit parameters from FORMAT statements or the data. The SPL/3000 user, however, must code the calls and the parameters by the methods described under "SPL/3000 Calling Sequences."

FORMAT STATEMENTS

FORMAT statements in a FORTRAN program enclose a series of format and/or edit specifications in parentheses. The specifications must be separated by commas or record terminators (see "/Edit Descriptor").

EXAMPLE:



These format and edit specifications can include another set of format and/or edit specifications enclosed in parentheses; this is called nesting. The HP 3000 Formatter allows nesting to a depth of four levels.

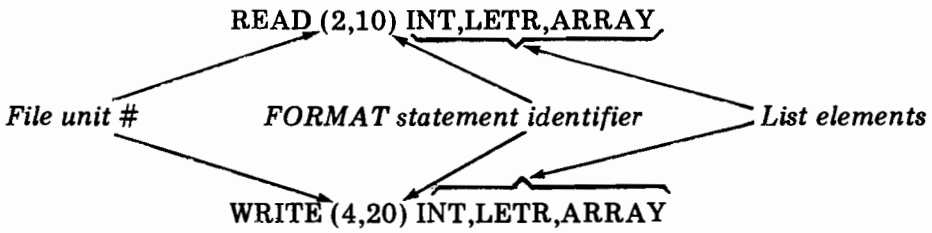
EXAMPLE:

```
20 FORMAT (I3,E12.5,3(D14.3,I6),4HSTOP)
```

READ or WRITE Statements

Formatted READ or WRITE statements in a FORTRAN program identify the list of variables that reference a FORMAT statement. (More than one READ or WRITE statement can reference a given FORMAT statement.)

EXAMPLE:



The list of variables can consist of any number of elements (including zero elements); there need not be a direct relationship to the number of list elements and the number of format and/or edit specifications. Refer to "Unlimited Groups," in this section.

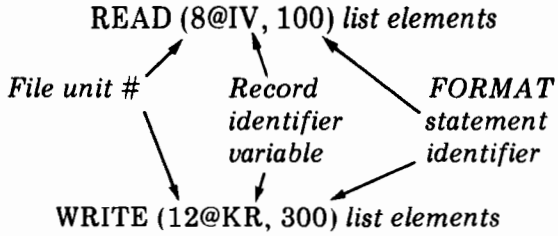
Disc Input/Output

Two types of access to files on disc devices are available through the MPE/3000 file system: sequential or direct. Either type can be established through the MPE/3000 file intrinsic FOPEN; direct access includes the capability of sequential access.

When formatted/sequential access is used, the READ or WRITE statements of a FORTRAN program are written as described above, under "READ or WRITE Statements."

When formatted/direct access is used, the READ or WRITE statements of a FORTRAN program must specify an integer, real, or double precision simple variable or a constant for the record identifier.

EXAMPLES:



When the file is opened (through the MPE/3000 file intrinsic FOPEN), the record size can be left at the system default value 128, or the user can specify a different size.

In sequential access, as many records as needed are used in sequence until the entire list of variables has been transmitted.

In direct access, only one record is transmitted.

FORMAT SPECIFICATIONS

Format specifications are written as

- A field descriptor
- A scale factor followed by a field descriptor
- A repeat specification followed by a field descriptor
- A scale factor followed by a repeat specification and a field descriptor

A brief discussion of field descriptors follows; detailed descriptions appear later in this section.

Field Descriptors

For output of data, the field descriptor determines the components of a data field into which a given list element will be written. For input, the field descriptor defines only the field width from which data can be read into an internal list element.

DECIMAL NUMERIC CONVERSIONS

Seven descriptor forms are provided:

D*w.d*	Output in double precision, floating point (with an exponent field) form.
E*w.d*	Output in real, floating point (with an exponent field) form.
F*w.d*	Output in real, fixed point (with *no* exponent field) form.
G*w.d*	Output in either the F*w.d* format or the E*w.d* format, depending on the relative size of the number to be converted.
M*w.d*	Output in monetary (business) form (real, fixed-point, plus $ and commas), e.g., $4,376.89.

N*w.d* Output in numeration form (same as the M*w.d* format, but without the $), e.g., 3,267.54.

I*w* Output in integer form.

where

w = the length of the external data field, in characters; must be greater than zero.

d = the number of fraction field digits in a floating or fixed point output (see detailed descriptions on the following pages). On input, *if* the external data does *not* include a decimal point, the integer is multiplied by 10^{-d}. If the external data does include a decimal point, this specification has no effect. Where listed above, d must be stated even if zero.

Rules for Input

All of the field descriptors listed on the preceding page accept ASCII numeric input in the following formats.

NOTE: I*w*, *on input, is interpreted as* F*w*.0

1. A series of integer number digits with or without a sign

   2314 or +56783 or –96

2. Any of the above with an exponent field with or without a sign

   2314+2 or +56783E–4 or –96D+4

3. A series of real number digits with or without a sign

   2.314 or +567.83 or –.96

4. Any of the above, with an exponent field with or without a sign

   2.314+2 or +567.83E–4 or –.96D+4

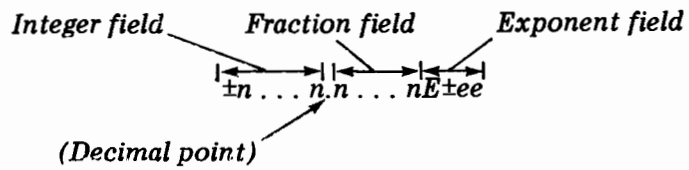
5. Either of the above items 1 and 3, in monetary (business) form

   $234 or $5,678.30 or –.96

6. Either of the above items 1 and 3, in numeration form

   2.314 or +5,678.30 or –961,534.873

In summary, the input field can include integer, fraction, and exponent subfields:



Rules: 1. The number of characters in the input field, including $ and commas, must not exceed *w* in the field descriptor used.

2. The exponent field input can be any of several forms:

+*e*	+*ee*	E*e*	E*ee*	D*e*	D*ee*
–*e*	–*ee*	E+*e*	E+*ee*	D+*e*	D+*ee*
		E–*e*	E–*ee*	D–*e*	D–*ee*

where *e* is an exponent value digit.

3. Embedded or trailing blanks (to the right of any character read as a value) are treated as zeros; leading blanks are ignored; a field of all blanks is treated as zero.

EXAMPLES:

1Δ23 = 1023	.2Δ56ΔE+Δ4 = .20560E+04
12.Δ34 = 12.034	2Δ2,Δ45.ΔΔ3 = 202045.003
–$1,Δ34.ΔΔ5 = –1034.005	2.Δ02–Δ13 = 2.002–013

4. The type of the internal storage is independent of either the ASCII numeric input or the field descriptor used to read the input. The data is stored according to the type of the list element (variable) currently using the field descriptor. The conversion rules are as follows:

- Type INTEGER truncates a fractional input.
- Type REAL rounds a fractional input.
- Type DOUBLE INTEGER (not recognized in FORTRAN) truncates a fractional input.
- Type DOUBLE PRECISION[1] rounds a fractional input.

OCTAL NUMERIC CONVERSION

One descriptor form is provided:

O*w* for octal numbers 0 through 177777_8

where

w is the length (in characters) of the external data field (must be greater than zero).

This field descriptor accepts ASCII numeric input up to six octal digits long. Non-numeric or non-octal characters cause a conversion error.

[1] In SPL/3000, type LONG real.

LOGICAL CONVERSION

One descriptor form is provided:

L*w*	for logical values (T or F followed by any other characters).

The field descriptor accepts any ASCII characters input that begins with either T or F.

ALPHAMERIC CONVERSIONS

Three descriptor forms are provided:

A*w*	for alphameric conversion to and from the leftmost bytes of a list element.
R*w*	for alphameric characters to and from the rightmost bytes of a list element.
S	for alphameric characters to and from a character string (user-defined character list element).

Each of the above field descriptors accepts (but provides differing storage of) any ASCII character's input, including blanks.

D*w.d*

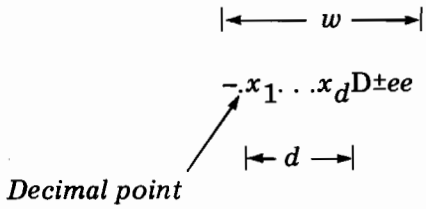
Double precision[1] numbers

FUNCTION: Define a field for a double precision[1] number with an exponent (floating-point).

OUTPUT

On output the D field descriptor causes normalized output of a variable (internal representation value: integer, double integer,[2] real, or double precision[1]) in ASCII character floating-point form, right-justified. The least significant digit of the output is rounded.

The external field is *w* positions of the record:



where

$x_1 . . . x_d$	=	the most significant digits of the value
ee	=	the digits of the exponent value
w	=	the width of the external field
d	=	in the number of significant digits allowed in *w*
– (minus)		is present if the value is negative

The field width *w* must follow the general rule

$$w \geqslant d + 6$$

to provide positions for the sign of the value, the decimal point, *d* digits, the letter D, the sign of the exponent, and the exponent's two digits. If *w* is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If *w* is less than the number of positions required for the value (with the sign, decimal point, and exponent field), the entire field is filled with #'s.

[1] In SPL/3000, type LONG real.
[2] Not recognized in FORTRAN.

EXAMPLES:

Descriptor	Internal Value	Output
D10.3	+12.342	△△.123D+02
D10.3	-12.341	△-.123D+02
D12.4	+12.340	△△△.1234D+02
D12.4	-12.345	△△-.1235D+02
D7.3	+12.343	#######
D5.1	+12.344	#####

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the rollover value is normalized and the exponent is adjusted.

EXAMPLES:

Descriptor	Internal Value	Output
D11.5	-999.997	-.10000D+04
D11.5	+999.996	△.10000D+04
D10.5	-99.9995	##########

INPUT

On input, the D field descriptor causes interpretation of the next *w* positions in an ASCII input record. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

E*w.d*

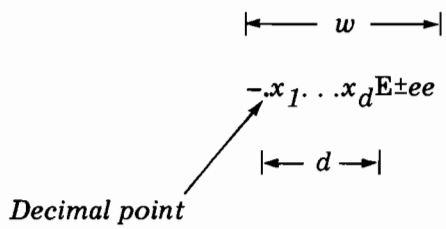
Real Numbers

FUNCTION: Define a field for a real number with an exponent (floating-point).

OUTPUT

On output, the E field descriptor causes normalized output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character floating-point form, right-justified. The least significant digit of the output is rounded.

The external field width is *w* positions in the record:



where

$x_1 . . x_d$	=	the most significant digits of the value
ee	=	the digits of the exponent value
w	=	the width of the external field
d	=	the number of significant digits allowed in *w* (for output, *d* must be greater than zero
–(minus)		is present if the value is negative

The field width *w* must follow the general rule

$$w \geqslant d + 6$$

to provide positions for the sign of the value, the decimal point, *d* digits, the letter E, the sign of the exponent, and the exponent's two digits. If *w* is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If *w* is less than the number of positions required for the value (with the sign, decimal point, and exponent field), the entire field is filled with #'s.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

EXAMPLES:

Descriptor	Internal Value	Output
E10.3	+12.342	ΔΔ.123E+02
E10.3	-12.341	Δ-.123E+02
E12.4	+12.340	ΔΔΔ.1234E+02
E12.4	-12.345	ΔΔ-.1235E+02
E7.3	+12.34	#######
E5.1	+12.34	#####

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the rollover value is normalized and the exponent is adjusted.

EXAMPLES:

Descriptor	Internal Value	Output
E11.5	-999.998	-.10000E+04
E11.5	999.995	Δ.10000E+04
E10.5	-99.9997	##########

INPUT

On input, the E field descriptor causes interpretation of the next *w* positions in an ASCII input record. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

F*w.d*

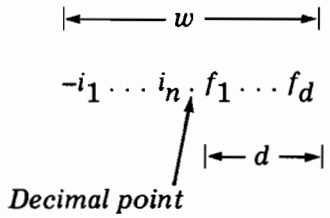
Real Numbers

FUNCTION: Define a field for a real number without an exponent (fixed-point).

OUTPUT

On output, the F field descriptor causes output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character fixed-point form, right-justified. The least significant digit of the output is rounded.

The external field width is w positions in the record:



where

$i_1 \ldots i_n$	=	the integer digits
$f_1 \ldots f_d$	=	the fraction digits
w	=	the width of the external field
d	=	the number of fractional digits allowed in w
n	=	the number of integer digits
– (minus)		is present if the value is negative.

The field width w must follow the general rule

$$w \geqslant d + n + 3$$

to provide positions for the sign, n digits, the decimal point, d digits, and a rollover digit if needed (see the following examples). If w is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If w is less than the number of positions required for the value (with the sign and decimal point), the entire field is filled with #s.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

EXAMPLES:

Descriptor	Internal Value	Output
F10.3	+12.3402	ΔΔΔ12.340
F10.3	-12.3413	ΔΔΔ-12.341
F12.3	+12.3434	ΔΔΔΔΔΔ12.343
F12.3	-12.3456	ΔΔΔΔΔ-12.346
F4.3	+12.34	####
F4.3	+12345.12	####

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the stated formula for *w* provides enough positions for the value.

EXAMPLES:

Descriptor	Internal Value	Output
F8.2	+999.997	Δ1000.00
F8.2	-999.996	-1000.00
F7.2	-999.995	#######

INPUT

On input, the F field descriptor causes interpretation of the next *w* positions in an ASCII input record. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

Gw.d

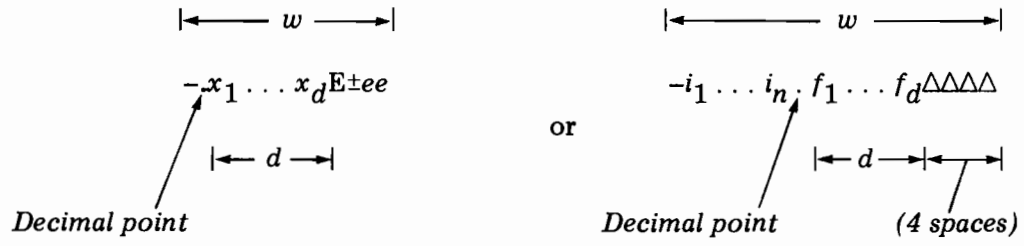
Real Numbers

FUNCTION: Define a field for a real number without an exponent (fixed-point) or, if needed, with an exponent (floating-point).

OUTPUT

On output, the G field descriptor causes output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character fixed-point form, or if needed, floating-point form, right-justified. The least significant digit of the output is rounded.

The external field is w positions in the record:



where

$i_1 \ldots i_n$ = the integer digits
$f_1 \ldots f_d$ = the fraction digits
} (Fw.d descriptor)

$x_1 \ldots x_d$ = the most significant digits of the value (Ew.d descriptor)

ee = the digits of the exponent value (Ew.d descriptor)

w = the width of the external field

d = the number of fractional digits allowed in w

n = the number of integer digits (Fw.d descriptor)

– (minus) is present if the value is negative

The G*w.d* field descriptor is interpreted as an F*w.d* descriptor for fixed-field form or as an E*w.d* descriptor for floating-point form, according to the internal representation absolute value (N) after rounding. If the number of integer digits in N is $> d$, or if N $< .1$, the E descriptor is used; otherwise the F descriptor is used (see following page).

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

IF		$N < 0.1$	THEN E*w.d*;
IF	0.1	$\leqslant N < 1$	THEN F(*w*–4) .*d* plus 4X (spaces);
IF	1	$\leqslant N < 10^1$	THEN F(*w*–4) .(*d*–1) plus 4X;
IF	10^1	$\leqslant N < 10^2$	THEN F(*w*–4) .(*d*–2) plus 4X;
IF	10^2	$\leqslant N < 10^3$	THEN F(*w*–4) . (*d*–3) plus 4X;
.	.	.	. .
.	.	.	. .
.	.	.	. .
IF	$10^{(d-1)}$	$\leqslant N < 10^d$	THEN F(*w*–4) .0 plus 4X;
IF	10^d	$\leqslant N$	THEN E*w.d*;

EXAMPLES:

G12.6, N = 1234.5: F(*w*–4) . (*d*–4) = F8.2, 4X: Δ1234.50ΔΔΔΔ

G13.7, N = 123456.7: F(*w*–4) . (*d*–6) = F9.1, 4X: Δ123456.7ΔΔΔΔ

G9.2, N = 123.4: E*w.d* = E9.2: ΔΔ.12E+03

The field width *w* must follow the general rule for the E*w.d* descriptor

$$w \geqslant d + 6$$

to provide positions for the sign of the value, *d* digits, the decimal point (preceding x_1), and, if needed, the letter E, the sign of the exponent, and the exponent's two digits. If *w* is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If *w* is less than the number of positions required for the value (with the sign, decimal point, and the exponent field—or 4 spaces), the entire field is filled with #'s.

EXAMPLES:

Descriptor	Internal Value	Output
G10.3 (E10.3)	+1234	ΔΔ.123E+04
G10.3 (E10.3)	–1234	Δ–.123E+04
G12.4 (E12.4)	+12345	ΔΔΔ.1235E+05
G12.4 (F8.0,4X)	+9999	ΔΔΔ9999.ΔΔΔΔ
G12.4 (F8.1,4X)	–999	ΔΔ–999.0ΔΔΔΔ
G7.1 (E7.1)	+.09	Δ.9E–01
G5.1 (E5.1)	–.09	#####

When the E descriptor is used, if rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the rollover value is normalized and the exponent is adjusted.

EXAMPLES:

Descriptor	Internal Value	Output
G12.1 (E12.2)	+9999	ΔΔΔΔΔ.10E+05
G8.2 (E8.2)	+999	Δ.10E+04
G7.2 (E7.2)	-999	######

INPUT

On input, the G field descriptor causes interpretation of the next *w* positions in an ASCII input record. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

Mw.d

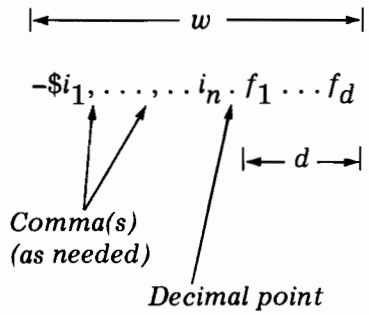
Real Numbers

FUNCTION: Define a field for a real number without an exponent (fixed-point) written in monetary (business) form.

OUTPUT

On output, the M field descriptor causes output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character fixed-point form, right-justified, with a dollar sign $ and commas. The least significant digit of the output is rounded.

The external field is w positions in the record:



where

$i_1 \ldots ..i_n$ = the integer digits (without commas)

$f_1 \ldots f_d$ = the fraction digits

$commas = c$ = the number of output commas needed: one to the left of every third digit left of the decimal point; see general rule for w below.

d = the number of fractional digits allowed in w

n = the number of integer digits

w = the width of the external field

– (minus) is present if the value is negative

The field width w must follow the general rule

$$w \geqslant d + n + c + 4$$

to provide positions for the sign, $, n digits, c commas, the decimal point, d digits, and a rollover digit if needed (see the following examples). If w is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

If *w* is less than the number of positions required for the output value (with the sign $, comma(s), and the decimal point), the entire field is filled with #'s.

EXAMPLES:

Descriptor	Internal Value	Output
M10.3	+12.3402	ΔΔΔ$12.340
M10.3	-12.3404	ΔΔ-$12.340
M13.3	+80175.3965	ΔΔ$80,175.397
M12.2	-80175.396	Δ-$80,175.40
M12.2	+28705352.563	###########

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the stated formula for *w* provides enough positions.

EXAMPLES:

Descriptor	Internal Value	Output
M12.2	+99999.996	Δ$100,000.00
M12.2	-99999.998	-$100,000.00
M11.2	-99999.995	###########

INPUT

On input, the M field descriptor causes interpretation of the next *w* positions in an ASCII input record. The field width is expected (but not required) to have a $ and comma(s) imbedded in the data as described above for M*w.d* outputs; the $ and comma(s) are ignored. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

N*w.d*

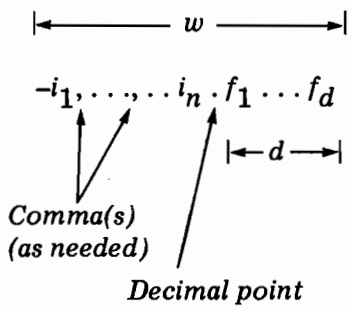
Real Numbers

FUNCTION: Define a field for a real number without exponent (fixed-point) written in numeration form (same as M*w.d* but without $ on output).

OUTPUT

On output, the N field descriptor causes output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character fixed-point form, right-justified, with commas. The least significant digit of the output is rounded.

The external field is *w* positions in the record:



where

$i_1 \ldots\ldots i_n$ = the integer digits (without commas)

$f_1 \ldots f_d$ = the fraction digits

commas = *c* = the number of output commas needed: one to the left of every third digit left of the decimal point; see general rule for *w* below.

d = the number of fractional digits allowed in *w*

n = the number of integer digits

w = the width of the external field

– (minus) is present if the value is negative

The field width *w* must follow the general rule

$$w \geq d + n + c + 3$$

to provide positions for the sign, *n* digits, *c* commas, the decimal point, *d* digits, and a rollover digit if needed (see the following examples). If *w* is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If *w* is less than the number of positions required for the output value (with the sign, comma(s), and the decimal point), the entire field is filled with #'s.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

EXAMPLES:

Descriptor	Internal Value	Output
N9.3	+12.3402	ΔΔΔ12.340
N9.3	-12.3404	ΔΔ-12.340
N12.3	+80175.3965	ΔΔ80,175.397
N11.2	-80175.396	Δ-80,175.40
N11.2	+28705352.563	##########

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the stated formula for *w* provides enough positions.

EXAMPLES:

Descriptor	Internal Value	Output
N11.2	+99999.995	Δ100,000.00
N11.2	-99999.997	-100,000.00
N10.2	-99999.999	##########

INPUT

On input, the N field descriptor causes interpretation of the next *w* positions in an ASCII input record as a real number without exponent (fixed-point). The field width is expected (but not required) to have comma(s) imbedded in the data as described above for N*w.d* outputs; the comma(s) are ignored. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

Iw

Integer Numbers

FUNCTION: Define a field for an integer number.

OUTPUT

On output, the I field descriptor causes output of a variable (internal representation value: integer, double integer,[1] real, or double precision[2]) in ASCII character integer form, right-justified. If the internal representation is real or double precision, the least significant digit of the output is rounded.

The external field is w positions of the record:

$$\vdash\!\!\leftarrow w \rightarrow\!\!\dashv$$

$$-i_1 \ldots i_n$$

where

$i_1 \ldots i_n$	=	the integer digits
n	=	the number of significant digits
w	=	the width of the external field
– (minus)		is present if the value is negative

The field width w must follow the general rule

$$w \geqslant n + 2$$

to provide positions for the sign, n digits, and a rollover digit if needed (see the following examples). If w is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If w is less than the number of positions required for the output (all digits of the integer and, when needed, the sign), the entire field is filled with #'s.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

EXAMPLES:

Descriptor	Internal Value	Output
I5	–123	Δ–123
I5	+123	ΔΔ123
I5	+12345	12345
I5	–12345	#####
I4	+12.4	ΔΔ12
I4	–12.7	Δ–13
I6	–.3765E+03	ΔΔ–377

If rounding of the least significant digit occurs and "rollover" results (for example, 99.99 becomes 100.00), the stated formula for *w* provides enough positions:

EXAMPLES:

Descriptor	Internal Value	Output
I5	–999.8	–1000
I5	+999.6	Δ1000
I4	–999.5	####

INPUT

On input, the I field descriptor functions as an F*w.d* descriptor with *d* = 0; it causes interpretation of the next *w* positions in the ASCII input record. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

All rules for input to decimal numeric conversions (see "Rules for Input") apply.

Ow

Octal Integer Number

FUNCTION: Define a field for an octal integer number.

OUTPUT

On output, the O field descriptor causes output of a variable (internal representation value: integer only) in ASCII-character octal integer form, right-justified.

The external field is w positions of the record:

$$\vdash\!\!\leftarrow w \rightarrow\!\!\dashv$$

$$i_1 \ldots i_n$$

where

$i_1 \ldots i_n$ = the octal integer digits

n = the number of significant digits (maximum: 6)

w = the width of the external field

The field width w can be any desired value but should be $\geqslant 6$ for complete accuracy. If w is greater than the number of positions required for the output value, the output is right-justified in the field with blank spaces to the left. If w is less than the number of positions required for the entire octal integer, only the w least significant digits are output.

EXAMPLES:

Descriptor	Internal Value	Output
O7	143567	Δ143567
O8	102077	ΔΔ102077
O4	027033	7033
O6	002004	002004

INPUT

On input, the O field descriptor causes interpretation of the next *w* positions in the ASCII input record as an octal integer number. The number is converted to an internal representation value for the variable (list element) currently using the field descriptor.

The input field can consist of only octal digits, no more than six digits (no larger than 177777_8) are interpreted. Any non-octal or non-numeric character (including a blank) anywhere in the field will produce a conversion error. If *w* is less than 6, *w* digits are right-justified in the internal representation (one word of memory).

EXAMPLES:

Descriptor	Input	Result
O6	134577	134577
O4	275674	002756
O10	1345367421	167421

Lw

Logical (Boolean) Values

FUNCTION: Define a field for a logical value.

OUTPUT

On output, the L field descriptor causes output of a variable (internal representation value: integer or logical (boolean)) in ASCII-character logical value form (T or F).

The external field is w positions of the record:

$$\vert\longleftarrow w \longrightarrow\vert$$

$$X_1 \ldots X_n c$$

where

$X_1 \ldots X_n$ = $w-1$ blanks

c = either of two logical characters: T (true) or F (false)

n = the number of blank spaces to the left of c

w = the width of the external field

The field width w can be any value $\geqslant 1$.

The logical character c is T if the least significant bit of the internal representation is 1; c is F if that bit is 0.

EXAMPLES:

Descriptor	Internal Value	Output
L1	102033_8	T
L13	$32767(77777_8)$	ΔΔΔΔΔΔΔΔΔΔΔΔT
L5	$+124(174_8)$	ΔΔΔΔF

INPUT

On input, the L field descriptor causes a scan of the next *w* positions in an ASCII input record to find a logical character (T or F). All positions to the left of the logical character must be blank; any other character(s) can follow the logical character. The character T is converted to –1 (177777_8), F is converted to 0 (000000_8).

EXAMPLES:

Descriptor	Input	Result
L8	ΔΔΔΔTRUE	177777_8
L1	F	000000_8
L6	ΔFALSE	000000_8

A*w*

Leftmost ASCII Characters

FUNCTION: Define a field for ASCII alphameric characters of a variable.

OUTPUT

On output, the A field descriptor causes output of one or more bytes of a variable in ASCII-character alphameric form. The maximum number *n* of bytes (thus, the maximum number of characters available to a single A*w* descriptor) depends on the type of the variable: for logical or integer, $n = 2$; for double integer[1] or real, $n = 4$; for double precision,[2] $n = 6$; for character, n = the length attribute[3] of the character variable (any integer in the range [1,255]).

The external field is *w* positions of the record:

$$\longleftarrow w \longrightarrow$$

$$s_1 \ldots s_r c_1 \ldots c_n$$

where

$c_1 \ldots c_n$ = the alphameric characters

n = the number of characters

w = the width of the external field

r = any remaining positions not used by n ($r = w-n$)

$s_1 \ldots s_r$ = blank spaces (when needed)

The field width *w* can be any value ≥ 1. If *w* is $\geq n$, the output is right-justified in the field with $w-n$ blanks to the left. If *w* is $< n$, the leftmost *w* bytes of the variable are output. The $n-w$ remaining bytes are ignored.

EXAMPLES:

Descriptor	Internal Characters	Variable Type (n =)	Output
A3	SA	Logical or Integer (2)	ΔSA
A3	SAMB	Double Integer[1] or Real (4)	SAM
A7	JANETW	Double Precision[2] (6)	ΔJANETW
A10	BG	Logical or Integer (2)	ΔΔΔΔΔΔΔΔBG
A4	DIXMCG	Double Precision[2] (6)	DIXM
A12	LEFTMOST	Character[3] (8)	ΔΔΔΔLEFTMOST
A6	LEFTMOST	Character[3] (8)	LEFTMO

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real
[3] As defined in a Type statement such as CHARACTER*8 LOCALE (see *HP 3000 FORTRAN (HP 03000-90007)*).

INPUT

On input, the A field descriptor causes transmittal of w positions in an ASCII input record to n bytes of the variable (list element) currently using the field descriptor. If $w \geq n$, the first $w-n$ characters of input are skipped, and n characters are transmitted. If $w < n$, w characters are transmitted to the leftmost bytes of the variable, and all remaining $n-w$ bytes are set to blank.

EXAMPLES:

Descriptor	External Characters	Variable Type (n =)	Internal Result
A3	CAB	Integer or Logical (2)	AB
A2	CA	Integer or Logical (2)	CA
A10	COMPLEMENT	Integer or Logical (2)	NT
A4	REAL	Double Precision[1] (6)	REAL△△
A4	REAL	Double Integer[2] or Real (4)	REAL
A7	PROGRAM	Character[3] (8)	PROGRAM△

[1] In SPL/3000, type LONG real.
[2] Not recognized in FORTRAN.
[3] As defined in a Type statement such as CHARACTER*8 LOCALE (see *HP 3000 FORTRAN (HP 03000-90007)*).

Rw

Rightmost ASCII Characters

FUNCTION: Define a field for ASCII alphameric characters of a variable.

OUTPUT

On output, the R field descriptor causes output of one or more bytes of a variable in ASCII character alphameric form. The maximum number n of bytes (thus, the maximum number of characters) available to a single Rw descriptor depends on the type of the variable: for logical or integer, $n = 2$; for double integer[1] or real, $n = 4$; for double precision,[2] $n = 6$; for character, n = the length attribute[3] of the character variable (any integer in the range [1,255]).

The external field is w positions of the record:

$$\longleftarrow w \longrightarrow$$

$$s_1 \ldots s_r c_1 \ldots c_n$$

where

$c_1 \ldots c_n$ = the alphameric characters

n = the number of characters

w = the width of the external field

r = any remaining positions not used by n ($r = w-n$)

$s_1 \ldots s_r$ = blank spaces (when needed)

The field width w can be any value ≥ 1. If w is $\geq n$, the output is right-justified in the field with $w-n$ blanks to the left. If w is $< n$, the rightmost bytes of the variable are output. The $n-w$ remaining bytes are ignored.

EXAMPLES:

Descriptor	Internal Characters	Variable Type (n =)	Output
R3	SA	Logical or Integer (2)	ΔSA
R3	SAMB	Double Integer[1] or Real (4)	AMB
R7	JANETG	Double Precision[2] (6)	ΔJANETG
R10	BG	Logical or Integer (2)	ΔΔΔΔΔΔΔΔBG
R4	DIXMCG	Double Precision (6)	XMCG
R12	RIGHTMOST	Character[3] (9)	ΔΔΔRIGHTMOST
R6	RIGHTMOST	Character[3] (9)	HTMOST

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.
[3] As defined in a Type statement such as CHARACTER*9 LOCALE (see *HP 3000 FORTRAN (HP 03000-90007)*).

INPUT

On input, the R field descriptor causes transmittal of w positions in an ASCII input record to n bytes of the variable currently using the field descriptor. If $w \geq n$, the first $w-n$ characters of input are skipped, and n characters are transmitted. If $w < n$, w characters are transmitted to the rightmost bytes of the variable, and all bits of the remaining $n-w$ bytes are set to 0 (ASCII Null).

EXAMPLE:

Descriptor	External Characters	Variable Type (n =)	Internal Result
R3	CAB	Integer or Logical (2)	AB
R2	CA	Integer or Logical (2)	CA
R10	COMPLEMENT	Integer or Logical (2)	NT
R4	REAL	Double Precision[1] (6)	*aa* REAL[2]
R4	REAL	Double Integer[3] or Real (4)	REAL
R7	PROGRAM	Character[4] (8)	*a*PROGRAM[2]

[1] In SPL/3000, type LONG real.
[2] *a* = ASCII Null.
[3] Not recognized in FORTRAN.
[4] As defined in a Type statement such as CHARACTER*8 LOCALE (see *HP 3000 FORTRAN (HP 03000-90007)*).

S

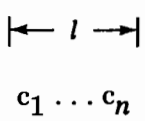
Strings of ASCII Characters

FUNCTION: Define a field for a string of ASCII alphameric characters.

OUTPUT

On output, the S field descriptor causes output of a variable[1] (internal value: character[2] only) in ASCII-character alphameric form.

The external field is *l* positions of the record:



where

$c_1 \ldots c_n$ = the alphameric characters

n = the number of characters

l = the length attribute of the character variable (list element); thus, the width of the external field

EXAMPLES:

NAME Internal Characters	Output
JIM	MY NAME IS JIM JONES
SAM	MY NAME IS SAM JONES

where the list element and length attribute are defined by the Type statement[3] CHARACTER*3 NAME and the format and edit specifications are

("MY NAME IS ",S," JONES")

[1] If the variable (list element) is not type character,[2] SOFTERROR' message FMT: STRING MISMATCH occurs (see "Library Errors").
[2] In SPL/3000, type byte.
[3] See (*HP 3000 FORTRAN (HP 03000-90007)*).

INPUT

On input, the S field descriptor causes transmittal of *l* positions in an ASCII input record to the character variable currently using the field descriptor.

EXAMPLES:

External Characters	DAY Internal Result
MONDAY Δ	MONDAY Δ
SATURDAY	SATURDA

where the list element and length attribute are defined by the Type statement CHARACTER*7 DAY and the format and edit specifications are

("TODAY IS",S)

Scale Factor

The scale factor is a format specification to modify the normalized *output* of the D*w.d*, E*w.d*, and the G*w.d*-selected E*w.d*[1] field descriptors and the fixed-point *output* of the F*w.d*, M*w.d*, and N*w.d* field descriptors. It also modifies the fixed-point and integer (no exponent field) *inputs* to the D*w.d*, E*w.d*, F*w.d*, G*w.d*, M*w.d*, and N*w.d* field descriptors. The scale factor has no effect on output of the G*w.d*-selected F*w.d*[1] field descriptor or floating-point (with exponent field) inputs.

A scale factor is written in one of two forms:

*n*P*f*

or

*n*P*rf*

where

- *n* = an integer constant or – (minus) followed by an integer constant: the scale value
- P = the scale factor identifier
- *f* = the field descriptor
- *r* = a repeat specification—for a field descriptor (described later in this section)

When the Formatter begins to interpret a FORMAT statement, the scale factor is set to zero. Each time a scale factor specification is encountered in that FORMAT statement, a new value is set. This scale value remains in effect for all subsequent affected field descriptors or until use of that FORMAT statement ends.

EXAMPLES:

Format Specifications	Comments
(E10.3,F12.4,I9)	No scale factor change, previous value remains in effect.
(E10.3,2PF12.4,I9)	Scale factor for E10.3 unchanged from previous value, changes to 2 for F12.3, has no effect on I9.

If the FORMAT statement includes one or more nested groups (see "Nesting," this section), the last scale factor value encountered remains in effect.

[1] See descriptions for G*w.d*.

EXAMPLE:

Format Specifications	Comments
(G9.2,2PF9.4,E7.1, 2(D10.2,–1PG8.1))	Scale values resulting are

Descriptor	Scale Value
G9.2	(Unchanged from previous value)
F9.4	2
E7.1	2
D10.2	2
G8.1	–1
D10.2	–1
G8.1	–1

OUTPUT

On output, the scale factor affects D*w.d*, E*w.d*, F*w.d*, M*w.d*, N*w.d*, and G*w.d*-selected E*w.d* field descriptors only.

D*w.d* and E*w.d*

The internal fraction is multiplied by 10^n, and the internal exponent value is reduced by n.

- If $n \leq 0$, the output fraction field has $-n$ leading zeros, followed by $d + n$ significant digits. The least significant digit is rounded.
- If $n > 0$, the output has n significant digits in the integer field, and $(d - n) + 1$ digits in the fraction field. The least significant digit field is rounded.
- The field width specification w normally required may have to be increased by 1.

EXAMPLES:

Scale Factor[1] and Field Descriptor	Internal Value	Output
E12.4	+12.345678	ΔΔΔ.1235E+02
3PE12.4	+12.345678	ΔΔ123.46E–01
–3PE12.4	+12.345678	ΔΔΔ.0001E+05

[1] In "Examples," no scale factor stated implies zero.

F*w.d*, M*w.d*, and N*w.d*

The internal value is multiplied by 10^n, then output in the normal manner.

EXAMPLES:

Scale Factor[1] and Field Descriptor	Internal Value	Output
F11.3	1234.500	ΔΔΔ1234.500
-2PF11.3	1234.500678	ΔΔΔΔΔ12.345
2PF11.3	1234.500678	Δ123450.068
1PM11.3	1234.500678	$12,345.007

G*w.d*-selected E*w.d*

The effect is exactly as described for E*w.d*.

G*w.d*-selected F*w.d*

The scale factor has no effect.

INPUT

On input, the scale factor effect is the same for integer or fixed-field (no exponent field) inputs to the D*w.d*, E*w.d*, F*w.d*, G*w.d*, M*w.d*, and N*w.d* field descriptors. The external value is multiplied by 10^{-n}, then converted in the usual manner.

If the input includes an exponent field, the scale factor has no effect.

EXAMPLES:

Scale Factor[1] and Field Descriptor	External Value	Internal Representation
E10.4	123.9678	.1239678E+03
2PD10.4	123.9678	.1239678E+01
-2PG11.5	123.96785	.12396785E+05
-2PE13.5	1239.6785E+02	.12396785E+06

[1] In "Examples," no scale factor stated implies zero.

Repeat Specification—For Field Descriptors

The repeat specification is a positive integer written to the left of the field descriptor it controls. If a scale factor is also needed, it is written to the left of the repeat specification.

The repeat specification allows one field descriptor to be used for several list elements. It can also be used for nested (groups of) format specifications.

EXAMPLES:

(4E12.4) = (E12.4,E12.4,E12.4,E12.4)

(–2P3D8.2,2I6) = (–2PD8.2,D8.2,D8.2,I6,I6)

(E8.2/3F7.1,3(I6,4HLOAD,D12.3))
= (E8.2/F7.1,F7.1,F7.1,I6,4HLOAD,D12.3,I6,4HLOAD,D12.3,I6,4HLOAD,D12.3)

(2(M8.2)) = (M8.2,M8.2)

EDIT SPECIFICATIONS

Edit specifications are written as an edit descriptor or a repeat specification followed by an edit descriptor.

NOTE: The repeat specification cannot be used directly on the nH or nX edit descriptors. See "Repeat Specification—For Edit Descriptors."

Edit Descriptors

There are six edit descriptors:

Descriptor	Function
" . . . "	Fix the next *n* characters of an edit specification.
' . . . '	Fix the next *n* characters of an edit specification.
*n*H	Initialize the next *n* characters of an edit specification.
*n*X	Skip *n* positions of the external record.
T*n*	Select the position in an external record where data input/output is to begin or resume.
/	Signal the end of a current record and the beginning of a new record.

Detailed descriptions of each edit descriptor follow.

" . . . "
ASCII String (Fixed)

FUNCTION: Fix *n* characters in the edit specification where *n* is the number of ASCII characters enclosed in the *quotation marks.* Any one or more of those characters can be a quotation mark if signaled by an adjacent quotation mark. Any other ASCII characters, including ' (apostrophe), can be used without restriction.

OUTPUT

On output, the " . . . " edit descriptor causes *n* characters to be transmitted to the external record; any adjacent pair of quotation marks is transmitted as one quotation mark.

EXAMPLES:

Edit Descriptor	Output
"OUTPUTΔ" "LOAD" "."	OUTPUTΔ"LOAD".
"USER'SΔPROGRAM"	USER'SΔPROGRAM

INPUT

On input, the " . . . " edit descriptor causes *n* positions of the input record to be skipped. Each pair of adjacent quotation marks counts as one position.

EXAMPLES:

Edit Descriptor	Input	Comment
"HEADINGΔHERE"	THISΔISΔTHEΔSTART	12 positions of the input are skipped.
"HEADINGΔ" "A" "Δ."	THISΔISΔTHEΔENDΔOF	13 positions of the input are skipped.

'. . .'

ASCII String (Fixed)

FUNCTION: Fix *n* characters in the edit specification, where *n* is the number of ASCII characters enclosed in the *apostrophes*. Any one or more of those characters can be an apostrophe if signaled by an adjacent apostrophe. Any other ASCII characters, including " (quotation mark), can be used without restriction.

OUTPUT

On output, the ' . . . ' edit descriptor causes *n* characters to be transmitted to the external record; any adjacent pair of apostrophes is transmitted as an apostrophe.

EXAMPLES:

Edit Descriptor	Output
'PRINTΔ' 'DATA' '.'	PRINTΔ 'DATA'.
'SAM' 'SΔ"SCORE" '	SAM'SΔ "SCORE"

INPUT

On input, the ' . . . ' edit descriptor causes *n* positions of the input record to be skipped. Each pair of adjacent apostrophes counts as one position.

EXAMPLES:

Edit Descriptor	Input	Comment
'COLUMNΔHEAD'	BEGINΔDATAΔINPUT	11 positions of the input are skipped
'ROWΔLABELΔ' 'B' '.'	ENDΔDATAΔINPUT	14 positions of the input are skipped.

*n*H

ASCII String (Variable)

FUNCTION: Initialize the next *n* characters of the edit specification. Any ASCII character is legal. If written, *n* must be a positive integer greater than zero (if omitted, its default value is 1).

OUTPUT

On output, the *n*H edit descriptor causes the current next *n* characters in the edit specification to be transmitted to the external record.

If the edit descriptor has *not* been referenced by a READ statement (see "Input"), the ASCII characters originally written into the edit descriptor are transmitted.

If the edit descriptor has been referenced by a READ statement, the ASCII characters read last are transmitted.

EXAMPLES:

Edit Descriptor	Input Last Read	Output
4HMULT	(None)	MULT
7HFORTRAN	ALGOL△△	ALGOL△△
12HPROGRAM△DATA	BINARY△LOADER	BINARY△LOADE
10HCALCULATED	PASSED△△△△	PASSED△△△△

INPUT

On input, the *n*H edit descriptor causes the next *n* characters of the external record to be transmitted to replace the next *n* characters in the edit specification.

*n*X
ASCII Blanks

FUNCTION: Skip *n* positions of the external record. If written, *n* must be a positive integer greater than zero; if omitted, the default value is 1.

OUTPUT

On output, the *n*X edit descriptor causes *n* positions of the external record to be skipped, typically to separate fields of data.

EXAMPLES:

Format/Edit Specifications	Contents of Numeric List Element(s)		Output
(E7.1,4X,"END")	34.1		△.3E+02△△△△END
		Fields:	7 4
(F8.2,2X,I6)	5.87,436		△△△△5.87△△△△△436
		Fields:	8 2 6

NOTE: This descriptor, when used with the Tn edit descriptor (described later in this section), may cause previous characters to be overlaid.

EXAMPLE:

Format/Edit Specifications	Output
("ABCDEFG", T1, "X", 2X, "Y")	XBCYEFG

INPUT

On input, the *n*X edit descriptor causes the next *n* positions of the input record to be skipped.

EXAMPLES:

Format/Edit Specifications	External Record Input	Data Transmitted to List Elements
(D8.2,3X,M9.2)	△.25E+02END$1,563.79	.25E+02, 1563.79
(5X,E9.2,I5)	54321–98.7563814581	–.9876538E+02, 14581

Tn

Position (Tabulate) Data

FUNCTION: Select the position (tabulation) in an external record where data input/output is to begin or resume.

The T*n* edit descriptor positions the record pointer to the *n*th position in the record.

OUTPUT EXAMPLES

1. **Format/Edit Specifications**
   (T10,"DESCRIPTION", T25, "QUANTITY", T1, "PART△NO.")

   Result
   PART△NO.△DESCRIPTION△△△△QUANTITY
   ↖ *position #1* ↖ *position #10* ↖ *position #25*

2. **Format/Edit Specifications**
   (T25,I3,T1,3A2,T10,3A4)

   Contents of List Elements
   125,HR124A,LOCK-WASHERS

   Result
   HR124A△△△LOCK-WASHERS△△△125
   ↖ *position #1* ↖ *position #10* ↖ *position #25*

INPUT EXAMPLE

Format/Edit Specifications
(T13,E8.2,T1,I4,T24,M12.3)

Input
1325COUNTED△△△525.78LBS△△$4,365.78△COST
↖ *position #1* ↖ *position #13* ↖ *position #24*

Results in List Elements
.52578E+03, 1325, .436578E+04

As can be seen in the above examples, the position numbers *n* need not be given in ascending order.

NOTE: This descriptor may cause previous characters to be overlaid (see nX descriptions, earlier in this section).

/

Record Terminator

FUNCTION: Terminate the current external record and begin a new record (on a line printer or a keyboard terminal, a new line; on a card device, a new card; etc.).

OUTPUT and INPUT

The / edit descriptor has the same result for both output and input: it terminates the current record and begins a new record.

If a series of two or more / edit descriptors are written into a FORMAT statement, the effect is to skip $n-1$ records, where n is the number of /'s in the series. A series of /'s can be written by using the repeat specification.

NOTE: If one or more / edit descriptors are the first item(s) in a series of format specifications, n (not n–1) records are skipped for that series of /'s.

EXAMPLES:

Format Specifications	Output	Record #
(E12.5,I3/"END")	ΔΔ.32456E+04Δ95	1
	END	2
(E12.5,I3///"END")	ΔΔ.32456E+04Δ96	1
		2
		3
	END	4
(I5,3HEND,4/"NEW DATA")	43592END	1
		2
		3
		4
	NEW DATA	5
(2/"END")		1
		2
	END	3

The / edit descriptor can also be used without a comma to separate it from other format and/or edit specifications; it has the same separating effect as a comma.

Repeat Specification—For Edit Descriptors

The repeat specification is a positive integer written to the left of the edit descriptor it controls. It is written as *r*" . . . " or *r*' . . . ' or *r*(*n*H) or *r*(*n*X) or *r*/, where *r* is the repetition value.

NOTE: The forms r(nH) and r(nX) may include other field and/or edit descriptors within the parentheses.

EXAMPLES:

(E9.2/3F7.1,2(4HDATA)) = (E9.2/F7.1,F7.1,F7.1,4HDATA,4HDATA)

(2(5HABORT2/)) = (5HABORT,//,5HABORT//)

(G10.3,3("READ"E12.4)) = (G10.3,"READ"E12.4,"READ"E12.4,"READ"E12.4)

SPECIFICATION INTERRELATIONSHIPS

Two or more specifications (E9.3,I6) in a FORMAT statement are concatenated: Data 12.3 and –30303 produces

△.123E+02	-30303

The *n*X edit specification (E9.3,4X,I6) can insert blank spaces between fields: The same data produces

△.123E+02	△△△△	-30303

Or the / edit specification (E9.3/I6) places each field on a different line: The same data produces

△.123E+02
-30303

Nesting

The group of format and edit specifications in a FORMAT statement can include one or more other groups enclosed in parentheses (in this text, called "group(s) at nested level *x*"). Each group at nested level 1 can include one or more other group(s) at nested level 2; those at level 2 can include group(s) at nested level 3; those at level 3 can include group(s) at level 4:

(E9.3,I6,(2X,I4))	One group at nested level 1.
(T12,"PERFORMANCES"3/(E10.3,2(A2,L4)))	One group at nested level 1, one at nested level 2.
(T5,5HCOSTS,2(M10.3,(I6,E10.3,(A2,F8.2))))	One group at nested level 1, one at level 2, one at level 3.

A FORTRAN READ or WRITE statement references each element of a series of list elements; the Formatter scans the corresponding FORMAT statement to find a field descriptor for each element. As long as a list element and field descriptor pair occurs, normal execution continues. Formatter execution continues until all list elements have been transmitted.

Unlimited Groups

If a program does not provide a one-to-one match between list elements and field descriptors, Formatter execution continues only until all list elements have been transmitted. If there are fewer written field descriptors than list elements, format specification groups at nested level 1 and deeper are used as "unlimited groups." After the effective rightmost field descriptor in a FORMAT statement has been referenced (see "Repeat Specifications—For Field Descriptors"), the Formatter performs three steps:

1. The current record is terminated: on output, the current field is completed, then the record is transmitted; on input, the rest of the record is ignored.
2. A new record is started.
3. Format control (field descriptor interpretation) is returned to the repeat specification for the rightmost specification group at nested level 1. Or, if there is no group at level 1, control returns to the first field descriptor (and its repeat specification) in the FORMAT statement.

NOTE: In any case, the current scale factor is not *changed until another scale factor is encountered (see "Scale Factor").*

EXAMPLES:

(I5,2(3X,F8.2,8(I2)))	Control returns to 2(3X.F8.2,8(I2))
(I5,2(3X,F8.2,8(12I2)),4X,(I6))	Control returns to (I6)
(I5,3X,4F8.2,3X)	Control returns to (I5,3X,4F8.2,3X)
("HEADER" /3(E10.2))	Control returns to 3(E10.2) to produce:

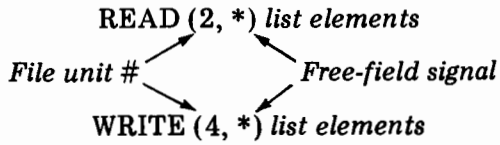
HEADER

E10.2	E10.2	E10.2
E10.2	E10.2	E10.2
E10.2	E10.2	E10.2

FREE-FIELD INPUT/OUTPUT

Free-field input/output is formatted conversion according to format and/or edit control characters imbedded in the data. That is, the Formatter converts data from or to external ASCII character form without using FORMAT statements. For free-field inputs, format and/or edit control characters are imbedded in the external data fields. For free-field outputs, predefined field and edit descriptions are used.

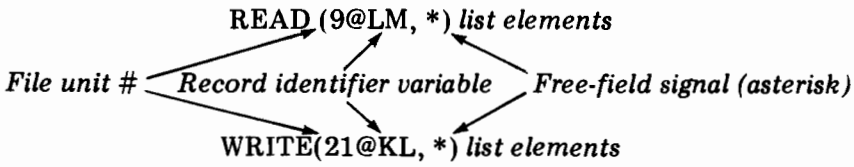
For free-field input/output, FORTRAN READ or WRITE statements are written with an asterisk instead of a FORMAT statement identifier:



For free-field input/output to or from disc devices (see "Disc Input/Output," earlier in this section), READ or WRITE statements in a FORTRAN program are written:

For sequential access: As described on the preceding page for free-field input/output.

For direct access:



Free-Field Control Characters

Special ASCII characters embedded in the external data fields control free-field input:

Character(s)	Function
(Blank space) or , (comma) or any ASCII character not part of the data item.	Data item delimiter (terminator)
/ (slash)	Record terminator (when not part of a character string data item)
+ (plus) or – (minus)	Sign of data item
. (period)	Define the beginning of the fraction subfield of the data item
E or + or – or D	Define the beginning of the exponent subfield of the data item
% (percent)	Define the data item as octal (not decimal)
" . . . "	A character string enclosed by quotation marks; to be input to a FORTRAN/3000 type character variable (or SPL/3000 type byte array)
<< . . . >>	A character string enclosed by << and >>; the characters are a comment only for the external record; the string and symbols are ignored on input

Free-Field Input

Six data types can be input to free-field conversion: octal, integer, double integer,[1] floating-point (real), double-precision floating point,[2] and character string. Numeric data types can be mixed freely with numeric list elements. For example, an integer data item can be input to a floating-point list element; the Formatter converts the integer to floating-point form and stores the double-word result.

All rules for input to numeric and alphameric conversions (see "Field Descriptors") apply.

[1] Not recognized in FORTRAN.
[2] In SPL/3000, type LONG real.

A character string item, however, must be input only to a character string list element; if not, SOFTERROR' message FMT: STRING MISMATCH: occurs (see "Library Errors") and Formatter execution is aborted.

DATA ITEM DELIMITERS

A data item is any numeric or character string field occurring between data item delimiters. A data item delimiter is a comma, *a blank space*, or any ASCII character that is not a part of the data item. The initial data item need not be preceded by a delimiter; the function of a delimiter is to signal the end of one data item and the beginning of another.

Two commas with no data item in between indicate that no data item is supplied for the corresponding list element, and the previous contents of that list element are to remain unchanged. Any other delimiter appearing two or more consecutive times is equivalent to one delimiter.

NOTE: Do not include a "no-data" field in a series of free-field data inputs. For example, a remark field such as REMARK: I=1234 IS CORRECT *will not prevent the digits 1234 from being interpreted as a free-field data item.*

DECIMAL DATA

Decimal data items are written in any of the forms described under "Field Descriptors," except the monetary or the numeration forms. Imbedded commas or the dollar sign are data item delimiters.

NOTES:

1. *Leading, imbedded, or trailing blanks or commas, $, etc., are data item delimiters.*
2. *All integer inputs have an implicit decimal point to the right of the last (least significant) digit.*
3. *The exponent field input can be any of several forms:*

+e	*+ee*	*Ee*	*Eee*	*De*	*Dee*
–e	*–ee*	*E+e*	*E+ee*	*D+e*	*D+ee*
		E–e	*E–ee*	*D–e*	*D–ee*

where e is an exponent value digit.

OCTAL DATA

Octal data items are written

$$\%i_1 \ldots i_n$$

where

$i_1 \ldots i_n$ = the octal integer digits

n = the number of octal digits (maximum: 6)

% is the octal data identifier

Non-octal digits are delimiters. The largest number allowed is 177777_8. If n is greater than 5, the first (most significant) digit must be 0 or 1.

CHARACTER STRING DATA

A character string data item is any series of ASCII characters, including blank spaces, enclosed in quotation marks. Any one or more of those characters can be a quotation mark if signaled by an adjacent quotation mark.

The corresponding list element must be of type CHARACTER in FORTRAN/3000 (or type BYTE ARRAY in SPL/3000) of a specified string length. If the number of characters in the data item is greater than the length attribute *n* of the list element, *n* characters are transmitted and the remaining characters are ignored. If there are fewer characters than *n*, all characters of the data item are transmitted, left-justified in the list element, followed by trailing blanks.

If an end-of-record condition occurs before the terminating quotation mark of a character string data item, the Formatter assumes the data item is continued in the next record and resumes transmission with the first character of the next record.

RECORD TERMINATOR

The character / (slash), if not part of a character data item, terminates the current record and delimits the current data item. If this occurs before all list elements have been satisfied, the remainder of the current record is skipped and transmission resumes with the first character of the next record.

LIST TERMINATION

If an end-of-record condition occurs without the record terminator /, the effect is to end the list of variables. Any list elements not satisfied are left unchanged.

Free-field Output

Five data types can be output under free-field conversion: integer, double-integer,[1] floating-point (real), double precision floating-point,[2] and character string. All output is compatible with the requirements of free-field input: it does not require external changes to be input using free-field conversion.

1. Integer data items are output under the I6 field description.
2. Double-integer data items are output under the I11 field description.
3. Floating-point data items are output under the G12.6 field description.
4. Double-precision floating-point data items are output under the G17.11 field description.
5. Character string data items are output under the " . . . " edit description; the adjacent quotation marks rule for included quotation marks is used. (If a quotation mark is included in the string, a double quote is output.)

[1] Not recognized in FORTRAN.
[2] In SPL/3000 type LONG real.

DATA ITEM DELIMITER

Each field in the output record is delimited by one blank space.

RECORD TERMINATORS

If the width of a current numeric data item is too great for the remainder of a current record, a record terminator character / is output, and a new record is started with the first character of the data item.

If a character string data item overlaps record boundaries, subsequent records are output (without record terminator slashes) until the entire character string has been transmitted.

ACCEPT/DISPLAY

FORTRAN/3000 ACCEPT and DISPLAY are alternate applications of free-field input and output. They are invoked by program statements such as

ACCEPT INT,ARRAY,LETR or DISPLAY INT,ARRAY,LETR

where INT, ARRAY, and LETR are typical list elements. The key words ACCEPT and DISPLAY are equivalent to READ(*i*, *) and WRITE(*o*, *), where *i* is the MPE/3000 file system name of a standard input device $STDIN and *o* is the name of a standard output device $STDLIST, and * is the free-field signal.

Transmissions by ACCEPT and DISPLAY conform to the descriptions given for free-field input and output, with one exception: the Formatter determines if the standard output device to be used is an interactive terminal (such as a teleprinter or a CRT keyboard/display); if the device is an interactive terminal, the ACCEPT routine prints a prompt character, ?, before accepting inputs.

CORE-TO-CORE CONVERSION

Conversions between external ASCII records and a list of variables use an input/output (I/O) buffer allocated to the Formatter. Core-to-core conversions, on the other hand, transfer to and from user-defined buffers (byte arrays). The user can manipulate the data, transmit it to or from external records, or return it to the original location or any other location.

To invoke core-to-core conversion FORTRAN READ and WRITE statements are written:

READ (*v*,*f*) *list elements* or WRITE (*v*,*f*) *list elements*

where

v = a character simple variable or a character array element

f = the FORMAT statement identifier

Core-to-core conversions are subject to the same rules, restrictions, and interactions as formatted or free-field conversions to and from external records, with the following exceptions:

1. Any signal to terminate the current record and start a new record (such as edit specification /, or free-field record terminator /, or the end of an unlimited group sequence) is taken to be an error; SOFTERROR' message FMT: BUFFER OVERFLOW occurs (see Section IV, "Library Errors").
2. If an end-of-record condition occurs before either a terminating quotation mark (") or a close comment symbol (>>) is encountered in free-field data, SOFTERROR' message FMT: BUFFER OVERFLOW occurs.

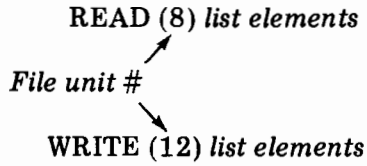
UNFORMATTED (BINARY) TRANSFER

Data can be transferred to and from disc or tape files in internal representation (binary) form without any conversion. Such transfers are faster and occupy less space than formatted data transfers.

Two types of access to files on disc devices are available through the MPE/3000 file system: sequential or direct. Either type can be established through the MPE/3000 file intrinsic FOPEN.

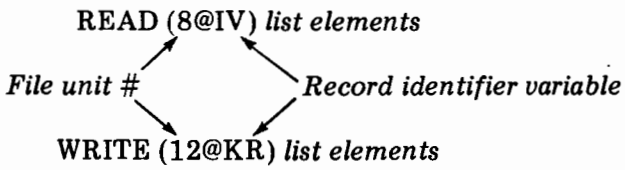
When binary /sequential access is used, the READ or WRITE statements of a FORTRAN program are written without a FORMAT statement identifier.

EXAMPLES:



When binary /direct access is used, the READ or WRITE statements of a FORTRAN program are written with an integer simple variable for the record identifier and without a FORMAT statement identifier.

EXAMPLES:



When the file is opened (through the MPE/3000 file intrinsic FOPEN), the record size can be left at the system default value 128, or the user can specify a different size.

In sequential access, as many records as needed are used in sequence until the entire list of elements has been transferred.

NOTE: If the storage required exceeds the size of the record, transfer continues into the next record; this usually leaves part of that next record unused.

In direct access, record access is terminated by the last element in the list. Any unused portion of the record just terminated is ignored.

If the storage required by all the elements in the list exceeds the record size, SOFTERROR' message FMT: DIRECT ACCESS OVERFLOW occurs (see "Library Errors").

Matching List Elements

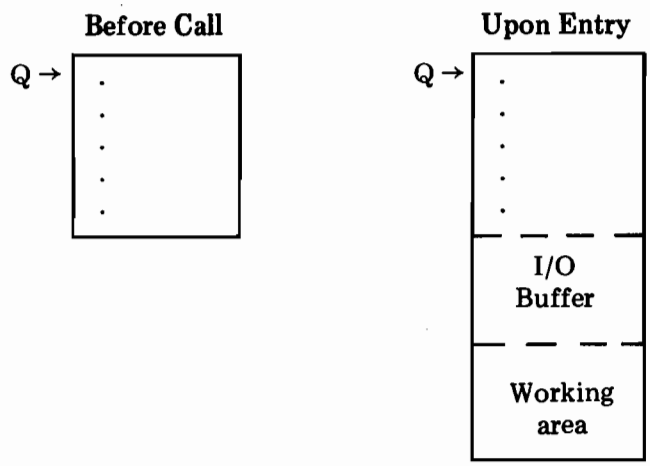
The binary transfer user must match list elements between corresponding READ and WRITE statements of a FORTRAN program. For example, if a list of elements is transferred to a disc, any corresponding return of the data to internal storage must do so to a list that matches each element by type and dimensions and by order of appearance in the list. The simplest method is to use the same element labels for input and output, if possible.

NOTE: Under binary /direct access, the Formatter begins each new list element output at a word boundary. If the list element is, for example, a byte array of an odd number of bytes, one byte of the record will not be used.

NOTE: The following descriptions assume the reader has a working knowledge of SPL/3000; see HP 3000 Systems Programming Language, (HP 03000-90002).

To summarize, Formatter executions follow these steps:

1. An initialization call is made (either by a compiler-generated code or by an SPL/3000 program). Parameters are included in the call (for example, a flag indicating input or output and a pointer to the format and/or edit parameters).
2. If the type of transfer is *not* to be core-to-core, the Formatter allocates space on the user's stack for the I/O buffer and working areas and saves the location of the working area in DB–2. The Q register and the stack marker's ΔQ entry are modified to prevent deallocation of the I/O buffer and working area upon initialization exit. If the direction of transfer is input, data transmission to the I/O buffer begins at this time, and control is returned to the user.
3. The user now makes a call for each element in the list of variables. Parameters in the Formatter's working area can be examined to determine the current positions in the series of format and/or edit parameters and in the I/O buffer.



Stack Conditions for the Formatter

4. When the list of variables has been satisfied, the user must make a termination call. If the direction of transfer is output, transmission of the last record begins at this time. The Q register and ΔQ in the stack marker are modified to assure deallocation of the I/O buffer and the working area upon termination exit.
5. When data transmission is complete, the user's stack and location DB-2 are restored to the conditions existing before the initialization call.

Calling Sequences

SPL/3000 calling sequences to the Formatter must be based on the Formatter procedure declarations as defined in the following paragraphs.

INITIALIZATION

Declaration:
```
PROCEDURE FMTINIT' (FORMAT, UNIT, REC, IOTYPE, LAST);
  VALUE UNIT, REC, IOTYPE, LAST; INTEGER UNIT, IOTYPE, LAST;
  DOUBLE REC; BYTE ARRAY FORMAT;
    OPTION EXTERNAL;
          .
          .
          .
```

Parameters:

FORMAT = For formatted conversions, a byte array containing format and edit parameters; or

for free-field conversions or unformatted (binary) transfers, ignored.

UNIT = For transfers to a FORTRAN/3000 logical unit numbered file, a positive integer in the range [1,99] to specify that unit number to be used; or

for transfers to a user-defined MPE/3000 file, the negated file number to be used; or

for core-to-core conversions, the size, in bytes, of the user's internal buffer.

NOTE: If UNIT is a FORTRAN/3000 logical unit number [1,99], the Formatter uses the FORTRAN/3000 Logical Unit Table (FLUT) to open the file.

If UNIT is a negated file number, the user must have previously opened the file through the MPE/3000 file intrinsic FOPEN.

In either case, see "File System Requirements" later in this section.

REC = For direct access to a file, a double integer record number; or

for core-to-core conversions, the second word of REC is a byte pointer to the user's internal buffer and the first word of REC is not used.

IOTYPE = Individual bits of this integer are used as follows:

Bit(s)	Function
15	Clear for output; set for input.
14	Set for ACCEPT/DISPLAY; clear for any other function.
13	Clear for sequential access to a file; set for direct access.
12	Clear for formatted or free-field conversions; set for unformatted (binary) transfers.
11	Clear to call SOFTERROR' routine (see "Library Errors") for end-of-file errors; set to not call SOFTERROR'.
10	Clear to call SOFTERROR' routine for irrecoverable file errors; set to not call SOFTERROR'.
9	Set for core-to-core conversions; clear for any other function.
8	Set for free-field conversions; clear for any other function.
7-0	Spares.

LAST = Label identifier of the instruction that immediately follows the Formatter termination call.

LIST ELEMENT TRANSFERS

Ten entry-point procedures to the Formatter are provided for transfers of various types of list elements. The procedures' declarations are written as follows:

```
PROCEDURE IIO' (LOC);
  INTEGER LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE DIO' (LOC);
  DOUBLE LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE RIO' (LOC);
  REAL LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE LIO' (LOC);
  LONG LOC;
    OPTION EXTERNAL;
          .
          .
```

For type integer, logical (boolean), octal, and two-byte ASCII character.

For type double-integer and four-byte ASCII character.

For type real (two-word floating point) and four-byte ASCII character.

For type LONG real (three-word floating point) and six-byte ASCII character.

```
PROCEDURE SIO' (SLEN, LOC);
  VALUE SLEN;
  INTEGER SLEN; BYTE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE AIIO' (DIM, LOC);
  VALUE DIM; INTEGER DIM;
  INTEGER ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ADIO' (DIM, LOC);
  VALUE DIM; INTEGER DIM;
  DOUBLE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ARIO' (DIM, LOC);
  VALUE DIM; INTEGER DIM;
  REAL ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ALIO' (DIM, LOC);
  VALUE DIM; INTEGER DIM;
  LONG ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ASIO' (SLEN, DIM, LOC);
  VALUE SLEN, DIM;
  INTEGER SLEN, DIM;
  BYTE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
```

For an ASCII character string.

For an array of the same types as IIO'.

For an array of the same types as DIO'.

For an array of the same types as RIO'.

For an array of the same types as LIO'.

For an array of ASCII character strings.

The parameters are

LOC = For a non-array list element, a reference parameter; or for an array list element, the array identifier.

SLEN = A positive integer to specify the string length in bytes.

DIM = The number of elements (not words or bytes) in the array.

TERMINATION

The call is written

```
TFORM';
LAST: (the next SPL/3000 program statement)
```

No parameters are required. On output, the data in the Formatter's I/O buffer is transmitted at this time. Then the user's stack is restored to the conditions existing before the initialization call. Now the user can check for a CCA error indication. If CCA = CCG, an end-of-file error occurred; if CCA = CCE, no error occurred; if CCA = CCL, an irrecoverable file error occurred.

EXAMPLE: A Complete Data Transfer

Statement No.	The Statement
1	FMTINIT'(FMT,10,ID,%74,@LAST);
2	X(0) := A + B;
3	X(1) := C/D;
4	ARIO'(X,2);
5	IIO'(I);
6	TFORM';
7	LAST: IF < THEN GO TO IOERROR;
8	IF > THEN GO TO EOTERROR;

Description

Statement 1 initializes the Formatter to

- Ignore label FMT
- Use file FTN10
- Use record number under label ID
- Call SOFTERROR' for end-of-file errors
- Call SOFTERROR' for irrecoverable file errors

Statements 2 and 3 demonstrate that computations can be made within a calling sequence, in this case, to prepare the contents of a two-element real array.

Statement 4 is a call to output the real array. In FORTRAN/3000, this is the method for output of a type complex quantity.

Statement 5 is a call for output of I (which could be integer, logical, two-byte ASCII character, or octal).

Statement 6 is the termination call.

Statements 7 and 8 are user-decisions to check for I/O error or end-of-tape error, respectively.

NOTE: The following descriptions assume the reader has a working knowledge of MPE/3000; see HP 3000 Multiprogramming Executive Operating System (HP 03000-90005).

FORTRAN/3000 LOGICAL UNIT TABLE (FLUT)

For FORTRAN/3000 programs using the Formatter, the MPE/3000 System loader prepares a FORTRAN Logical Unit Table (FLUT). The SPL/3000 user, however, must prepare a FLUT in his DB Data Area and initialize location DB-1 to reference the byte address of the FLUT:

DB-1 → | I_{fa} |

where

I_{fa} is a positive integer to specify the FLUT byte displacement from DB.

The FLUT is written:

DB + I_{fa} →		
	U_1	F_1
	U_2	F_2
	.	.
	.	.
	U_n	F_n
	255	//////

← *The terminal entry (required)*

where I_{fa} is defined above and

$U_1 \ldots U_n$ = the UNIT numbers (integers in the range [1,99]) in the left byte of each entry, to be specified in Formatter initialization calls

$F_1 \ldots F_n$ = 0, when the FLUT is prepared

The last U entry must be 255 to signal the end of the FLUT

For the special free-field conversions ACCEPT and DISPLAY, two U entries must be included in the FLUT: 5 for $STDIN and 6 for $STDLIST. For full details of these standard device names, including the ability to equate FORTRAN file names FTN05 and FTN06 to other device names, see *HP 3000 Multiprogramming Executive Operating System (HP 03000-90005).*

When the Formatter is initialized, it must determine if the file to be used has been opened, and if it has, what the file parameters are (such as the file options, the access options, etc.). Thus, a global data area is required for storage of the file data.

The Formatter first checks the FLUT for a U entry corresponding to the UNIT specified in the initialization call. If such an entry does not exist, SOFTERROR' message FMT: FILE NOT IN TABLE occurs and the Formatter terminates. If one does exist, the F entry is checked.

If the F entry is zero, the file has not been opened and the Formatter makes a call to the MPE/3000 file intrinsic FOPEN. The nominal FORTRAN/3000 parameters (as described below) are used in the FOPEN call. These include the file name created by appending the UNIT number to the ASCII characters FTN. For example, the file name for UNIT 3 is FTN03. The FOPEN intrinsic returns an integer (stored in the FLUT) as the F entry for the UNIT referenced.

If the F entry is not zero, the file has already been opened and the Formatter calls the MPE/3000 file intrinsic FGETINFO to extract the file parameters and store them in the global data area. The Formatter also allocates space on the stack for its I/O buffer, according to the size indicated in the file parameter RECSIZE.

NOMINAL FORTRAN/3000 PARAMETERS

The following parameters can be superseded with an MPE/3000 :FILE command.

FILEDESIGNATOR	FTN*dd*, where *dd* is the UNIT number in the FLUT (for example, FTN03).
FOPTIONS	All 16 bits are cleared for the following file options: Domain Specifications (bits 15 and 14): NEW BINARY file (bit 13) Default File Designator (bits 12, 11, 10): 000[1] Record Format (bits 9 and 8): VARIABLE Carriage Control (bit 7): none[2] Nolabel Option (bit 6): no, use standard label (Bits 5 through 0 are spares.)
AOPTIONS	The bits are set to a pattern for the following access options: Access (bit 15 clear): SEQUENTIAL Input Possible (bit 14 set): YES Output Possible (bit 13 set): YES Update (bit 12 clear): NO Append (bit 11 clear): NO Inhibit (bit 10 set): NO Exclusive Access (bit 9 clear; bit 8 set): all modes Retain (bit 7 set): YES Dynamic Locking (bit 6 clear): NO (Bits 5 through 0 are spares.)
RECSIZE	System default value: 128 words.
DEVICE	System default: DISC.

[1] Except for FTN05: 100 for $STDIN, and FTN06: 001 for $STDLIST.
[2] Except for FTN06, yes (set).

FORMMSG	None.
RECMODE	Clear (800 BPI—magnetic tape, packed binary—card reader).
BLOCKFACTOR	System default value: 1.
NUMBUFFERS	System default: 1.
FILESIZE	System default: (see *HP 3000 Multiprogramming Executive Operating System (HP 03000-90005)*)
NUMEXTENTS	System default: (see *HP 3000 Multiprogramming Executive Operating System (HP 03000-90005)*)
INITIALLOC	System default value: 0.
FILECODE	System default: 0.

ACCEPT/DISPLAY OPTION

The ACCEPT/DISPLAY option is assumed to be used with an interactive terminal, such as a teleprinter or a CRT keyboard/display (devices used for both input reading and output listing). However, any two separate devices can be used instead by predefining the file parameters for FTN05 and FTN06 (see the preceding subsection "Nominal FORTRAN/3000 Parameters"). When the Formatter becomes aware (by examination of the file parameters through the FGETINFO intrinsic) that the terminal is not an interactive device (the terminal cannot output information), it only reads inputs; it does not output a ? and then read an input.

SECTION II
MATHEMATICAL PROCEDURES

To find the descriptions for any given procedure in this section, see the Function Directory.

DABS'

FUNCTION: Calculate the absolute value of a double precision (LONG real) number.

Declaration:

```
LONG PROCEDURE DABS' (Y);
  VALUE Y;LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameter: Any double precision number except the smallest negative number (-2^{-256}).

Result: A double precision number.

FORTRAN: Intrinsic Function: DABS (*Y*).

Error: The absolute value of the smallest negative number is not representable; the result is given as zero.

CABS (or CABS')

FUNCTION: Calculate the absolute value of a complex number.

Declaration:
```
REAL PROCEDURE CABS(Y); or CABS'(Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method:

$Y = a + bi$

$Y(0) = a$(real part)

$Y(1) = b$(imaginary part)

when $|a| > |b|$, CABS = $|a| \sqrt{1 + \left(\frac{b}{a}\right)^2}$

when $|b| \geq |a|$, CABS = $|b| \sqrt{1 + \left(\frac{a}{b}\right)^2}$

Accuracy: Depends on accuracy of SQRT.

ATTRIBUTES:

Parameter: Any complex number representable in two real numbers, one for a and one for b.

Result: A non-negative real number.

FORTRAN: Basic External Function: CABS (Y).

Error: If a and b are near the overflow threshold (a and $b \approx 10^{77}$), the SOFTERROR' message CABS: OVERFLOW occurs (see "Library Errors").

ISIGN'

FUNCTION: Calculate the absolute value of a first integer number and give it the sign of a second integer number.

Declaration:
```
INTEGER PROCEDURE ISIGN' (J,K);
  VALUE J,K;INTEGER J,K;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: ISIGN' (J,K) = sign of K times |J|

ATTRIBUTES:

Parameters: Both arguments are integer numbers, if the second is zero, the sign is assumed to be positive.

Result: An integer number.

FORTRAN: Intrinsic Function: ISIGN (*J,K*).

Error: None.

SIGN'

FUNCTION: Calculate the absolute value of a first real number and give it the sign of a second real number.

Declaration:
```
REAL PROCEDURE SIGN' (Y,Z);
  VALUE Y,Z;REAL Y,Z;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: SIGN' (Y,Z) = sign of Z times |Y|

ATTRIBUTES:

Parameters: Both arguments are real numbers; if the second is zero, the sign is assumed to be positive.

Result: A real number.

FORTRAN: Intrinsic Function: SIGN (*Y,Z*).

Error: None.

DSIGN'

FUNCTION: Calculate the absolute value of a first double precision (LONG real) number and give it the sign of a second double precision (LONG real) number.

Declaration:
```
LONG PROCEDURE DSIGN' (Y,Z);
  VALUE Y,Z;LONG Y,Z;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: DSIGN'(Y,Z) = sign of Z times |Y|

ATTRIBUTES:

Parameters: Both arguments are double precision numbers; if the second is zero, the sign is assumed to be positive.

Result: A double precision number.

FORTRAN: Intrinsic Function: DSIGN (*Y,Z*).

Error: None.

INT' (or IFIX')

FUNCTION: Truncate a real number to an integer number.

Declaration:
```
INTEGER PROCEDURE INT'(Y); or IFIX'(Y);
  VALUE Y; REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: INT'(Y) or IFIX'(Y) = sign of Y times largest integer $\leq |Y|$

ATTRIBUTES:

Parameter: A representable[1] real number in the range [-32768.0, 32767.0].

Result: An integer number.

FORTRAN: Intrinsic Function: INT (*Y*) or IFIX (*Y*).

Error: If the real number is outside the range stated, the arithmetic trap INT OFLW (integer overflow) occurs (if traps are enabled).

[1] See "Introduction."

AINT'

FUNCTION: Truncate a real number to an integer number in real representation.

Declaration:
```
REAL PROCEDURE AINT' (Y);
  VALUE Y; REAL Y;
    OPTION EXTERNAL;
              .
              .
              .
```

Method: AINT' (Y) = sign of Y times largest integer $\leq |Y|$

ATTRIBUTES:

Parameter: A real number.

Result: A real number.

FORTRAN: Intrinsic Function: AINT (*Y*).

Error: None.

DDINT'

FUNCTION: Truncate a double precision (LONG real) number to an integer number in double precision (LONG real) representation.

Declaration:

```
LONG PROCEDURE DDINT' (Y);
  VALUE Y; LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: DDINT' (Y) = sign of Y times largest integer $\leq |Y|$

ATTRIBUTES:

Parameter: A double precision number.

Result: A double precision number.

FORTRAN: Intrinsic Function: DDINT (*Y*).

Error: None.

DFIX'

FUNCTION: Truncate a LONG real number to a double integer number.

Declaration:

```
DOUBLE PROCEDURE DFIX' (Y);
  VALUE Y; LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: DFIX' = sign of Y times largest double integer $\leq |Y|$

ATTRIBUTES:

Parameter: A LONG real number.

Result: A double integer number.

FORTRAN: Not callable.

Error: If the truncated LONG real number cannot be represented in the two words of the double integer, arithmetic trap INT OFLW (integer overflow) occurs (if traps are enabled).

DFLOAT'

FUNCTION: Convert a double integer to LONG real representation.

Declaration:
```
LONG PROCEDURE DFLOAT' (J);
  VALUE J; DOUBLE J;
    OPTION EXTERNAL;
          .
          .
          .
```

ATTRIBUTES:

Parameter: A double integer number.

Result: A LONG real number.

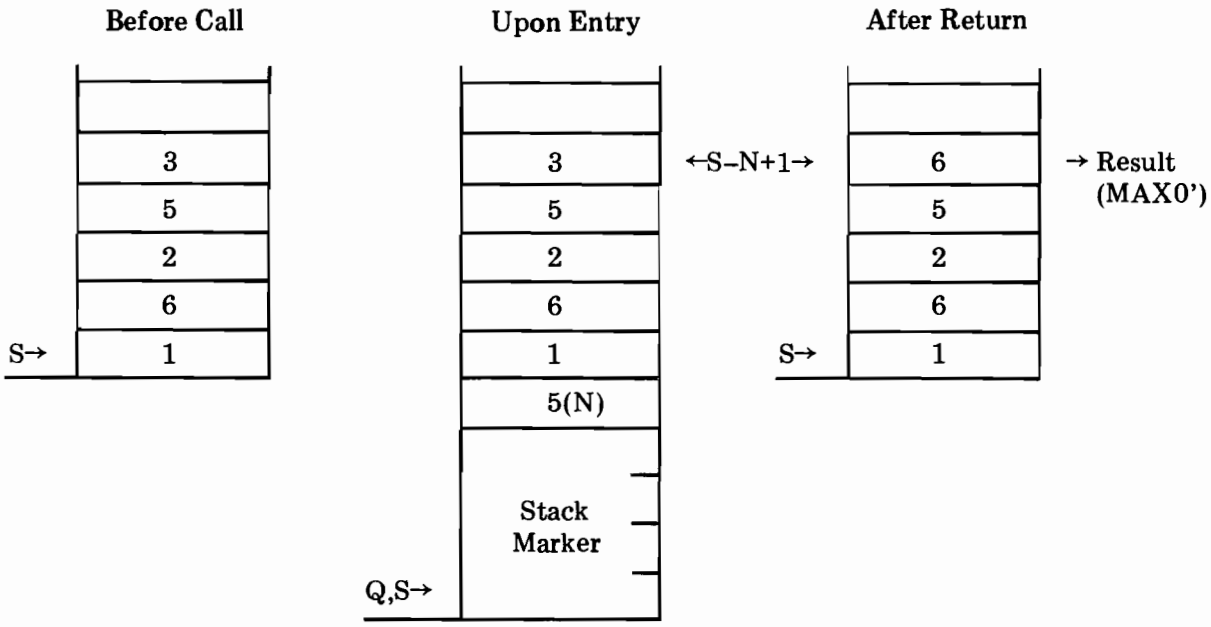
FORTRAN: Not callable.

Error: None.

MAX0'/MIN0'

FUNCTION: Calculate the largest (MAX0') or smallest (MIN0') of N integers on top-of-stack and return that integer in S–N+1.

Declaration:
```
PROCEDURE MAX0'(N); or MIN0'(N);
  VALUE N; INTEGER N;
    OPTION EXTERNAL;
          .
          .
          .
```



ATTRIBUTES:

Parameter: An integer number ⩾ 2.

Result: An integer number.

FORTRAN: Intrinsic Function: MAX0 (*A,B,C,. . .*) or MIN0 (*A,B,C,. . .*).

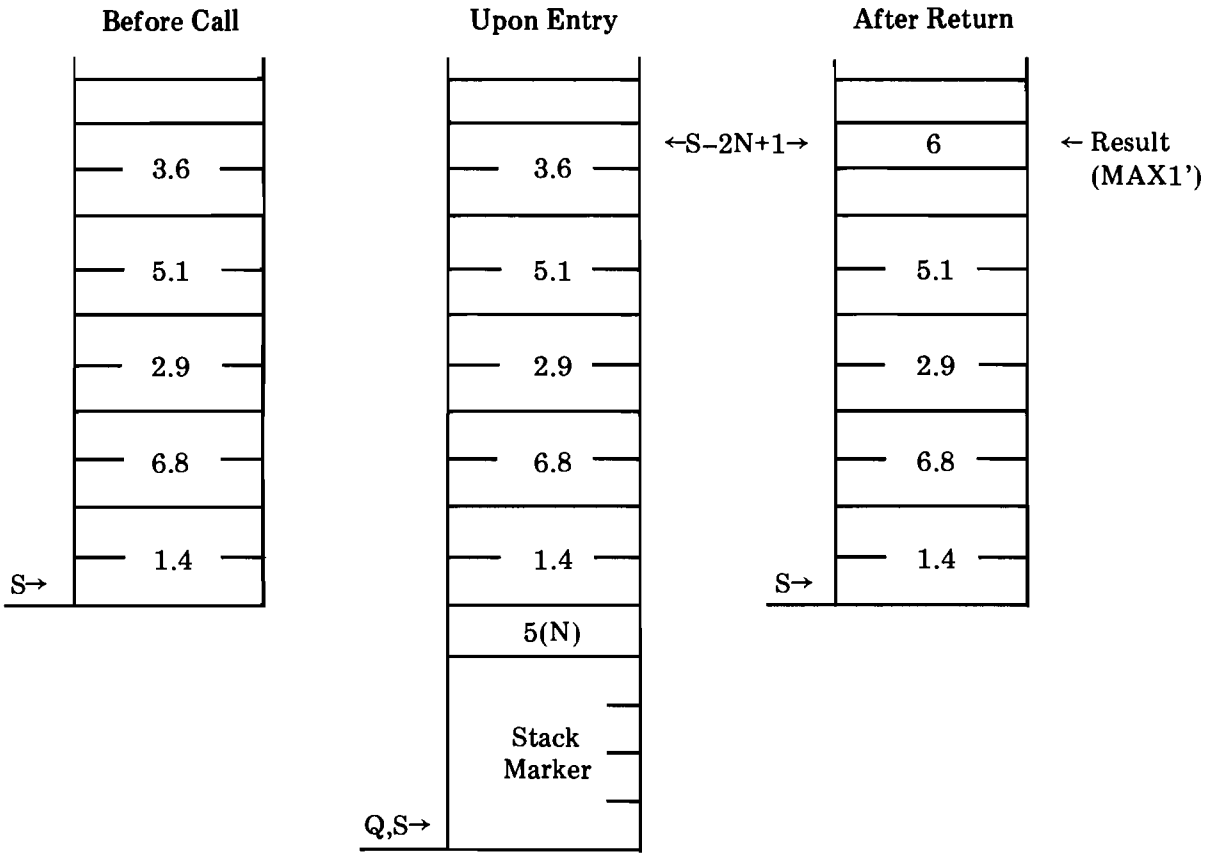
Error: If the argument (or number of parameters) is less than 2, no action occurs.

COMMENT: The SPL/3000 caller must cut back the stack after return (for example, use an ASSEMBLE (SUBS 4); statement).

MAX1'/MIN1'

FUNCTION: Calculate the largest (MAX1') or smallest (MIN1') of N real numbers on top-of-stack and return the integer of that number in S–2N+1.

Declaration:
```
PROCEDURE MAX1'(N); or MIN1'(N);
  VALUE N; INTEGER N;
    OPTIONAL EXTERNAL;
```



ATTRIBUTES:

Parameter: An integer number ⩾ 2.

Result: An integer number.

FORTRAN: Intrinsic Function: MAX1 (*A,B,C,...*) or MIN1 (*A,B,C,...*).

Error: See "Comments."

COMMENTS:

1. If the argument (or number of parameters) is less than 2, no action occurs.

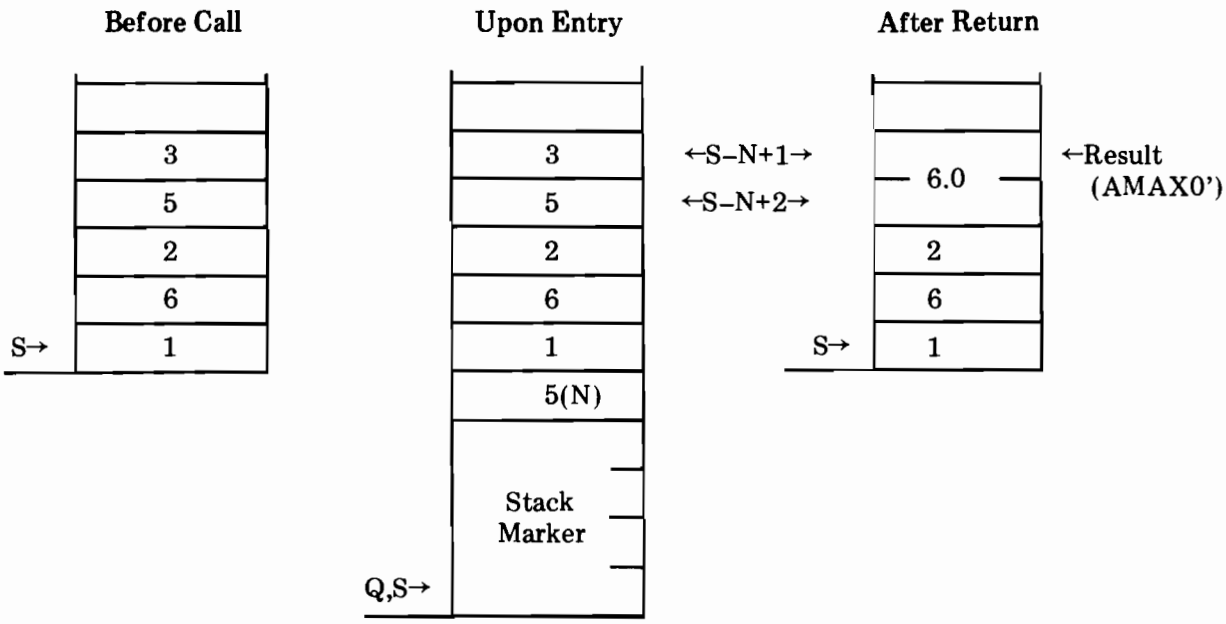
2. If the largest (or smallest) real number is outside the range [-32768.0, 32767.0], the arithmetic trap INT OFLW (integer overflow) occurs (if traps are enabled).

3. The SPL/3000 caller must cut the stack back after return (for example, use an ASSEMBLE (SUBS 9); statement).

AMAX0'/AMIN0'

FUNCTION: Calculate the largest (AMAX0') or smallest (AMIN0') of N integers on top-of-stack and return that integer in S-N+1 and S-N+2 in real representation.

Declaration:
```
PROCEDURE AMAX0'(N); or AMIN0'(N);
  VALUE N; INTEGER N;
    OPTION EXTERNAL;
```



ATTRIBUTES:

Parameter: An integer number $\geqslant 2$.

Result: A real number.

FORTRAN: Intrinsic Function: AMAX0 (*A,B,C,...*) or AMIN0 (*A,B,C,...*).

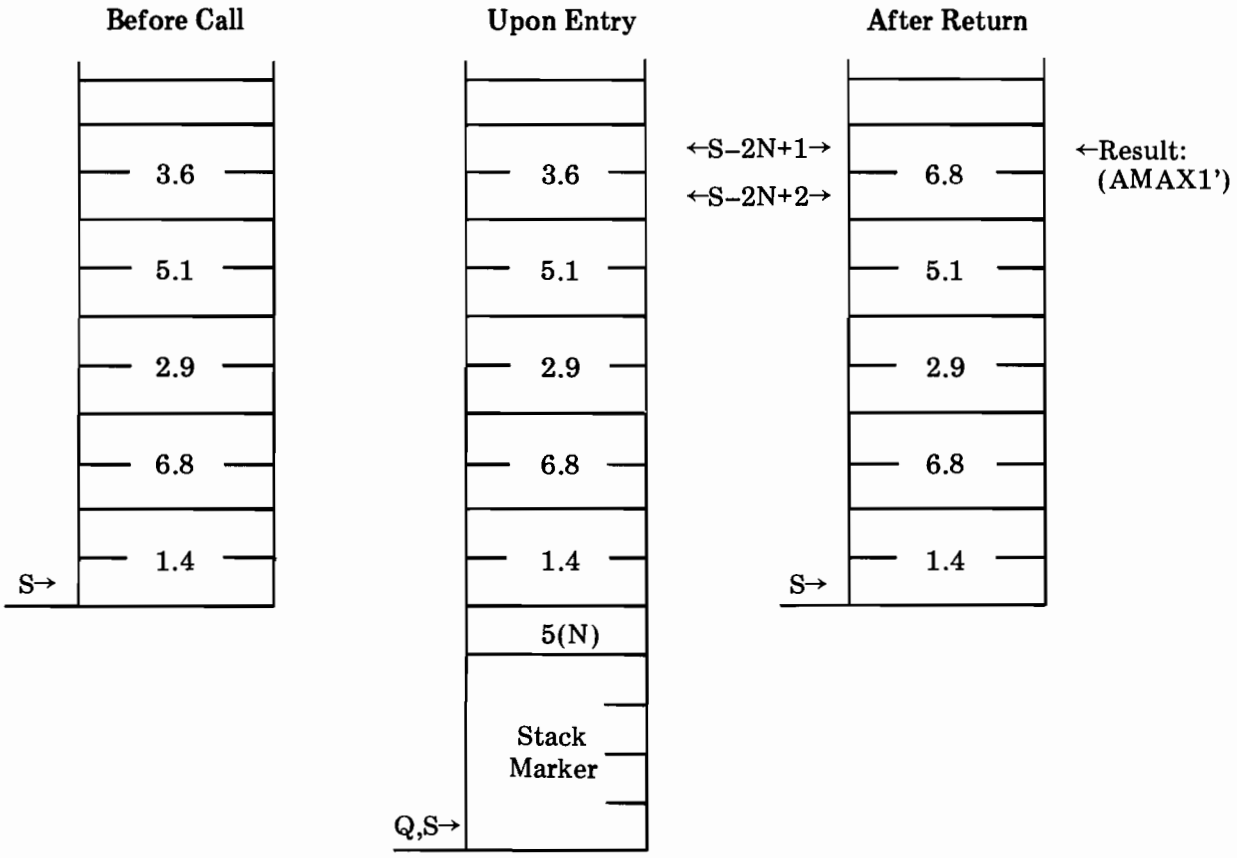
Error: If the argument (or number of parameters) is less than 2, no action occurs.

COMMENT: The SPL/3000 caller must cut the stack back after return (for example, use an ASSEMBLE (SUBS 3); statement).

AMAX1'/AMIN1'

FUNCTION: Calculate the largest (AMAX1') or smallest (AMIN1') of N real numbers on top-of-stack and return that result in S–2N+1 and S–2N+2.

Declaration
```
PROCEDURE AMAX1'(N); or AMIN1'(N);
  VALUE N; INTEGER N;
    OPTION EXTERNAL;
```



ATTRIBUTES:

Parameter: An integer number ≥ 2

Result: A real number.

FORTRAN: Intrinsic Function: AMAX1 (*A,B,C,. . .*) or AMIN1 (*A,B,C,. . .*).

Error: If the argument (or number of parameters) is less than 2, no action occurs.

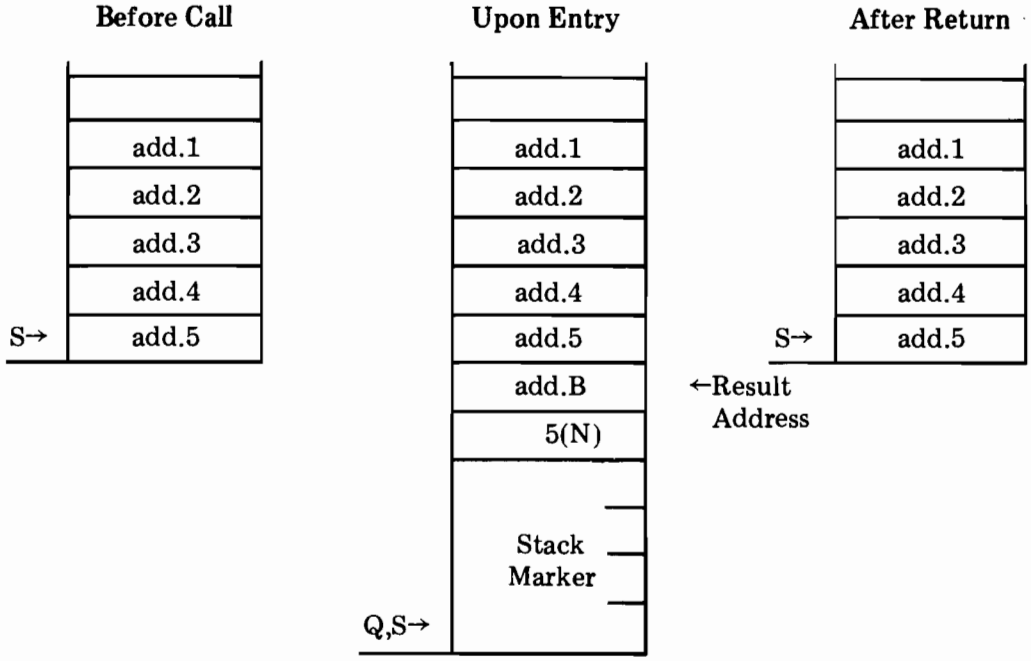
COMMENT: The SPL/3000 caller must cut back the stack after return (for example, use an ASSEMBLE (SUBS 8); statement).

DMAX1'/DMIN1'

FUNCTION: Calculate the largest (DMAX1') or smallest (DMIN1') of N double precision (LONG real) numbers addressed in the N words on top-of-stack and return that result in the address referenced by B.

Declaration:

```
PROCEDURE DMAX1' (B,N); or DMIN1' (B,N);
  VALUE N; LONG B; INTEGER N;
    OPTION EXTERNAL;
```



ATTRIBUTES:

Parameters: For N, an integer ≥ 2; for B, a double precision identifier.

Result: A double precision number.

FORTRAN: Intrinsic Function: DMAX1 (*A,B,C,...*) or DMIN1 (*A,B,C,...*).

Error: If the N argument (or number of parameters) is less than 2, no action occurs.

COMMENT: The SPL/3000 caller must cut back the stack after return (for example, use an ASSEMBLE (SUBS 5); statement.)

AMOD'

FUNCTION: Calculate a first real number modulus a second real number.

Declaration:
```
REAL PROCEDURE AMOD' (Y,Z);
  VALUE Y,Z; REAL Y,Z;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: X = Y – AINT(Y/Z)*Z

ATTRIBUTES:

Parameters: Both arguments are real numbers, the second must not be zero.

Result: A real number.

FORTRAN: Intrinsic Function: AMOD (*Y,Z*).

Error: None.

CAUTION: The arithmetic traps FLPT OFLW (floating point overflow), FLPT UFLW (floating point underflow), or FLPT DIVZ (floating point divide by zero) may occur (if traps are enabled).

DMOD

FUNCTION: Calculate a first double precision (LONG real) number modulus a second double precision (LONG real) number.

Declaration:

```
LONG PROCEDURE DMOD (Y,Z);
  LONG Y,Z;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: X = Y – DDINT(Y/Z)*Z

ATTRIBUTES:

Parameters: Both arguments are double precision numbers; the second must not be zero.

Result: A double precision number.

FORTRAN: Basic External Function: DMOD (*Y,Z*).

Error: None.

CAUTION: The arithmetic traps FLPT OFLW (floating point overflow), FLPT UFLW (floating point underflow), or FLPT DIVZ (floating point divide by zero) may occur (if traps are enabled).

EXP (or EXP')

FUNCTION: Calculate e^x, where x is a real number.

Declaration:
```
REAL PROCEDURE EXP (Y); or EXP' (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

when $|x - y| \sim \epsilon$, maximum $\left|\frac{f - g}{f}\right| \sim \epsilon$

ATTRIBUTES:

Parameter: A representable[1] real number in the range [–177.4455, 177.4455].

Result: A representable[1] positive real number.

FORTRAN: Basic External Function: EXP (*Y*).

Error: If the argument is $\geqslant 177.4456$, the result cannot be represented and SOFTERROR' message EXP: OVERFLOW occurs (see "Library Errors"). If the argument $\leqslant -177.4456$, the result is set to zero.

[1] See "Introduction."

DEXP (or DEXP')

FUNCTION: Calculate e^x, where x is a double precision (LONG real) number.

Declaration:
```
LONG PROCEDURE DEXP (Y); or DEXP' (Y);
  LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

$$\text{when } |x - y| \sim \epsilon, \text{ maximum } \left|\frac{f - g}{f}\right| \sim \epsilon$$

ATTRIBUTES:

Parameter: A representable[1] number in the range [– 177.4455, 177.4455].

Result: A representable[1] positive double precision number.

FORTRAN: Basic External Function: DEXP (*Y*).

Error: If the argument is $\geqslant 177.4456$, the result cannot be represented and the SOFTERROR' message DEXP: OVERFLOW occurs (see "Library Errors"). If the argument is $\leqslant -177.4456$, the result is set to zero.

[1] See "Introduction."

CEXP

FUNCTION: Calculate e^x, where x is a complex number.

Declaration: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part a and one for the imaginary part b. Thus, complex numbers occupy four words (see "Internal Representation," in the "Introduction").

```
PROCEDURE CEXP (Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
                .
                .
                .
```

An SPL/3000 caller must use the statement "TOS := 0D," twice, to set two double integers "0" on top-of-stack in four words; then use "CEXP (Y);" to call the procedure which overlays the result on those four words. (See sample below.)

Method: $e^{a+bi} = e^a (\cos(b) + i \sin(b))$
Y (0) = a (real part)
Y (1) = b (imaginary part)

Accuracy: Depends on accuracy of EXP, COS, and SIN.

ATTRIBUTES:

Parameter: Any complex number representable in two representable[1] real numbers, one for a and one for b; a must be in the range [– 177.4455, 177.4455].

Result: A complex number, stored in 4 words on TOS (for SPL/3000 caller).

FORTRAN: Basic External Function: CEXP (*Y*).

Error: See EXP.

Sample SPL/3000 calling sequence:

```
REAL ARRAY Y(0:1);
  TOS := 0D;
  TOS := 0D;
  CEXP(Y);
```

[1] See "Introduction."

SQRT (or SQRT')

FUNCTION: Calculate the square root of a real number.

Declaration:
```
REAL PROCEDURE SQRT (Y); or SQRT' (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: An appropriate starting point for two Newton iterations is reached through a minimax approximation.

Accuracy: (See "Introduction"):

$$\text{when} \left| \frac{x - y}{x} \right| \sim \epsilon, \text{ maximum} \left| \frac{f - g}{f} \right| \sim (1/2)\,\epsilon$$

ATTRIBUTES:

Parameter: A non-negative real number.

Result: A non-negative real number.

FORTRAN: Basic External Function: SQRT (*Y*).

Error: SOFTERROR' message SQRT: ARGUMENT NEGATIVE occurs if the argument is negative (see "Library Errors").

DSQRT

FUNCTION: Calculate the square root of a double precision (LONG real) number.

Declaration:
```
REAL PROCEDURE DSQRT (Y);
  REAL Y;
    OPTION EXTERNAL;
              .
              .
              .
```

Method: An appropriate starting point for three Newton iterations is reached through a minimax approximation.

Accuracy: (See "Introduction"):

$$\text{when} \left| \frac{x - y}{f} \right| \sim \epsilon, \text{ maximum} \left| \frac{f - g}{f} \right| \sim (1/2)\,\epsilon$$

ATTRIBUTES:

Parameter: A non-negative double precision number.

Result: A non-negative double precision number.

FORTRAN: Basic External Function: DSQRT (*Y*).

Error: SOFTERROR' message DSQRT: ARGUMENT NEGATIVE occurs if the argument is negative (see "Library Errors").

CSQRT

FUNCTION: Calculate the square root of a complex number.

Declaration: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

```
PROCEDURE CSQRT (Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
              .
              .
              .
```

An SPL/3000 caller must use the statement "TOS := 0D;" twice to set two double integers "0" on top-of-stack in four words; then use "CSQRT (Y);" to call the procedure which overlays the result on those four words. (See sample on the next page.)

Method: Either step 1 or step 2, then steps 3 and 4:

Y(0) = *a* (real part)

Y(1) = *b* (imaginary part)

CSQRT (a + bi) = x + yi

1. IF $|a| \geq |b|$ THEN $T1 = 1 + \sqrt{1 + (|b| / |a|)^2}$;

   IF $|a| < 2^{-252}$ THEN T1 = (T1) / 4 ELSE a = |a| / 4;

   $T1 = \sqrt{2} * \sqrt{a * T1}$;

2. ELSE $T1 = (|a| / |b|) + \sqrt{1 + (|a| / |b|)^2}$;

   IF $|b| < 2^{-252}$ THEN T1 = (T1) / 4 ELSE b = |b| / 4;

   $T1 = \sqrt{2} * \sqrt{b * T1}$;

3. T2 = b / (T1 * 2);

4. IF $a \geq 0$ THEN x = T1; y = T2 ELSE

   x = T2; y = T1;

Accuracy: Depends on accuracy of SQRT.

ATTRIBUTES:

Parameter:	Any complex number representable in two real numbers, one for *a* and one for *b*.
Result:	A complex number, as just defined, left in four words on TOS (for SPL/3000 caller).
FORTRAN:	Basic External Function: CSQRT (*Y*).
Error:	See "SQRT."

Sample SPL/3000 calling sequence:

```
REAL ARRAY Y(0:1);
  TOS := 0D;
  TOS := 0D;
  CSQRT (Y);
```

ALOG (or ALOG')/ALOG10

FUNCTION: Calculate the natural (ALOG or ALOG') or the base 10 (ALOG10) logarithm of a positive real number.

Declaration:

```
REAL PROCEDURE ALOG (Y); [or ALOG' (Y);] or ALOG10 (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Accuracy: (See "Introduction"):

$$\text{when } \left|\frac{x-y}{f}\right| \sim \epsilon, \text{ maximum } \left|\frac{f-g}{f}\right| \sim \epsilon \,/\, |\ln(x)|$$

Method: A minimax approximation.

ATTRIBUTES:

Parameter: A positive real number.

Result: A real number (ALOG10 = ALOG*$\log_{10}(e)$).

FORTRAN: Basic External Function: ALOG (*Y*) or ALOG10 (*Y*).

Error: If the argument is negative or zero, SOFTERROR' message ALOG: ARGUMENT NOT POSITIVE occurs for either ALOG or ALOG10 (see "Library Errors").

DLOG (or DLOG')/DLOG10

FUNCTION: Calculate the natural (DLOG or DLOG') or the base 10 (DLOG10) logarithm of a positive double precision (LONG real) number.

Declaration:

```
LONG PROCEDURE DLOG (Y);[or DLOG' (Y);] or DLOG10 (Y);
  LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Accuracy: (See "Introduction"):

$$\text{when } \left|\frac{x-y}{x}\right| \sim \epsilon, \text{ maximum } \left|\frac{f-g}{f}\right| \sim \epsilon \,/\, |\ln(x)|$$

Method: A minimax approximation.

ATTRIBUTES:

Parameter: A representable[1] positive double precision number.

Result: A double precision number (DLOG10 = DLOG*$\log_{10}(e)$).

FORTRAN: Basic External Function: DLOG (*Y*) or DLOG10 (*Y*).

Error: If the argument is negative or zero, SOFTERROR' message DLOG: ARGUMENT NOT POSITIVE occurs for either DLOG or DLOG10 (see "Library Errors").

[1] See "Introduction."

CLOG

FUNCTION: Calculate the natural logarithm of a complex number.

Declaration: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part a and one for the imaginary part b. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

```
PROCEDURE CLOG (Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
                .
                .
                .
```

An SPL/3000 caller must use the statement "TOS := 0D;" twice to set two double integers "0" on top-of-stack in four words; then use "CLOG (Y);" to call the procedure which overlays the result on those four words. (See sample below.)

Method: CLOG $(a + bi) = x + yi$
Y(0) = a (real part)
Y(1) = b (imaginary part)

where

x = ALOG (CABS $(a + bi)$)
y = ATAN2 (b, a)

Accuracy: For a, depends on accuracy of ALOG and SQRT; accuracy for b depends on accuracy of ATAN2.

ATTRIBUTES:

Parameter: Any non-zero complex number representable in two real numbers, one for a and one for b; both parts must not be zero.

Result: A complex number, left in four words on TOS (for SPL caller).

FORTRAN: Basic External Function: CLOG (Y).

Errors: If a and b are zero, SOFTERROR' message ALOG: ARGUMENT NOT POSITIVE occurs. If $\frac{\min(a, b)}{\max(a, b)}$ underflows, SOFTERROR' message ATAN2: UNDERFLOW occurs (see "Library Errors").

Sample SPL/3000 calling sequence:

```
REAL ARRAY Y(0:1);
  TOS := 0D;
  TOS := 0D;
  CLOG (Y);
```

TAN

FUNCTION: Calculate the tangent of a real number in radians.

Declaration:
```
REAL PROCEDURE TAN (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Accuracy: (See "Introduction"):

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \sec^2 x$

Method: A minimax approximation.

ATTRIBUTES:

Parameter: A real number in radians.

Result: A real number.

FORTRAN: Basic External Function: TAN (*Y*)

Error: Let

$$M = \frac{(2k + 1)\pi}{2}$$

where k is any non-negative integer. Then, if

$$\left||\text{argument}| - M\right| < 2^{-23} * M$$

SOFTERROR' message TAN: OVERFLOW occurs (see "Library Errors").

SIN (or SIN')

FUNCTION: Calculate the sine of a real number in radians

Declaration:
```
REAL PROCEDURE SIN (Y); or SIN' (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

When $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \cos x$

ATTRIBUTES:

Parameter: A real number in radians.

Result: A representable[1] real number in the range [–1.0, 1.0].

FORTRAN: Basic External Function: SIN (*Y*).

Error: None.

[1] See "Introduction."

COS (or COS')

FUNCTION: Calculate the cosine of a real number in radians.

Declaration:
```
REAL PROCEDURE COS (Y); or COS' (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

When $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \sin x$

ATTRIBUTES:

Parameter: A real number in radians.

Result: A representable[1] real number in the range [–1.0, 1.0].

FORTRAN: Basic External Function: COS (*Y*).

Error: None.

[1] See "Introduction."

DTAN

FUNCTION: Calculate the tangent of a double precision (LONG real) number in radians.

Declaration:

```
LONG PROCEDURE DTAN (Y);
  LONG Y;
    OPTION EXTERNAL;
              .
              .
              .
```

Accuracy: (See "Introduction"):

When $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \sec^2 x$

Method: A minimax approximation.

ATTRIBUTES:

Parameter: A double precision number in radians.

Result: A double precision number.

FORTRAN: Basic External Function: TAN (*Y*).

Error: Let

$$M = \frac{(2k + 1)\pi}{2}$$

where k is any non-negative integer. Then, if

$$\Big||\text{argument}| - M\Big| < 2^{-39} * M$$

SOFTERROR' message DTAN: OVERFLOW occurs (see "Library Errors").

DSIN

FUNCTION: Calculate the sine of a double precision (LONG real) number in radians.

Declaration:
```
LONG PROCEDURE DSIN (Y);
  LONG Y;
    OPTION EXTERNAL;
              .
              .
              .
```

Method: A minimax approximation.

Accuracy: (See "Introduction")

When $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \cos x$

ATTRIBUTES:

Parameter: A double precision number in radians.

Result: A representable[1] double precision number in the range [–1.0,1.0].

FORTRAN: Basic External Function: SIN (*Y*).

Error: None.

[1] See "Introduction."

DCOS

FUNCTION: Calculate the cosine of a double precision (LONG real) number in radians.

Declaration:

```
LONG PROCEDURE DCOS (Y);
  LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

When $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \sin x$

ATTRIBUTES:

Parameter: A double precision number in radians.

Result: A representable[1] double precision number in the range [-1.0,1.0].

FORTRAN: Basic External Function: DCOS (*Y*).

Error: None.

[1] See "Introduction."

FUNCTION: Calculate the sine of a complex number.

Declaration: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part a and one for the imaginary part b. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

```
PROCEDURE CSIN (Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
              .
              .
              .
```

An SPL/3000 caller must use the statement "TOS := 0D;" twice to set two double integers "0" on top-of-stack in four words; then use "CSIN (Y);" to call the procedure which overlays the result on those four words. (See sample below.)

Method: CSIN $(a + bi)$ = sin (a) cosh (b) + i cos (a) sinh (b)
Y(0) = a (real part)
Y(1) = b (imaginary part)

where

If $b < 0.5$, sinh (b) is determined by a minimax approximation.

If $b \geqslant 0.5$, sinh $(b) = (e^{b} - e^{-b}) / 2$;

$$\cosh(b) = \sinh|b| + \frac{1}{e^{|b|}}$$

Accuracy: Depends on accuracy of SIN, COS, and EXP.

ATTRIBUTES:

Parameter: A complex number.

Result: A complex number, left in four words on TOS (for SPL/3000 caller).

FORTRAN: Basic External Function: CSIN (Y).

Error: See EXP.

Sample SPL/3000 calling sequence:

```
REAL ARRAY Y(0:1);
  TOS := 0D;
  TOS := 0D;
  CSIN (Y);
```

CCOS

FUNCTION: Calculate the cosine of a complex number.

Declaration: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part a and one for the imaginary part b. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

```
PROCEDURE CCOS (Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
                .
                .
                .
```

An SPL/3000 caller must use the statement "TOS := 0D;" twice to set two double integers "0" on top-of-stack in four words; then use "CCOS (Y);" to call the procedure which overlays the result on those four words. (See sample below.)

Method: CCOS $(a + bi)$ = cos (a) cosh $(b) - i$ sin (a) sinh (b)
Y(0) = a (real part)
Y(1) = b (imaginary part)

where

If $b < 0.5$, sinh (b) is determined by a minimax approximation.

If $b \geqslant 0.5$, sinh $(b) = (e^{b} - e^{-b}) / 2$;

$$\cosh(b) = \sinh|b| + \frac{1}{e^{|b|}}$$

Accuracy: Depends on accuracy of SIN, COS, and EXP.

ATTRIBUTES:

Parameter: A complex number.

Result: A complex number, left in four words on TOS (for SPL/3000 caller).

FORTRAN: Basic External Function: CCOS (Y).

Error: See EXP.

Sample SPL/3000 calling sequence:

```
REAL ARRAY Y(0:1);
  TOS := 0D;
  TOS := 0D;
  CCOS(Y);
```

TANH

FUNCTION: Calculate the hyperbolic tangent of a real number.

Declaration:

```
REAL PROCEDURE TANH (Y);
  REAL Y;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: $\text{TANH}(Y) = \frac{e^{Y} - e^{-Y}}{e^{Y} + e^{-Y}}$ unless $|Y| < 0.4812118$. In that case, a minimax approximation is used.

Accuracy: (See "Introduction")

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \operatorname{sech}^2 x$

ATTRIBUTES:

Parameter: A real number.

Result: A representable[1] real number in the range [0.0, 1.0].

FORTRAN: Basic External Function: TANH (*Y*).

Error: None.

[1] See "Introduction."

SINH

FUNCTION: Calculate the hyperbolic sine of a real number.

Declaration:

```
REAL PROCEDURE SINH (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: SINH (Y) = $\frac{e^{Y} - e^{-Y}}{2}$ unless $Y < 0.5$. In that case, a minimax approximation is used.

Accuracy: (See "Introduction")

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \cosh x$

ATTRIBUTES:

Parameter: A real number.

Result: A real number.

FORTRAN: Basic External Function: SINH (*Y*).

Error: See EXP.

COSH

FUNCTION: Calculate the hyperbolic cosine of a real number.

Declaration:

```
REAL PROCEDURE COSH (Y);
  REAL Y;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: $\text{COSH}(Y) = \frac{e^{Y} + e^{-Y}}{2}$

Accuracy: (See "Introduction"):

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \epsilon \sinh x$

ATTRIBUTES:

Parameter: A real number.

Result: A real number.

FORTRAN: Basic External Function: COSH (*Y*).

Error: See EXP.

ATAN (or ATAN')

FUNCTION: Calculate the arctangent of a real number.

Declaration:
```
REAL PROCEDURE ATAN (Y); or ATAN' (Y);
  REAL Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction")

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \frac{\epsilon}{1 + x^2}$

ATTRIBUTES:

Parameter: A real number.

Result: A representable[1] real number in the range $[-\pi / 2, \pi / 2]$.

FORTRAN: Basic External Function: ATAN (*Y*).

Error: None.

[1] See "Introduction."

DATAN (or DATAN')

FUNCTION: Calculate the arctangent of a double precision (LONG real) number.

Declaration:
```
LONG PROCEDURE DATAN (Y); or DATAN' (Y);
  LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \frac{\epsilon}{1 + x^2}$

ATTRIBUTES:

Parameter: A double precision number.

Result: A representable[1] double precision number in the range $[-\pi / 2, \pi / 2]$.

FORTRAN: Basic External Function: DATAN (*Y*).

Error: None.

[1] See "Introduction."

ATAN2 (or ATAN2')

FUNCTION: Calculate the arctangent of the quotient of two real numbers.

Declaration:
```
REAL PROCEDURE ATAN2 (Y,Z); or ATAN2' (Y,Z);
  REAL Y,Z;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: Calls ATAN (|min (Y,Z) / max (Y,Z)|), then determines the proper quandrant.

Accuracy: (See "Introduction"):

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \frac{3\epsilon}{1 + w^2}$

where

$w = \min(Y,Z) / \max(Y,Z)$

ATTRIBUTES:

Parameters: Real numbers. Both must not be zero.

Result: A representable[1] real number in one of the following ranges:

	$Z \geqslant 0$	$Z < 0$
$Y \geqslant 0$	$[0, \pi / 2]$	$[-\pi / 2, 0)$
$Y < 0$	$(\pi / 2, \pi]$	$(-\pi, -\pi / 2)$

FORTRAN: Basic External Function: ATAN2 (*Y,Z*).

Error: SOFTERROR' message ATAN2: ARGUMENTS ZERO occurs if both arguments are zero; SOFTERROR' message ATAN2: UNDERFLOW occurs if

|smaller argument| / |larger argument|

causes underflow (see "Library Errors").

[1] See "Introduction."

DATAN2

FUNCTION: Calculate the arctangent of the quotient of two double precision (LONG real) numbers.

Declaration:

```
LONG PROCEDURE DATAN2 (Y,Z);
  LONG Y,Z;
    OPTION EXTERNAL;
              .
              .
              .
```

Method: Calls DATAN (|min (Y,Z) / max (Y,Z)|), then determines the proper quadrant.

Accuracy: (See "Introduction")

when $|x - y| \sim \epsilon$, maximum $|f - g| \sim \frac{3\epsilon}{1 + w^2}$

where

$w = \min(Y,Z) / \max(Y,Z)$

ATTRIBUTES:

Parameters: Double precision numbers. Both must not be zero.

Result: A representable[1] double precision number in one of the following ranges:

	$Z \geqslant 0$	$Z < 0$
$Y \geqslant 0$	$[0, \pi / 2]$	$[-\pi / 2, 0)$
$Y < 0$	$(\pi / 2, \pi]$	$(-\pi, -\pi / 2)$

FORTRAN: Basic External Function: DATAN2 (*Y,Z*)

Error: SOFTERROR' message DATAN2: ARGUMENTS ZERO occurs if both arguments are zero; SOFTERROR' message DATAN2: UNDERFLOW occurs if

|smaller argument| / |larger argument|

causes underflow (see "Library Errors").

[1] See "Introduction."

INVERT

FUNCTION: Invert a square matrix containing real numbers stored by rows; the resulting inverse is stored over the input matrix. (Required by BASIC/3000.)

Declaration:

```
PROCEDURE INVERT (N, A, SFLG);
  VALUE N; INTEGER N, SFLG; REAL ARRAY A;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: For N, an integer for the order of the matrix; for A, a real identifier of the matrix; for SFLG, an integer identifier.

Results: Inverse replaces original matrix, and SFLG is 1 if the matrix is nonsingular or 0 if the matrix is singular.

FORTRAN: Callable as an external subroutine.

Error: None.

CAUTION: If the matrix is singular (SFLG is 0), all or part of the matrix is overlaid with undefined results.

DINVERT

FUNCTION: Invert a square matrix containing double precision (LONG real) numbers, stored by rows; the resulting inverse is stored over the input matrix. (Required by BASIC/3000.)

Declaration:

```
PROCEDURE DINVERT (N, A, SFLG);
  VALUE N; INTEGER N, SFLG; LONG ARRAY A;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: For N, an integer for the order of the matrix; for A, a double precision identifier of the matrix; for SFLG, an integer identifier.

Results: Inverse replaces original matrix, and SFLG is 1 if the matrix is nonsingular or 0 if the matrix is singular.

FORTRAN: Callable as an external subroutine.

Error: None.

CAUTION: If the matrix is singular (SFLG is 0), all or part of the matrix is overlaid with undefined results.

CINVERT

FUNCTION: Invert a square matrix containing complex elements (pairs of real elements) stored real part *a* then imaginary part *b*, by rows; the resulting inverse is stored over the input matrix. (Required by BASIC/3000.)

Declaration:

```
PROCEDURE CINVERT (N, A, SFLG);
  VALUE N; INTEGER N, SFLG; REAL ARRAY A;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: For N, an integer for the order of the matrix; for A, a real identifier of the matrix; for SFLG, an integer identifier.

Results: Inverse replaces original matrix, and SFLG is 1 if the matrix is nonsingular or 0 if the matrix is singular.

FORTRAN: Callable as an external subroutine.

Error: None.

CAUTION: If the matrix is singular (SFLG is 0), all or part of the matrix is overlaid with undefined results.

RAND1

FUNCTION: Generate a pseudo-random number, which may be used as the starting point for RAND. (Required by BASIC/3000.)

Declaration:

```
REAL PROCEDURE RAND1;
  OPTION EXTERNAL;
        .
        .
        .

        or

DOUBLE PROCEDURE RAND1;
  OPTION EXTERNAL;
        .
        .
        .
```

ATTRIBUTES:

Parameter: None.

Result: A 32-bit quantity, which can be identified as either a real number or a double integer number.

FORTRAN: Not callable.

Error: None.

RAND

FUNCTION: Generate the next element of a sequence of pseudo-random numbers (see Comment 1). (Required by BASIC/3000.)

Declaration:

```
REAL PROCEDURE RAND (X);
  REAL X;
    OPTION EXTERNAL;
                .
                .
                .
              or
DOUBLE PROCEDURE RAND (X);
  DOUBLE X;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: Either a real number or a double integer number, which initially can be obtained for RAND1.

Results: A representable[1] real number in the range (0.0, 1.0) returned as the value of the routine, and a 32-bit quantity replacing the original value of the parameter (see Comment 1).

FORTRAN: Callable as an external subroutine (see Comment 2).

Error: None.

COMMENTS:

1. The parameter value at the initial call to RAND completely determines a sequence of pseudo-random real numbers. Each time RAND returns a new value to the calling program it also sets a new 32-bit value in place of the parameter. To continue the pseudo-random sequence thus initiated, that 32-bit value must be used as the parameter in the next call to RAND.

2. The FORTRAN caller must provide his own initial value.

[1] See "Introduction."

SECTION III
UTILITY PROCEDURES

To find the descriptions for any given procedure in this section, see the Function Directory.

EXTIN'

FUNCTION: Convert a byte array containing an input string of ASCII digits (see "Comments") into one of four internal representations:

1. Integer.
2. Real
3. Double integer
4. LONG real

Declaration:

```
PROCEDURE EXTIN' (STRING,W,D,TYPE,SCALE,BLANKS,
  RESULT,ERROR);
    VALUE D,TYPE,SCALE,BLANKS,RESULT;
      BYTE ARRAY STRING;
      INTEGER W,D,TYPE,SCALE,ERROR;
      INTEGER POINTER RESULT;
      LOGICAL BLANKS;
        OPTION EXTERNAL;
            .
            .
            .
```


ATTRIBUTES:

Parameters (Input):

STRING = Pointer to the first byte of the byte array to be converted.

W = Upon entry, the field width w of the ASCII input string including all special characters (see Comment 1).

D = The number of digits d to be interpreted as fraction digits (multiply the integer field by 10^{-d} *if* the input string does *not* include a decimal point (see Comment 1). If a decimal point is included in the input string, this parameter has no effect. If D is given as < 0. the procedure assumes D is 0. If TYPE is 0 or –1, this parameter is ignored.

TYPE = The internal representation desired:

0 = integer

1 = real

–1 = double integer

–2 = LONG real

SCALE = The scale factor (see "Comments"). Ignored if TYPE = 0 (integer) or –1 (double integer).

BLANKS = Treatment of imbedded blanks, a dollar sign,[1] and commas[1] in the input:

[1] See "M$w.d$" and "N$w.d$" in Section I.

False: $ and/or commas and/or imbedded blanks are delimiters.

True: Imbedded blanks are treated as zeros; a $ and/or a comma to the left of every third digit to the left of the decimal point are allowed.

Parameters (Output):

RESULT = Pointer to the first word of result storage (in 1, 2, or 3 words) according to the TYPE specified.

ERROR = Error indicator (see "Comments"):

- 0 = Valid result, no error
- 1 = An illegal character was detected (see Comment 5)
- 2 = No integer or fraction value was detected (see Comment 6)
- −2 = Resulting number > largest representable value of TYPE (see "Introduction")
- −4 = Resulting number < smallest representable value of TYPE (see "Introduction")
- −1 = Number > largest representable *and* illegal character
- −3 = Number < smallest representable *and* illegal character

W = Upon exit, the number of string characters (see Comment 5) used to compute the result.

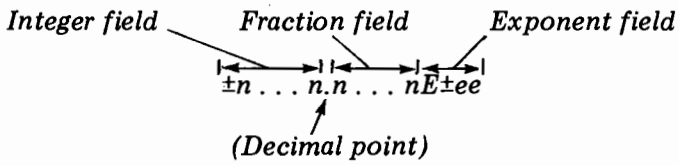
Results: See "Parameters (Output)."

FORTRAN: Not callable.

Errors: See "Parameters (Output)."

COMMENTS:

1. The external form of the input is a string of ASCII digits which can include integer, fraction, and exponent subfields:



NOTES: 1. *A $ and comma(s) (for monetary or numeration form) in the input are ignored, but must be provided for in parameter W.*

2. *The exponent field input can be any of several forms:*

+e	*+ee*	E*e*	E*ee*	D*e*	D*ee*
–e	*–ee*	E+*e*	E+*ee*	D+*e*	D+*ee*
		E–*e*	E–*ee*	D–*e*	D–*ee*

where e is an exponent value digit.

2. SCALE has no effect if the input string includes an exponent field. Otherwise, a SCALE of *n* sets the result to the input string value * 10^{-n}

EXAMPLES:

STRING *Array*	SCALE	RESULT
4398.76	3	4.39876
543.21	–3	543210.

3. The type of the result is independent of the input string format. For example, the input 4398.76 can be converted to integer form. The conversion rules are as follows:

   Integer (TYPE = 0) truncates a fractional input.

   Real (TYPE = 1) rounds a fractional input.

   Double integer (TYPE = –1) truncates a fractional input.

   LONG real (TYPE = –2) rounds a fractional input.

4. Leading blanks in the input string are ignored; if BLANK is true, trailing blanks are treated as 0s.

5. If ERROR is set to an odd value, an illegal character was input; if ERROR is odd and negative, an illegal character and illegal value was detected. The RESULT is computed from the input string characters that preceded the delimiting digit or illegal character. Parameter W can be used as an index into STRING to locate that delimiter or illegal character. Here are two examples of illegal character inputs:

   +1.345A (A is illegal).

   7543CUP (C, U, and P are illegal)

6. If ERROR is set to 2, no integer or fraction value was detected. Thus, no result can be computed. Here are two examples of non-value inputs:

   +.E5 (the exponent E5 has no base)

   –.A (no base, no exponent)

INEXT'

FUNCTION: Convert a number in storage (in one of four internal representations) to a byte array for an output string of ASCII digits (see "Comments"). The four internal representations are as follows:

1. Integer
2. Real
3. Double integer
4. LONG real

Declaration:

```
PROCEDURE INEXT' (N,TYPE,W,D,KIND,SCALE,STRING,TROUBLE);
  VALUE N,TYPE,W,D,KIND,SCALE;
    INTEGER POINTER N;
    INTEGER TYPE,W,D,KIND,SCALE;
    BYTE ARRAY STRING;
    LOGICAL TROUBLE;
      OPTION EXTERNAL;
            .
            .
            .
```

ATTRIBUTES:

Parameters (Input):

N = Pointer to the first word (in 1, 2, or 3 words) of the internal representation to be converted.

TYPE = The type of internal representation:

0 = integer

1 = real

–1 = double integer

–2 = LONG real

W = Field width *w* of the ASCII string, including all special characters (see KIND).

NOTE: Set W to at least D + 6 to allow for special characters in KIND D, E, or G. Set W to at least D + 7 if a positive SCALE factor is used too.

D = The number of fractional digits *d* in the ASCII string. If D is given as 0, no fractional digits are included in the output, even though a decimal point is included. If $W \leq 0$ or $D < 0$, an error is implied and TROUBLE is set TRUE.

KIND = The kind of conversion desired:

3 for the G*w.d* format:	(see "Comments")
2 for the D*w.d* format:	.12345D+04
1 for the E*w.d* format:	.12345E+04
0 for the I*w* format:	1234
–1 for the N*w.d* format:	1,234.5
–2 for the M*w.d* format:	$1,234.5
–3 for the F*w.d* format:	1234.5

SCALE = The scale factor (see "Comments").

Parameters (Output):

STRING = Pointer to the first byte array for the ASCII string output. The result occupies the first W characters (bytes) in this array.

TROUBLE = TRUE if the field width W is too small for the result in the specified KIND, if $D < 0$, or if $W \leq 0$; the byte array is filled with #'s. FALSE if result is valid.

Results: See "Parameters (Output)."

FORTRAN: Not callable.

Error: See "Parameters (Output)."

COMMENTS:

1. The result STRING is an array of ASCII digits; STRING can also include the sign character –, a decimal point, and an exponent field, for KIND = 1 or 2 (E*w.d* or D*w.d* formats). The exponent field always includes the letter E or D followed by a signed two-digit integer. Or, STRING can include a sign character –, a dollar sign (for KIND = –2, M*w.d* format) and/or commas for KIND = –2 or –3 (M*w.d* or N*w.d* formats).

NOTE: w = the parameter W and d = the parameter D.

2. To use KIND = 3 (G*w.d* format), set D to the number of significant digits and set W to D + 6 to allow for special characters. Then KIND = 3 is used as KIND = –3 or 1 (F*w.d* or E*w.d* format), according to the absolute value of the internal representation value N:

IF		$N < 0.1$	THEN E*w.d*;
IF	0.1	$\leqslant N < 1$	THEN F(*w*-4) . *d* plus 4X (spaces);
IF	1	$\leqslant N < 10^1$	THEN F(*w*-4) . (*d*–1) plus 4X;
IF	10^1	$\leqslant N < 10^2$	THEN F(*w*-4) . (*d*–2) plus 4X;
IF	10^2	$\leqslant N < 10^3$	THEN F(*w*-4) . (*d*–3) plus 4X;
.	.	.	.
.	.	.	.
.	.	.	.
IF	$10^{(d-1)}$	$\leqslant N < 10^d$	THEN F(*w*-4) . 0 plus 4X;
IF	10^d	$\leqslant N$	THEN E*w.d*;

In general, if the number of integer digits in N is > D or = 0, KIND = 1 (the E*w.d* format) is used.

EXAMPLES:

G12.6,N = 1234.5:F(*w*–4) . (*d*–4) = F8.2,4X:Δ1234.50ΔΔΔΔ

G13.7,N = 123456.7:F(*w*–4) . (*d*–4) = F9.1,4X:Δ123456.7ΔΔΔΔ

G9.2,N = 123.4:E*w.d* = E9.2:ΔΔ.12E+03

3. SCALE does not affect KIND = 0:

When KIND = 2 or 1,

the result STRING uses these factors:

a. The internal representation N fraction is multiplied by 10^S (where *s* is SCALE).

b. The internal representation N exponent is reduced by SCALE.

c. When SCALE is $\leqslant 0$, the STRING fraction has –SCALE leading 0s followed by D + SCALE significant digits.

d. When SCALE is > 0, STRING has SCALE significant digits left of the decimal point and (D – SCALE) + 1 significant digits right of the decimal point.

e. The least significant digit in STRING is rounded.

EXAMPLES:

For each, N = 1234.5, KIND = 1, W = 11, D = 3

SCALE = 0, STRING = ΔΔΔ.123E+04

SCALE = –2, STRING = ΔΔΔ.001E+06

SCALE = 2, STRING = ΔΔ12.35E+02

When KIND = –3 or –2 or –1,

the result STRING is the internal representation N multiplied by 10^S (where s is SCALE) then converted.

EXAMPLES:

For each, N = 1234.5, KIND = –3, W = 11, D = 3

SCALE = 0, STRING = ΔΔΔ1234.500

SCALE = –2, STRING = ΔΔΔΔΔ12.345

SCALE = 2, STRING = Δ123450.000

When KIND = 3 (see Comment 2),

if KIND = 3 (G*w.d*) is used as KIND = –3 (F*w.d*), SCALE has no effect.

If KIND = 3 is used as KIND = 1 (E*w.d*), SCALE affects STRING as described above for KIND = 2 or 1.

FUNCTION: Raise an integer number base to an integer number power.

Declaration:
```
PROCEDURE ITOI'(B,P);
  VALUE B,P;INTEGER B,P;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: P is factored into powers of 2; then the result is obtained by successive multiplications.

EXAMPLE:

$$B^7 = B^1 B^2 B^4 = B^1 B^2 (B^2)^2$$

ATTRIBUTES:

Parameters: B = any integer number $\neq 0$ *(the base).*

P = any integer number > 0 *(the power).*

Result: An integer number.

FORTRAN: Not callable.

Error: If B = 0 or P $\leqslant$ 0, SOFTERROR' message ITOI: ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result exceeds the range of integer numbers [–32768, 32767], the arithmetic traps INT OFLW (integer overflow) or INT UFLW (integer underflow) may occur (if traps are enabled).

RTOI'

FUNCTION: Raise a real number base to an integer number power.

Declaration:
```
PROCEDURE RTOI' (B,P);
  VALUE B,P;REAL B;INTEGER P;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: P is factored into powers of 2; then the result is obtained by successive multiplications.

EXAMPLE:

$$B^7 = B^1 B^2 B^4 = B^1 B^2 (B^2)^2$$

ATTRIBUTES:

Parameters: B = any real number $\neq 0.0$ *(the base).*

P = any integer number > 0 *(the power).*

Result: A real number.

FORTRAN: Not callable.

Error: If B = 0.0 or if $P \leqslant 0$, SOFTERROR' message RTOI': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result is outside the range of real numbers (see "Introduction"), the arithmetic traps FLPT OFLW (floating point overflow) or FLPT UFLW (floating point underflow) may occur (if traps are enabled).

RTOR'

FUNCTION: Raise a real number base to a real number power.

Declaration:
```
PROCEDURE RTOR'(B,P);
  VALUE B,P;REAL B,P;
    OPTION EXTERNAL;
          .
          .
          .
```

Method: One of three methods is used:

1. If B = 0.0 and P > 0.0, the result is set to 0.0.
2. If B > 0.0 and P = 0.0, the result is set to 1.0.
3. If B > 0.0 and P ≠ 0.0, result = EXP(P*ALOG(B)).

Accuracy: See EXP and ALOG.

ATTRIBUTES:

Parameters: B = 0.0, P > 0.0, or

B > 0.0, P = any real number *(B is the base, P is the power).*

Result: A non-negative real number.

FORTRAN: Not callable.

Error: If B = 0.0 and P ⩽ 0.0 or if B < 0.0 SOFTERROR' message RTOR': ILLEGAL ARGUMENTS occurs (see "Library Errors"). Or, see EXP and ALOG.

FUNCTION: Raise a real number base to a LONG real number power and return the result as a LONG real number.

Declarations: LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representation" in the "Introduction"). A real number B is raised to a LONG real power P by one of two procedures called by compiler-generated code. Each of the procedures is declared in the following format, to allow P to be call-by-reference or call-by-value:

```
PROCEDURE RTOLf';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f = V or R, for the second parameter (P):

V = call-by-value; use SPL/3000 statement "TOS := B;" to set B value on top-of-stack in two words, use "TOS := P;" to set P value on top-of-stack in three more words, overlay the result on the first three words, then delete the remaining two words from the stack.

R = call-by-reference; use SPL/3000 statement "TOS := B;" to set B value on top-of-stack in two words, use "TOS := @P;" to set P reference address on top-of-stack in one more word, then overlay the result on those three words.

Method: One of three methods is used:

1. If B = 0.0 and P > 0.0, the result is set to 0.0.
2. If B > 0.0 and P = 0.0, the result is set to 1.0.
3. If B > 0.0 and P ≠ 0.0, result = DEXP(P*DLOG(LONG(B))).

Accuracy: See EXP and ALOG.

ATTRIBUTES:

Parameters: B = 0.0, P > 0.0, or

B > 0.0, P = any LONG real number *(B is the base, P is the power).*

Result: A non-negative LONG real number.

FORTRAN: Not callable.

Errors: If B = 0.0 and P ≤ 0.0, or if B < 0.0, SOFTERROR' message RTOL': ILLEGAL ARGUMENTS occurs (see "Library Errors"). Or, see EXP and ALOG.

FUNCTION: Raise a LONG real number base to an integer number power.

Declarations: LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representation" in the "Introduction"). A LONG real base B is raised to an integer power P by one of two procedures called by compiler-generated code. Each of the procedures is declared in the following format to allow P to be call-by-reference or call-by-value:

```
PROCEDURE LTOIf';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f = V or R for the first parameter B:

V = call-by-value; use SPL/3000 statement "TOS := B;" to set B value on top-of-stack in three words, use "TOS := P;" to set P value on top-of-stack in one more word, overlay the result on the first three words, then delete the remaining word from the stack.

R = call-by-reference; use SPL/3000 statement "TOS := @B;" to set B reference address on top-of-stack in one word, use "TOS := P;" to set P value on top-of-stack in one more word, use "TOS := 0;" to set the integer "0" on top-of-stack in one more word, then overlay the result on those three words.

Method: P is factored into powers of 2, then the result is obtained by successive multiplications.

EXAMPLE:

$$B^7 = B^1 B^2 B^4 = B^1 B^2 (B^2)^2$$

ATTRIBUTES:

Parameters: B = any LONG real number $\neq 0.0$ *(the base).*

P = any integer number > 0 *(the power).*

Result: A LONG real number.

FORTRAN: Not callable.

Errors: If B = 0.0 or if P ≤ 0, SOFTERROR' message LTOI': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If result is outside the range of LONG real numbers (see "Introduction"), the arithmetic traps FLPT OFLW (floating point overflow) or FLPT UFLW (floating point underflow) may occur (if traps are enabled).

FUNCTION: Raise a LONG real number base to a LONG real number power.

Declarations: LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representation" in the "Introduction"). A LONG real base B is raised to a LONG real power P by one of four procedures called by compiler-generated code. Each of the procedures is declared in the following format to allow any combination of call-by-reference or call-by-value parameters:

```
PROCEDURE LTOLf1f2';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f_1 = V or R for the first parameter B:

V = call-by-value; use SPL/3000 statement "TOS := B;" to set B value on top-of-stack in three words.

R = call-by-reference; use SPL/3000 statement "TOS := @B;" to set B reference address on top-of-stack in one word.

f_2 = V or R, for second parameter P:

V = call-by-value:

1. If f_1 is R, one word is on top-of-stack, use SPL/3000 statement "TOS := P;" to set P value on top-of-stack in three more words (four words total), overlay result on the first three words, then delete the remaining word from the stack.
2. If f_1 is V, three words are on top-of-stack, use SPL/3000 statement "TOS := P;" to set P value on top-of-stack in three more words (six words total), overlay result on the first three words, then delete the remaining three words from the stack.

R = call-by-reference:

1. If f_1 is R, one word is on top-of-stack, use SPL/3000 statement "TOS := @P;" to set P reference address on top-of-stack in one word, use "TOS := 0;" to set the integer "0" on the top-of-stack in one more word (three words total), then overlay the result on those three words.
2. If f_1 is V, three words are on top-of-stack, use SPL/3000 statement "TOS := @P;" to set P reference address on top-of-stack in one more word (four words total), overlay result on the first three words, then delete the remaining word from the stack.

Method: One of three methods is used:

1. If B = 0.0 and P > 0.0, the result is set to 0.0.
2. If B > 0.0 and P = 0.0, the result is set to 1.0.
3. If B > 0.0 and P ≠ 0.0, result = DEXP(P*DLOG(B)).

Accuracy: See DEXP and DLOG.

ATTRIBUTES:

Parameters: B = 0.0, P > 0.0 or
B > 0.0, P = any LONG real number *(B is the base, P is the power).*

Result: A LONG real number.

FORTRAN: Not callable.

Errors: If B = 0.0 and P ≤ 0.0, or if B < 0.0, SOFTERROR' message LTOL': ILLEGAL ARGUMENTS occurs (see "Library Errors"). Or, see DEXP and DLOG.

FUNCTION: Raise a complex number base to an integer number power.

Declarations: Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a 2-element array) of real numbers, one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

A complex base B is raised to an integer power P by one of two procedures called by compiler-generated code. Each of the procedures is declared in the following format, to allow B to be call-by-reference or call-by-value:

```
PROCEDURE CTOIf';
  OPTION EXTERNAL;
          .
          .
          .
```


where

f = V or R, for the first parameter B:

V = call-by-value; use SPL/3000 statement "TOS := B(0);" to set B real-part value on top-of-stack in two words, use "TOS := B(1);" to set B imaginary part value on top-of-stack in two more words, use "TOS :=P;" to set P value on top-of-stack in one more word, overlay result on the first four words, then delete the remaining word from the stack.

R = call-by-reference; use SPL/3000 statement "TOS := @B;" to set B reference address on top-of-stack in one word, use "TOS := P;" to set P value on top-of-stack in one more word, use "TOS := 0D;" to set the double integer "0" on top-of-stack in two more words, then overlay the result on those four words.

Method: P is factored into powers of 2, then the result is obtained by successive multiplications:

EXAMPLE:

$$B^7 = B^1 B^2 B^4 = B^1 B^2 (B^2)^2$$

ATTRIBUTES:

Parameters: B = any complex number representable in two real numbers, one for *a* and one for *b*; neither *a* nor *b* may be 0.0 *(the base).*

P = any integer number > 0 *(the power).*

Result: A complex number.

FORTRAN: Not callable.

Errors: If *a* or *b* of B = 0.0, or if P $\leqslant$ 0, SOFTERROR' message CTOI': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If a or b of the result exceeds the range of real numbers (see "Introduction"), the arithmetic traps FLPT OFLW (floating point overflow) or FLPT UFLW (floating point underflow) may occur (if traps are enabled).

LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representation" in the "Introduction"). LONG real expressions (for example, X := Y + Z) are evaluated through primary LONG-arithmetic procedures: ADDLONG',SUBLONG',MLTLONG', and DIVLONG', all called by compiler-generated code.

Declarations

For each of the arithmetic operations there are eight entry-point procedures to allow any combination of call-by-reference or call-by-value parameters. Each of these entry point procedures is declared as follows:

```
PROCEDURE ADDLf1f2f3';
  OPTION EXTERNAL;
```

or

```
PROCEDURE SUBLf1f2f3';
  OPTION EXTERNAL;
```

or

```
PROCEDURE MLTLf1f2f3';
  OPTION EXTERNAL;
```

or

```
PROCEDURE DIVLf1f2f3';
  OPTION EXTERNAL;
```

where

f_1 = V or R, for the first parameter (subtraction minued or division dividend):

- V = call-by-value; use SPL/3000 statement "TOS := Y;" to set Y value on top-of-stack in three words.
- R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

f_2 = V or R, for the second parameter (subtraction subtrahend or division divisor):

- V = call-by-value; use SPL/3000 statement "TOS := Z;" to set Z value on top-of-stack in three words.
- R = call-by-reference; use SPL/3000 statement "TOS := @Z;" to set Z reference address on top-of-stack in one word.

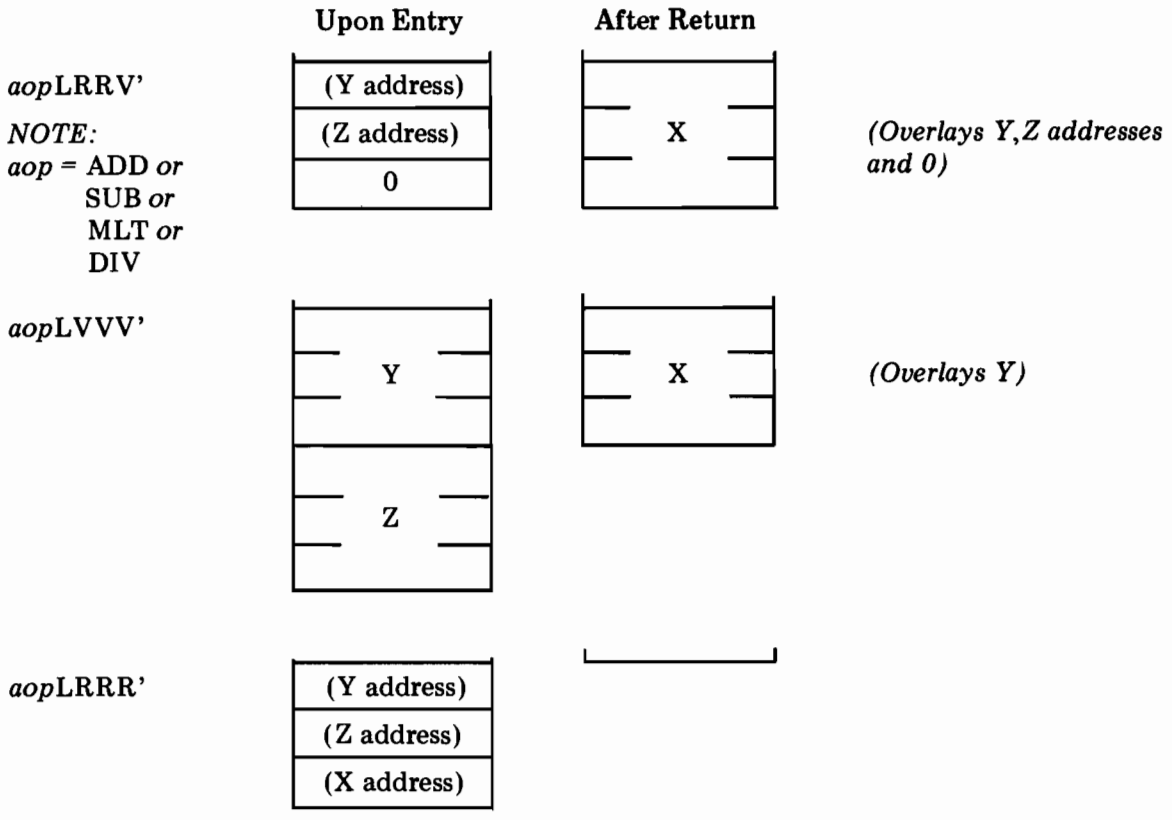
f_3 = V or R, for the result parameter

V = call-by-value:

1. If f_1 and f_2 are R, two words are on top-of-stack; use SPL/3000 statement "TOS := 0;" to set the integer "0" on top-of-stack in one more word, then overlay the result value on those three words.
2. If f_1 or f_2 is V, four or more words are on top-of-stack; overlay the result value on the first three words, then delete the remaining word(s) from the stack.

R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set result reference address on top-of-stack in one word, return result to that address, then delete the address and all of the first and second parameters from the stack.

EXAMPLES (of three of eight possible combinations):



A call to any of the entry point procedures sets a stack marker. The selected procedure then calls the primary LONG-arithmetic procedure, which sets another stack marker. The primary procedure changes indicators in the entry point procedure's stack marker in preparation for the arithmetic operation.

ATTRIBUTES:

Parameters: Any LONG real number.

Result: A LONG real number.

FORTRAN: Not callable.

Errors: (see "Internal Representations" in the "Introduction").

If the result is outside the range $\pm 1.1579208924 \times 10^{77}$, SOFTERROR' message LONG OVERFLOW occurs (see "Library Errors"). The correct result can be obtained by one of these methods:

1. If bit 1 (MSB of the exponent field) is clear (0), add 512 to the exponent.
2. If bit 1 is set, subtract 512 from the exponent.

If the result is inside the range $\pm 8.6361685551 \times 10^{-78}$, SOFTERROR' message LONG UNDERFLOW occurs. If the exponent and the fraction are both 0, the result is $(-1)^S * 2^{-256}$.

If the parameter Z is 0 for DIVLONG' (Y,Z,X), SOFTERROR' message LONG DIVIDE BY ZERO occurs.

Accuracy: The result is rounded to 39 significant bits of internal representation.

COMMENT: Indicators are CCA and overflow. If traps are enabled, the SOFTERROR' messages occur as just described. If the traps are disabled, control is returned to the user.

Long Negate

LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representation" in the "Introduction"). LONG real negate operations (for example, X := -Y) are evaluated through one of four procedures called by compiler-generated code.

Declarations

Each of the procedures is declared in the following format to allow any combination of call-by-reference or call-by-value parameters.

```
PROCEDURE NEGL$f_1 f_2$';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f_1 = V or R, for the first parameter:

V = call-by-value; use SPL/3000 statement "TOS := Y;" to set Y value on top-of-stack in three words.

R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

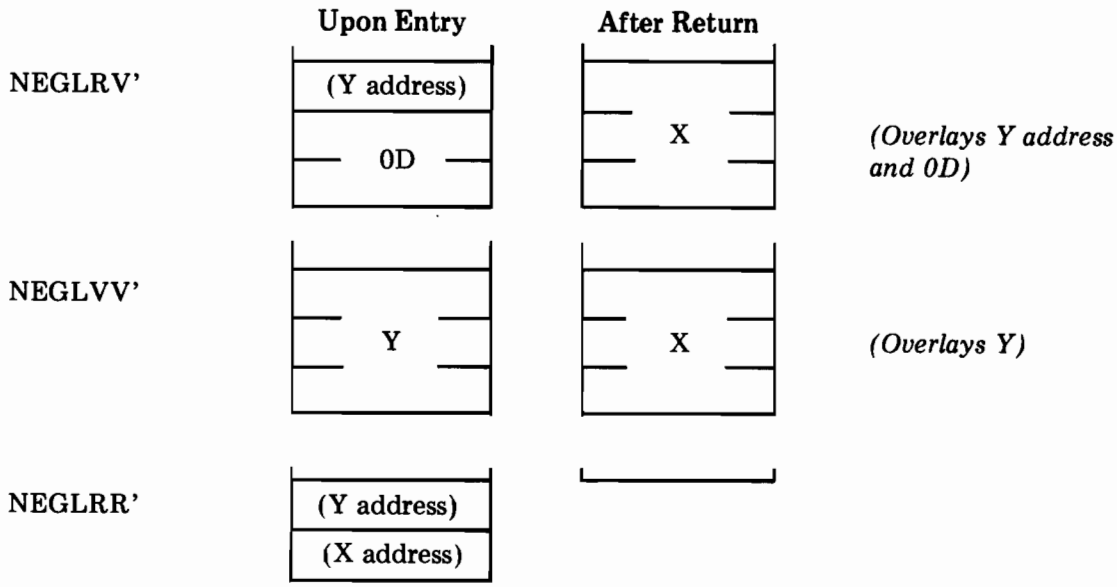
f_2 = V or R, for the result parameter:

V = call-by-value:

1. If f_1 is R, one word is on top-of-stack; use SPL/3000 statement "TOS := 0D;" to set the double integer "0" in two more words, then overlay the result value on those three words.
2. If f_1 is V, three words are on top-of-stack; overlay the result value on those three words.

R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set the result reference address on top-of-stack in one word, return result to that address, then delete the address and all of the first parameter from the stack.

EXAMPLES (of three of the four possible combinations):



ATTRIBUTES:

Parameter: Any LONG real number.

Result: Any LONG real number, except -2^{-256} which is internally represented by a 1 followed by 47 0's; there is no positive counterpart. (See "Internal Representation" in the "Introduction").

FORTRAN: Not callable.

Error: None.

COMMENT: Indicator is CCA.

Long Compare

LONG real numbers in SPL/3000 programs are represented in three words (see "Internal Representations" in the "Introduction"). LONG real compare operations (for example, X:Y) are evaluated through one of four procedures called by compiler-generated code.

Declaration

Each of the procedures is declared in the following format, to allow any combination of call-by-reference or call-by-value parameters:

```
PROCEDURE CMPLf1f2';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f_1 = V or R, for first parameter:

V = call-by-value; use SPL/3000 statement "TOS := Y;" to set Y value on top-of-stack in three words.

R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

f_2 = V or R, for second parameter:

V = call-by-value; use SPL/3000 statement "TOS := X;" to set X value on top-of-stack in three words.

R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set X reference address on top-of-stack in one word.

NOTE: All words of the first and second parameters are deleted from the stack after the result is set as defined in "Attributes."

ATTRIBUTES:

Parameters: Any LONG real numbers.

Result: Condition code:

If X < Y, CC = CCL

If X = Y, CC = CCE

If X > Y, CC = CCG

FORTRAN: Not callable.

Error: None.

Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a two-element array) of real numbers, one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

Complex expressions (for example, X := Y + Z) are evaluated through primary complex-arithmetic procedures: ADDC. . .', SUBC. . .', MLTC. . .', and DIVC. . .' called by compiler-generated code.

Declaration

For each of the arithmetic operations there are eight entry point procedures to allow any combination of call-by-reference or call-by-value parameters. Each of the entry-point procedures is declared as follows:

```
PROCEDURE ADDCf1f2f3'
  OPTION EXTERNAL;
```

or


```
PROCEDURE SUBCf1f2f3';
  OPTION EXTERNAL;
```

or

```
PROCEDURE MLTCf1f2f3';
  OPTION EXTERNAL;
```

or

```
PROCEDURE DIVCf1f2f3';
  OPTION EXTERNAL;
```

where

f_1 = V or R, for first parameter (subtraction minuend or division dividend):

- V = call-by-value; use SPL/3000 statement "TOS := Y(0);" to set Y real-part value on top-of-stack in two words, then use "TOS := Y(1);" to set Y imaginary-part value on top-of-stack in two more words (four words total).
- R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

f_2 = V or R, for second parameter (subtraction subtrahend or division divisor):

- V = call-by-value; use SPL/3000 statement "TOS := Z(0);" to set Z real-part value on top-of-stack in two words, then use "TOS := Z(1);" to set Z imaginary-part value on top-of-stack in two more words (four words total).
- R = call-by-reference; use SPL/3000 statement "TOS := @Z;" to set Z reference address on top-of-stack in one word.

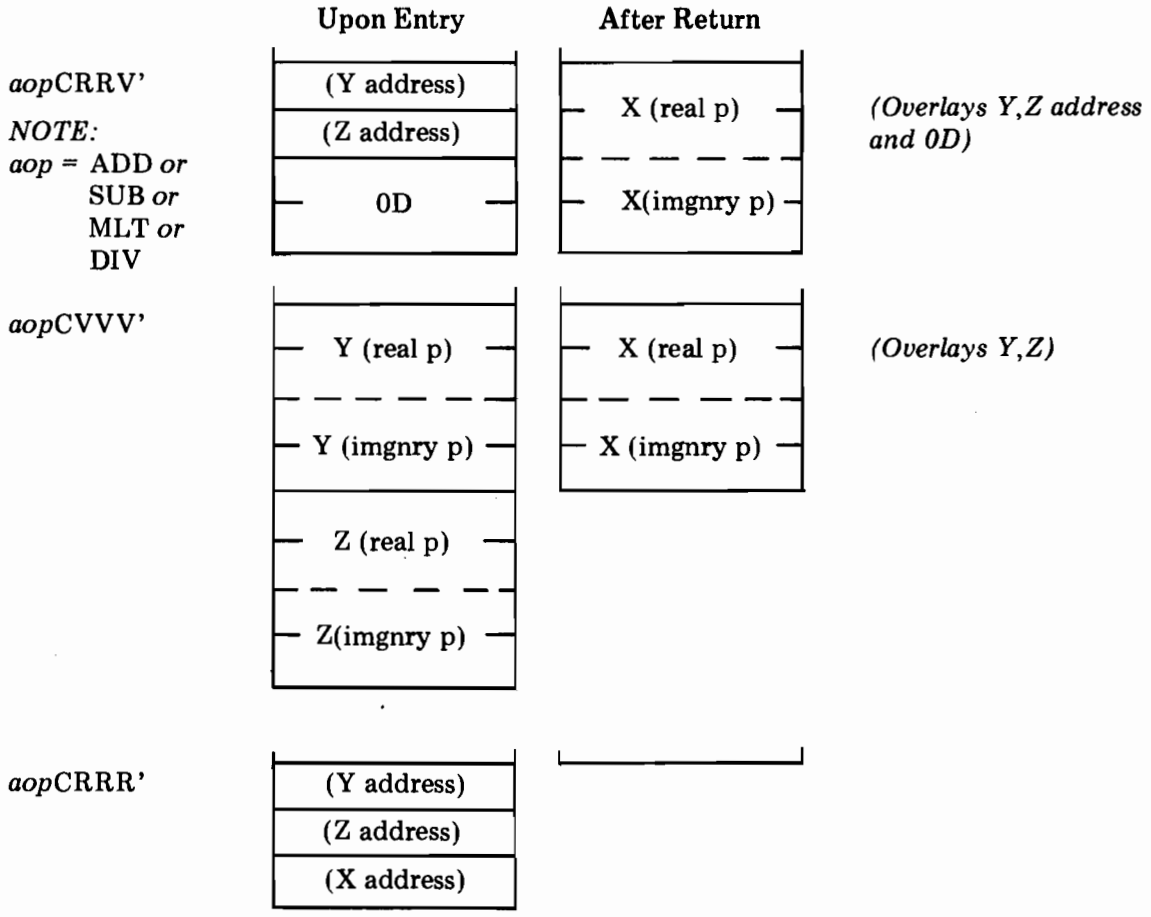
f_3 = V or R, for result parameter:

V = call-by-value:

1. If f_1 and f_2 are R, two words are on top-of-stack; use SPL/3000 statement "TOS := 0D;" to set the double integer "0" on top-of-stack in two more words, then overlay the result value on those four words.
2. If f_1 or f_2 is V, five or more words are on top-of-stack; overlay the result value on the first four words, then delete the remaining word(s) from the stack.

R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set result reference address on top-of-stack in one word, return result to that address, then delete the address and all of the first and second parameters from the stack.

EXAMPLES (of three of eight possible combinations):



ATTRIBUTES:

Parameters: Any complex number representable in two real numbers, one for *a* and one for *b*.

Result: Any complex number representable in two real numbers, one for *a* and one for *b*.

FORTRAN: Not callable.

Error: None.

CAUTION: The arithmetic traps FLPT OFLW (floating point overflow), FLPT UFLW (floating point underflow), or FLPT DIVZ (floating point divide by zero) may occur (if traps are enabled).

Complex Negate

Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a two-element array) of real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

Complex negate operations (for example, X := –Y) are evaluated through one of four procedures called by compiler-generated code.

Declaration

Each of these procedures is declared in the following format, to allow any combination of call-by-reference or call-by-value parameters:

```
PROCEDURE NEGC'f1 f2 ';
  OPTION EXTERNAL;
          .
          .
          .
```

where

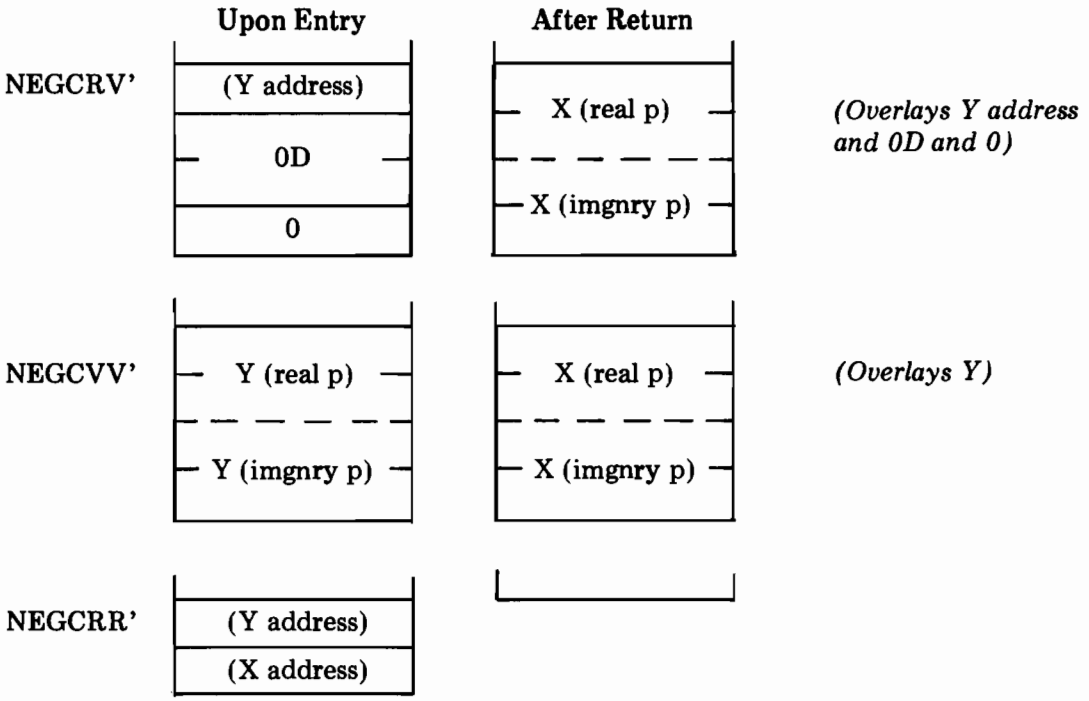
f_1 = V or R, for the first parameter:

- V = call-by-value; use SPL/3000 statement "TOS := Y(0);" set Y real-part value on top-of-stack in two words, then use "TOS := Y(1);" to set Y imaginary-part value on top-of-stack in two more words (four words total).
- R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

f_2 = V or R, for the result parameter:

- V = call-by-value:
  1. If f_1 is R, one word is on top-of-stack; use SPL/3000 statement "TOS := 0D;" to set the double integer "0" on top-of-stack in two more words, use "TOS := 0;" to set the integer "0" on top-of-stack in one more word, then overlay the result on those four words.
  2. If f_1 is V, four words are on top-of-stack; overlay the result value on those four words.
- R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set the result reference address on top-of-stack in one word, return result to that address, then delete the address and all of the first parameter from the stack.

EXAMPLES (of three of four possible combinations):



ATTRIBUTES:

Parameter: Any complex number representable in two real numbers, one for *a* and one for *b*.

Result: Any complex number in which neither the real part *a* nor the imaginary part *b* is -2^{-256}; that value is internally represented by a 1 followed by 47 0's; there is no positive counterpart. (See "Internal Representation" in the "Introduction").

FORTRAN: Not callable.

Error: None.

COMMENT: Indicator is CCA.

Complex Compare

Complex numbers in FORTRAN/3000 programs are represented as an ordered pair (a two-element array) of real numbers, one for the real part *a* and one for the imaginary part *b*. Thus, complex numbers occupy four words (see "Internal Representation" in the "Introduction").

Complex compare operations (for example, X:Y) are evaluated through one of four procedures called by compiler-generated code.

Declaration

Each of the procedures is declared in the following format, to allow any combination of call-by-reference or call-by-value parameters:

```
PROCEDURE CMPC$f_1 f_2$';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f_1 = V or R, for the first parameter:

- V = call-by-value; use SPL/3000 statement "TOS := Y(0);" to set Y real-part value on top-of-stack in two words, then use "TOS := Y(1);" to set Y imaginary-part value on top-of-stack in two more words (four words total).
- R = call-by-reference; use SPL/3000 statement "TOS := @Y;" to set Y reference address on top-of-stack in one word.

f_2 = V or R, for the second parameter:

- V = call-by-value; use SPL/3000 statement "TOS := X(0);" to set X real-part value on top-of-stack in two words, then use "TOS := X(1);" to set X imaginary-part value on top-of-stack in two more words (four words total).
- R = call-by-reference; use SPL/3000 statement "TOS := @X;" to set X reference address on top-of-stack in one word.

NOTE: All words of the first and second parameters are deleted from the stack after the result is set as defined in "Attributes."

ATTRIBUTES:

Parameters: Any complex numbers each representable in two real numbers, one for *a* and one for *b*.

Result: Condition code:

If X(0) < Y(0), CC = CCL

If X = Y, CC = CCE

If X(0) > Y(0), CC = CCG

FORTRAN: Not callable.

Error: None.

FTNAUX'

FUNCTION: Normally called only by FORTRAN/3000 compiler generated code to implement the FORTRAN auxiliary I/O statements REWIND, BACKSPACE, and ENDFILE (see "Comments").

Declaration:

```
PROCEDURE FTNAUX' (OPT,UNIT);
  VALUE OPT, UNIT;INTEGER OPT, UNIT;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: OPT: An integer to specify the action

- < 0: REWIND
- = 0: BACKSPACE
- > 0: ENDFILE

UNIT: A positive integer in the range [1,99] to specify the FORTRAN Logical Unit Table (FLUT) entry (see "Comments") or a negated MPE/3000 file number (SPL/3000 callers only).

Result: (See "Comments").

FORTRAN: Callable only through the auxiliary I/O statements.

Error: (See "Comments").

COMMENTS:

1. The following comments refer to descriptions in the *HP 3000 Multiprogramming Executive Operating System (HP 03000-90005)* and the *HP 3000 Systems Programming Language (HP 03000-90002).*

2. If the FORTRAN/3000 compiler generates the code to call FTNAUX', a FORTRAN Logical Unit Table is prepared in the user's DB Data Area by the MPE/3000 system loader.

3 SPL/3000 users can call FTNAUX' directly, if desired. If UNIT is a negated file number, that file number must have been opened by use of the MPE/3000 file intrinsic FOPEN. If UNIT is a positive integer in the range [1,99] (a FORTRAN Logical Unit), the SPL/3000 user must have created a FLUT, as described in Section I, "File System Requirements."

4. If UNIT is a positive integer in the range [1,99], FTNAUX' checks the FLUT for that UNIT number. If there is no corresponding U entry, SOFTERROR' message FMT: FILE NOT IN TABLE occurs (see "Library Errors") and FTNAUX' terminates. If a corresponding U entry is found and the F entry for that is 0, an MPE/3000 file intrinsic FOPEN call is made with nominal FORTRAN file parameters (see Section I, "File System Requirements").

Those parameters include the file name built by appending the UNIT number (in the left byte of the FLUT entry) to the ASCII characters FTN. For example, the file name for UNIT 3 is FTN03. There are two exceptions to the construction of file names: FORTRAN defines UNIT 5 to be $STDIN and UNIT 6 to be $STDLIST. If the FOPEN file intrinsic is not successful (indicated by condition code CCL), SOFTERROR' message "FMT: OPEN FILE NAME ERROR" occurs (see "Library Errors").

A second entry point, FNUM, is provided in this FTNAUX' procedure. FNUM allows a FORTRAN/3000 programmer to use the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number for the program in use. For example, FNUM can be called, then an MPE/3000 file intrinsic such as FGETINFO can be used with the file number returned by the entry point FNUM. For further details, see "FNUM'" in this section.

5. REWIND or BACKSPACE actions are provided, historically, for control of magnetic tape files. If the device referenced has no physical capability corresponding to the OPTION (action) request, no action occurs.

6. For a REWIND request an MPE/3000 file intrinsic FCONTROL call is made for control code 5 (rewind file). This may invoke a physical operation on the device referenced.

7. For a BACKSPACE request an MPE/3000 file intrinsic FSPACE call is made for –1 DISPLACEMENT. This may invoke a physical operation on the device referenced.

8. For an ENDFILE request an MPE/3000 file intrinsic FCLOSE call is made for 0 DISPOSITION.

FNUM

FUNCTION: A secondary entry point to procedure FTNAUX'. FNUM enables a FORTRAN/3000 programmer to extract the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number from the FORTRAN Logical Unit Table. See "FTNAUX' " in this section.

Declaration:

```
INTEGER PROCEDURE FNUM(UNIT);
  INTEGER UNIT;
    OPTION EXTERNAL;
          .
          .
          .
```

ATTRIBUTES:

Parameter: A positive integer number in the range [1,99] (same as the parameter UNIT for FTNAUX').

Result: An integer number, the MPE/3000 system file number.

FORTRAN: Callable as an external function: I = FNUM(UNIT)

Errors: See "Comments."

COMMENTS:

1. If UNIT is < 1 or > 99, SOFTERROR' message FMT: FILE NOT IN TABLE occurs (see "Library Errors").
2. If UNIT is in the range [1,99], execution continues as described in FTNAUX' Comment 4.

FSET

FUNCTION: A secondary entry point to procedure FTNAUX'. FSET enables a FORTRAN/3000 programmer to change the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number in the FORTRAN Logical Unit Table. (See "FTNAUX' " in this section.)

Declaration:

```
PROCEDURE FSET(UNIT,NEWFILE,OLDFILE);
  INTEGER UNIT,NEWFILE,OLDFILE;
    OPTION EXTERNAL;
              .
              .
              .
```

ATTRIBUTES:

Parameters: (input) UNIT, a positive integer in the range [1,99] (same as the parameter UNIT for FTNAUX').

NEWFILE, a positive integer in the range [0,255] to specify the new MPE/3000 system file number to be assigned to the UNIT specified above.

Parameter: (output) OLDFILE, a positive integer; the previous MPE/3000 system file assigned to the UNIT specified above.

Result: (See "Parameter (output)," above.)

FORTRAN: Callable as an external subroutine, for example,

```
CALL FSET(3,FNUMB,OLD)
```

Errors: If UNIT or NEWFILE is not in the range stated above, or if the UNIT number is not in the FORTRAN Logical Unit Table, SOFTERROR' message FMT: FILE NOT IN TABLE occurs (see "Library Errors").

Procedures from the following list are called by FORTRAN/3000 compiler-generator code at run time to perform the functions listed for users' programs.

Procedure(s) Identifier	Function
ACHRLL' ACHRLS' ACHRSL' ACHRSS'	Assigns a character string. The different entry points are for long (L) and short (S) target or source strings. "Long" means substring parameters are included. SUBSTR' and BLANKFILL' may be called.
ACHRLPB' ACHRSPB'	Assigns a character string. The source string is a PB string, the target string may be long (L) or short (S). BLANKFILL' is called if the target is longer than source, the stack is set up to do a MOVE PB after return.
BCA'1 BCA'2 BCA'3	Checks, when a program executes, the declaration subscript bounds for a 1, 2 or 3 (and more) dimension local array (if the BOUNDS option is used).
BFA'1 BFA'2 BFA'3	Checks, when a program executes, the declaration subscript bounds for a 1, 2 or 3 (and more) dimension formal (dummy) array (if the BOUNDS option is used).
BLANKFILL'	Blankfills a string in character assignment when the target string is longer than the source string.
BNDCHK1' BNDCHK2' BNDCHK3'	Checks, when a program executes, an assignment or I/O statement for subscript within bounds for a 1, 2 or 3 dimension array. An integer procedure that returns the index in the array. Parameters are the array bounds and subscript values.
BNDCHKN'	Checks, when a program executes, an assignment or I/O statement for subscript within bounds for multidimensional (i.e. more than 3) arrays. An integer procedure that returns the index in the array. Parameters are the array bounds, the subscript values and the number of subscripts.
CCHRLPB'	Compares character strings where source string is a PB string. The stack is set up for a CMPB PB.
CCHRLL' CCHRLS' CCHRSL' CCHRSS'	Compares character strings. Different entry points are for long (L) and short (S) strings. "Long" means substring parameters are included.

Procedure(s) Identifier	Function
INUM' RNUM' DNUM'	Converts a character expression into an integer, real, or double precision value. Procedure EXTIN' (see "Function Directory") is called with parameters set as follows:

Procedure	W	D	TYPE
INUM'	Field width *w*	0	0 (integer)
RNUM'	Field width *w*	0	1 (real)
DNUM'	Field width *w*	0	-2 (LONG[1] real)

IFIXT'	Fixes and truncates a real value to an integer value. Used for index expressions.
INDEX'	Searches a first argument, a character variable, for a subpart matching its second argument, a character expression. Returns 0 if not found, or returns the position of the first character of the matching subpart (integer value).
STR'	Converts a linear expression to a string of length specified by the second argument (integer constant).
SUBSTR'	Generates a byte address of a substring in a character string. Procedure also checks to see if a substring is contained in the source string.

[1] In SPL/3000; same as FORTRAN type double precision.

SECTION IV
Library Errors

Many of the routines in the HP 3000 Compiler Library, particularly mathematical routines, can detect error(s) in the data processed. These routines make a call to the library error routine SOFTERROR'. The normal function of SOFTERROR' is to produce an error message and abort the program.

SOFTERROR'

The SOFTERROR' routine should not be called directly by a user; it is intended for use only by the library routines. This presentation of the SOFTERROR' declaration is for information only:

```
PROCEDURE SOFTERROR'(NUM);
  VALUE NUM;INTEGER NUM;
      .
      .
      .
```

The parameter NUM, replaced by an actual parameter from the calling library routine, indicates the error message to be produced, as listed in Table 4-1.

XLIBTRAP

The user can override the normal functions of SOFTERROR' and specify his own error procedure(s). To do so, the MPE/3000 intrinsic function XLIBTRAP is used. That function is declared.

```
PROCEDURE XLIBTRAP (PLABEL,OLDPLABEL);
  VALUE PLABEL;
  INTEGER PLABEL,OLDPLABEL;
    OPTION EXTERNAL;
          .
          .
          .
```

where

PLABEL = external label of the user-written error procedure or 0 to disarm the library trap mechanism. If 0, control is *not* passed to a user's error procedure.

OLDPLABEL = original PLABEL, returned to permit the user to return to the previous conditions.

Execution proceeds as follows:

1. A library procedure finds an error and calls SOFTERROR'.
2. SOFTERROR' checks for a user-written error procedure.
3. If no user-written error procedure has been specified, SOFTERROR' produces the appropriate error message then aborts the current program. If a user-written error procedure has been specified, SOFTERROR' disarms the library trap mechanism then calls the user-written procedure.
4. When the user-written error procedure returns control to SOFTERROR', the library trap mechanism is re-armed.
5. As defined below, the user-written error procedure must also set a flag, QUIT, to direct SOFTERROR' to abort the current program or to return control to the procedure that found the error.

The user-written error procedure (in this example, the hypothetical name ERROR) should be declared:

```
PROCEDURE ERROR(MARKER,ERRORNUM,QUIT);
  LOGICAL ARRAY MARKER;INTEGER ERRORNUM;
    LOGICAL QUIT;
              .
              .
              .
```

where

MARKER is a four-word array containing the stack marker created for the library error routine that detected the error. Thus, MARKER (1) is the PB relative address in the user program where the error occurred.

ERRORNUM indicates which error occurred (see the list at the end of this section).

QUIT is a flag set by the user-written procedure ERROR. If QUIT = FALSE, SOFTERROR' will return directly to the user program without printing an error message; if QUIT = TRUE, SOFTERROR' will abort the user program.

EXAMPLE: XLIBTRAP USE

Assume that in the procedure USER, the user-written error procedure MINE is to be substituted for SOFTERROR' whenever a library error is detected. A program might be written as shown on the following page.

```
BEGIN                <<MAIN PROGRAM>>
    (declarations)
    .
    .
    .
    PROCEDURE XLIBTRAP(NEW,OLD);VALUE NEW;LOGICAL NEW,OLD;
      OPTION EXTERNAL;
    PROCEDURE MINE(MARK,ERNUM,QUIT);
      LOGICAL ARRAY MARK;INTEGER ERNUM;
      LOGICAL QUIT;
          BEGIN
              .
              .
              .
          END;<<MINE>>
PROCEDURE USER;
  BEGIN
      LOGICAL OLD;
      TOS:=@MINE;
      XLIBTRAP(*,OLD);
              .
              .
              .
      XLIBTRAP(OLD,OLD);<<RESTORE INITIAL PROCEDURE>>
  END;<<USER>>
      .
      .
      .
END:  <<MAIN PROGRAM>>
```

Table 4-1. HP 3000 Compiler Library Errors

Error number	Library routine name	Error description	SOFTERROR' message
1	ATAN2	Both arguments = 0	ATAN2: ARGUMENTS ZERO
2	ATAN2	Underflow when arguments divided	ATAN2: UNDERFLOW
3	DATAN2	Both arguments = 0	DATAN2: ARGUMENTS ZERO
4	DATAN2	Underflow when arguments divided	DATAN2: UNDERFLOW
5	EXP (or EXP'	Result overflow	EXP: OVERFLOW
6	DEXP (or DEXP'	Result overflow	DEXP: OVERFLOW
7	ALOG (or ALOG')	Argument ≤ 0	ALOG: ARGUMENT NOT POSITIVE
8	DLOG (or DLOG')	Argument ≤ 0	DLOG: ARGUMENT NOT POSITIVE
9	CABS	Result overflow	CABS: OVERFLOW
10	SQRT	Argument < 0	SQRT: ARGUMENT NEGATIVE
11	DSQRT	Argument < 0	DSQRT: ARGUMENT NEGATIVE
12	TAN	Argument near $\frac{(2k+1)\pi}{2}$ (see text)	TAN: OVERFLOW
13	DTAN	Argument near $\frac{(2k+1)\pi}{2}$ (see text)	DTAN: OVERFLOW
14 . . . 50	(Unassigned)		
51	All LONG real arithmetic	Result overflow	LONG OVERFLOW
52		Result underflow	LONG UNDERFLOW
53		Divide by 0	LONG DIVIDE BY ZERO
54	ITOI'	Base = 0 and power ≤ 0	ITOI': ILLEGAL ARGUMENTS
55	RTOI'	Base = 0 and power ≤ 0	RTOI': ILLEGAL ARGUMENTS
56	RTOR'	Base = 0 and power ≤ 0 or base < 0 and power $\neq 0$	RTOR': ILLEGAL ARGUMENTS
57	RTOL'	Base = 0 and power ≤ 0	RTOL': ILLEGAL ARGUMENTS

Table 4-1. *(cont.)* HP 3000 Compiler Library Errors

Error number	Library routine name	Error description	SOFTERROR' message
58	LTOI'	Base = 0 and power $\leq$ 0	LTOI': ILLEGAL ARGUMENTS
59	LTOL'	Base = 0 and power $\leq$ 0 or base $<$0 and power $\neq$ 0	LTOL': ILLEGAL ARGUMENTS
60	CTOI'	Base = 0 and power $\leq$ 0	CTOI': ILLEGAL ARGUMENTS
61	INUM' RNUM' DNUM'	Illegal character in string being converted.	NUM: ILLEGAL CHARACTER
62	INUM' RNUM' DNUM'	Number out of representable range (see "Introduction").	NUM: NUMBER OUT OF RANGE
63	INUM' RNUM' DNUM'	(both of the above)	NUM: RANGE AND CHAR ERROR
64	(any)	Illegal EXIT label.	INVALID EXIT ON RETURN
65	BNDCHK*x*' BCA'*x* BFA'*x*	Subscript out of range (not detected unless $CONTROL BOUNDS is requested).	INVALID SUBSCRIPT VALUE
66	SUBSTR'	Designator defines a substring not contained in the source string.	INVALID SUBSTRING DESIGNATOR
67 . . . 100	(Unassigned)		
101	Formatter	Illegal format character	FMT: ILLEGAL FORMAT CHARACTER
102	Formatter	*w* field too small for number	FMT: FIELD WIDTH TOO SMALL
103	Formatter	Specification group(s) nested deeper than level 4	FMT: NESTING TOO DEEP
104	Formatter	List and character string specification do not match	FMT: STRING MISMATCH
105	Formatter	Illegal character in input field	FMT: BAD INPUT CHARACTER
106	Formatter	Numeric input field unrepresentable	FMT: NUMBER OUT OF RANGE
107	Formatter	Format specification exceeds record length	FMT: FORMAT BEYOND RECORD
108	Formatter	Core-to-core conversion exceeds user-defined buffer	FMT: BUFFER OVERFLOW

Table 4-1. *(cont.)* HP 3000 Compiler Library Errors

Error number	Library routine name	Error description	SOFTERROR' message
109	Formatter	Binary direct access exceeds record length	FMT: DIRECT ACCESS OVER-FLOW
110	Formatter	File name not in FLUT	FMT: FILE NOT IN TABLE
111	Formatter	File access problem.	FMT: FILE SYSTEM RETURNED CCL
112	Formatter	File access problem.	FMT: FILE SYSTEM RETURNED CCG