HP 3000 Computer Systems

Compiler Library

Reference Manual

HEWLETT hp PACKARD

5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA, 95050

Part No. 30000-90028
Product No. 32211D

Printed in U.S.A. 11/76

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

Pages	Effective Date
Title	Nov 1976
ii	Jun 1976
iii to iv	Nov 1976
v	Jun 1976
vii	Jun 1976
viii	Nov 1976
ix to xiii	Jun 1976
xv to xvi	Jun 1976
xvii to xxiii	Nov 1976
1-1	Jun 1976
1-2	Nov 1976
1-3 to 1-4	Jun 1976
1-5 to 1-6	Nov 1976
1-7 to 1-21	Jun 1976
1-22 to 1-25	Nov 1976
1-26 to 1-27	Jun 1976
1-28	Nov 1976
1-29	Jun 1976
1-30	Nov 1976
1-31	Jun 1976
1-32	Nov 1976
1-33 to 1-36	Jun 1976
1-37	Nov 1976
1-38 to 1-40	Jun 1976
1-41	Nov 1976
1-42 to 1-46	Jun 1976
1-47 to 1-48	Nov 1976
1-49 to 1-50	Jun 1976
1-51 to 1-52	Nov 1976
1-53 to 1-55	Jun 1976
1-56	Nov 1976
1-57 to 1-58	Jun 1976
1-59	Nov 1976
1-60	Jun 1976
1-61	Nov 1976
1-62 to 1-64	Jun 1976
2-1 to 2-3	Jun 1976
2-4	Nov 1976
2-5 to 2-6	Jun 1976
2-7	Nov 1976
2-8 to 2-9	Jun 1976
2-10 to 2-13	Nov 1976
2-14	Jun 1976
2-15 to 2-17	Nov 1976
2-18	Jun 1976
2-19	Nov 1976
2-20 to 2-21	Jun 1976
2-22 to 2-23	Nov 1976
2-24 to 2-55	Jun 1976
2-56 to 2-59	Nov 1976
2-60	Jun 1976
2-61 to 2-68	Nov 1976
2-69 to 2-79	Jun 1976
3-1 to 3-7	Jun 1976
3-8 to 3-12	Nov 1976
3-13 to 3-14	Jun 1976
3-15 to 3-16	Nov 1976
3-17 to 3-18	Jun 1976
3-19	Nov 1976
3-20	Jun 1976
3-21	Nov 1976
3-22 to 3-23	Jun 1976
3-24	Nov 1976
3-25 to 3-35	Jun 1976
3-36 to 3-37	Nov 1976
3-38 to 3-40	Jun 1976
3-41	Nov 1976
3-42 to 3-54	Jun 1976
3-55 to 3-56	Nov 1976
4-1	Nov 1976
4-2 to 4-7	Jun 1976
A-1 to A-2	Nov 1976
I-1 to I-5	Nov 1976

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

The software product part number printed alongside the date indicates the version and update level of the software product at the time the manual edition or update was issued. Many product updates and fixes do not require manual changes, and conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	June 1976
Second Edition	November 1976

PREFACE

The *Compiler Library Reference Manual* 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.

MPE Intrinsics Reference Manual, 30000-90010

Systems Programming Language Reference Manual 30000-90024

FORTRAN/3000 Reference Manual, 30000-90040

System Manager/Supervisor Manual, 30000-90014

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.

Purposes of the Library and its relationship to the MPE/3000 Operating Systems are stated in the Introduction; so are the structural elements of this book. A Function Directory follows, to categorize each function provided in the Library and point to the description of the procedure. Section I presents a detailed analysis of the Formatter. Functions and attributes of other procedures are defined in a standard format, in Section II for Mathematical Procedures and Section III for Utility Procedures. Section IV outlines the special utilities for Library Errors.

NOTE: A special procedure, HP32211, is included in the HP 3000 Compiler Library, to report version identification for the Library in use. This procedure can be called, for example, from a FORTRAN/3000 program:

CALL HP 32211

CONTENTS

TABLES

INTRODUCTION

Introduction

HP 3000 Compiler Library routines perform input/output, internal data conversion, mathematical, data plotting, 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 *Systems Programming Language Reference Manual.*

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)

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

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

$\left|\frac{f-g}{f}\right|$ = 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: When applicable, how FORTRAN/3000 uses or calls the procedure.

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

COMMENTS: (When needed, special comments.)

Parameter Checking for Procedures

SPL/3000 procedures declared option external or called as an external procedure by other programs can include a CHECK option for attributes of the procedure. For SPL/3000 callers, the CHECK option can be specified in the declaration in the calling program, as described in the *Systems Programming Language Reference Manual* under "PROCEDURE DECLARATION." The CHECK option levels (values) are:

0 — no checking

1 — check procedure type only

2 — check procedure type and number of parameters

3 — check procedure type, number of parameters, and type of each parameter.

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

The following procedures in the HP 3000 Compiler Library do specify, in their declarations, the CHECK option for level 3:

ALOG
ALOG10
ATAN
ATAN2
COSH
DATAN
DATAN2
DCOS
DEXP
DINVERT
DLOG
DLOG10
DSIN
DSQRT
DTAN
EXP
FACTOR
INVERT
PLOT
PLOTS
SIN
SINH
SQRT
SYMBOL
TAN
TANH
WHERE

One Library procedure specifies, in its declaration, the CHECK option for level 2:

RAND

Text Conventions

The following conventions are used throughout this 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 *Systems Programming Language Reference Manual.*
4. The notation := means "is replaced by."
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 $a \leqslant x < b$

Internal Representations

Data Characteristics

Integer:

$[-32768, 32767]$,
in 1 word:

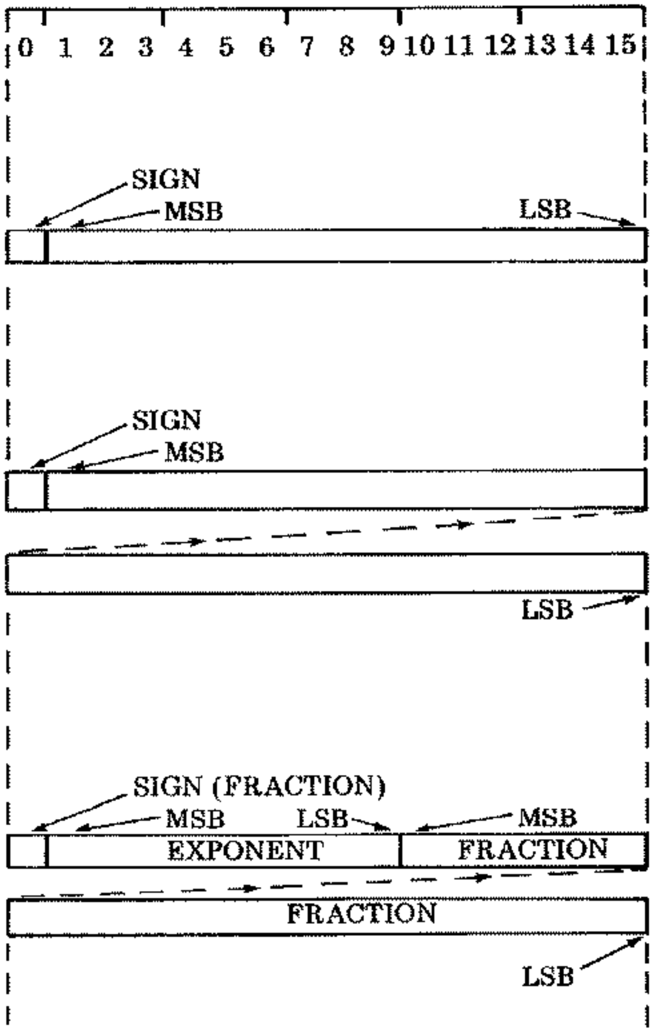
Double Integer:

$[-2147483648, 2147483647]$,
in 2 words, concatenated:

Real:

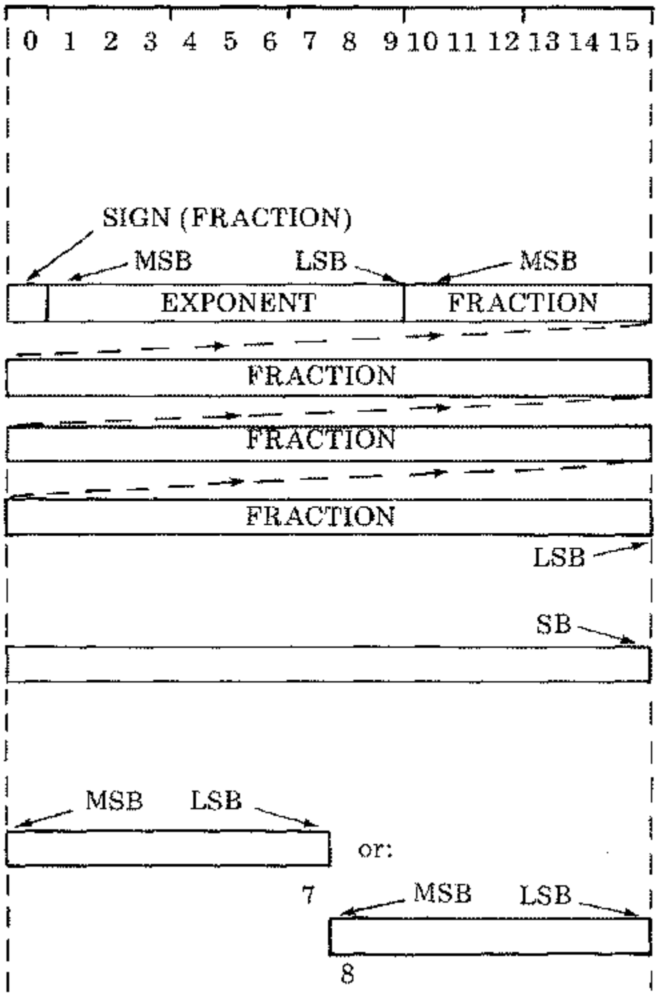
$[-1.15792 \cdot 10^{77}$
$-8.63617 \cdot 10^{-78}]$ and 0.0 and
$[8.63617 \cdot 10^{-78},$
$1.15792 \cdot 10^{77}]$ in 2 words,
concatenated (see definitions
on the next page):

Internal Representation Format



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

Data Characteristics **Internal Representation Format**



Double Precision:[1]
[$-1.157920892373162 \cdot 10^{77}$
$-8.636168555094445 \cdot 10^{-78}$] and
0.0 and [$8.636168555094445 \cdot$
10^{-78}, $1.157920892373162 \cdot 10^{77}$]
in 4 words, concatenated (see definitions below):

Logical (Boolean):

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

Byte (ASCII character code):

[any 8-bit code],
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

$(-1)^S * 2^{E-256} * (1.+F*2^{-54})$ 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

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.

Function	Identifier	Page
Data Format Definitions		
For a double precision[1] number with an exponent (floating-point).	FORMATTER, see D*w.d* Field Descriptor	1-7
For a real number with an exponent (floating-point).	FORMATTER, see E*w.d* Field Descriptor.	1-9
For a real number without an exponent (fixed-point).	FORMATTER, see F*w.d* Field Descriptor.	1-11
For a real number with or without an exponent, according to the relative size of the number.	FORMATTER, see G*w.d* Field Descriptor.	1-13
For a real number written in monetary (business) form.	FORMATTER, see M*w.d* Field Descriptor.	1-16
For a real number written in numeration form.	FORMATTER, see N*w.d* Field Descriptor.	1-18
For an integer number.	FORMATTER, see I*w* Field Descriptor.	1-20
For an octal integer number.	FORMATTER, see O*w* Field Descriptor.	1-22
For a hexidecimal integer number	FORMATTER, see Z*w* Field Descriptor	1-24
For a logical value.	FORMATTER, see L*w* Field Descriptor.	1-26
For the leftmost ASCII alphameric characters of a variable.	FORMATTER, see A*w* Field Descriptor.	1-28

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

Function	Identifier	Page
Data Manipulation *(cont.)*		
Transfer data between files and a list of variables without conversion.	FORMATTER, see Unformatted (Binary) Transfer	1-52
Convert a byte array of ASCII numeric data to an internal representation.	EXTIN'	3-1
Convert an internal representation of a number to a byte array for ASCII numeric data.	INEXT'	3-4
Absolute Values		
Calculate the absolute value of a double precision[1] number.	DABS'	2-1
Calculate the absolute value of a complex[2] number.	CABS (or CABS')	2-2
Calculate the absolute value of one integer number and give it the sign of a second integer number.	ISIGN'	2-3
Calculate the absolute value of one double integer number and give it the sign of a second double integer number.	JSIGN'	2-4
Calculate the absolute value of one real number and give it the sign of a second real number.	SIGN'	2-5
Calculate the absolute value of one double precision[1] number and give it the sign of a second double precision[1] number.	DSIGN'	2-6
Number Conversion		
Truncate a real number to an integer number.	INT' IFIX'	2-7 2-68
Truncate a real number to an integer number (result in real representation).	AINT'	2-8
Truncate a double precision[1] number to an integer number (result in double precision[1] representation).	DDINT'	2-9
Truncate a LONG real number to a double-integer number	DFIX (or DFIX')	2-10
Convert a double-integer number to a LONG real number	DFLOAT (or DFLOAT')	2-11

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

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

Function	Identifier	Page
Number Comparison		
Calculate the largest (MAX0') or the smallest (MIN0') of N integer numbers.	MAX0'/MIN0'	2-12
Calculate the largest (JMAX0') or the smallest (JMIN0') of N double integer numbers.	JMAX0'/JMIN0'	2-13
Calculate the largest (MAX1') or the smallest (MIN1') of N real numbers (result in integer representation).	MAX1'/MIN1'	2-14
Calculate the largest (JMAX1') or the smallest (JMIN1') of N real numbers (result in double integer representation).	JMAX1'/JMIN1'	2-16
Calculate the largest (AMAX0') or the smallest (AMIN0') of N integer numbers (result in real representation).	AMAX0'/AMIN0'	2-18
Calculate the largest (AJMAX0') or the smallest (AJMIN0') of N double integer numbers (result in real representation).	AJMAX0'/AJMIN0'	2-19
Calculate the largest (AMAX1') or the smallest (AMIN1') of N real numbers.	AMAX1'/AMIN1'	2-20
Calculate the largest (DMAX1') or the smallest (DMIN1') of N double precision[1] numbers.	DMAX1'/DMIN1'	2-21
Calculate one real number modulus a second real number.	AMOD'	2-22
Calculate one double precision[1] number modulus a second double precision[1] number.	DMOD	2-23
Exponents, Roots, Logarithms		
Calculate e^X, where x is a real number.	EXP (or EXP')	2-24
Calculate e^X, where x is a double precision[1] number.	DEXP (or DEXP')	2-25
Calculate e^X, where x is a complex[2] number.	CEXP (or CEXP')	2-26
Calculate the square root of a real number.	SQRT (or SQRT')	2-27
Calculate the square root of a double precision[1] number.	DSQRT (or DSQRT')	2-28
Calculate the square root of a complex[2] number.	CSQRT (or CSQRT')	2-29
Raise an integer number to an integer power.	ITOI'	3-8
Raise a double integer number to an integer power.	DTOI'	3-9
Raise a double integer number to a double integer power	DTOD'	3-10
Raise a real number to an integer power.	RTOI'	3-11
Raise a real number to a double integer power.	RTOD'	3-12
Raise a real number to a real power.	RTOR'	3-13
Raise a real number to a double precision[1] power.	RTOL'	3-14

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

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

Function	Identifier	Page
Exponents, Roots, Logarithms *(cont.)*		
Raise a double precision[1] number to an integer power.	LTOI'	3-15
Raise a double precision[1] number to a double integer power.	LTOD'	3-16
Raise a double precision[1] number to a double precision[1] power.	LTOL'	3-17
Raise a complex[2] number to an integer power.	CTOI'	3-19
Raise a complex[2] number to a double integer power.	CTOD'	3-20
Calculate the natural logarithm of a positive real number.	ALOG (or ALOG')	2-31
Calculate the base 10 logarithm of a positive real number.	ALOG10	2-31
Calculate the natural logarithm of a positive double precision[1] number.	DLOG (or DLOG')	2-32
Calculate the base 10 logarithm of a positive double precision[1] number.	DLOG10	2-32
Calculate the natural logarithm of a complex[2] number.	CLOG (or CLOG')	2-33
Trigonometry		
Calculate the tangent of a real number in radians.	TAN (or TAN')	2-34
Calculate the sine of a real number in radians.	SIN (or SIN')	2-35
Calculate the cosine of a real number in radians.	COS (or COS')	2-36
Calculate the tangent of a double precision[1] number in radians.	DTAN (OR DTAN')	2-37
Calculate the sine of a double precision[1] number in radians.	DSIN (or DSIN')	2-38
Calculate the cosine of a double precision[1] number in radians.	DCOS (or DCOS')	2-39
Calculate the tangent of a complex[2] number.	CTAN (or CTAN')	2-40
Calculate the sine of a complex[2] number.	CSIN (or CSIN')	2-41
Calculate the cosine of a complex[2] number.	CCOS (or CCOS')	2-42
Calculate the hyperbolic tangent of a real number.	TANH (or TANH')	2-43
Calculate the hyperbolic sine of a real number.	SINH (or SINH')	2-44

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

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

Function	Identifier	Page
Trigonometry *(cont.)*		
Calculate the hyperbolic cosine of a real number.	COSH (or COSH')	2-45
Calculate the hyperbolic tangent of a double precision[1] number.	DTANH (or DTANH')	2-46
Calculate the hyperbolic sine of a double precision[1] number.	DSINH (or DSINH')	2-47
Calculate the hyperbolic cosine of a double precision[1] number.	DCOSH (or DCOSH')	2-48
Calculate the hyperbolic tangent of a complex[2] number.	CTANH (or CTANH')	2-49
Calculate the hyperbolic sine of a complex[2] number.	CSINH (or CSINH')	2-50
Calculate the hyperbolic cosine of a complex[2] number.	CCOSH (or CCOSH')	2-51
Calculate the arctangent of a real number.	ATAN (or ATAN')	2-52
Calculate the arctangent of a double precision[1] number.	DATAN (or DATAN')	2-53
Calculate the arctangent of the quotient of two real numbers.	ATAN2 (or ATAN2')	2-54
Calculate the arctangent of the quotient of two double precision[1] numbers.	DATAN2	2-55
Matrices		
Invert a square matrix of real numbers.	INVERT	2-56
Invert a square matrix of double precision[1] numbers.	DINVERT	2-57
Invert a square matrix of complex[2] numbers.	CINVERT	2-58
Random Numbers		
Generate a random number, for use as a starting point for RAND.	RAND1 (or RAND1')	2-59
Generate the next element of a sequence of pseudo-random numbers.	RAND (or RAND')	2-60
Double Integer Arithmetic		
Calculate the sum of two double integer numbers.	DADD	2-61
Calculate the difference between two double integer numbers.	DSUB	2-62
Calculate the product of two double integer numbers.	DMPY (or DMPY')	2-63

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

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

Double Integer Arithmetic *(cont.)*

Complex[1] Arithmetic

Data Plotting

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

Data Plotting *(cont.)*

Function	Identifier	Page
Change the plot factor (the ratio of the plot physical size to the plot command size).	FACTOR	3-49
Return the X-axis and Y-axis coordinates of the present pen position (with respect to the current origin) and return the current plot factor.	WHERE	3-50
Write plot annotation in the form of ASCII characters and special symbols	SYMBOL	3-51

Packed Decimal Numbers[1]

Function	Identifier	Page
Calculate the product of a Packed Decimal multiplicand and a Packed Decimal multiplier.	MPYD	2-69
Calculate the quotient, or the quotient and remainder, of a Packed Decimal dividend and a Packed Decimal divisor.	DIVD or LONGDIVD	2-71 2-71
Unpack a Packed Decimal Number (numeric edit), or do an alphanumeric only edit.	R'EDIT	2-74

Miscellaneous Functions

Function	Identifier	Page
Call the FORMATTER from an SPL/3000 program.	SPL/3000 Calling Sequences	1-54
Implement the FORTRAN auxiliary I/O statements: REWIND, BACKSPACE, and ENDFILE (normally called only by FORTRAN/3000 compiler-generated code).	FTNAUX'	3-29
Request any of ten actions available for a FORTRAN Logical Unit file.	UNITCONTROL	3-32
Extract the MPE/3000 file number assigned to a given FORTRAN Logical Unit Number from the FORTRAN Logical Unit Table.	FNUM	3-34
Change the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number in the FORTRAN Logical Unit Table.	FSET	3-35
A collection of procedures called by FORTRAN/3000 compiler-generated code at run-time, to perform various functions for a user's program.	FORTRAN Run-time Procedures	3-55
Specify a user-defined procedure to process library errors.	XLIBTRAP	4-2

[1] In SPL/3000, a BYTE array.

Miscellaneous Functions *(cont.)*

SECTION I
THE FORMATTER

To find the descriptions for any given feature of the Formatter, see the Function Directory or Appendix A.

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:

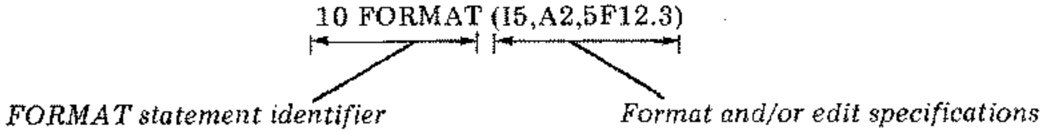
1. Convert between external ASCII numeric and/or character records and an internally represented list of variables. Formatting proceeds according to implicit parameters derived from a FORTRAN program's FORMAT statements or explicit parameters written into an SPL/3000 program.
2. Convert free-field external 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:



```
10 FORMAT (I5,A2,5F12.3)
```

FORMAT statement identifier
Format and/or edit specifications

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.

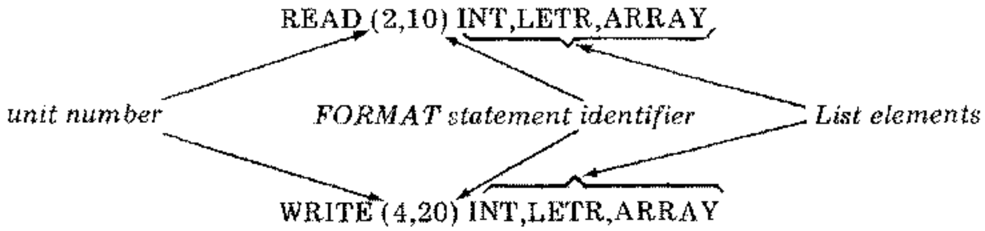
EXAMPLE:

```
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:



```
READ (2,10) INT,LETR,ARRAY
WRITE (4,20) INT,LETR,ARRAY
```

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.

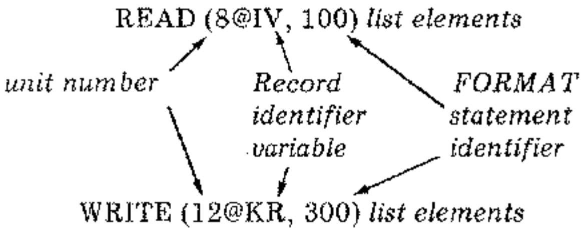
Disc Input/Output

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, double integer, real, or double precision simple variable or a constant for the record identifier.

EXAMPLES:



READ (8@IV, 100) *list elements*

WRITE (12@KR, 300) *list elements*

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. If the list elements specified require storage space greater than the record size of the file device used, the report FORMAT BEYOND RECORD occurs (see "FORMATTER ERROR REPORTS").

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 above 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

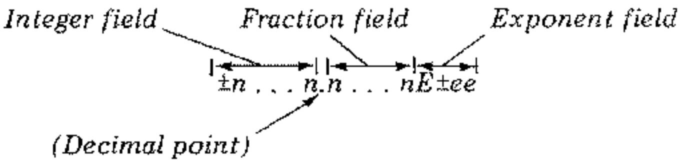
5. Either of the above items 1 and 3, in monetary (business) form

   $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 truncates a fractional input.
- Type DOUBLE PRECISION[1] rounds a fractional input.

OCTAL NUMERIC CONVERSION

One descriptor form is provided:

Ow for octal numbers 0 through 1777777777777777777777_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 22 octal digits long. Non-numeric or non-octal characters cause a conversion error.

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

HEXIDECIMAL NUMERIC CONVERSION

One descriptor form is provided:

Z*w* for hexidecimal numbers 0 through $\text{FFFFFFFFFFFFFFFF}_{16}$

where

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

This field descriptor accepts ASCII inputs up to 16 hexidecimal digits long. Non-hexidecimal characters cause a conversion error.

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 characters 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*

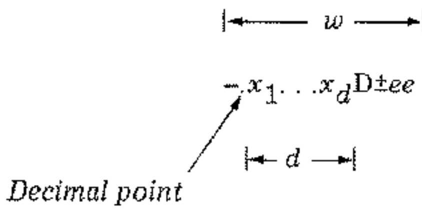
Double precision[1] numbers

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, 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.

EXAMPLES:

Descriptor	Internal Value	Output
D10.3	+12.342	ΔΔ.123D+02
D10.3	−12.341	Δ−.123D+02
D12.4	+12.340	ΔΔΔ.1234D+02
D12.4	−12.345	ΔΔ−.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	Δ.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, 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 width is w positions in the record:

$$\overbrace{-\underset{\text{Decimal point}}{.}\underbrace{x_1 \ldots x_d}_{d}\mathrm{E}\pm ee}^{w}$$

where

$x_1 \ldots x_d$ = 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] 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*

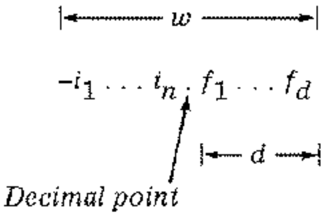
Real Numbers

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, real, or double precision[1]) 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

$i_1 \dots i_n$ = the integer digits

$f_1 \dots f_d$ = 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] In SPL/3000, type LONG real.

EXAMPLES:

Descriptor	Internal Value	Output
F10.3	+12.3402	ΔΔΔΔ12.340
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.

G*w*.*d*

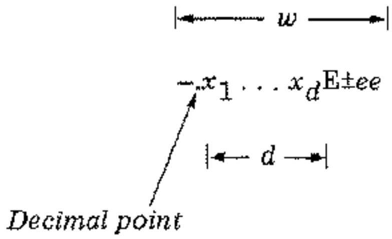
Real Numbers

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

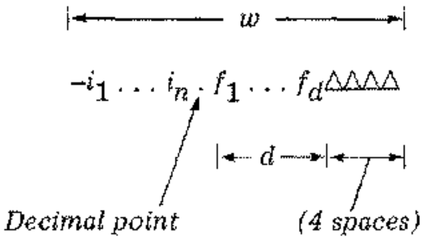
OUTPUT

On output, the G field descriptor causes output of a variable (internal representation value: integer, double integer, real, or double precision[1]) 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:



or



where

$i_1 \ldots i_n$ = the integer digits } (F*w*.*d* descriptor)
$f_1 \ldots f_d$ = the fraction digits }
$x_1 \ldots x_d$ = the most significant digits of the value (E*w*.*d* descriptor)
ee = the digits of the exponent value (E*w*.*d* descriptor)
w = the width of the external field
d = the number of fractional digits allowed in *w*
n = the number of integer digits (F*w*.*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] In SPL/3000, type LONG real.

Gw.d (cont.)

IF		$N < 0.1$	THEN Ew.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 Ew.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: Ew.d = E9.2: ΔΔ.12E+03

The field width w must follow the general rule for the Ew.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
G12.2 (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.

M*w.d*

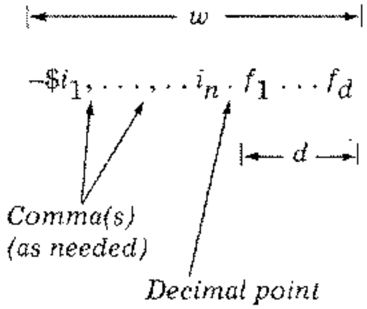
Real Numbers

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, real, or double precision[1]) 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

$i_1 \ldots\ldots i_n$ = 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] 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*

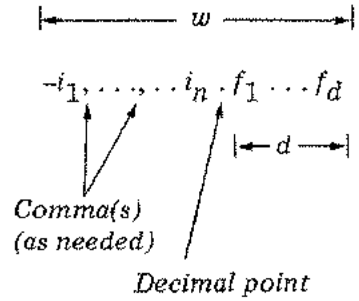
Real Numbers

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, real, or double precision[1]) 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

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

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] 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.

I*w*

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, real, or double precision[1]) 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:

$$|\!\longleftarrow w \longrightarrow\!|$$

$$-i_1 \ldots i_n$$

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] 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.

O*w*

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, or real, or double integer, or double precision[1]) in ASCII-character octal integer form, right-justified.

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

$$|\longleftarrow w \longrightarrow|$$

$$i_1 \ldots i_n$$

where

$i_1 \ . \ . \ . \ i_n$ = the octal integer digits

n = the number of significant digits
(maximums: 6 for an integer variable,
11 for a real or double integer variable,
22 for a double precision variable)

w = the width of the external field

The field width *w* can be any desired value but should be $\geqslant 6$ or $\geqslant 11$ or $\geqslant 22$, for an integer or real (or double integer) or double precision variable, respectively, 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
O8	102077	ΔΔ102077
O4	30554677321	7321
O16	56774532673	ΔΔΔΔΔ56774532673
O11	3435645327422113	45327422113

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

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) for an integer variable, or no more than 11 digits (no larger than 37777777777_8) for a real or double integer variable, or no more than 22 digits (no larger than 1777777777777777777777_8) for a double precision[1] variable, 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 the maximum number allowed by the variable using the descriptor, *w* digits are right-justified in that variable's internal representation (one or two or four words of memory).

EXAMPLES:

Descriptor	Input	Result
O6	134577	134577 or 00000134577 or 0000000000000000134577
O9	545563274	563274 or 00545563274 or 0000000000000545563274
O13	4367436521051	521051 or 67436521051 or 0000000004367436521051

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

Z*w*

Hexidecimal Integer Number

FUNCTION: Define a field for a hexidecimal integer number.

OUTPUT

On output, the Z field descriptor causes output of a variable (internal representation value: integer, or real, or double integer, or double precision[1]) in ASCII-character hexidecimal integer form, right-justified.

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

|← *w* →|

$i_1 \; . \; . \; . \; i_n$

where

$i_1 \; . \; . \; . \; i_n$ = the hexidecimal integer digits
n = the number of significant digits
(maximums: 4 for an integer variable,
8 for a real or double integer variable,
16 for a double precision variable)
w = the width of the external field

The field width *w* can be any desired value but should be ⩾ 4 or ⩾ 8 or ⩾ 16, for an integer or a real or double integer or a double precision variable, respectively, 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 hexidecimal integer, only the *w* least significant digits are output.

EXAMPLES:

Descriptor	Internal Value	Output
Z6	5AFC	ΔΔ5AFC
Z4	FCD473BE	73BE
Z12	32AB698A	ΔΔΔΔ32AB698A
Z8	9BE84893E6FF	4893E6FF

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

Z*w* (cont.)

INPUT

On input, the Z field descriptor causes interpretation of the next *w* positions in the ASCII input record as a hexidecimal 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 only of hexidecimal digits: no more than four digits (no larger than $FFFF_{16}$) for an integer variable, or no more than eight digits (no larger than $FFFFFFFF_{16}$) for a real variable, or no more than 16 digits (no larger than $FFFFFFFFFFFFFFFF_{16}$) for a double precision[1] variable, are interpreted. Any non-hexidecimal character (including a blank) anywhere in the field will produce a conversion error. If *w* is less than the maximum number allowed by the variable using the descriptor, *w* digits are right-justified in that variable's internal representation (one or two or four words of memory).

EXAMPLES:

Descriptor	Input	Result
Z4	1AD6	1AD6 or 00001AD6 or 0000000000001AD6
Z6	AB12F6	12F6 or 00AB12F6 or 0000000000AB12F6
Z10	5489BB3A6C	3A6C or 89BB3A6C or 0000005489BB3A6C

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

L*w*

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:

$$|\longleftarrow w \longrightarrow|$$

$$X_1 \ldots X_n c$$

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	$32767(77777_8)$	$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$T
L5	$+124(174_8)$	$\underline{\Delta\Delta\Delta\Delta}$F

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	△△△△TRUE	177777_8
L1	F	000000_8
L6	△FALSE	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 or real, $n = 4$; for double precision[1], $n = 8$; for character, n = the length attribute[2] 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 $\geqslant 1$. If w is $\geqslant n$, 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)	△SA
A3	SAMB	Double Integer or Real (4)	SAM
A7	JANETW	Double Precision[1] (8)	△JANETW
A10	BG	Logical or Integer (2)	△△△△△△△△BG
A4	DIXMCG	Double Precision[1] (8)	DIXM
A12	LEFTMOST	Character[2] (8)	△△△△LEFTMOST
A6	LEFTMOST	Character[2] (8)	LEFTMO

[1] In SPL/3000, type LONG real

[2] As defined in a Type statement such as CHARACTER*8 LOCALE (see *FORTRAN/3000 Reference Manual*).

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 $w \geqslant n$, 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] (8)	REALΔΔ
A4	REAL	Double Integer or Real (4)	REAL
A7	PROGRAM	Character[2] (8)	PROGRAMΔ

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

[2] As defined in a Type statement such as CHARACTER*8 LOCAL (see *FORTRAN/3000 Reference Manual*).

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 or real, $n = 4$; for double precision[1], $n = 8$; for character, $n =$ the length attribute[2] 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 $\geqslant 1$. If w is $\geqslant n$, 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 or Real (4)	AMB
R7	JANETG	Double Precision[1] (8)	ΔJANETG
R10	BG	Logical or Integer (2)	ΔΔΔΔΔΔΔΔBG
R4	DIXMCG	Double Precision (8)	XMCG
R12	RIGHTMOST	Character[2] (9)	ΔΔΔRIGHTMOST
R6	RIGHTMOST	Character[2] (9)	HTMOST

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

[2] As defined in a Type statement such as CHARACTER*8 LOCAL (see *FORTRAN/3000 Reference Manual*).

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 $w \geqslant n$, 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] (8)	*aa* REAL[2]
R4	REAL	Double Integer or Real (4)	REAL
R7	PROGRAM	Character[3] (8)	*a*PROGRAM[2]

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

[2] a = ASCII Null.

[3] As defined in a Type statement such as CHARACTER*8 LOCAL (see *FORTRAN/3000 Reference Manual*).

S

Strings of ASCII Characters

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

OUTPUT

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

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

$$\vdash\!\leftarrow l \rightarrow\!\dashv$$

$$c_1 \ldots c_n$$

where

$c_1 \ldots c_n$ = the ASCII 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
GEORGE	MY NAME IS GEORGE JONES

where the list element and length attribute are defined by the Type statement[3] CHARACTER*3 NAME or CHARACTER*6 NAME and edit specifications are

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

[1] If the variable (list element) is not type character,[2] the report STRING MISMATCH occurs (see "FORMATTER ERROR REPORTS").

[2] In SPL/3000, type byte.

[3] See *FORTRAN/3000, Reference Manual.*

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	MONDAY
SATURDAY	SATURD

where the list element and length attribute are defined by the Type statement CHARACTER*6 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 $n \leqslant 0$, 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,-2PD8.2,-2PD8.2,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.
%*n*C	Use the octal number *n* as a byte character.

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
"OUTPUT∆" "LOAD" "."	OUTPUT∆"LOAD".
"USER'S∆PROGRAM"	USER'S∆PROGRAM

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
"HEADING∆HERE"	THIS∆IS∆THE∆START	12 positions of the input are skipped.
"HEADING∆" "A" "∆."	THIS∆IS∆THE∆END∆OF	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
‘PRINT∆‘ ‘DATA’ ’.’	PRINT∆ ‘DATA’.
‘SAM’ ’S∆“SCORE” ’	SAM’S∆ “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
‘COLUMN∆HEAD’	BEGIN∆DATA∆INPUT	11 positions of the input are skipped
‘ROW∆LABEL∆‘ ‘B’ ’.’	END∆DATA∆INPUT	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. Those positions skipped are filled with ASCII blanks.

EXAMPLES:

Format/Edit Specifications	Contents of Numeric List Element(s)		Output
(E7.1,4X,"END")	34.1	*Fields:*	$\underbrace{\Delta.3E{+}02}_{7}\underbrace{\Delta\Delta\Delta\Delta}_{4}\mathrm{END}$
(F8.2,2X,I6)	5.87,436	*Fields:*	$\underbrace{\Delta\Delta\Delta\Delta5.87}_{8}\underbrace{\Delta\Delta}_{2}\underbrace{\Delta\Delta\Delta436}_{6}$

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)	Δ.25E+02END$1,563.79	.25E+02, 1563.79
(5X,E9.2,I5)	54321–98.7563814581	–.9876538E+02, 14581

T*n*

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.

%*n*C
Single Character

FUNCTION: To use an octal number *n* in the range 0 - 377 as a byte character. The primary purpose is to represent a carriage control character, especially where a particular number does not represent a printing ASCII character.

OUTPUT

On output, the %*n*C edit descriptor causes the character in the edit specification to be transmitted to the external record. If the character is in the first position of a record to a device using carriage control, the character will be used as a carriage control character.

If the edit descriptor has not been referenced by a READ statement (See "INPUT"), the original character is transmitted.

If the edit descriptor has been referenced by a READ statement, the character which was last read is transmitted.

EXAMPLES:

Edit Descriptor	Action Taken If Carriage Control Character
%306C	Space 1/4 page
%301C	Skip to bottom of the form
%53C	Suppress line advance (equivalent to "+")

INPUT

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

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

```
| Δ.123E+02|-30303 |
```

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

```
| Δ.123E+02|ΔΔΔΔ|-30303 |
```

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

```
| Δ.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:

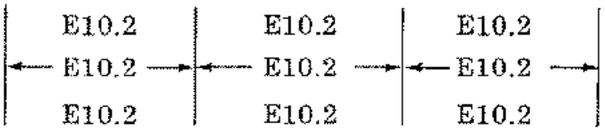
1. The current record is terminated: on output, the current field is completed, then the record is transmitted; on input, the rest of the record is ignored.
2. A new record is started.
3. Format control (field descriptor interpretation) is returned to the repeat specification for the rightmost specification 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:

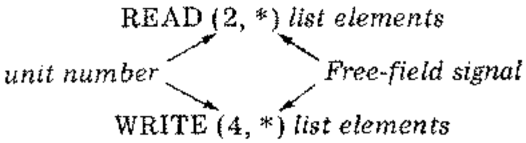
HEADER



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.

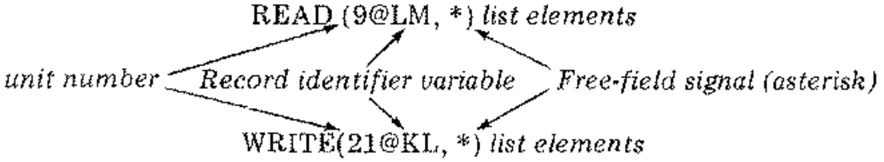
For free-field input/output, FORTRAN READ or WRITE statements are written with an asterisk instead of a FORMAT statement identifier:



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:



READ (9@LM, *) *list elements*

unit number *Record identifier variable* *Free-field signal (asterisk)*

WRITE(21@KL, *) *list elements*

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)
". . ."	An "enclosed" character string, in quotation marks; to be input only to a FORTRAN/3000 type character variable (or SPL/3000 type byte array)
'. . .'	An "enclosed" character string, in apostrophes; to be input only to a FORTRAN/3000 type character variable (or SPL/3000 type byte array)
. . .	A "non-enclosed" character string; to be input only to a FORTRAN/3000 type character variable (or SPL/3000 type byte array)
<<. . .>>	A "comment" 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, floating-point (real), double-precision floating point[1], and character string. Numeric data

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

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.

A character string item, however, must be input only to a character string list element; if not, the report STRING MISMATCH occurs (see "FORMATTER ERROR REPORTS") and the user's program 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: 22)

% is the octal data identifier

Non-octal digits are delimiters. The largest number allowed is 1777777777777777777777_8. If n is greater than 21, the first (most significant) digit must be 0 or 1.

CHARACTER STRING DATA

An "enclosed" character string data item is any series of ASCII characters, including blank spaces, enclosed either in quotation marks or in apostrophes. Any one or more of the characters enclosed in quotation marks can be a quotation mark if signalled by an adjacent quotation mark; any one or more of the characters enclosed in apostrophes can be an apostrophe if signalled by an adjacent apostrophe:

"SETS " "UNIT" " VALUE"	transmits SETS "UNIT" VALUE
'CLEARS ' 'OPT' ' VALUE'	transmits CLEARS 'OPT' VALUE

A "non-enclosed" character string data item is any series of ASCII characters that does not begin with:

a comma, or
a quotation mark, or
two consecutive left symbols <<, or
a blank, or
an apostrophe, or

does not contain a slash /. Otherwise, any ASCII characters are permitted. Such a data item ends with:

an end-of-record condition, or
when *n* characters have been transmitted (*n* is the length attribute of the list element), or
a slash / (record terminator).

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 or apostrophe of an "enclosed" 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 an "enclosed" 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.

INPUT EXAMPLES

Given the READ statement READ(5,*)STR,I where STR is declared CHARACTER*10 STR and I is declared INTEGER I:

External Input	STR Result	I Result
"ABC"	ABCΔΔΔΔΔΔΔ	unchanged
'ABC"DE'	ABC"DEΔΔΔΔ	unchanged
ABC*cr* (*cr* = carriage return)	ABCΔΔΔΔΔΔΔ	unchanged
<<COMMENT>>ABC123	ABC123ΔΔΔΔ	unchanged
"1234567890123"	1234567890	unchanged
1234567890123	1234567890	123
YES 256	YESΔ256ΔΔΔ	unchanged*
"YES" 256	YESΔΔΔΔΔΔΔ	256
YES 256	YESΔΔΔΔΔΔΔ	256
ABC/ 6	ABCΔΔΔΔΔΔΔ	6

*This may not be the desired result, the next example (above) shows a recommended method.

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, floating-point (real), double precision floating-point,[1] 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 G22.16 field description.
5. Character string data items are output under the S field description; all characters are transmitted without modification, including blanks, quotation marks and apostrophes.

[1] 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 new record is started with the first character of the data item.

If a character string data item is too long for the current record, the string will continue to be written, but onto the next record. No record terminator is output.

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(5,*) and WRITE(6,*), where 5 is typically the FORTRAN logical unit number of the MPE/3000 standard input file $STDIN, and 6 is typically the FORTRAN logical unit number of the MPE/3000 standard output file $STDLIST, and * (asterisk) 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 a terminal (such as a teleprinter or a CRT keyboard/display); if the device is such a terminal, the ACCEPT routine prints a carriage return, a line feed then 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; the report BUFFER OVERFLOW occurs (see "FORMATTER ERROR REPORTS").

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, BUFFER OVERFLOW occurs (see "FORMATTER ERROR REPORTS").

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:

READ (8) *list elements*

unit number

WRITE (12) *list elements*

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

EXAMPLES:

READ (8@IV) *list elements*

unit number *Record identifier variable*

WRITE (12@KR) *list elements*

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, the report DIRECT ACCESS OVERFLOW occurs (see "FORMATTER ERROR REPORTS").

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.

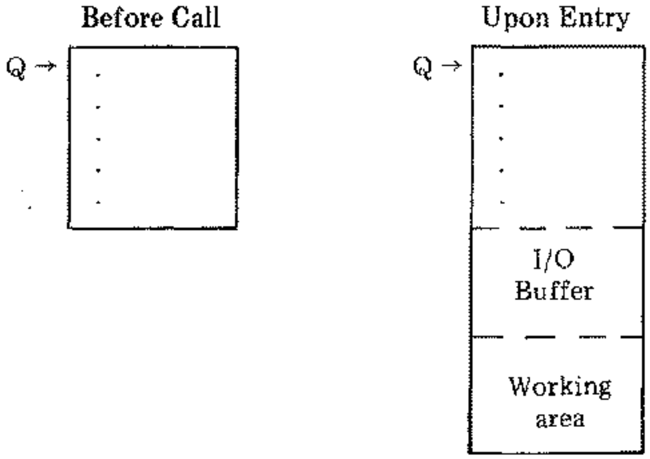
> *NOTE: Under binary /direct access, the Formatter begins each new list element output at a word boundary. If the list element is, for example, a byte array of an odd number of bytes, one byte of the record will not be used.*

SPL/3000 CALLING SEQUENCES

NOTE: The following descriptions assume the reader has a working knowledge of SPL/3000; see the Systems Programming Language Reference Manual.

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 (a positive integer), 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 Formatter Error Report routine for end-of-file errors; set to not call.
10	Clear to call Formatter Error Report routine for irrecoverable file errors; set to not call.
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;
           .
           .
```

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

```
PROCEDURE DIO' (LOC);
  DOUBLE LOC;
    OPTION EXTERNAL;
           .
           .
```

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

```
PROCEDURE RIO' (LOC);
  REAL LOC;
    OPTION EXTERNAL;
           .
           .
```

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

```
PROCEDURE LIO' (LOC);
  LONG LOC;
    OPTION EXTERNAL;
           .
           .
```

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

```
PROCEDURE SIO' (SLEN, LOC);                 For an ASCII character string.
  VALUE SLEN;
  INTEGER SLEN; BYTE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE AIIO' (DIM, LOC);                 For an array of the same types as IIO'.
  VALUE DIM; INTEGER DIM;
  INTEGER ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ADIO' (DIM, LOC);                 For an array of the same types as DIO'.
  VALUE DIM; INTEGER DIM;
  DOUBLE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ARIO' (DIM, LOC);                 For an array of the same types as RIO'.
  VALUE DIM; INTEGER DIM;
  REAL ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ALIO' (DIM, LOC);                 For an array of the same types as LIO'.
  VALUE DIM; INTEGER DIM;
  LONG ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
PROCEDURE ASIO' (SLEN, DIM, LOC);           For an array of ASCII character strings.
  VALUE SLEN, DIM;
  INTEGER SLEN, DIM;
  BYTE ARRAY LOC;
    OPTION EXTERNAL;
          .
          .
```

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,%34,@LAST);
2	X(0) := A + B;
3	X(1) := C/D;
4	ARIO'(2,X);
5	IIO'(I);
6	TFORM';
7	LAST:
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
- Not call Formatter Error Report routine for end-of-file errors
- Call Formatter Error Report routine 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.

Statement 8 is a user-decision to check for end-of-tape error.

File System Requirements

NOTE: The following descriptions assume the reader has a working knowledge of MPE/3000; see the MPE Intrinsics Reference Manual.

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 word address of the FLUT:

DB-1 →	I_{fa}

where

I_{fa} is a positive integer to specify the FLUT word 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 in the right byte, 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 or DISPLAY, one or the other (or both) of two U entries must be included in the FLUT: 5 for $STDIN and 6 for $STDLIST. For full details of these standard file names, including the ability to equate FORTRAN file names FTN05 and FTN06 to other file names, see the *MPE Intrinsics Reference Manual.*

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, the Formatter Error Report FILE NOT IN TABLE FOR UNIT # *xx* occurs and the user's program is aborted. 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.

formaldesignator FTN*dd*, where *dd* is the UNIT number in the FLUT (for example, FTN03).

foptions

Bit(s)	Field Name and Setting(s)
14:2	Domain: 00, this is a new file.
13:1	ASCII/BINARY: 0, this is a BINARY file[1].
10:3	Default File Designator: 000, the default file designator is the same as the formal file designator[2].
8:2	Record Format: 00, fixed-length records for direct-access; or 01, variable-length records for sequential-access.
7:1	Carriage Control: 0, no carriage control character expected[3].
6:1	(Reserved for MPE/3000 system use.)
5:1	Dissallow File Equation: 0, allow :FILE commands.
0:5	(Reserved for MPE/3000 system use.)

aoptions

Bit(s)	Field Name and Setting(s)
12:4	Access Type: 0100, input/output access.
11:1	Multirecord: 0, non-multirecord mode.
10:1	Dynamic Locking: 0, disallow dynamic locking/unlocking.
8:2	Exclusive: 00, default value related to Access Type aoption.
7:1	Inhibit Buffering: 0, allow normal buffering.
0:7	(Reserved for MPE/3000 system use.)

[1] Except for FTN05 or FTN06: 1, this is an ASCII file.
[2] Except for FTN05: 100, for $STDIN; or for FTN06: 001, for $STDLIST.
[3] Except for FTN06: 1, carriage control character expected.

recsize	System default value.
device	System default: DISC.
formmsg	None.
userlabels	System default value: 0.
blockfactor	System default value.
numbuffers	System default value: 2.
filesize	System default value: 1023.
numextents	System default value: 8.
initalloc	System default value: 1.
filecode	System default value: 0.

ACCEPT/DISPLAY OPTION

The ACCEPT/DISPLAY option is assumed to be used with a 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 device is not a terminal (the device cannot output information), it only reads inputs; it does not print a carriage return, line feed then a prompt character ? before it reads inputs.

FORMATTER ERROR REPORTS

Errors detected during Formatter execution call the error procedure FMTERROR' in the Compiler Library. The error is analyzed and control either remains in FMTERROR', to print reports then abort the user's program, or is passed to a user-defined error procedure (see Section IV, Library Errors). Reports by FMTERROR' are printed on the standard list device $STDLIST; starting with an error identifying line and continuing with further information as described below. The reports end with a "stack trace-back report" that uses the format specified by the MPE Stack Dump facility. For further information, see *MPE Debug/Stack Dump Reference Manual* (HP Part No. 30000-90012).

The Formatter error reports are:

BAD INPUT CHARACTER

This message is followed by a portion of the input buffer that includes the bad character, then a caret ^, positioned under that bad character.

BUFFER OVERFLOW

Occurs only in a Core-To-Core Conversion that transfers data to a user-defined buffer. This message is followed by a portion of the input buffer that includes the character at which overflow was detected, then a caret ^, positioned under that first overflow character.

DIRECT ACCESS OVERFLOW

Occurs only in an Unformatted (Binary) Transfer to a file on a direct access device, when the storage required by all the list elements exceeds the file record size. If the file is a FORTRAN/3000 logical unit, ON UNIT #*xx* is appended to the above message. If the file is a user-defined MPE/3000 file, a File Information Display is printed (a sample appears at the end of this section).

END OF FILE DETECTED

The file system returned CCA = CCG. If the file is a FORTRAN/3000 logical unit, ON UNIT #*xx* is appended to the above message. In any case, a File Information Display is printed (a sample appears at the end of this section).

NOTE: *The user can choose to handle this error another way:*

SPL/3000 user: to ignore the error, set bit 11 of parameter IOTYPE for procedure FMTINIT' (see "SPL/3000 CALLING SEQUENCE").

FORTRAN/3000 user: to transfer program control to another statement, include END = label *in the READ statement or the WRITE statement.*

FILE NOT IN TABLE FOR UNIT #*xx*

The FORTRAN/3000 logical unit accessed has no corresponding entry in the FLUT (see "File System Requirements").

FILE SYSTEM ERROR

The file system returned CCA = CCL. If the file is a FORTRAN/3000 logical unit, ON UNIT #*xx* is appended to the above message. In any case, a File Information Display is printed (a sample appears at the end of this section).

NOTE: The user can choose to handle this error in another way:

SPL/3000 user: to ignore the error, set bit 10 of parameter IOTYPE for procedure FMTINIT' (see "SPL/3000 CALLING SEQUENCES").

FORTRAN/3000 user: to transfer program control to another statement, include ERR = label *in the READ statement or the WRITE statement.*

FORMAT BEYOND RECORD

The number of characters required by the list elements exceeds the record length of the device to be used. For example, if the device is a line printer limited to 132 characters per line (per record), the number of characters required by the list elements is more than 132. This message is followed by a portion of the format statement that includes the specification related to the error, then a caret ^, positioned under that invalid specification.

ILLEGAL FORMAT CHARACTER

This message is followed by a portion of the format statement that includes the illegal character, then a caret ^, positioned under that illegal character.

INVALID FILE NUMBER FOR UNIT #*xx*

Procedure FSET (see the Function Directory) was called with the parameter NEWFILE set to a value outside the range [1,254].

NESTING TOO DEEP

This message is followed by a portion of the format statement that includes the start of the illegally nested group (see "Nesting"), then a caret ^, positioned under the start of that illegally nested group.

NUMBER OUT OF RANGE

A value in the input buffer is too small or too large for the ranges representable by the corresponding list element type (see "Introduction"). This message is followed by a portion of the input buffer that includes the invalid number, then a caret ^, positioned under the last character of that invalid number.

STRING MISMATCH

A character data item is not directed to a FORTRAN/3000 type CHARACTER (or an SPL/3000 type BYTE ARRAY) list element. This message is followed by a portion of the input buffer that includes the misdirected character data item, then a caret ^, positioned under the start of that character data item (usually a quotation mark or an apostrophe).

NOTE: *This error is detected for Free-Field Input or ACCEPT only if the character data item is "enclosed" (in quotation marks or apostrophes). Otherwise, any group (one or more) of non-numeric characters is treated as a delimiter for a numeric data item; any group of numeric characters is transmitted as a numeric data item.*

UNDEFINED OPTION ON UNIT #*xx*

Procedure UNITCONTROL or FTNAUX' (see the Function Directory) was called with the parameter OPT set to a value outside the range [-1,8].

File Information Display

As described in the preceding text, certain Formatter Error Reports are followed by a printed File Information Display. Either one of two possible formats is used:

If access to the MPE/3000 file referenced is blocked, or if that file is undefined in the MPE/3000 file system in use:

```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!  FILE  NUMBER #      IS UNDEFINED.             !
!  ERROR NUMBER: 56     RESIDUE: 0               !
!  BLOCK NUMBER: 0             NUMREC: 0         !
+------------------------------------------------+
```

For a file that returned either a CCA = CCG (end-of-file error) or a CCA = CCL (irrecoverable file error):

```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!  FILE NAME IS FTN05     .                      !
!  FOPTIONS: NEW,A,  $STDIN,U,N,SL,FEQ           !
!  AOPTIONS:   INPUT,SREC,NOLOCK,DEF,NOBUFF      !
!  DEVICE TYPE: 16  LU: 17   DRT: 18   UNIT: 6   !
!  RECORD SIZE: 72     BLOCK SIZE: 72    (BYTES) !
!  EXTENT SIZE: 0      MAX EXTENTS: 0            !
!  RECPTR: 0           RECLIMIT: 0               !
!  LOGCOUNT: 0            PHYSCOUNT: 0           !
!  EOF AT: 0           LABEL ADDR: %02100000000  !
!  FILE CODE: 0      ID IS          ULABELS: 0   !
!  PHYSICAL STATUS: 0001000100000000             !
!  ERROR NUMBER: 0     RESIDUE: 0                !
!  BLOCK NUMBER: 0             NUMREC: 1         !
+------------------------------------------------+
```

The contents of either display are explained in Section III of this manual, under "PRINTFILEINFO."

SECTION II
MATHEMATICAL PROCEDURES

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

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)

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

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

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.

JSIGN'

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

Declaration:

```
INTEGER PROCEDURE JSIGN' (J,K),
   VALUE J,K;INTEGER J,K;
      OPTION EXTERNAL;
                 .
                 .
                 .
```

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

ATTRIBUTES:

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

Result: A double integer number.

FORTRAN: Intrinsic Function: JSIGN (*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'

FUNCTION: Truncate a real number to an integer number.

Declaration:

```
INTEGER PROCEDURE INT'(Y);
   VALUE Y; REAL Y;
      OPTION EXTERNAL;
                .
                .
                .
```

Method: INT'(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*).

Error: If the real number is outside the range stated, the arithmetic trap 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 $\leqslant |Y|$

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 $\leqslant |Y|$

ATTRIBUTES:

Parameter: A double precision number.

Result: A double precision number.

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

Error: None.

DFIX (or DFIX')

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

Declaration:

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

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

ATTRIBUTES:

Parameter: A LONG real number.

Result: A double integer number.

FORTRAN: Callable as an external function:

X = DFIX (\Y\\)

or through use of a SYSTEM INTRINSIC statement as:

X = DFIX (*Y*)

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

DFLOAT (or DFLOAT')

FUNCTION: Convert a double integer number to a double precision (LONG real) number.

Declaration:

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

ATTRIBUTES:

Parameter: A double integer number.

Result: A LONG real number.

FORTRAN: Callable as an external function:

X = DFLOAT (\Y\\)

or through use of the SYSTEM INTRINSIC statement as:

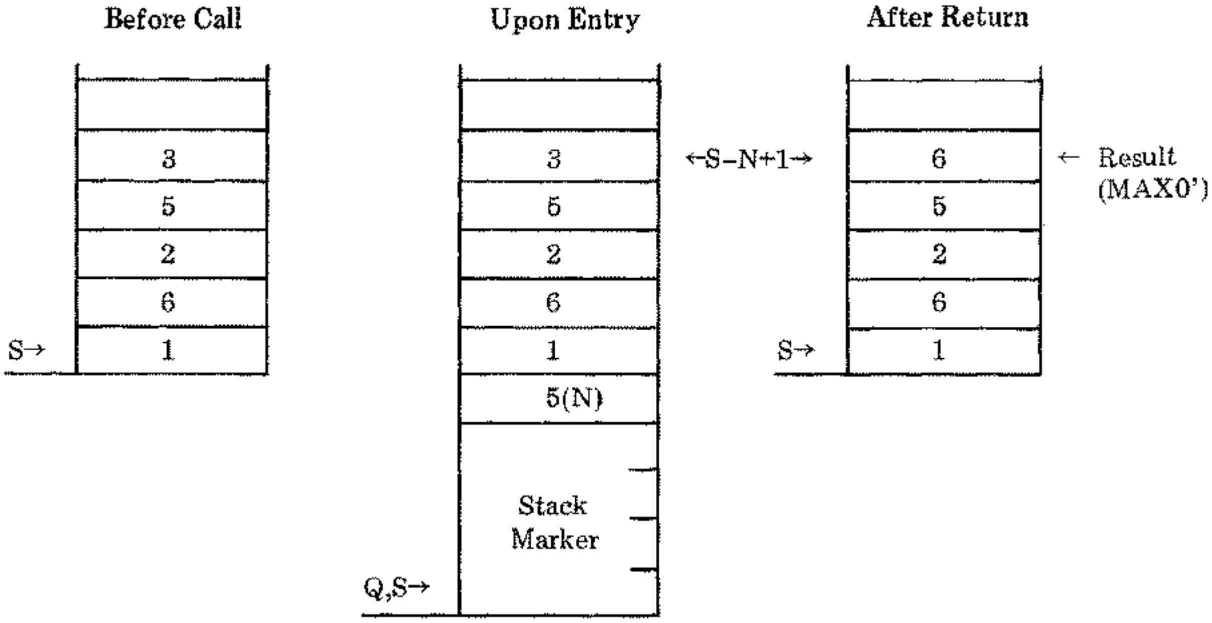
X = DFLOAT (*Y*)

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*,. . .).

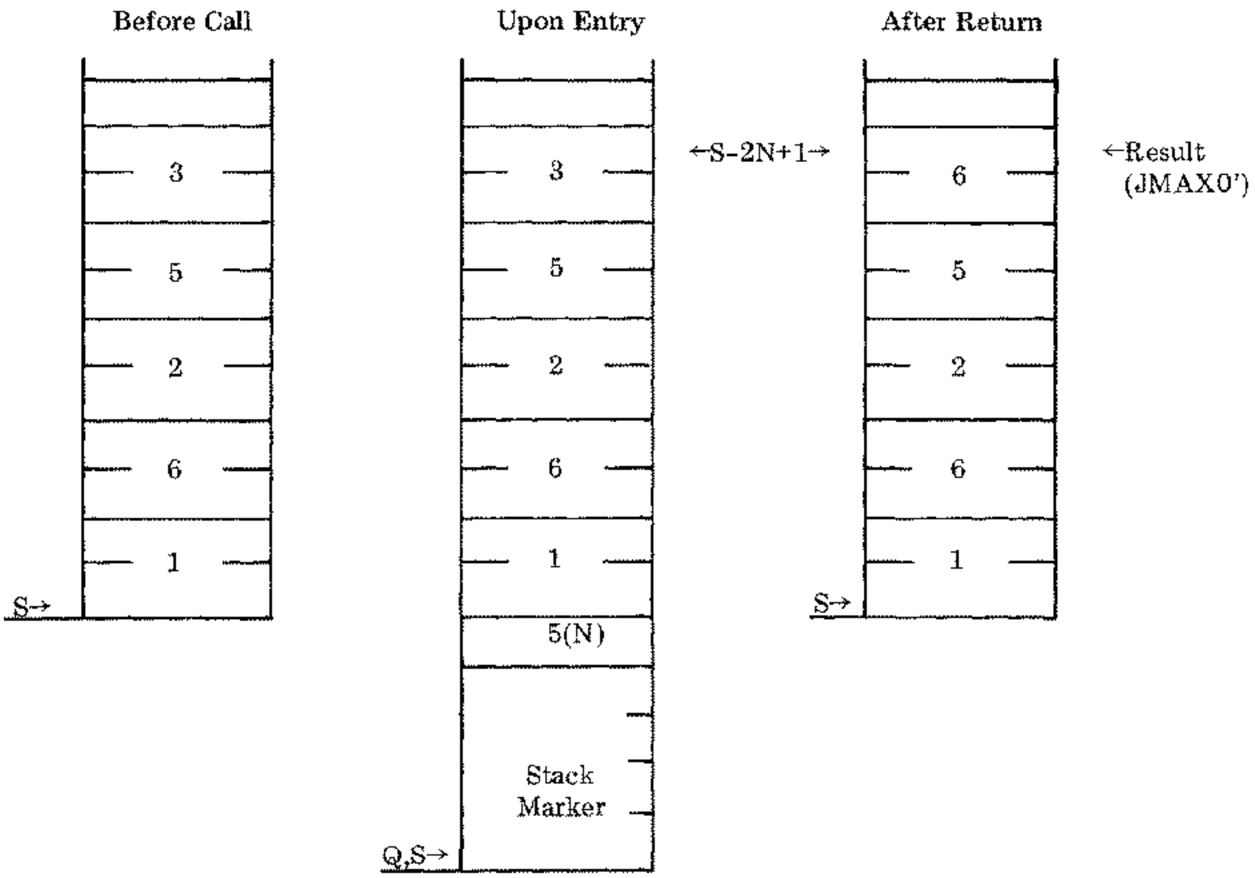
Error: If the argument (or number of parameters) is less than 2, no action occurs.

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

FUNCTION: Calculate the largest (JMAX0') or smallest (JMIN0') of N double integers on top-of-stack and return that double integer in S–2N+1.

Declaration:

```
PROCEDURE JMAX0'(N); or JMIN0'(N);
   VALUE N; INTEGER N;
      OPTION EXTERNAL;
           .
           .
           .
```



ATTRIBUTES:

Parameter: A single precision integer number ⩾ 2.

Result: A double integer number.

FORTRAN: Intrinsic Function: JMAX0 (*A*,*B*,*C*,...) or JMIN0 (*A*,*B*,*C*,...).

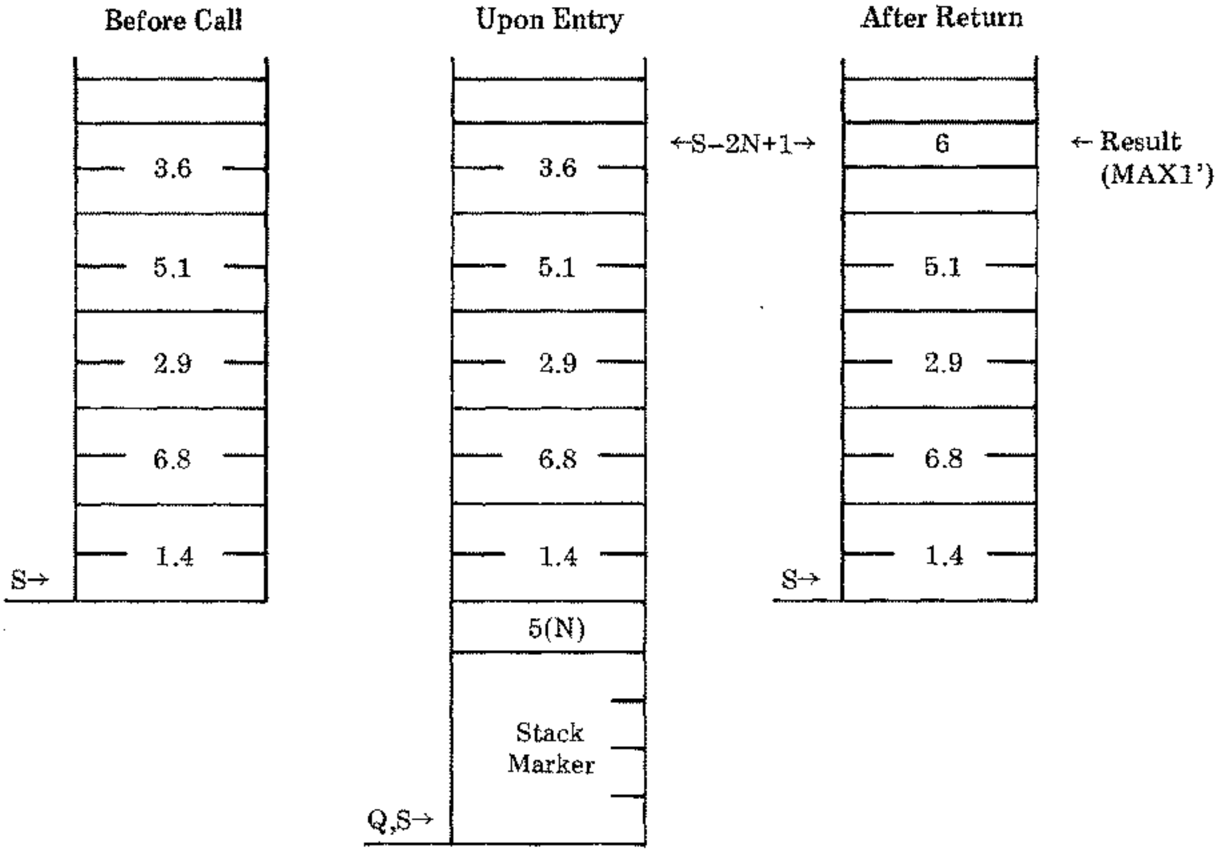
Error: If the argument (or number of parameters) is less than 2, no action occurs.

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

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:

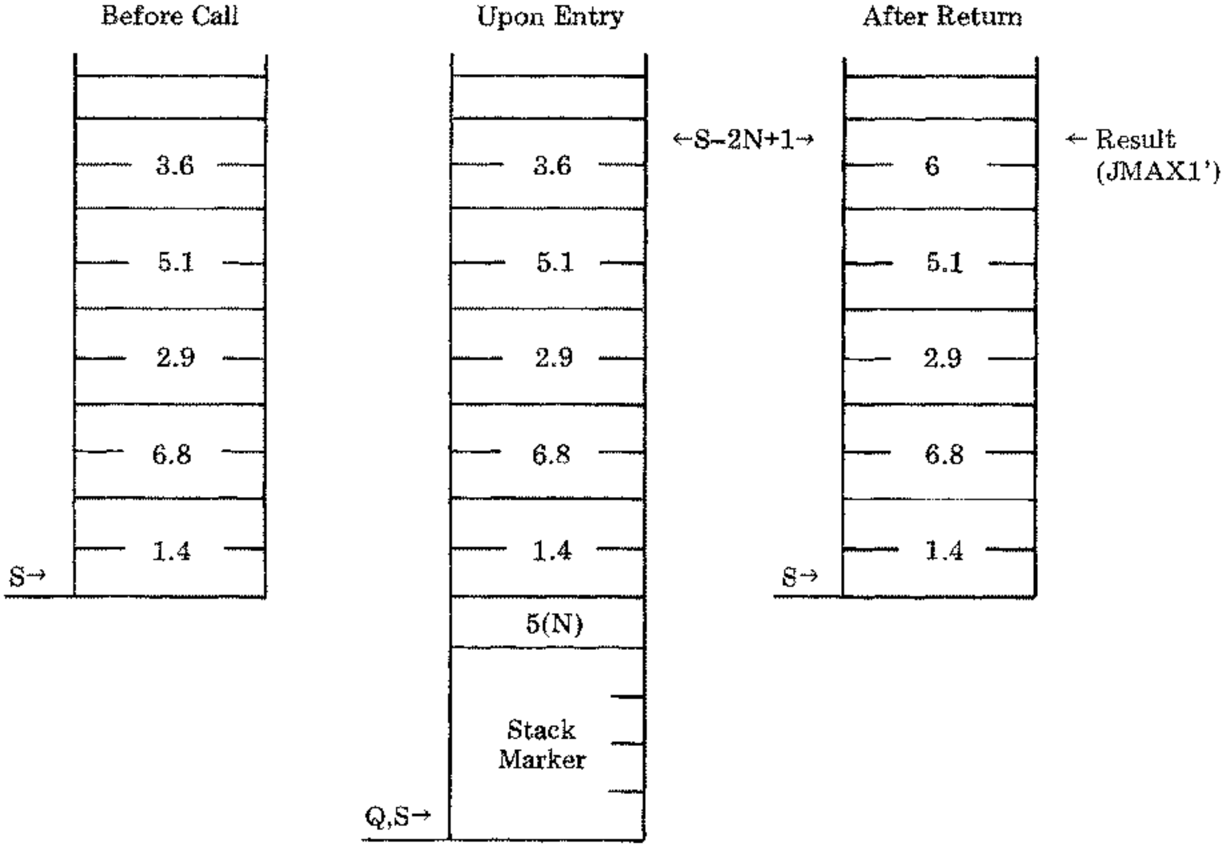
1. If the argument (or number of parameters) is less than 2, no action occurs.
2. If the largest (or smallest) real number is outside the range [–32768.0, 32767.0], the arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).
3. The SPL/3000 caller must cut the stack back after return (for example, use an ASSEMBLE (SUBS 9); statement).

JMAX1'/JMIN1'

FUNCTION: Calculate the largest (JMAX1') or smallest (JMIN1') of N real numbers on top-of-stack and return the double integer of that number in S-2N+1.

Declaration:

```
PROCEDURE JMAX1'(N); or JMIN1'(N);
  VALUE N; INTEGER N;
    OPTIONAL EXTERNAL;
              .
              .
              .
```



ATTRIBUTES:

Parameter: A single precision integer number ≥ 2.

Result: A double integer number.

FORTRAN: Intrinsic Function: JMAX1 (*A*,*B*,*C*,...) or JMIN1 (*A*,*B*,*C*,...).

Error: See "Comments."

COMMENTS:

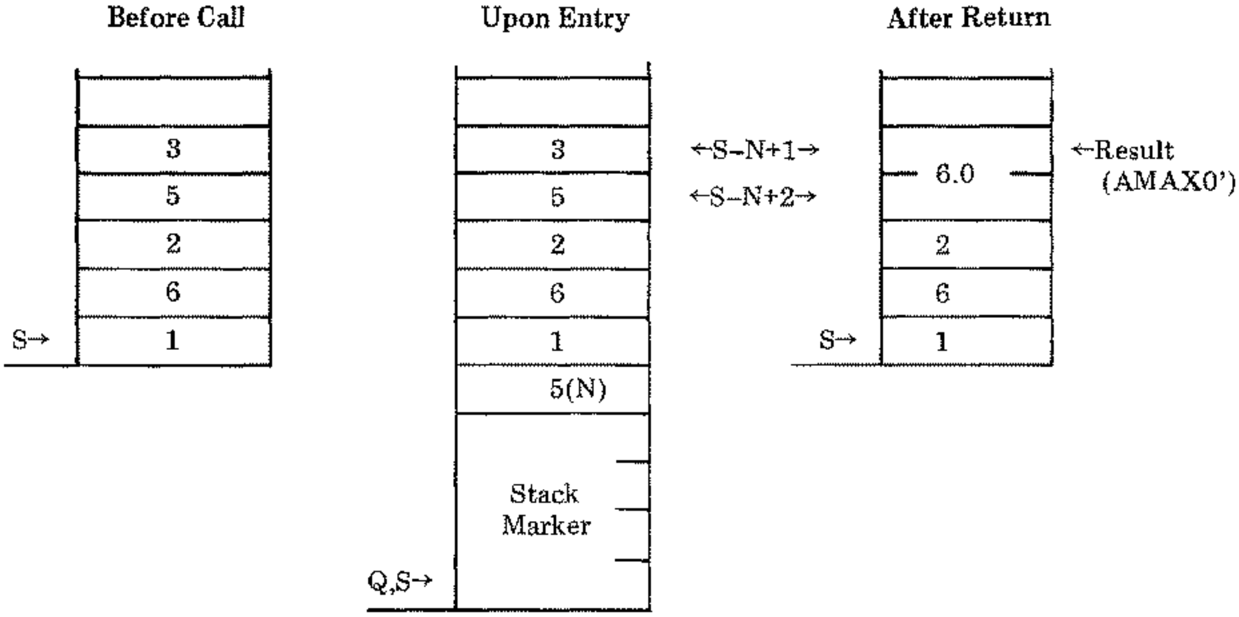
1. If the argument (or number of parameters) is less than 2, no action occurs.
2. If the largest (or smallest) real number is outside the range [-2147483648, 2147483647], the arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).
3. The SPL/3000 caller must cut the stack back after return (for example, use an ASSEMBLE (SUBS 8); 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 ⩾ 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.

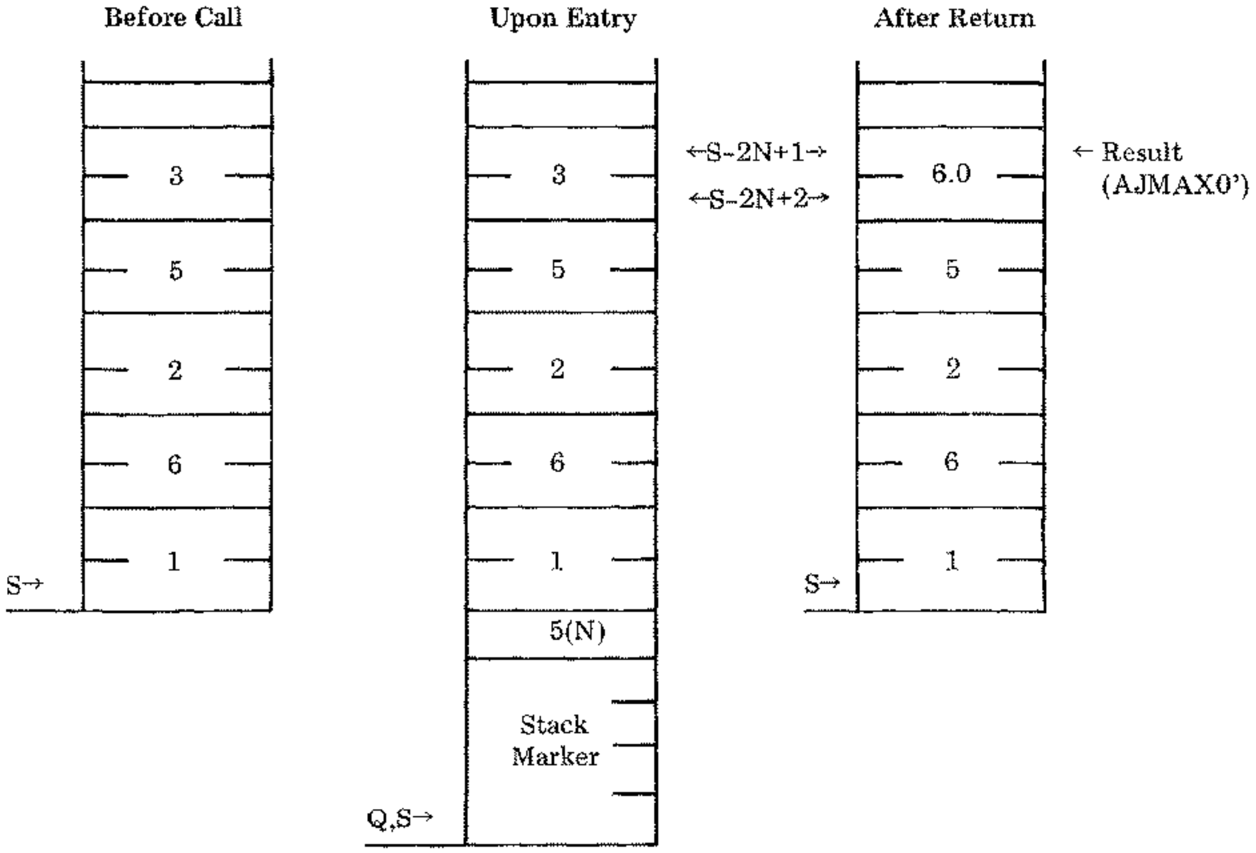
COMMENT: The SPL/3000 caller must cut the stack back after return (for example, use an ASSEMBLE (SUBS 3); statement).

AJMAX0'/AJMIN0'

FUNCTION: Calculate the largest (AJMAX0') or smallest (AJMIN0') of N double integers on top-of-stack and return that double integer in S-2N+1 and S-2N+2 in real representation.

Declaration:

```
PROCEDURE AJMAX0'(N); or AJMIN0'(N);
   VALUE N; INTEGER N;
      OPTION EXTERNAL;
                   .
                   .
                   .
```



ATTRIBUTES:

Parameter: A single precision integer number ⩾ 2.

Result: A real number.

FORTRAN: Intrinsic Function: AJMAX0 (*A*,*B*,*C*,...) or AJMIN0 (*A*,*B*,*C*,...).

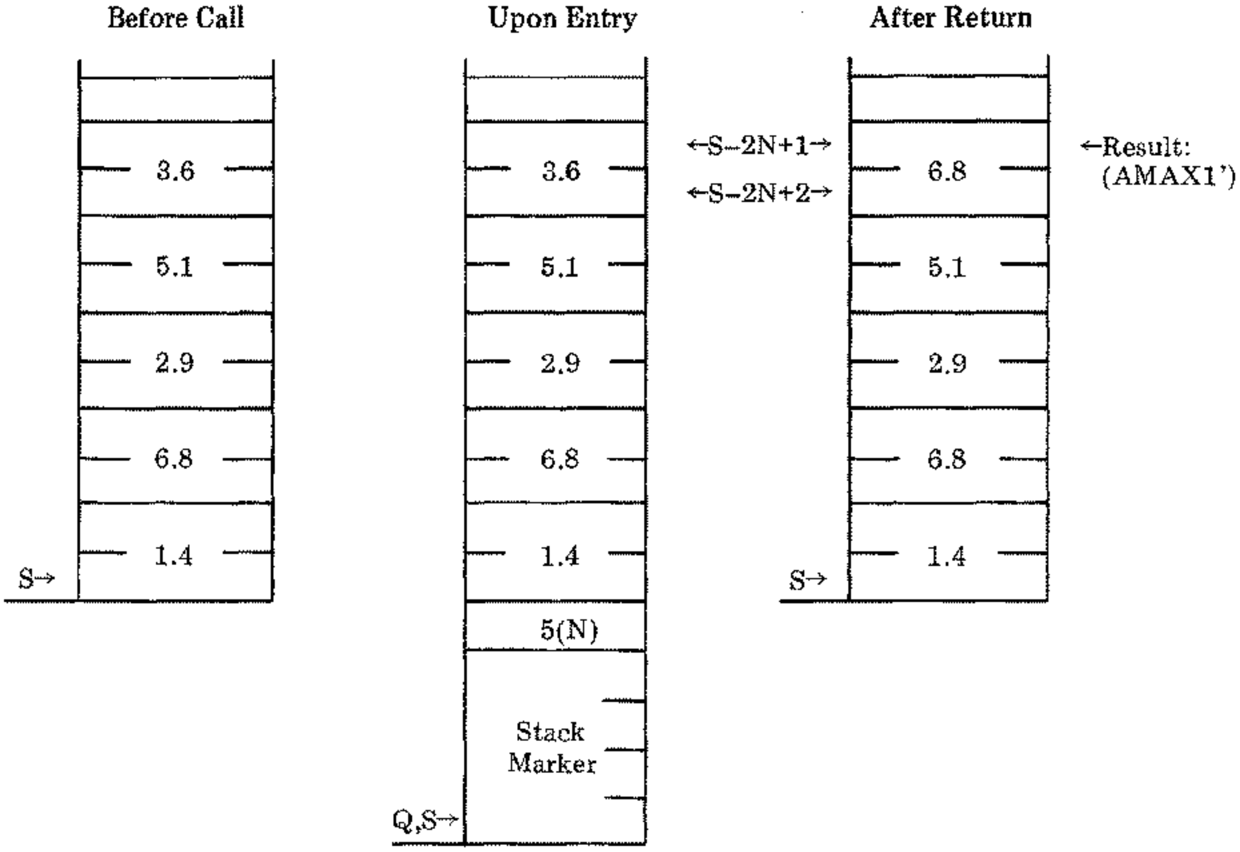
Error: If the argument (or number of parameters) is less than 2, no action occurs.

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

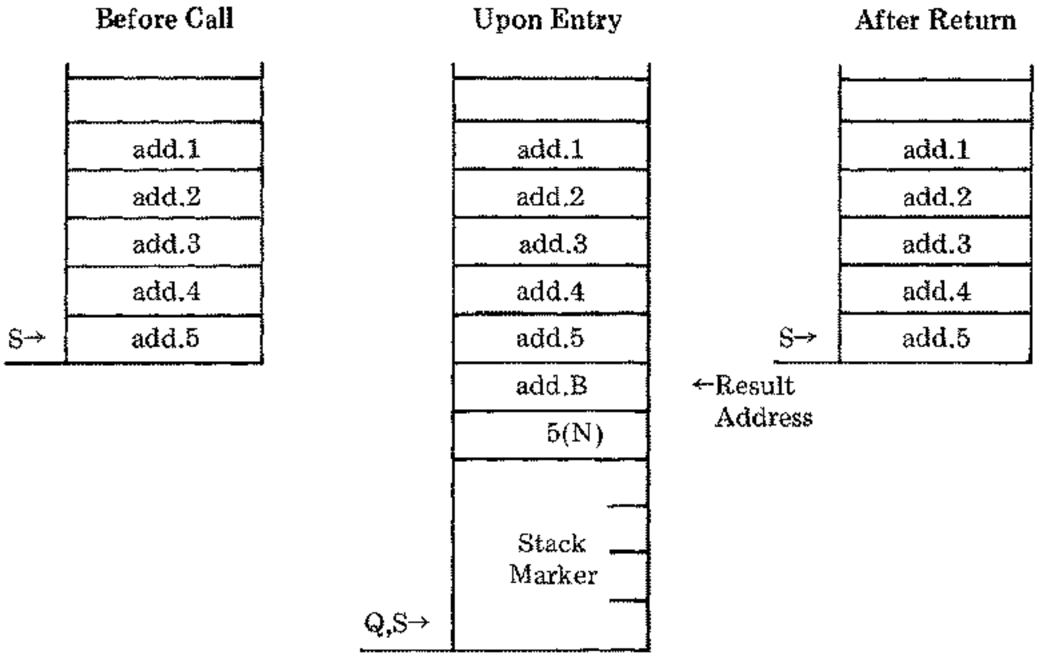
COMMENT: The SPL/3000 caller must cut back the stack after return (for example, use an ASSEMBLE (SUBS 8); statement).

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 FLOATING POINT OVERFLOW, FLOATING POINT UNDERFLOW, or 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 EXTENDED PRECISION OVERFLOW, EXTENDED PRECISION UNDERFLOW, or EXTENDED PRECISION 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 [-176.7525,176.7525].

Result: A representable[1] positive real number.

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

Error: If the argument is ⩾ 176.7526, the result cannot be represented and SOFTERROR' message EXP: OVERFLOW occurs (see "Library Errors"). If the argument ⩽ -176.7526, 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"):

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

ATTRIBUTES:

Parameter: A representable[1] double precision number in the range [-176.75253104, 176.75253104].

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

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

Error: If the argument is ⩾ 176.75253105, the result cannot be represented and the SOFTERROR' message DEXP: OVERFLOW occurs (see "Library Errors"). If the argument is ⩽ -176.75253105, the result is set to zero.

[1] See "Introduction."

CEXP (or 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); or 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 (or DSQRT')

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

Declaration:

```
LONG PROCEDURE DSQRT (Y); or DSQRT' (Y);
  LONG Y;
    OPTION EXTERNAL;
             .
             .
             .
```

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

Accuracy: (See "Introduction"):

when $\left|\frac{x - y}{f}\right| \sim \epsilon$, 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 (or 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); or 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 $|a| \geqslant |b|$ 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 $a \geqslant 0$ 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;
                 .
                 .
                 .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

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

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;
          .
          .
          .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

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

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 (or 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); or 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 (or TAN')

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

Declaration:

```
REAL PROCEDURE TAN (Y); or TAN' (Y);
   REAL Y;
      OPTION EXTERNAL;
                .
                .
                .
```

Method: A minimax approximation.

Accuracy: (See "Introduction"):

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

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

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

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 (or DTAN')

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

Declaration:

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

Method: A minimax approximation.

Accuracy: (See "Introduction"):

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

ATTRIBUTES:

Parameter: A double precision number in radians.

Result: A double precision number.

FORTRAN: Basic External Function: DTAN (*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 (or DSIN')

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

Declaration:

```
LONG PROCEDURE DSIN (Y); or 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: DSIN (*Y*).

Error: None.

[1] See "Introduction."

DCOS (or DCOS')

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

Declaration:

```
LONG PROCEDURE DCOS (Y); or 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."

CTAN (or CTAN')

FUNCTION: Calculate the tangent 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 CTAN(Y); or CTAN'(Y);
  REAL ARRAY Y;
    OPTION EXTERNAL;
              .
              .
              .
```

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

Method: $\mathrm{CTAN(Y)} = \dfrac{\sin(\mathrm{Y})}{\cos(\mathrm{Y})}$

Accuracy: Depends on accuracy of CSIN, and CCOS.

ATTRIBUTES:

Parameter: A complex number.

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

FORTRAN: Basic External Function: CTAN(Y).

Error: Underflow, overflow, divide by zero; see DIVCVVR.

Sample SPL/3000 calling sequence:

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

CSIN (or CSIN')

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); or 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 (or 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); or 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: $\text{CCOS}(a + bi) = \cos(a)\cosh(b) - i\sin(a)\sinh(b)$
$Y(0) = a$ (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 (or TANH')

FUNCTION: Calculate the hyperbolic tangent of a real number.

Declaration:

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

Method: $\text{TANH (Y)} = \dfrac{e^{Y} - e^{-Y}}{e^{Y} + e^{-Y}}$ 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 (or SINH')

FUNCTION: Calculate the hyperbolic sine of a real number.

Declaration:

```
REAL PROCEDURE SINH (Y); or 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 (or COSH')

FUNCTION: Calculate the hyperbolic cosine of a real number.

Declaration:

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

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

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.

DTANH (or DTANH')

FUNCTION: Calculate the hyperbolic tangent of a double precision number.

Declaration:

```
LONG PROCEDURE DTANH(Y); or DTANH'(Y);
  LONG Y;
    OPTION EXTERNAL;
                .
                .
                .
```

Method: $\text{DTANH(Y)} = \dfrac{\sinh(Y)}{\cosh(Y)}$

Accuracy: (See "Introduction")

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

ATTRIBUTES:

Parameter: A double precision number.

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

FORTRAN: Basic External Function: DTANH(Y).

Error: See DSINH and DCOSH.

[1] See "Introduction."

DSINH (or DSINH')

FUNCTION: Calculate the hyperbolic sine of a double precision number.

Declaration:

```
LONG PROCEDURE DSINH(Y); or DSINH'(Y);
  LONG Y;
    OPTION EXTERNAL;
               .
               .
               .
```

Method: $\mathrm{DSINH}(Y) = \dfrac{e^{Y} - e^{-Y}}{2}$

unless $Y < 0.1$. 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 double precision number.

Result: A double precision number.

FORTRAN: Basic External Function: DSINH(Y).

Error: See DEXP.

DCOSH (or DCOSH')

FUNCTION: Calculate the hyperbolic cosine of a double precision number.

Declaration:

```
LONG PROCEDURE DCOSH(Y); or DCOSH'(Y);
   LONG Y;
      OPTION EXTERNAL;
              .
              .
              .
```

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

Accuracy: (See "Introduction")

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

ATTRIBUTES:

Parameter: A double precision number.

Result: A double precision number.

FORTRAN: Basic External Function: DCOSH(Y).

Error: See DEXP.

CTANH (or CTANH')

FUNCTION: Calculate the hyperbolic tangent 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 CTANH(Y); or CTANH'(Y);
   REAL ARRAY Y;
      OPTION EXTERNAL;
                .
                .
                .
```

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

Method: $\text{CTANH(Y)} = \dfrac{\sinh(\text{Y})}{\cosh(\text{Y})}$

Accuracy: Depends on accuracy of SIN, COS, COSH, and SINH.

ATTRIBUTES:

Parameter: A complex number.

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

FORTRAN: Basic External Function: CTANH(Y).

Error: None.

Sample SPL/3000 calling sequence:

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

CSINH (or CSINH')

FUNCTION: Calculate the hyperbolic 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 CSINH(Y); or CSINH'(Y);
   REAL ARRAY Y;
      OPTION EXTERNAL;
                  .
                  .
                  .
```

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

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

Accuracy: Depends on accuracy of SIN, COS, COSH, and SINH.

ATTRIBUTES:

Parameter: A complex number.

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

FORTRAN: Basic External Function: CSINH(Y).

Error: None.

Sample SPL/3000 calling sequence:

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

CCOSH (or CCOSH')

FUNCTION: Calculate the hyperbolic 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 CCOSH(Y); or CCOSH'(Y);
   REAL ARRAY Y;
      OPTION EXTERNAL;
                 .
                 .
                 .
```

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

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

Accuracy: Depends on accuracy of SIN, COS, SINH, and COSH.

ATTRIBUTES:

Parameter: A complex number.

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

FORTRAN: Basic External Function: CCOSH(Y).

Error: None.

Sample SPL/3000 calling sequence:

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

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 $|f - g| \sim \dfrac{\epsilon}{1 + x^2}$

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:

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

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 all the conditions below occur:

|smaller argument| / |larger argument|

causes underflow and $Y \geqslant 0$ and $Z \geqslant 0$ and $Y < Z$.
See Section IV, 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 $|f - g| \sim \dfrac{3\epsilon}{1 + w^2}$

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:

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

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 all the conditions below occur:

|smaller argument| / |larger argument|

causes underflow and $Y \geqslant 0$ and $Z \geqslant 0$ and $Y < Z$.
See Section IV, 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:

```
CALL INVERT (\/\,C,K)
```

or through use of the SYSTEM INTRINSIC statement.

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:

```
CALL DINVERT (\J\,D,L)
```

or through use of the SYSTEM INTRINSIC statement.

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:

CALL CINVERT (\K\\,*E*,*M*)

or through use of the SYSTEM INTRINSIC statement.

Error: None.

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

RAND1 (or RAND1')

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

Declaration:

```
REAL PROCEDURE RAND1; or 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: Callable through use of the SYSTEM INTRINSIC statement.

Error: None.

COMMENT: This random value is derived from the 31-bit logical quantity changed every millisecond by the MPE/3000 system timer.

RAND (or RAND')

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

Declaration:

```
REAL PROCEDURE RAND (X); or 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.

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).

FORTRAN: Callable as an external function: $Y = \text{RAND}\ (Z)$

Error: None.

COMMENTS: The parameter value at the initial call to RAND completely determines a sequence of pseudo-random 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.

[1] See "Introduction."

DADD

FUNCTION: Calculate the sum of two double integer numbers.

Declaration:

```
DOUBLE PROCEDURE DADD(D1,D2);
  DOUBLE D1,D2;
    OPTION EXTERNAL;
           .
           .
           .
```

ATTRIBUTES:

Parameters: Double integer numbers.

Result: A double integer number.

FORTRAN: Callable as an external function: X = DADD(Y,Z)

Errors: If the result cannot be represented in the two words of a double integer, arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).

NOTE: This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.

DSUB

FUNCTION: Calculate the difference between two double integer numbers.

Declaration:

```
DOUBLE PROCEDURE DSUB(D1,D2);
  DOUBLE D1,D2;
    OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: Double integer number.

Result: A double integer number.

FORTRAN: Callable as an external function: $X = \text{DSUB}\ (Y,Z)$

Errors: If the result cannot be represented in the two words of a double integer, arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).

NOTE: *This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.*

DMPY (or DMPY')

FUNCTION: Calculate the product of two double integer numbers.

Declaration:

```
DOUBLE PROCEDURE DMPY(D1,D2);
   DOUBLE D1,D2;
      OPTION EXTERNAL;
                 .
                 .
                 .
                 or

PROCEDURE DMPY' (D1,D2);            (see Comment)
   VALUE D1,D2;
      OPTION EXTERNAL;
                 .
                 .
                 .
```

ATTRIBUTES:

Parameters: Double integer numbers.

Result: A double integer number (the result from DMPY' replaces parameter D1 on TOS, as in the hardware instruction DADD).

FORTRAN: Callable as an external function: X = DMPY(Y,Z)

Error: If the result cannot be represented in the two words of a double integer, arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).

COMMENT: A sample SPL/3000 calling sequence:

```
DOUBLE A,B,C;
      .
      .
      .
TOS := A;
TOS := B;
DMPY'(*,*);
C    := TOS;
```

which is equivalent to C := A*B; in double integer form.

NOTE: This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.

DDIV (or DDIV')

FUNCTION: Calculate the quotient only of one double integer number divided by another double integer number. See procedure DREM (or DREM') for the remainder.

Declaration:

```
DOUBLE PROCEDURE DDIV(D1,D2);
   DOUBLE D1,D2;
      OPTION EXTERNAL;
               .
               .
               .
              or

PROCEDURE DDIV' (D1,D2);               (see Comment)
   VALUE D1,D2: DOUBLE D1,D2;
      OPTION EXTERNAL;
               .
               .
               .
```

ATTRIBUTES:

Parameters: Double integer numbers.

Result: A double integer number, the quotient only (the result from DDIV' replaces parameter D1 on TOS, as in the hardware instruction DADD).

FORTRAN: Callable as an external function: $X = \mathrm{DDIV}(Y,Z)$

Error: If parameter D2 = 0, the arithmetic trap INTEGER DIVIDE BY ZERO occurs (if traps are enabled).

COMMENT: A sample SPL/3000 calling sequence:

```
DOUBLE A,B,C;
      .
      .
      .
TOS := A;
TOS := B;
DDIV'(*,*);
C    := TOS;
```

which is equivalent to C := A/B; in double integer form.

NOTE: This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.

DREM (or DREM')

FUNCTION: Calculate the remainder only of one double integer number divided by another double integer number. See procedure DDIV (or DDIV') for the quotient.

Declaration:

```
DOUBLE PROCEDURE DREM(D1,D2);
   DOUBLE D1,D2;
      OPTION EXTERNAL;
                  .
                  .
                  .
                 or
PROCEDURE DREM'(D1,D2);              (see Comment)
   VALUE D1,D2; DOUBLE D1,D2;
      OPTION EXTERNAL;
                  .
                  .
                  .
```

ATTRIBUTES:

Parameters: Double integer numbers.

Result: A double integer number, the remainder only (the result from DREM' replaces parameter D1 on TOS, as in the hardware instruction DADD).

FORTRAN: Callable as an external function: $X = \text{DREM}(Y,Z)$
where $X = Y$ MOD Z in double integer form.

Error: If parameter D2 = 0, the arithmetic trap INTEGER DIVIDE BY ZERO occurs (if traps are enabled).

COMMENT: A sample SPL/3000 calling sequence;

```
DOUBLE A,B,C;
      .
      .
      .
TOS := A;
TOS := B;
DREM'(*,*);
C    := TOS;
which is equivalent to C := A MOD B; in double integer form.
```

NOTE: This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.

DNEG

FUNCTION: Negate a double integer number.

Declaration:

```
DOUBLE PROCEDURE DNEG(D);
  DOUBLE D;
    OPTION EXTERNAL;
            .
            .
            .
```

ATTRIBUTES:

Parameter: A double integer number.

Result: The double integer number with the opposite sign.

FORTRAN: Callable as an external function: X = DNEG(Y)

Error: None.

NOTE: *This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.*

DCMP

FUNCTION: Compare two double integer numbers.

Declaration:

```
INTEGER PROCEDURE DCMP(D1,D2);
   DOUBLE D1,D2;
      OPTION EXTERNAL;
                 .
                 .
                 .
```

ATTRIBUTES:

Parameters: Double integer numbers.

Result:

-1 if D1 < D2
0 if D1 = D2
1 if D1 > D2

FORTRAN: Callable as an external function: I = DCMP(X, Y) for use in, for example, an arithmetic IF statement:

IF (I) 10,20,30

to direct the program as follows:

go to 10 if $X < Y$
go to 20 if $X = Y$
go to 30 if $X > Y$

Error: None.

NOTE: This procedure is maintained in the Compiler Library only for compatibility with previous versions of the FORTRAN/3000 Compiler which did not recognize double integers.

IFIX'

FUNCTION: Truncate a real number to an integer number.

Declaration:

```
INTEGER PROCEDURE IFIX'(Y);
   VALUE Y; REAL Y;
      OPTION EXTERNAL;
```

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

ATTRIBUTES:

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

Result: An integer number.

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

Error: If the real number is outside the range stated, the arithmetic trap INTEGER OVERFLOW occurs (if traps are enabled).

[1] See "Introduction."

MPYD

FUNCTION: Calculate the product of a decimal multiplicand and a decimal multiplier, such as those of RPG/3000 or COBOL/3000 Packed Decimal Numbers. See COMMENT.

This procedure is called normally only by code emitted by a compiler. However, this procedure can be called explicitly by a user's program.

Declaration:

```
PROCEDURE MPYD(OP2,OP2DIGS,OP1,OP1DIGS,SDEC);
   VALUE OP1DIGS,OP2DIGS,SDEC;
      BYTE ARRAY OP1,OP2;
      INTEGER OP1DIGS,OP2DIGS,SDEC;
         OPTION EXTERNAL;
                .
                .
                .
```

Method: Both operands are converted to multi-word binary integers then multiplied; the product is converted back to a Packed Decimal Number. If the signs of the operands do not match and the product is not zero, the sign of the result is set to - (minus).

The result includes the product and the sign. The result overlays OP2, the multiplier.

ATTRIBUTES:

Parameters (Input):

OP1 = The multiplicand decimal value, including its sign, in a byte array; 1 to 28 digits and sign. See COMMENT.

OP2 = The multiplier decimal value, including its sign, in a byte array; 1 to 28 digits and sign. See COMMENT and OP2DIGS.

OP1DIGS = The number of digits in decimal multiplicand OP1; a positive integer.

OP2DIGS = The number of digits in decimal multiplier OP2 and in the result overlaying OP2. This positive integer must provide enough digits for the largest possible product.

SDEC = The calling program's request for TOS Top-Of-Stack condition after the procedure returns control, in the two lowest order bits of an integer:
0 requests: leave both byte array pointers OP1 and OP2, and both integers OP1DIGS and OP2DIGS on TOS.
1 requests: leave only OP2 and OP2DIGS on TOS.
2 or 3 request: clear TOS of all residue from this procedure.

Parameter
(Output): OP2 = The decimal result, overlaying the multiplier OP2 byte array; OP2DIGS digit and sign.

Results: A decimal product and sign; and
Condition code:
CCE, product = 0
CCG, produce is + (plus);
CCL,product is – (minus).

Errors: Either OP1DIGS or OP2DIGS = 0; Only the SDEC request is answered (OP2 then remains the multiplier and is not overlaid by the result).

Either OP1DIGS or OP2DIGS < 0 or > 28: arithmetic trap INVALID DECIMAL OPERAND LENGTH occurs.

A non-decimal numeric character in the numeric digits of either array OP1 or OP2: arithmetic trap INVALID DECIMAL DIGIT occurs.

The result number of digits > 28: arithmetic trap DECIMAL OVERFLOW occurs; the result is meaningless.

The result number of digits > OP2DIGS but =< 28: arithmetic trap DECIMAL OVERFLOW occurs, but only the result MSD's (Most Significant Digits) are truncated.

COMMENT: Packed Decimal Numbers are byte arrays:

```
         BYTE:  [ FIRST  \ SECOND  \ ... \ LAST - 1\  LAST  ]
         BITS:  [  0-7   \   0-7   \ ... \   0-7   \   0-7  ]
 MAY CONTAIN:   BBBB\BBBB\BBBB\BBBB\ ... \BBBB\BBBB\BBBB\SSSS
 FOR DECIMAL:    D  \ D  \ D  \ D  \ ... \ D  \ D  \ D  \
      AS THE:    MSD\2MSD\3MSD\4MSD\ ... \3LSD\2LSD\ LSD\SIGN
                 OR \ OR \ OR \ OR \ ... \ OR \ OR \    \
                 LZ \ LZ \ LZ \ LZ \ ... \ LZ \ LZ \    \
```

```
WHERE:
 BBBB = ANY BINARY PATTERN 0000 THROUGH 1001, WEIGHTED 8-4-2-1
    D = ANY DECIMAL DIGIT   0  THROUGH   9
  MSD = MOST SIGNIFICANT DIGIT
  LSD = LEAST SIGNIFICANT DIGIT
  LZ  = LEADING ZERO
 SSSS = 1101 FOR - (MINUS); ALL ELSE = + (PLUS)
```

LONGDIVD and DIVD

FUNCTION: Calculate the quotient, or the quotient and remainder, of a decimal dividend and a decimal divisor, such as those of RPG/3000 or COBOL/3000 Packed Decimal Numbers. See COMMENT.

Two entry points are provided: The main entry, LONGDIVD, allows a dividend of up to 36 digits for COBOL/3000. The secondary entry, DIVD, allows a dividend of up to 28 digits for RPG/3000 or COBOL/3000.

The divisor can never exceed 28 digits.

This procedure is called normally only by code emitted by a compiler. Or this procedure can be called explicitly by a user's program.

Declaration:

```
PROCEDURE LONGDIVD(OP1,OP2DIGS,OP1,OP1DIGS,SDEC);
                          or
PROCEDURE DIVD(OP2,OP2DIGS,OP1,OP1DIGS,SDEC);
   VALUE OP1DIGS,OP2DIGS,SDEC;
      BYTE ARRAY OP1, OP2;
      INTEGER OP1DIGS,OP2DIGS,SDEC;
         OPTION EXTERNAL;
                  .
                  .
                  .
```

Method: Both operands are converted to multi-word binary integers then divided; then the quotient is – or the quotient and the remainder are – converted back to a Packed Decimal Number(s). If the signs of the operands do not match and the quotient, or the quotient and the remainder, are not zero, the sign of the result is set to – (minus).

When OP2DIGS is a positive integer, the result includes only the quotient and the sign. The result overlays OP2, the divisor.

When OP2DIGS is a negative integer, the result includes the quotient value and sign and the remainder value and sign. The quotient and sign overlay OP2, the divisor. The remainder and sign are appended to the original array OP2, in OP1DIGS more bytes (higher addresses).

NOTE: A user's program to calculate a quotient and remainder must allocate space beyond the Least Significant Digit of array OP2. See COMMENT.

LONGDIVD and DIVD (cont.)

ATTRIBUTES:

Parameters (Input):

OP1 = The dividend decimal value, including its sign, in a byte array:
For entry LONGDIVD: 1 to 36 digits and sign;
For entry DIVD: 1 to 28 digits and sign.
See COMMENT.

OP2 = The divisor decimal value, including its sign, in a byte array:
1 to 28 digits and sign.
See COMMENT.

OP1DIGS = The number of digits in decimal dividend OP1 and, if OP2DIGS is negative, in the remainder to be appended to array OP2 (See OP2DIGS).

OP2DIGS = The number of digits in decimal divisor OP2 and in the quotient overlaying OP2:

A positive integer to direct the procedure to return only the quotient and sign.

A negative integer to direct the procedure to return the quotient and sign and the remainder and sign.

SDEC = The calling program's request for TOS Top-Of-Stack condition after the procedure returns control, in the two lowest order bits of an integer:

0 requests: leave both byte array pointers OP1 and OP2, and both integers OP1DIGS and OP2DIGS on TOS.

1 requests: leave only OP2 and OP2DIGS on TOS.

2 or 3 request: clear TOS of all residue from this procedure.

Parameter (Output):

OP2 = Either one of two results possible:

The quotient only result, overlaying the divisor byte array OP2: OP2DIGS digits and sign.

The quotient and remainder result, overlaying the divisor array OP2 (the quotient and sign) then OP1DIGS more array digits appended to OP2 (the remainder and sign), in higher addresses.

Results: A decimal quotient and sign, or a decimal quotient and sign and a decimal remainder and sign; and
Condition code:
 CCE, quotient = 0;
 CCG, quotient is + (plus);
 CCL, quotient is – (minus).

Errors: Either OP1DIGS or OP2DIGS = 0; Only the SDEC request is answered (OP2 then remains the divisor and is not overlaid by the quotient).

OP2 = 0: arithmetic trap DECIMAL DIVIDE BY ZERO occurs.

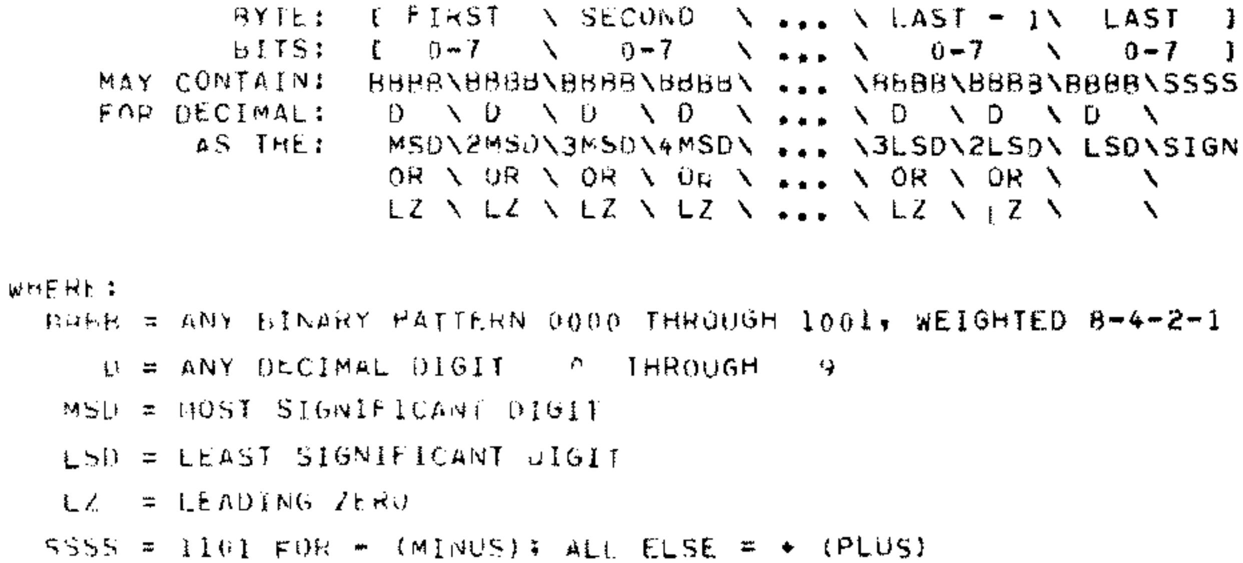
Entry DIVD tests for OP1DIGS < 0, or for OP1DIGS or OP2DIGS > 28: arithmetic trap INVALID DECIMAL OPERAND LENGTH occurs. Entry LONGDIVD does not test OP1DIGS or OP2DIGS because it expects to be used only by COBOL/3000.

A non-decimal numeric character in the numeric digits of either array OP1 or OP2: arithmetic trap INVALID DECIMAL DIGIT occurs.

The result number of digits > 28: arithmetic trap DECIMAL OVERFLOW occurs; the result is meaningless.

Reference: Knuth, Donald E. "The Classical Algorithms" Chapter 4.3.1, The Art of Computer Programming, Vol. 2., Reading, Mass. : Addison-Wesley Publishing Co., 1969.

COMMENT: Packed Decimal Numbers are byte arrays:



```
           BYTE:  [ FIRST   \ SECOND   \ ... \ LAST - 1\  LAST   ]
           BITS:  [   0-7   \    0-7   \ ... \    0-7   \   0-7  ]
   MAY CONTAIN:  BBBB\BBBB\BBBB\BBBB\ ... \BBBB\BBBB\BBBB\SSSS
   FOR DECIMAL:   D  \ D  \ D  \ D  \ ... \ D  \ D  \ D  \
        AS THE:   MSD\2MSD\3MSD\4MSD\ ... \3LSD\2LSD\ LSD\SIGN
                  OR \ OR \ OR \ OR \ ... \ OR \ OR \    \
                  LZ \ LZ \ LZ \ LZ \ ... \ LZ \ LZ \    \

WHERE:
  BBBB = ANY BINARY PATTERN 0000 THROUGH 1001, WEIGHTED 8-4-2-1
     D = ANY DECIMAL DIGIT   0  THROUGH   9
   MSD = MOST SIGNIFICANT DIGIT
   LSD = LEAST SIGNIFICANT DIGIT
   LZ  = LEADING ZERO
  SSSS = 1101 FOR - (MINUS); ALL ELSE = + (PLUS)
```

R'EDIT

FUNCTION: Either one of two possible:

NUMERIC EDIT: unpack a SOURCE array containing a Packed Decimal Number of an RPG/3000 program. That is, copy numeric digits and the sign from the SOURCE array, and copy alphanumeric characters from an EDITARRAY array, into a RESULT array of ASCII characters. See COMMENT.

ALPHANUMERIC EDIT: copy alphanumeric characters from a SOURCE array containing only ASCII characters into a RESULT array of ASCII characters.

This procedure is called normally only by code emitted by the RPG/3000 compiler for an EDIT WORD. However, this procedure can be called explicitly by a user's program.

Declaration:

```
PROCEDURE R'EDIT(RESULT,SOURCE,EDITARRAY,DIGCOUNT);
  VALUE DIGCOUNT;
    INTEGER DIGCOUNT;
    BYTE ARRAY RESULT,SOURCE,EDITARRAY;
      OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters (Input):

SOURCE = Numeric Edit: a Packed Decimal Number byte array.
Alphanumeric Edit: a byte array of ASCII characters.

EDITARRAY = Numeric or Alphanumeric Edit: a byte array of R'EDIT commands and, when needed, ASCII characters.
Details are given later, under DESCRIPTION.

DIGCOUNT = Numeric Edit: the integer number of digits in the SOURCE array Packed Decimal Number.
Alphanumeric Edit: 0 to signal the SOURCE array contains only ASCII characters.

Parameters (Output):

RESULT = a byte array of ASCII characters.

Result: A byte array of ASCII characters, RESULT.

Errors: See DESCRIPTION.

COMMENT: Packed Decimal Numbers are byte arrays:

```
          BYTE:  [ FIRST   \ SECOND   \ ... \ LAST - 1\  LAST   ]
          BITS:  [   0-7   \    0-7   \ ... \    0-7   \    0-7   ]
  MAY CONTAIN:  BBBB\BBBB\BBBB\BBBB\ ... \BBBB\BBBB\BBBB\SSSS
  FOR DECIMAL:   D  \ D   \ D   \ D   \ ... \ D   \ D   \ D   \
       AS THE:   MSD\2MSD\3MSD\4MSD\ ... \3LSD\2LSD\ LSD\SIGN
                 OR \ OR  \ OR  \ OR  \ ... \ OR  \ OR  \     \
                 LZ \ LZ  \ LZ  \ L7  \ ... \ LZ  \ LZ  \     \
```

```
WHERE:
  BBBB = ANY BINARY PATTERN 0000 THROUGH 1001, WEIGHTED 8-4-2-1
     D = ANY DECIMAL DIGIT   0   THROUGH   9
   MSD = MOST SIGNIFICANT DIGIT
   LSD = LEAST SIGNIFICANT DIGIT
   LZ  = LEADING ZERO
  SSSS = 1101 FOR - (MINUS); ALL ELSE = + (PLUS)
```

DESCRIPTION: Assume an array labelled PDN contains a Packed Decimal number
0057426+
that is to be unpacked by the RPG/3000 EDIT WORD
"bb,$0b.bb&CR"
into an array labelled TARGET. The number of digits in array PDN is reported by an integer PDNCNT.

The compiler then constructs a byte array that might be labelled EDCONTROL to contain R'EDIT commands and ASCII characters:

```
  \   0 :1100\     $     \   1 :1001\   0 :1011\   1 :1001\   0 :1000\>>
>>\   1 :1010\   1 :1000\   4 :1110\   2 :0111\     C     \     R     \>>
>>\   0 :0000\
```

from the R'EDIT command set listed later and from the standard ASCII character set. In this example, bytes are shown delimited by a | (vertical bar), a : (colon) signals each of ten R'EDIT commands, and three ASCII characters are included: $, C and R.

Then the compiler emits a calling sequence to procedure R'EDIT:

PROCEDURE R'EDIT(TARGET,PDN,EDCONTROL,PDNCNT);

to obtain the step-by-step execution of R'EDIT shown after the R'EDIT Command Set.

Procedure R'EDIT sets one internal flag when it begins:

The Significant Digit flag, set FALSE if the DIGCOUNT integer < > 0 to call Numeric Edit. Later, this flag is set TRUE when R'EDIT detects a significant digit in the SOURCE array, or by an explicit R'EDIT command defined later.

Or, the Significant Digit flag is set TRUE if the DIGCOUNT integer = 0 to call Alphanumeric Edit.

Procedure R'EDIT also sets two special characters when it begins:

The Fill character, set to ASCII   (SPACE) and reset to any ASCII character by the R'EDIT command 0110 described later.

The Float character, set to ASCII   (SPACE) and reset to any ASCII character by the R'EDIT command 1100 described later.

R'EDIT Command Set

The complete command set for procedure R'EDIT includes 13 commands. Each command occupies a full byte of the EDITARRAY: the left-most four bits (0-3) contain a REPEAT integer, and the right-most bits (4-7) are the R'EDIT command field:

Command Field Content — **Command Name, R'EDIT Action**

0000 TERMINATE EDIT
Numeric Edit:
If the current Float character is not ASCII   (SPACE), replace the last non-significant digit in the RESULT array with the Float character. Then, unconditionally terminate R'EDIT.

Alphanumeric Edit:
Terminate R'EDIT; do nothing else.

0001 COPY EDITARRAY BYTES
Numeric or Alphanumeric Edit:
Copy the next REPEAT + 1 bytes of the EDITARRAY into the RESULT array, unconditionally.

0010 SET SIGNIFICANCE
Intended for Numeric Edit only:
Set the Significant Digit flag, subject to the REPEAT integer and the first significant digit in the SOURCE array:

If REPEAT = 1, set (or leave) the Significant Digit flag TRUE and save the current byte pointer to the RESULT array as the destination for the Float character, regardless of the SOURCE content.

Command Field Content	Command Name, R'EDIT Action
0010 (cont)	If REPEAT <> 1 and the Significant Digit flag is FALSE and the SOURCE content is a non-zero decimal number, respond as if REPEAT = 1 defined above. But if any one of these three conditions does not exist, do nothing.
0011	COPY CHARACTERS SELECTED Intended for Numeric Edit only: Copy REPEAT + 1 characters into the RESULT array, subject to the Significant Digit flag: If the Significant Digit flag is TRUE, copy the next REPEAT + 1 bytes of the EDITARRAY. If the Significant Digit flag is FALSE, copy the current Fill Character REPEAT + 1 times.
0100	COPY ONE EDITARRAY BYTE Numeric or Alphanumeric Edit: Copy the next byte of the EDITARRAY into the RESULT array REPEAT + 1 times.
0101	COPY ONE CHARACTER Intended for Numeric Edit only: Copy a single ASCII character, subject to the Significant Digit flag: If the Significant Digit flag is TRUE, copy the next byte of the EDITARRAY REPEAT + 1 times. If the Significant Digit flag is FALSE, copy the Fill character REPEAT + 1 times.
0110	CHANGE FILL CHARACTER Intended for Numeric Edit only: Change the Fill character to a copy of the next EDITARRAY byte.
0111	COPY SIGN CHARACTER Intended for Numeric Edit only: Copy one or two ASCII characters into the RESULT array, subject to the sign of the Packed Decimal Number and the REPEAT integer: When the SOURCE array sign is – (minus): If REPEAT = 0 or 1, copy the next byte in the EDITARRAY once. If REPEAT = 2, copy the next two bytes in the EDITARRAY once.

Command Field Content — **Command Name, R'EDIT Action**

0111 (cont)

When the SOURCE array sign is + (plus):

If REPEAT = 0, copy a + (plus) once.

If REPEAT = 1, copy a (SPACE) once.

If REPEAT = 2, copy a (SPACE) twice.

1000

COPY SOURCE ARRAY DIGITS

Numeric Edit:
Copy the next REPEAT + 1 bytes of the SOURCE array into the RESULT array then set (or leave) the Significant Digit flag TRUE.

Alphanumeric Edit:
Copy the next REPEAT + 1 bytes of the SOURCE array into the RESULT array; do nothing else.

1001

COPY SOURCE ARRAY DIGITS, SET SIGNIFICANCE

Intended for Numeric Edit only:
Copy REPEAT + 1 bytes from the SOURCE array into the RESULT array. If the Significant Digit is FALSE, replace any leading zeroes with the current Fill character. If a significant digit is found in the SOURCE array during this function, set the Significant Digit flag TRUE.

1010

COPY PUNCTUATION

Intended for Numeric Edit only:
Set (or leave) the Significant Digit flag TRUE, then copy one punctuation character into the RESULT array, subject to the REPEAT integer:

If REPEAT = 0, copy a , comma.

If REPEAT = 1, copy a . period.

If REPEAT = 2, copy a / slash.

If REPEAT = 3, copy a – (minus).

If REPEAT = 4, copy a (SPACE).

1011

COPY PUNCTUATION SELECTED

Intended for Numeric Edit only:
Copy one punctuation character into the RESULT array, subject to the Significant Digit flag:

If the Significant Digit flag is TRUE, respond as if the R'EDIT command was 1010.

If the Significant Digit flag is FALSE, copy the current Fill character.

Command Field Content — **Command Name, R'EDIT Action**

1100 CHANGE FLOAT CHARACTER

Intended for Numeric Edit only:
Change the Float character subject to the REPEAT integer and the sign in the SOURCE array:

When REPEAT = 0, copy the next byte of the EDITARRAY regardless of the SOURCE content.

When the SOURCE array sign is – (minus):
If REPEAT = 1, copy a – (minus).
If REPEAT = 2, copy the next EDITARRAY byte.
If REPEAT > 2, respond as if it = 2.

When the SOURCE array sign is + (plus):
If REPEAT = 1, copy the next EDITARRAY byte.
If REPEAT = 2, copy a + (plus).
If REPEAT > 2, respond as if it = 2.

1101 **DO NOT USE. RESERVED FOR R'EDIT.

1110 **DO NOT USE. RESERVED FOR R'EDIT.

1111 **DO NOT USE. RESERVED FOR R'EDIT.

Returning now to the examples assumed earlier, step-by-step execution of procedure R'EDIT is:

EDCONTROL or EDITARRAY Content	PDN or SOURCE Content	Significant Digit flag Before/After	TARGET or RESULT Content
0 :1100 $	n/u	FALSE/FALSE	(empty)
1 :1001	00	FALSE/FALSE	bb
0 :1011	n/u	FALSE/FALSE	bbb
1 :1001	57	FALSE/TRUE	bbb57
0 :1000	4	TRUE/TRUE	bbb574
1 :1010	n/u	TRUE/TRUE	bbb574.
1 :1000	26	TRUE/TRUE	bbb574.26
4 :1010	n/u	TRUE/TRUE	bbb574.26b
2 :0111 C R	n/u	TRUE/TRUE	bbb574.26bbb
0 :0000	n/u	TRUE/TRUE	bb$574.26bbb

b means ASCII (SPACE)
n/u means "not used."

SECTION III
UTILITY PROCEDURES

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

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.

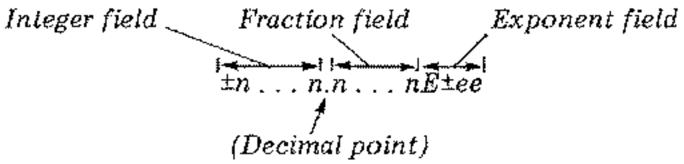
Results: See "Parameters (Output)."

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 4 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 when KIND = 3 (Gw.d format) or = 2 (Dw.d format) or = 1 (Ew.d format). If a positive scale factor is also used, set W at least D + 7.

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 ⩽ 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 $\leqslant$ 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.

INEXT' (cont.)

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 ⩽ 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.

ITOI'

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

Declaration:

```
PROCEDURE ITOI';
   OPTION EXTERNAL;
              .
              .
              .
```

An integer number B is raised to an integer power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in one word. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. The integer result overlays the first word and the remaining word is deleted from the stack.

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:

Values: B = 0, P = any integer number $\geqslant 0$, or
B = any integer number $\neq 0$, P = any integer number
(*B is the base, P is the power*).

Result: An integer number.

FORTRAN: Not callable.

Error: If B = 0 and P < 0, SOFTERROR' message: ITOI': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result exceeds the range of integer numbers [-32768, 32767], the arithmetic trap INTEGER OVERFLOW may occur (if traps are enabled).

DTOI'

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

Declaration:

```
PROCEDURE DTOI';
  OPTION EXTERNAL;
                .
                .
                .
```

A double integer number B is raised to an integer power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in two words. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. The double integer result overlays the first two words and the remaining word is deleted from the stack.

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:

Values: B = 0 D, P = any integer number ⩾ 0, or
B = any double integer number ≠ 0, P = any integer number
(*B is the base, P is the power*).

Result: A double integer number.

FORTRAN: Not callable.

Error: If B = 0 and P < 0, SOFTERROR' message: DTOI': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result exceeds the range of double integer numbers [-2147483648, 2147483647], the arithmetic trap INTEGER OVERFLOW may occur (if traps are enabled).

DTOD'

FUNCTION: Raise a double integer number base to a double integer number power.

Declaration:

```
PROCEDURE DTOD';
  OPTION EXTERNAL;
          .
          .
          .
```

A double integer number B is raised to a double integer power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in two words. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. The double integer result overlays the first two words and the remaining two words are deleted from the stack.

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:

Values: B = 0 D, P = any double integer number $\geqslant 0$, or
B = any double integer number $\neq 0$, P = any double integer number
(*B is the base, P is the power*).

Result: A double integer number.

FORTRAN: Not callable.

Error: If B = 0 and P < 0, SOFTERROR' message: DTOD': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result exceeds the range of double integer numbers [-2147483648, 2147483647], the arithmetic trap INTEGER OVERFLOW may occur (if traps are enabled).

RTOI'

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

Declaration:

```
PROCEDURE RTOI';
  OPTION EXTERNAL;
          .
          .
          .
```

A real number B is raised to an integer power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in two words. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. The real result overlays the first two words and the remaining word is deleted from the stack.

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:

Values: B = 0.0, P = any integer number $\geqslant 0$, or
B = any real number $\neq 0.0$, P = any integer number
(*B is the base, P is the power*).

Result: A real number.

FORTRAN: Not callable.

Error: If B = 0.0 and P < 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 FLOATING POINT OVERFLOW or FLOATING POINT UNDERFLOW may occur (if traps are enabled).

RTOD'

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

Declaration:

```
PROCEDURE RTOD';
   OPTION EXTERNAL;
          .
          .
          .
```

A real number B is raised to a double integer power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in two words. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. The real result overlays the first two words and the remaining two words are deleted from the stack.

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:

Values: B = 0.0, P = any double integer number $\geqslant 0$, or
B = any real number $\neq 0.0$, P = any double integer number
(B is the base, P is the power).

Result: A real number.

FORTRAN: Not callable.

Error: If B = 0.0 and P < 0, SOFTERROR' message: RTOD': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result is outside the range of real numbers (see "Introduction"), the arithmetic traps FLOATING POINT OVERFLOW or FLOATING POINT UNDERFLOW may occur (if traps are enabled).

RTOR'

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

Declaration:

```
PROCEDURE RTOR';
   OPTION EXTERNAL;
         .
         .
         .
```

A real number B is raised to a real power P. Use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in two words. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. The real result overlays the first two words and the remaining two words are deleted 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 = EXP(P*ALOG(B)).

Accuracy: See EXP and ALOG.

ATTRIBUTES:

Values: B = 0.0, P = any real number ⩾ 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.

RTOL'

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 four 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 four more words, overlay the result on the first four 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. Use "TOS := 0;" to put an integer zero onto the stack in one more word. The result overlays these four 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.

LTOI'

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

Declarations: LONG real numbers in SPL/3000 programs are represented in four 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 B 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 the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in four words. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. The result overlays the first four words and the remaining word is deleted from the stack.

R = call-by-reference; use the SPL/3000 statement "TOS:=@B;" to put the address of B onto the top of the stack in one word. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. Use "TOS := 0D;" to put a double integer "0" onto the top of the stack in two more words. The result overlays these 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:

Values: B = 0.0L, P = any integer number $\geqslant 0$, or
B = any LONG real number $\neq 0.0$, P = any integer 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, SOFTERROR' message LTOD': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result is outside the range of LONG real numbers (see "Introduction"), the arithmetic traps EXTENDED PRECISION OVERFLOW or EXTENDED PRECISION UNDERFLOW may occur (if traps are enabled).

LTOD'

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

Declarations: LONG real numbers in SPL/3000 programs are represented in four words (see "Internal Representation" in the "Introduction"). A LONG real base B is raised to a double 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 LTODf';
  OPTION EXTERNAL;
          .
          .
          .
```

where

f = V or R for the first parameter B:

V = call-by-value; use the SPL/3000 statement "TOS:=B;" to put the value of B onto the top of the stack in four words. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. The result overlays the first four words and the remaining two words are deleted from the stack.

R = call-by-reference; use the SPL/3000 statement "TOS :=@B;" to put the address of B onto the top of the stack in one word. Use "TOS :=P;" to put the value of P onto the top of the stack in two more words. Use "TOS := 0;" to put an integer zero onto the stack in one more word. The result overlays these 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:

Values: B = 0.0L, P = any double integer number $\geqslant 0$, or
B = any LONG real number $\neq 0.0$, P = any double integer 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, SOFTERROR' message LTOD': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If the result is outside the range of LONG real numbers (see "Introduction"), the arithmetic traps EXTENDED PRECISION OVERFLOW or EXTENDED PRECISION UNDERFLOW may occur (if traps are enabled).

LTOL'

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

Declarations: LONG real numbers in SPL/3000 programs are represented in four 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 four 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 four more words (five words total), overlay result on the first four words, then delete the remaining word from the stack.
2. If f_1 is V, four words are on top-of-stack, use SPL/3000 statement "TOS := P;" to set P value on top-of-stack in four more words (eight words total), overlay result on the first four words, then delete the remaining four 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 := 0D;" to set a double integer "0" on the top-of-stack in two more words (four words total), then overlay the result on those four words.
2. If f_1 is V, four 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 (five words total), overlay result on the first four 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.

CTOI'

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

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").

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 the SPL/3000 statement "TOS:=B(0);" to put the value of the real part of B onto the top of the stack in two words. Use "TOS:=B(1);" to put the imaginary part of B onto the top of the stack in two more words. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. The result overlays the first four words and the remaining word is deleted from the stack.

R = call-by-reference; use the SPL/3000 statement "TOS:=@B;" to put the address of B onto the top of the stack in one word. Use "TOS:=P;" to put the value of P onto the top of the stack in one more word. Use "TOS:= 0D;" to put a double integer "0" onto the top of the stack in two more words. The result overlays these 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:

Values: B = a complex number = 0.0 ($a = b = 0.0$), P = any integer ⩾ 0, or
B = any complex number representable in two real numbers, one for *a* and one for *b*; *a* and *b* are not both 0.0, P = any integer number
(*B is the base, P is the power*).

Result: A complex number.

FORTRAN: Not callable.

Errors: If *a* and *b* of B = 0.0 and P < 0, SOFTERROR' message CTOI':ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If a or b of the result is outside the range of real numbers (see "Introduction"), the arithmetic traps FLOATING POINT OVERFLOW or FLOATING POINT UNDERFLOW may occur (if traps are enabled).

CTOD'

FUNCTION: Raise a complex number base to a double integer number power.

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").

A complex base B is raised to a double 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 CTODf';
  OPTION EXTERNAL;
            .
            .
            .
```

where

f = V or R for the first parameter B:

V = call-by-value; use the SPL/3000 statement "TOS:=B(0);" to put the value of the real part of B onto the top of the stack in two words. Use "TOS:=B(1);" to put the imaginary part of B onto the top of the stack in two more words. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. The result overlays the first four words and the remaining two words are deleted from the stack.

R = call-by-reference; use the SPL/3000 statement "TOS:=@B;" to put the address of B onto the top of the stack in one word. Use "TOS:=P;" to put the value of P onto the top of the stack in two more words. Use "TOS:=0;" to put an integer "0" onto the top of the stack in one more word. The result overlays these 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:

Values: B = a complex number = 0.0 ($a = b = 0.0$), P = any double integer ⩾ 0, or
B = any complex number representable in two real numbers, one for *a* and one for *b*; *a* and *b* are not both 0.0, P = any double integer number
(*B is the base, P is the power*).

Result: A complex number.

FORTRAN: Not callable.

Errors: If a and b of B = 0.0 and P < 0, SOFTERROR' message CTOD': ILLEGAL ARGUMENTS occurs (see "Library Errors").

CAUTION: If a or b of the result is outside the range of real numbers (see "Introduction"), the arithmetic traps FLOATING POINT OVERFLOW or FLOATING POINT UNDERFLOW may occur (if traps are enabled).

Complex Arithmetic

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 ADDC$f_1 f_2 f_3$'
  OPTION EXTERNAL;
```

or

```
PROCEDURE SUBC$f_1 f_2 f_3$';
  OPTION EXTERNAL;
```

or

```
PROCEDURE MLTC$f_1 f_2 f_3$';
  OPTION EXTERNAL;
```

or

```
PROCEDURE DIVC$f_1 f_2 f_3$';
  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.

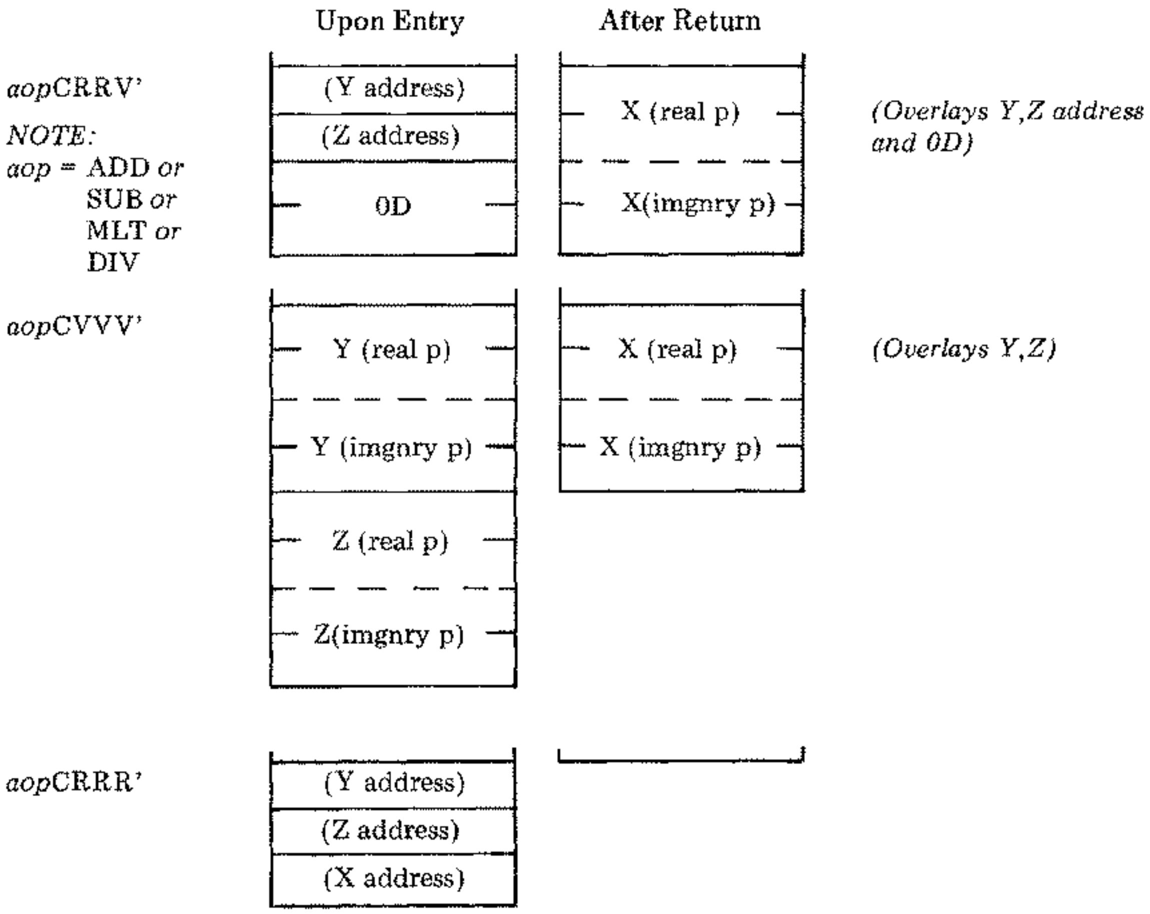
f_3 = V or R, for result parameter:

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):



NOTE:
aop = ADD *or*
SUB *or*
MLT *or*
DIV

Complex Arithmetic (cont.)

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 FLOATING POINT OVERFLOW, FLOATING POINT UNDERFLOW, or 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$f_1 f_2$';
  OPTION EXTERNAL;

 .
 .
 .

where

f_1 = V or R, for the first parameter:

V = call-by-value; use SPL/3000 statement "TOS := Y(0);" 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:

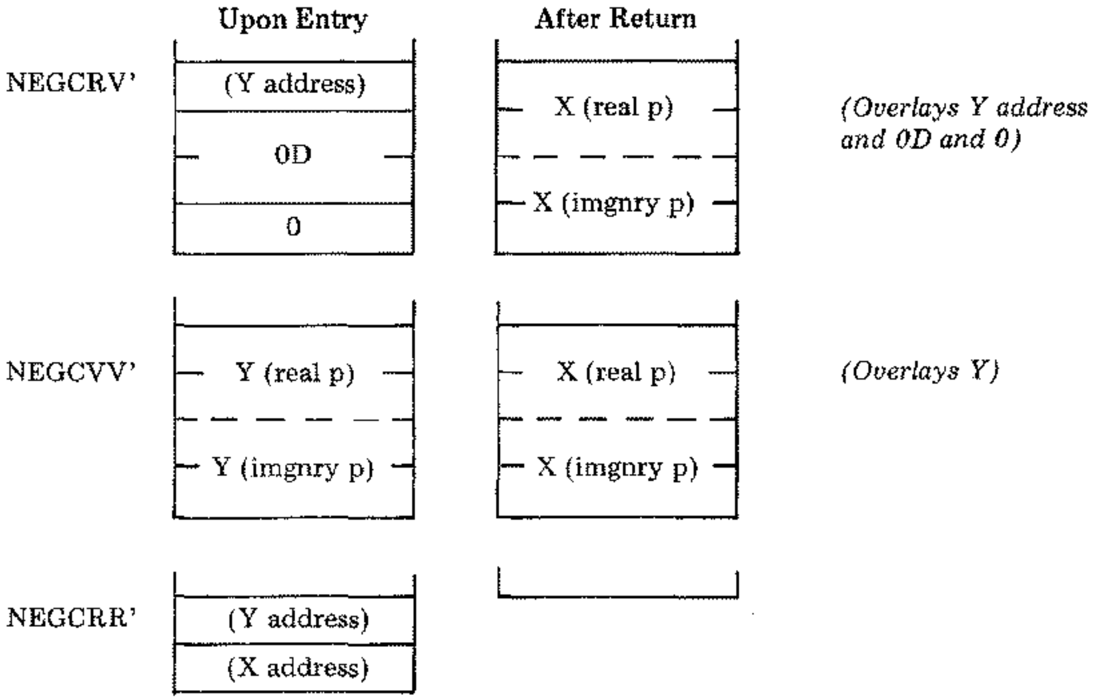
V = call-by-value:

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.

Complex Negate (cont.)

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$f_1f_2$';
  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*.

Complex Compare (cont.)

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. A FORTRAN/3000 program can request other actions too, through the procedure UNITCONTROL described later in this section.

Declaration:

```
PROCEDURE FTNAUX' (OPT,UNIT);
   VALUE OPT,UNIT;INTEGER OPT,UNIT;
     OPTION EXTERNAL;
                .
                .
                .
```

ATTRIBUTES:

Parameters: OPT: An integer to specify the action:

-1: REWIND (but don't close the file)
0: BACKSPACE
1: ENDFILE (write an EOF mark)
2: SKIP BACKWARD TO A TAPE MARK
3: SKIP FORWARD TO A TAPE MARK
4: UNLOAD TAPE AND CLOSE THE FILE
5: LEAVE TAPE AND CLOSE THE FILE
6: CONVERT FILE TO PRE-SPACING*
7: CONVERT FILE TO POST-SPACING*
8: CLOSE FILE

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: Implicitly called through the auxiliary I/O statements REWIND, BACKSPACE and ENDFILE.

Errors: See "Comments."

COMMENTS:

1. The following comments refer to descriptions in the *MPE Intrinsics Reference Manual* and the *Systems Programming Language Reference Manual.*

2. If the FORTRAN/3000 compiler generates the call to FTNAUX' (from the FORTRAN auxiliary I/O statements REWIND, BACKSPACE and ENDFILE), the parameter OPT is set to -1, 0 or 1, respectively. Further, a FORTRAN Logical Unit Table (FLUT) is prepared in the user's DB Data Area by the MPE/3000 system loader.

*See the discussion of file control operations in the *MPE Intrinsics Reference Manual.*

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 = 0 or UNIT > 99, the report FILE NOT IN TABLE FOR UNIT #*xx* occurs (see Section I, "FORMATTER ERROR REPORTS") and the user's program is aborted.

   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, the Formatter error report FILE NOT IN TABLE FOR UNIT #*xx* occurs. 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 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/3000 defines UNIT 5 to be $STDIN and UNIT 6 to be $STDLIST. If the FOPEN intrinsic is not successful (indicated by condition code CCL), the Formatter Error Report FILE SYSTEM ERROR occurs.

5. Three other entries to this procedure FTNAUX' are available to FORTRAN/3000 users:

   UNITCONTROL provides any of the actions described under parameter OPT.

   FNUM returns the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number.

   FSET enables the user to change the MPE/3000 system file number assigned to a given FORTRAN Logical Unit Number.

   For further details, see procedures UNITCONTROL, FNUM and FSET, later in this section.

6. REWIND and BACKSPACE actions are provided, historically, for control of magnetic tape files. If the device referenced has no physical capability corresponding to the OPT (action) request, no action occurs.

7. For the OPT value -1, an MPE/3000 file intrinsic FCONTROL call is made with *controlcode* = 5. This may invoke a physical operation on the device referenced.

8. For the OPT value 0, an MPE/3000 file intrinsic FSPACE call is made with *displacement* = -1. This may invoke a physical operation on the device referenced.

9. For the OPT value 1, an MPE/3000 file intrinsic FCONTROL call is made with *controlcode* = 6.

10. For the OPT values listed below, MPE/3000 file intrinsic FCONTROL calls are made with *controlcode* only or *controlcode* and *param* set as follows:

OPT	controlcode	param
2	7	(not used)
3	8	(not used)
6	1	401_8
7	1	400_8

11. For the OPT values listed below, MPE/3000 file intrinsic FCLOSE calls are made with *disposition* set to:

OPT	disposition
4	1
5	3
8	0

12. If OPT is a value outside the range [-1,8], the Formatter Error Report UNDEFINED OPTION ON UNIT #*xx* occurs.

13. Either of the Formatter Error Reports FILE SYSTEM ERROR ON UNIT #*xx* or END OF FILE ERROR ON UNIT #*xx* can occur.

UNITCONTROL

FUNCTION: A secondary entry point to procedure FTNAUX'. UNITCONTROL enables a FORTRAN/3000 program to request any of the actions listed below under OPT for any FORTRAN Logical Unit.

Declaration:

```
PROCEDURE UNITCONTROL(UNIT,OPT);
   INTEGER UNIT,OPT;
      OPTION EXTERNAL;
            .
            .
            .
```

ATTRIBUTES:

Parameters: UNIT: A positive integer in the range [1,99] to specify the FORTRAN Logical Unit Table (FLUT) entry (see "Comments") of the file device to be used.

OPT: An integer to specify the action:

-1:	REWIND (but don't close the file)
0:	BACKSPACE
1:	ENDFILE (write an EOF mark)
2:	SKIP BACKWARD TO A TAPE MARK
3:	SKIP FORWARD TO A TAPE MARK
4:	UNLOAD TAPE AND CLOSE THE FILE
5:	LEAVE TAPE AND CLOSE THE FILE
6:	CONVERT FILE TO PRE-SPACING*
7:	CONVERT FILE TO POST-SPACING*
8:	CLOSE FILE

Result: See "Comments."

FORTRAN: Callable as an external subroutine:

CALL UNITCONTROL(*12,6*)

Errors: See "Comments."

COMMENTS:

1. If UNIT ≤ 0 or UNIT > 99, the report FILE NOT IN TABLE FOR UNIT #*xx* occurs (see Section I, "FORMATTER ERROR REPORTS") and the user's program is aborted.

   If UNIT is in the range required, UNITCONTROL checks the FLUT (see Section I, "File System Requirements") for that UNIT number. If there is no corresponding U entry, the Formatter Error Report FILE NOT IN TABLE FOR UNIT #*xx* occurs and the user's program is aborted. If a

*See the discussion of file control operations in the *MPE Intrinsics Reference Manual.*

corresponding U entry is found and the F entry for that is 0, an MPE/3000 file intrinsic FOPEN call is made, as described in Comment 4 for FTNAUX'.

2. For each of the actions available through this procedure, one or another MPE/3000 file intrinsic is called, as described in Comments 7 through 11 for FTNAUX'.

3. If OPT is a value outside the range [-1,8], the Formatter Error Report UNDEFINED OPTION ON UNIT #*xx* occurs and the user's program is aborted.

4. Either of the Formatter Error Reports FILE SYSTEM ERROR ON UNIT #*xx* or END OF FILE ON UNIT #*xx* can occur.

FNUM

FUNCTION: A secondary entry point to procedure FTNAUX'. FNUM enables a FORTRAN/3000 program 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: UNIT, a positive integer in the range [1,99] to specify the FORTRAN Logical Unit Table (FLUT) entry (see "Comments" for procedure FTNAUX') to be used.

Result: An integer number, the MPE/3000 system file number for the UNIT specified.

FORTRAN: Callable as an external function: *I* = FNUM(*UNIT*)

NOTE: FNUM must be declared an INTEGER FUNCTION.

Errors: If UNIT is not in the range required, or if there is no corresponding U entry in the FLUT, the report FILE NOT IN TABLE FOR UNIT #*xx* (see Section I, "FORMATTER ERROR REPORTS") occurs and the user's program is aborted.

FSET

FUNCTION: A secondary entry point to procedure FTNAUX'. FSET enables a FORTRAN/3000 program 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] to specify the FORTRAN Logical Unit Table (FLUT) entry (see "Comments" for procedure FTNAUX') for which the change is to be made.

NEWFILE, a positive integer in the range [1,254] 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 number assigned to the UNIT specified above.

Result: See "Parameter (output)," above.

FORTRAN: Callable as an external subroutine:

CALL FSET(*3,FNUMB,OLD*)

Errors: See "Comments."

COMMENTS:

1. If UNIT is not in the range required, or if there is no corresponding U entry in the FLUT, the report FILE NOT IN TABLE FOR UNIT #*xx* (see Section I, "FORMATTER ERROR REPORTS") occurs and the user's program is aborted.
2. If NEWFILE is not in the range required, the Formatter Error Report INVALID FILE NUMBER FOR UNIT #*xx* occurs and the user's program is aborted.
3. If the value returned to OLDFILE is 0, that file was not open and remains unopened.

DATELINE

FUNCTION: Fill a byte array with formatted date and time information.

Declaration:

```
PROCEDURE DATELINE(BUF);
   BYTE ARRAY BUF;
      OPTION EXTERNAL;
              .
              .
              .
```

ATTRIBUTES:

Parameter: A pointer to the first byte of the array.

Result: The byte array is filled as described in Comment 1.

FORTRAN: Callable as an external subroutine, as described in Comment 2.

Error: None.

COMMENTS:

1. The byte array must be at least 27 bytes (characters) long; the first 27 bytes are filled as follows:

Byte(s)	Contain(s)
1–3	Day of the week (SUN,MON,TUE,WED,THU,FRI,SAT)
4–5	A comma and a blank (,Δ)
6–8	Month of the year (JAN,FEB,MAR,APR,MAY,JUN, JUL,AUG,SEP,OCT,NOV,DEC)
9	A blank (Δ)
10–11	Day of the month (Δ1 through 31)
12–13	A comma and a blank (,Δ)
14–17	The year
18–19	A comma and a blank (,Δ)
20–21	The hour (Δ1 through 12)
22	A colon (:)
23–24	The minute (00 through 59)
25	A blank (Δ)
26–27	AM or PM

2. A sample FORTRAN use:

```
CHARACTER S*27
          :
CALL DATELINE(S)
DISPLAY S[6:11]
```

which displays the month, day and year only.

PRINTFILEINFO (or PRINT 'FILE' INFO)

FUNCTION: Print a File Information Display on the job or session list device $STDLIST.

This procedure is called normally only by an MPE/3000 subsystem or an MPE/3000 utility program. However, this procedure can be called explicitly by a user's program.

Declaration:

```
PROCEDURE PRINTFILEINFO(FNUM);
                    or
PROCEDURE PRINT 'FILE' INFO(FNUM);
   VALUE FNUM; INTEGER FNUM;
      OPTION EXTERNAL;
                    .
                    .
                    .
```

ATTRIBUTES:

Parameter: Any MPE/3000 file number currently available to the calling program.

Result: A File Information Display in either of two formats described under Comment.

FORTRAN: Callable as an external subroutine:

CALL PRINTFILEINFO(\FILENUM\)

where:
FILENUM is the MPE/3000 file number, and
\\ (the two backslashes) tell FORTRAN/3000 to pass this parameter by value rather than by reference.

Also callable through use of the SYSTEM INTRINSIC statement.

Error: None.

COMMENT: A short display of only two or three lines occurs if access to the MPE/3000 file number (FNUM or FILENUM) is blocked or if that file number is undefined in the MPE/3000 file system in use.

```
         +-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
Line 1 → !  FILE NUMBER #       IS UNDEFINED.              !
Line 2 → !  ERROR NUMBER: 56     RESIDUE: 0                !
Line 3 → !  BLOCK NUMBER: 0                NUMREC: 0       !
         +-------------------------------------------------+
```

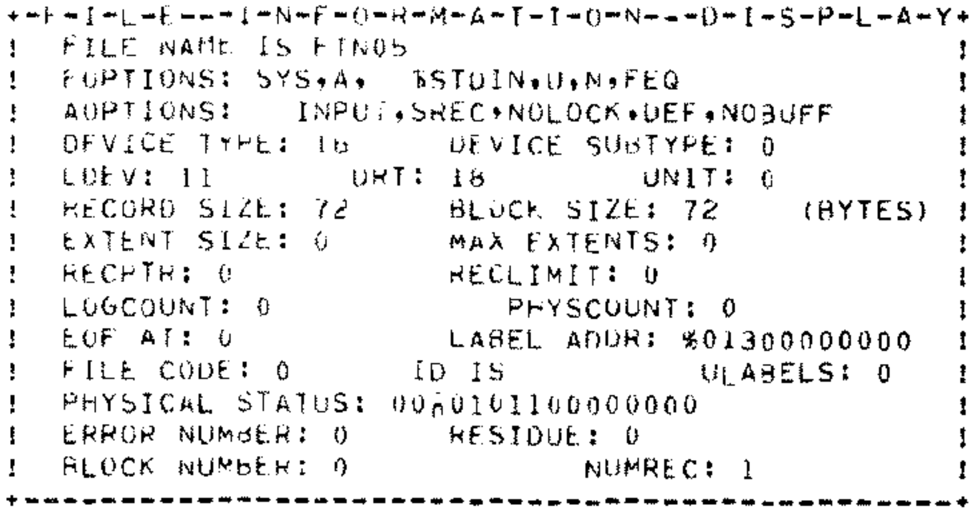
where lines 1 through 3 are explained on the next page.

Line 1 is not included if access to the file is blocked. However, line 1 does report the file number if that number is undefined in the MPE/3000 file system in use.

Line 2 reports an ERROR NUMBER that is explained in the *MPE Intrinsics Reference Manual*, and a RESIDUE integer number of bytes not transmitted for an input/output request (in this case, no input/output request was made, hence RESIDUE: 0).

Line 3 reports the BLOCK NUMBER of the physical record and the RECNUM (number of logical records) in the current block of the file (not opened, hence both integers are 0 to signal "unknown").

For files opened but a CCG (end-of-file error) or a CCL (irrecoverable file error) condition code occurred or an explicit call to this procedure was made by a user's program, a display of 14 lines occurs:



```
                 +-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
Line 1   →   !  FILE NAME IS FIN05                                  !
Line 2   →   !  FOPTIONS: SYS,A,   $STDIN,U,N,FEQ                   !
Line 3   →   !  AOPTIONS:    INPUT,SREC,NOLOCK,DEF,NOBUFF           !
Line 4   →   !  DEVICE TYPE: 16      DEVICE SUBTYPE: 0              !
Line 5   →   !  LDEV: 11        DRT: 18         UNIT: 0             !
Line 6   →   !  RECORD SIZE: 72      BLOCK SIZE: 72     (BYTES)     !
Line 7   →   !  EXTENT SIZE: 0       MAX EXTENTS: 0                 !
Line 8   →   !  RECPTR: 0            RECLIMIT: 0                    !
Line 9   →   !  LOGCOUNT: 0             PHYSCOUNT: 0                !
Line 10  →   !  EOF AT: 0            LABEL ADDR: %01300000000       !
Line 11  →   !  FILE CODE: 0       ID IS           ULABELS: 0       !
Line 12  →   !  PHYSICAL STATUS: 000101100000000                    !
Line 13  →   !  ERROR NUMBER: 0      RESIDUE: 0                     !
Line 14  →   !  BLOCK NUMBER: 0              NUMREC: 1              !
                 +--------------------------------------------------+
```

where:

Line 1 reports the name of the file.

Line 2 reports the FOPTIONS in effect:

Domain:

NEW	=	a new file, or
SYS	=	the system file domain, or
JOB	=	the job temporary domain, or
ALL	=	both system and job temporary domains.

Type:

A	=	an ASCII file, or
B	=	a binary file.

Line 2 (cont.)

Default file designator:

FORMAL = the actual file designator is the same as the formal file designator.

Record Format:

F = fixed length, or
V = variable length, or
U = undefined length, or
? = unknown format.

Carriage control:

N = none, or
C = carriage control character expected.

File equation option:

FEQ = :FILE (the MPE/3000 command) allowed, or
DEQ = :FILE disallowed.

Line 3 reports the AOPTIONS in effect:

Access type:
INPUT = read access, or
OUTPUT = write access, or
OUTKEEP = write-only access, without deleting, or
APPEND = append access, or
IN/OUT = input and output access, or
UPDATE = update access.

Multi-record option:
SREC = single record access, or
MREC = multi-record access.

Dynamic locking option:
NOLOCK = no locking permitted, or
LOCK = locking permitted.

Exclusive access option:
DEF = default specification, or
EXC = exclusive access allowed, or
SEA = semi-exclusive access allowed, or
SHR = sharable file.

Buffering:
BUFFER = automatic buffering, or
NOBUFF = inhibit buffering.

Lines 4 and 5 report the DEVICE TYPE, the DEVICE SUB-TYPE, the LDEV (logical device number), the DRT (device reference table entry number), and the UNIT number of the device on which the file resides.

Line 6 reports the RECORD SIZE and the BLOCK SIZE of the current record.

Line 7 reports the EXTENT SIZE of the current extent and the MAX EXTENTS (maximum number of extents) allowed.

Line 8 reports the RECPTR (the current record pointer) and the RECLIMIT (limit on the number of records).

Line 9 reports the LOGCOUNT (present count of logical records) and the PHYSCOUNT (present count of physical records).

Line 10 reports the EOF AT (location of the current end-of-file) and the LABEL ADDR (location of the header label).

Line 11 reports the FILE CODE and ID (identity for the user who created the file) and ULABELS (the number of user-defined labels).

Line 12 reports the PHYSICAL STATUS code (bit pattern).

Line 13 reports an ERROR NUMBER that is explained in the *MPE Intrinsics Reference Manual,* and a RESIDUE integer number of bytes not transmitted for an input/output request (in this case, no input/output request was made, hence RESIDUE: 0).

Line 14 reports the BLOCK NUMBER of the physical record and the RECNUM (number of logical records) in the current block of the file.

COMPLIBINFO (or COMPLIBINFO')

FUNCTION: Identify whether the Compiler Library in use is a three-word, extended precision, floating point version or a four-word, extended precision, floating point version. (Used primarily by compilers.)

Declaration:

```
PROCEDURE COMPLIBINFO (INFOWORD); or COMPLIBINFO' (INFOWORD);
   LOGICAL INFOWORD;
      OPTION EXTERNAL
                 .
                 .
                 .
```

ATTRIBUTES:

Result:

Bit 15 = 0	implies 3-word
Bit 15 = 1	implies 4-word
Bits 0 through 14	reserved

FORTRAN: Callable as an external subroutine.

Error: None.

PLOTS

FUNCTION: Initialize plotter variables, initialize a user-defined plot commands buffer, and use the MPE/3000 file intrinsic FOPEN[1] to open the plotter file.

Declaration:

```
PROCEDURE PLOTS(BUFF,BUFFSIZE);
   INTEGER ARRAY BUFF; INTEGER BUFFSIZE;
      OPTION EXTERNAL;
            .
            .
            .
```

ATTRIBUTES:

Parameters: BUFF: A pointer to the first word of the user-defined plot buffer.

BUFFSIZE: The size, in words, of BUFF.

FORTRAN: Callable as an external subroutine:

CALL PLOTS(*IBUFF,500*)

Errors:

1. SOFTERROR’ message PLOTS: INVALID BUFFER SIZE occurs if the BUFFSIZE parameter is not large enough.

2. PLOTS can report one of four errors. After any one, a File Information Display (see PRINT'FILE'INFO in this section) is printed and the program is aborted:

   ****ERROR ON PLOT FILE OPEN****

   The FOPEN intrinsic was unable to open the plotter file.

   ****INVALID PLOTTER SUBTYPE****

   The FCONTROL intrinsic obtained an undefined sub-type value (see Comment 6).

   ****PLOTS: FCONTROL ERROR****

   The FCONTROL intrinsic encountered an error (see Comment 6).

   ****PLOTS: FGETINFO ERROR****

   The FGETINFO intrinsic encountered an error or an undefined device type integer (see Comment 5).

[1] See *MPE Intrinsics Reference Manual.*

COMMENTS:

1. The integer array BUFF must be defined before PLOTS is called:

   In an SPL/3000 program, use an array declaration:

   INTEGER ARRAY *IBUFF(0:500)*

   In a FORTRAN/3000 program, use a DIMENSION statement:

   DIMENSION *IBUFF(500)*

2. The size of BUFF must provide for storage of plot information between successive calls to the PLOT procedure (described later in this section) as well as a series of plotter commands. Typically, parameter BUFFSIZE should be given a value of at least 100 words and agree with the size specified in the array declaration. Zip mode plotting is more efficient for large values of the BUFFSIZE parameter.

3. PLOTS should be called only once during a given plotting sequence and before any other plotter procedures are called. A plotting sequence is terminated by a call to the procedure PLOT with parameter PEN = 999. Any call to PLOTS after the first call and before the terminating call to procedure PLOT is ignored.

4. To open the plotter file, the MPE/3000 file intrinsic FOPEN is used with its parameters set as follows (these settings can be superceded by an MPE/3000 file command :FILE):

   formaldesignator PLOTFILE

   foptions

Bit(s)	Field Name and Setting(s)
14:2	Domain: 00, this is a new file.
13:1	ASCII/BINARY: 0, this is a BINARY file.
10:3	Default File Designator: 000, the default file designator is the same as the formal file designator.
8:2	Record Format: 10, undefined-length records.
7:1	Carriage Control: 0, no carriage control character expected.
6:1	(Reserved for MPE/3000 system use.)
5:1	Disallow File Equation: 0, allow :FILE commands.
0:5	(Reserved for MPE/3000 system use.)

aoptions

Bit(s)	Field Name and Setting(s)
12:4	Access Type: 0001, write access only.
11:1	Multirecord: 0, non-multirecord mode.
10:1	Dynamic Locking: 0, disallow dynamic locking/unlocking.
8:2	Exclusive: 01, exclusive access.
7:1	Inhibit Buffering: 0, allow normal buffering.
0:7	(Reserved for MPE/3000 system use.)

device = PLOTTER

(All other parameters assume the default value.)

5. The increment (i.e., the minimum pen movement) of any given plotter is determined by the device itself. When the device is added to the MPE/3000 system, its software interface is configured by the procedures described in the *System Manager/Supervisor Manual.* The device type and device sub-type are set as follows:

Cal Comp Plotter Series	Device Type	Increment	Device Sub-Type
500	35	.010"	0
600	36	.005"	1
700	37	.0025"	2
		.00125"	3
		.002"	4
		.1 mm	5
		.05 mm	6
		.025 mm	7

NOTE: If the increment is not one of the values listed above, PLOTS must be re-compiled with the EQUATE statement for variable INCR changed to specify the number of pen movements per inch of total displacement. For example, INCR equated to 1000 implies an increment of .001". In such a case, the non-zero (equated) value of INCR takes precedence over the sub-type value to determine the increment size.

6. When PLOTS executes, it interrogates the software interface through the MPE/3000 file intrinsics FGETINFO (to obtain the device type) and FCONTROL (with *controlcode* = 0, to obtain the plotter sub-type and therefore the increment size).

   If the device type is not 35 or 36 or 37, the plotter commands are to be written onto a disc or magnetic tape for later read-back to a plotter. In such a case, the sub-type is not examined. Rather, either a default increment of .01" or an equated value of the variable INCR (see the note above) is used. In either case, FACTOR (described later in this section) can be called to produce plot commands in the correct proportions for the plotter that will ultimately produce the plots.

PLOT

FUNCTION: Convert general X-axis and Y-axis coordinates into distinct plotter commands, manage buffering of the plotter commands and close the plotter file when the plotting sequence is complete.

Declaration:

```
PROCEDURE PLOT(X,Y,PEN);
  REAL X,Y; INTEGER PEN;
    OPTION EXTERNAL;
          .
          .
          .
```

ATTRIBUTES:

Parameters:

X: The X-axis position, in inches from the current origin, where the pen is to be moved.

Y: The Y-axis position, in inches from the current origin, where the pen is to be moved.

PEN: An integer, to specify pen down/pen up status, use of the plot command buffer, origin definition, and termination of the plotting sequence:

= 2, pen down during movement, accumulate plot commands in the buffer.

= 3, pen up during movement, accumulate plot commands in the buffer.

= -2, pen down during movement, transmit all plot commands accumulated from prior calls and this call, define the terminal pen position to be the new origin for subsequent calls.

= -3, pen up during movement, transmit all plot commands accumulated from prior calls and this call, define the terminal pen position to be the new origin for subsequent calls.

= 12, pen down during movement, transmit all plot commands accumulated from prior calls and this call.

= 13, pen up during movement, transmit all plot commands accumulated from prior calls and this call.

= -12, same as 12, plus define the terminal pen position as the new origin for subsequent calls.

= -13, same as 13, plus define the terminal pen position as the new origin for subsequent calls.

= 999, terminate the plotting sequence: pen up during movement, transmit all plot commands accumulated from prior calls and this call, define the terminal pen position as the new origin for subsequent calls, then close the plotter file.

FORTRAN: Callable as an external subroutine:

CALL PLOT(*XAXIS*, *YAXIS*,*IPEN*)

Errors:

1. SOFTERROR' message PLOT: PLOTS NOT CALLED occurs if the PLOTS procedure has not been called before PLOT is called.

2. SOFTERROR' message PLOT: INVALID PEN PARAMETER occurs if the PEN value is not one of those listed under "ATTRIBUTES."

3. PLOT can report one of two errors. After either one, a File Information Display (see PRINT'FILE'INFO in this section) is printed and the program is aborted:

   ```
   ****PLOTTER WRITE ERROR****
   ```

   The MPE/3000 file intrinsic FWRITE (called by PLOT) found an error.

   ```
   ****ERROR ON PLOT FILE CLOSE****
   ```

   The FCLOSE intrinsic found an error (see Comment 5).

COMMENTS:

1. The PLOTS procedure must be called before any call can be made to PLOT.

2. If the plot buffer BUFF (see PLOTS in this section) is filled by a series of calls to PLOT with PEN = 2 or 3, all plot commands accumulated up to that point are transmitted to the plotter file. Then accumulation resumes at the beginning of the buffer.

3. PEN values 12, 13, -12, and -13 are provided for interactive use, such as testing the accuracy of individual PLOT calls before accumulating a series of calls.

4. Any negative value legal to PEN includes the specification "define the terminal position of the pen as the new origin for subsequent X and Y values." That is, at that position, the logical X and Y coordinates are set to 0.0. Thus, all plotter commands accumulated up to and including the call with a negative PEN value are transmitted to the plotter file before the new origin is defined.

5. The PEN value 999 is identical to the value -3, plus the specification "close the plotter file by use of the MPE/3000 file intrinsic FCLOSE with *disposition* = 0".

6. The PEN value 999 can be used only once during a plotting sequence; it terminates the sequence. To begin another sequence, the PLOTS procedure must be called first.

FACTOR

FUNCTION: Change the plot factor (the ratio of the plot physical size to the plot command size).

Declaration:

```
PROCEDURE FACTOR(FACT);
  REAL FACT;
    OPTION EXTERNAL;
              .
              .
              .
```

ATTRIBUTES:

Parameter: FACT: The desired plot factor.

FORTRAN: Callable as an external subroutine:

CALL FACTOR(*RATIO*)

Errors:

1. SOFTERROR' message FACTOR: PLOTS NOT CALLED occurs if procedure PLOTS has not been called before FACTOR is called.

2. SOFTERROR' message FACTOR: INVALID FACTOR occurs if the value of FACT is less than or equal to 0.

COMMENTS:

1. The procedure PLOTS initializes the factor to 1.0, the normal plot ratio.

2. If FACT = 2, all physical pen movements are twice the distance specified by parameters X and Y in procedure PLOT. Conversely, if FACT = .5, all physical pen movements are half those distances specified.

WHERE

FUNCTION: Return the current X-axis and Y-axis positions, in inches from the current origin, of the present pen location and return the current plot factor.

Declaration:

```
PROCEDURE WHERE(RXPAGE,RYPAGE,RFACT);
   REAL RXPAGE,RYPAGE,RFACT;
      OPTION EXTERNAL;
                  .
                  .
                  .
```

ATTRIBUTES:

Parameters: RXPAGE: A real identifier.

RYPAGE: A real identifier.

RFACT: A real identifier.

Results: RXPAGE and RYPAGE and RFACT, real numbers.

FORTRAN: Callable as an external subroutine:

CALL WHERE(*XPOSIT, YPOSIT,PLTFCT*)

Error: SOFTERROR' message WHERE: PLOTS NOT CALLED occurs if the PLOTS procedure has not been called before WHERE is called.

SYMBOL

FUNCTION: Write plot annotation in the form of ASCII characters and special symbols listed in Table 3-1.

Declaration:

```
PROCEDURE SYMBOL(XPAGE,YPAGE,HEIGHT,IBCD,ANGLE,NCHAR);
   REAL XPAGE,YPAGE,HEIGHT,ANGLE; ARRAY IBCD; INTEGER NCHAR;
      OPTION EXTERNAL;
                  .
                  .
                  .
```

ATTRIBUTES:

Parameters:

XPAGE and YPAGE: Real numbers, the position before rotation (see ANGLE), in inches from the current origin, where writing of the character(s) is to begin. See Comment 1 for further details.

HEIGHT: The height in inches, for the character(s) to be written. See Comment 2 for further details.

IBCD: A pointer to the storage area that contains either a symbol code integer or an ASCII character string. See Comment 3 for further details.

ANGLE: The rotation angle in counter-clockwise degrees from the X-axis, for the base line of the character(s) to be written. If ANGLE = 0.0, the base line will be parallel to the X-axis and the character(s) will be rightside-up. If ANGLE = 180.0, the character(s) will be upside-down.

NCHAR: An integer, the number of characters to be written, from those in array IBCD:

If NCHAR > 0, an ASCII character string is to be written; the first character is the first one in IBCD.

If NCHAR = 0, only one (or the only) character in IBCD is to be written.

If NCHAR < 0, a special symbol is to be written; that symbol is selected by an integer in the first (or only) word of IBCD.

See Comment 4 for further details.

FORTRAN: Callable as an external subroutine:

CALL SYMBOL(*ATX,ATY,HIGH,IANN,DEGREES,ICHARS)*

Errors:

1. SOFTERROR' message SYMBOL: PLOTS NOT CALLED occurs if the PLOTS procedure has not been called before SYMBOL is called.

2. SOFTERROR' message SYMBOL: INVALID CHARACTER OR SYMBOL occurs if an ASCII character or special symbol integer not listed in Table 3-1 is specified in IBCD.

NOTE: If a user-written error procedure is furnished to override the normal function of SOFTERROR' and return control to SYMBOL (see Section IV), an error symbol will be written in place of a legitimate symbol. That error symbol is ‽ (a question mark ? superimposed on an exclamation mark !).

COMMENTS:

1. XPAGE and YPAGE specify the position of the first or only character to be written, according to the value of NCHAR:

   If NCHAR < 0 and the special symbol is a centered one (see Table 3-1), the position is that of the center of the symbol. If the special symbol is not a centered one, the position is that of the lower left corner.

   If NCHAR ⩾ 0, the position is that of the lower left corner of the first (or only) character to be written.

   Further, XPAGE and/or YPAGE can be set to 999.0 to specify that the position is that at which a next character would start. This special value 999.0 can be used for both XPAGE and YPAGE or for either one independently of the other.

2. For best results from HEIGHT, for non-centered symbols (see Table 3-1) specify a multiple of seven times the plotter increment. For centered symbols specify a multiple of four times the plotter increment.

   EXAMPLE:

   To write the character A approximately 1/2" high (tall), with a plotter increment of .010", specify HEIGHT = .049 (= .010 * 7 * 7).

3. The array IBCD must contain one of three types of data, according to the value of NCHAR:

   If NCHAR $>$ 0, IBCD must contain an ASCII character string left-justified in the array (starting in the left byte of the first word).

   If NCHAR = 0, IBCD must contain the desired ASCII character right-justified in the first (or only) word (i.e., in the right byte).

   If NCHAR $<$ 0, IBCD must contain an integer listed in Table 3-1 for the desired special character in the first (or only) word.

4. NCHAR, in addition to its affect on XPAGE, YPAGE, and IBCD (see Comments 1 and 3), controls the pen during movement:

   If NCHAR $\geqslant$ 0, the pen is up during movement to the starting position of the first character.

   If NCHAR = –1, the pen is up during the move to the starting position, after which the special character is written.

   If NCHAR $<$ –1, the pen is down during the move to the starting position, after which the special character is written.

5. Table 3-1 appears on the following page.

Table 3-1. Plotter Characters/Symbols

ASCII Characters

1.	space	26.	9	51.	R	76.	k
2.	!	27.	:	52.	S	77.	l
3.	”	28.	;	53.	T	78.	m
4.	#	29.	<	54.	U	79.	n
5.	$	30.	=	55.	V	80.	o
6.	%	31.	>	56.	W	81.	p
7.	&	32.	?	57.	X	82.	q
8.	'	33.	@	58.	Y	83.	r
9.	(	34.	A	59.	Z	84.	s
10.	)	35.	B	60.	[	85.	t
11.	*	36.	C	61.	\	86.	u
12.	+	37.	D	62.	]	87.	v
13.	,	38.	E	63.	^	88.	w
14.	–	39.	F	64.	—	89.	x
15.	.	40.	G	65.	`	90.	y
16.	/	41.	H	66.	a	91.	z
17.	0	42.	I	67.	b	92.	{
18.	1	43.	J	68.	c	93.	|
19.	2	44.	K	69.	d	94.	}
20.	3	45.	L	70.	e	95.	∿
21.	4	46.	M	71.	f		
22.	5	47.	N	72.	g		
23.	6	48.	O	73.	h		
24.	7	49.	P	74.	i		
25.	8	50.	Q	75.	j		

Special Symbols

Integer	Symbol	Integer	Symbol	Integer	Symbol	Integer	Symbol
0	⧄	11	✳	21	Σ	31	√
1	⏀	12	⧗	22	≥	32	¢
2	⍋	13	|	23	▵	33	?
3	+	14	✡	24	≠		
4	×	15	–	25	±		
5	⟐	16	|	26	←		
6	⍏	17	↓	27	↑		
7	X̄	18	≤	28	∫		
8	Z̄	19	=	29	Π		
9	Y	20	→	30	÷		
10	⌧						

Special symbols 0-14 are centered symbols.

FORTRAN Run-time Procedures

Procedures from the following list are called by FORTRAN/3000 compiler-generated code at run time to perform the functions listed for users' programs.

> *CAUTION: The operation and calling sequences of these procedures are optimized for the code generation needs of the FORTRAN compiler and should not be used explicitly by any other user program.*

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
DFIXRT' DFIXT'	Fixes and truncates a double precision number to a double integer number.
DFLOATT' DFLT'	Converts a double integer number to a double precision number.
F'SYSTRAP F'LIBTRAP F'CONTRAP F'ARITRAP F'SYSTRAPPROC F'CONTRAPPROC F'ARITRAPPROC F'LIBTRAPPROC	Handles run-time details of FORTRAN trap handling statements.
INUM' RNUM' JNUM' 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)
JNUM'	Field width *w*	0	-1 (double integer)
DNUM'	Field width *w*	0	-2 (LONG[1] real)

Procedure(s) Identifier	Function
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).
OVFL'	Generates integer overflow for invalid type transfers.
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

SECTION IV
Library Errors

Many routines in the Compiler Library, especially some mathematical routines and the Formatter, can detect error(s) in the data processed. Those routines, when error detection occurs, call one of two library error routines: SOFTERROR' or FMTERROR'. The normal function of either error routine is to report the error conditions and abort the user's program. All error messages are listed in Table 4-1, in this section; further details of the Formatter error messages (from FMTERROR') are given in Section I, under "Formatter Error Reports."

All SOFTERROR' messages are followed by one or two additional reports: an "illegal parameter value(s) report" (if needed) then a "stack trace-back report." The former occurs for most library procedures that find such an error.

The SOFTERROR' message format is (see Table 4-1):

procedure name: *message*

The illegal parameter value(s) report format is:

X = *first or only parameter value*

Y = *second parameter value* (this line is omitted if not needed)

NOTE: If either or both values are complex, two values are reported for each; the first value is the real part, the second value is the imaginary part.

The stack trace-back report, which also occurs after any Formatter Error Report, uses the same format as is provided by the MPE Stack Dump facility. For further information, see *MPE Debug/Stack Dump Reference Manual* (HP Part No. 30000-90012).

XLIBTRAP

The user can override the normal functions of SOFTERROR' (or FMTERROR') 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', or the Formatter finds an error and calls FMTERROR'.

2. SOFTERROR' or FMTERROR' checks for a user-written error procedure.

3. If no user-written error procedure has been specified, the appropriate error report is produced and the current program is aborted. If a user-written error procedure has been specified, the library trap mechanism is disarmed then the user-written procedure is called.

4. When the user-written error procedure returns control to SOFTERROR' or FMTERROR', 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' or FMTERROR' 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' or FMTERROR' will return to the user program without printing an error message; if QUIT = TRUE, SOFTERROR' or FMTERROR' will abort the user program.

EXAMPLE: XLIBTRAP USE

Assume that in the procedure USER, the user-written error procedure MINE is to be called from SOFTERROR' (or FMTERROR') whenever an 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;
     XLIBTRAP(MINE,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	Message
1	ATAN2 (or ATAN2')	Both arguments = 0	ATAN2: ARGUMENTS ZERO
2	ATAN2 (or 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 $\leqslant 0$	ALOG: ARGUMENT NOT POSITIVE
8	DLOG (or DLOG')	Argument $\leqslant 0$	DLOG: ARGUMENT NOT POSITIVE
9	CABS (or CABS')	Result overflow	CABS: OVERFLOW
10	SQRT (or SQRT')	Argument < 0	SQRT: ARGUMENT NEGATIVE
11	DSQRT (or DSQRT')	Argument < 0	DSQRT: ARGUMENT NEGATIVE
12	TAN (or TAN')	Argument near $\frac{(2k+1)\pi}{2}$ (see text)	TAN: OVERFLOW
13	DTAN (or DTAN')	Argument near $\frac{(2k+1)\pi}{2}$ (see text)	DTAN: OVERFLOW
14 · · · 53	(Unassigned)		
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	RTOR': ILLEGAL ARGUMENTS
57	RTOL'	Base = 0 and power < 0	RTOL': ILLEGAL ARGUMENTS
58	LTOI'	Base = 0 and power < 0	LTOI': ILLEGAL ARGUMENTS
59	LTOL'	Base = 0 and power < 0 or base < 0	LTOL': ILLEGAL ARGUMENTS
60	CTOI'	Base = 0 and power < 0	CTOI': ILLEGAL ARGUMENTS

Table 4-1. HP 3000 Compiler Library Errors (cont.)

Error number	Library routine name	Error description	Message
61	INUM' RNUM' JNUM' DNUM'	Illegal character in string being converted.	NUM: ILLEGAL CHARACTER
62	INUM' RNUM' JNUM' DNUM'	Number out of representable range (see "Introduction").	NUM: NUMBER OUT OF RANGE
63	INUM' RNUM' JNUM' 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	DTOI'	Base = 0 and power < 0	DTOI': ILLEGAL ARGUMENTS
68	DTOD'	Base = 0 and power < 0	DTOD': ILLEGAL ARGUMENTS
69	RTOD'	Base = 0 and power < 0	ROTD': ILLEGAL ARGUMENTS
70	LTOD'	Base = 0 and power < 0	LTOD': ILLEGAL ARGUMENTS
71	CTOD'	Base = 0 and power < 0	CTOD': ILLEGAL ARGUMENTS
72 . . . 100	(Unassigned)		
101	Formatter	Illegal format character	ILLEGAL FORMAT CHARACTER
102	UNITCONTROL or FTNAUX'	Parameter OPT is outside the range [-1,8].	UNDEFINED OPTION ON UNIT #*xx*
103	Formatter	Specification group(s) nested deeper than level 4.	NESTING TOO DEEP
104	Formatter	List and character string specification do not match.	STRING MISMATCH
105	Formatter	Illegal character in input field.	BAD INPUT CHARACTER
106	Formatter	Numeric input field unrepresentable.	NUMBER OUT OF RANGE
107	Formatter	Format specification exceeds record length.	FORMAT BEYOND RECORD

Table 4-1. HP 3000 Compiler Library Errors (cont.)

Error number	Library routine name	Error description	Message
108	Formatter	Core-to-core conversion exceeds user-defined buffer.	BUFFER OVERFLOW
109	Formatter	Binary direct access exceeds record length.	DIRECT ACCESS OVERFLOW ON UNIT #*xx*
110	Formatter	File name not in FLUT.	FILE NOT IN TABLE FOR UNIT #*xx*
111	Formatter	File access problem.	FILE SYSTEM ERROR ON UNIT #xx
112	Formatter	File access problem.	END OF FILE DETECTED ON UNIT #*xx*
113	FSET	Parameter NEWFILE is outside the range [1,254].	INVALID FILE NUMBER FOR UNIT #*xx*
114 . . . 149	(Unassigned)		
150	PLOTS	Invalid plot buffer size.	PLOTS: INVALID BUFFER SIZE
151	PLOT	PLOTS procedure has not been called.	PLOT: PLOTS NOT CALLED
152	PLOT	Unrecognized PEN value.	PLOT: INVALID PEN PARAMETER
153	FACTOR	PLOTS procedure has not been called.	FACTOR: PLOTS NOT CALLED
154	FACTOR	Invalid plot factor (FACT ⩽0)	FACTOR: INVALID FACTOR
155	WHERE	PLOTS procedure has not been called.	WHERE: PLOTS NOT CALLED
156	SYMBOL	PLOTS procedure has not been called.	SYMBOL: PLOTS NOT CALLED
157	SYMBOL	Unrecognized input symbol.	SYMBOL: INVALID CHARACTER OR SYMBOL

APPENDIX A
Library Procedure Names

Index