RTE FORTRAN IV

Reference Manual

HEWLETT
PACKARD

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 92060-90023
Printed in U.S.A. July 1980

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain the latest replacement pages and write-in instructions to be merged into the manual, including an updated copy of this Printing History page.

To replenish stock, this manual will be reprinted as necessary. Each such reprinting will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information.

To determine the specific manual edition and update which is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog.

Seventh Edition .... Mar 1980
Update 1 .... Jul 1980
Reprint .... Jul 1980 (Update 1 incorporated)

PREFACE

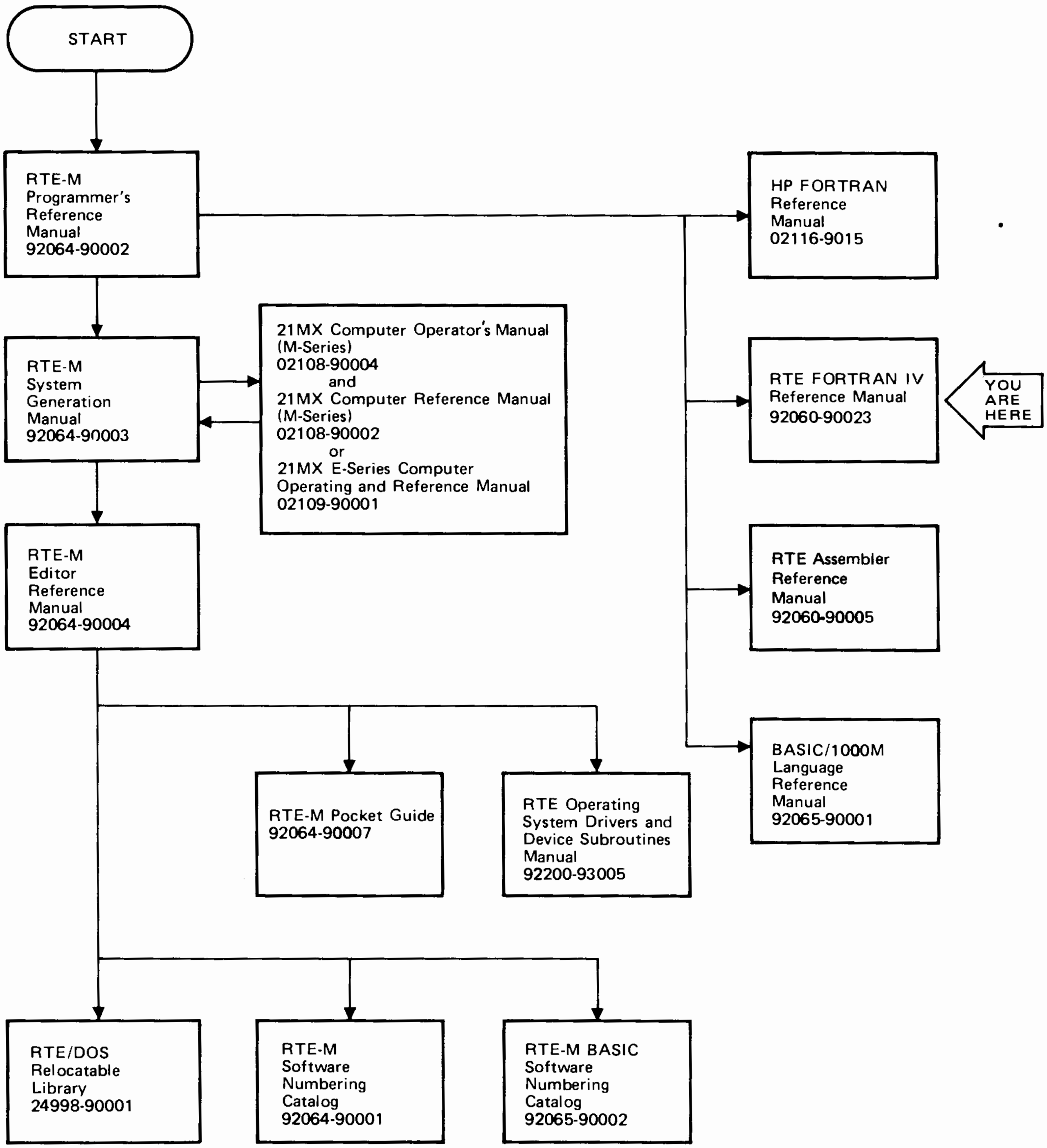
The front matter includes a Table of Contents and an Introduction to the manual. Sections I through III describe the form of source programs and the types, identification, and format of data and expressions used in RTE FORTRAN IV. Sections IV through IX describe the language elements used to code a source program, including the formats and uses of RTE FORTRAN IV statements. The Appendixes describe the format of data in memory, the form of RTE FORTRAN IV jobs, departures from and extensions of ANSI FORTRAN IV specifications, features included in RTE FORTRAN IV for compatibility with HP FORTRAN, RTE FORTRAN IV Compiler error diagnostics, the HP character set for computer systems, and the RTE FORTRAN IV invocation command for RTE-II, RTE-III, RTE-IV, and RTE-M Operating Systems.

> NOTE: Throughout the manual are special boxed notes that explain departures from ANSI FORTRAN IV specifications or features for compatibility with HP FORTRAN.

This manual is a reference text for programmers who have had FORTRAN programming experience, either with HP FORTRAN or with other FORTRAN compilers.

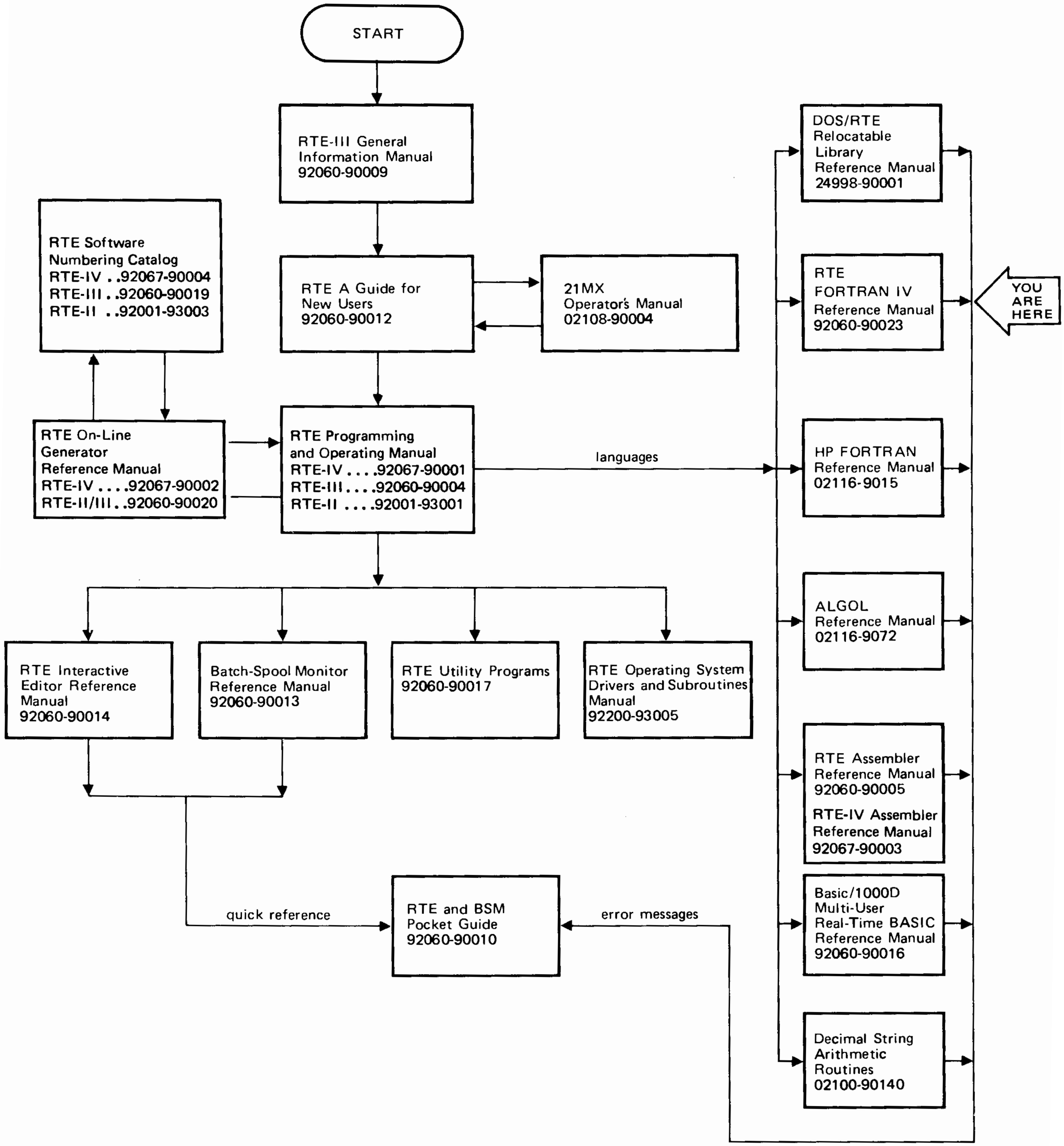
The documentation maps on the following pages are a guide to HP documentation pertinent to the use of RTE FORTRAN IV.

RTE-M OPERATING SYSTEM DOCUMENTATION MAP



7700-24

RTE-II/III/IV OPERATING SYSTEMS DOCUMENTATION MAP



7700-23

CONTENTS

SECTION III

SECTION IV

SECTION V

SECTION VI

SECTION VIII (cont.)

THE FORMAT STATEMENT

SECTION IX

TABLES

COMPILER PURPOSE

The RTE FORTRAN IV Compiler is used to construct object language programs from source language programs written according to the rules of the RTE FORTRAN IV language described in this manual.

FILE DEFINITION

In the following discussion, a file is defined to be a sequential access device which may be either on a mass storage device such as a disc, or an external device such as a card reader.

COMPILER SYNOPSIS

The RTE FORTRAN IV Compiler reads source input from a source file. The compiler writes the resultant object program on a standard binary output file in a format acceptable to the Relocating Loader. Exact detail for specifying these files is found in the Reference Manuals for the Operating System being used (see the Documentation Maps on pages iv and v).

RTE FORTRAN IV is a multi-pass compiler. A pass is defined as a processing cycle of the source program. In the initial pass, the source program is processed, a symbol table is constructed, and a set of intermediate machine code is generated. During subsequent passes, the compiler searches the symbol table for object code references, completes translation of the intermediate object code on the disc and produces a relocatable binary object program. It produces the object program as directed at invocation. Source and object listings may be produced, if specified in the FORTRAN IV control statement (see Appendix B), or the Operating System program invocation command (see Appendix J).

COMPILER ENVIRONMENT

The RTE FORTRAN IV Compiler is available in the HP 92001 RTE-II, HP 92060 RTE-III, HP 92067 RTE-IV, and HP 92064 RTE-M Operating Systems. The hardware configurations required for compiling and executing RTE FORTRAN IV programs under control of these systems are described in the appropriate system documentation.

The libraries of relocatable subroutines available to RTE FORTRAN IV are described in the HP DOS/RTE Relocatable Library Reference Manual. See the documentation maps on p. IV or V for the part number of this manual.

SECTION I
THE FORM OF A FORTRAN IV PROGRAM

The RTE FORTRAN IV Compiler accepts as input a source program written according to the specifications contained in this manual. Each source program is constructed from characters grouped into lines and statements. Appendix F shows a sample program listing. The elements used to construct a source language program are defined in the following text.

FORTRAN IV SOURCE PROGRAMS

The following terms define FORTRAN IV source programs:

Executable Program: A program that can be used as a self-contained computing procedure. An executable program consists of precisely one main program and its subprograms and segments*, if any.

Main Program: A set of statements and comments not containing a FUNCTION, SUBROUTINE, or BLOCK DATA statement, beginning with a program statement and ending with an END statement.

Subprogram: A set of statements and comments containing a FUNCTION, SUBROUTINE, or a BLOCK DATA statement. When defined by FORTRAN statements and headed by a FUNCTION statement, it is called a function subprogram. When defined by FORTRAN statements and headed by a SUBROUTINE statement, it is called a subroutine subprogram. When defined by FORTRAN statements and headed by a BLOCK DATA statement, it is called a block data subprogram. Subprograms also can be written in HP FORTRAN, HP ALGOL, or HP Assembler languages.

*Segmented programs may not be supported in some operating systems.

Program Unit: A main program or a subprogram.

Segments *: An overlayable set of statements beginning with a PROGRAM statement which specifies Type 5, and ending with an END statement.

FORTRAN IV CHARACTER SET

A source language program is written using the following character set.

Letters: The twenty-six letters A through Z.

Digits: The ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Unless specified otherwise, a string of digits is interpreted in the decimal base number system when a number system base interpretation is appropriate.

Alphanumeric Character: A letter or a digit.

Blank Character: Has no meaning and may be used to improve the appearance of a program with the following exceptions:

a. A continuation line cannot contain a blank in column 6.

b. A blank character is valid and significant in Hollerith data strings.

c. In numeric input conversions, leading blanks are not significant, but embedded blanks are converted to zeros. A field of all blanks is converted to all zeros.

---

*Segmented programs may not be supported in some operating systems.

Special Characters: Used for special program functions. They are:

SYMBOL	REPRESENTING
	blank
=	equals
+	plus
-	minus
*	asterisk
/	slash
(	left parenthesis
)	right parenthesis
,	comma
.	decimal point
$	currency symbol
"	quote - string delimiter

SOURCE PROGRAM LINES

Source program lines are written according to the following rules.

Lines: A line is a string of 72 characters. All characters must be from the HP ASCII character set (see Appendix I). The character positions in a line are called columns, and are consecutively numbered 1, 2, 3, ..., 72. The number indicates the sequential position of a character in the line, starting at the left and proceeding to the right.

Comment Line: A comment line is denoted by a "C" or by an "*" in column 1. A comment line is not a statement and does not effect the program in any way. A comment line beginning with "*" will be listed in mixed listings.

> EXTENSIONS TO THE STANDARD
>
> Comment lines may appear at any point in a program, including between lines of a continued statement. Comment lines beginning with a "C" will not be included in mixed listings.

Initial Line: An initial line is a line that is neither a comment line nor an end line, and that contains the digit 0 or the character blank in column 6. Columns 1 through 5 may contain a statement label or the character blank.

Debug Line: The letter D in column 1 of a line designates that line as a debug line. Compilation of debug lines is optional. Unless specifically directed to compile debug lines, the RTE FORTRAN IV compiler will treat debug lines the same as comment lines.

To cause compilation of debug lines, specify the character D as a parameter either in the FTN4 control statement (see Appendix B) or as an FTN4 invocation command option (see Appendix J). In either case, when the character D is specified, the debug lines are compiled.

Continuation Line: A continuation line is a line that contains any characters other than the digit 0 or the character blank in column 6, and does not contain the character C or $ in column 1. Any other character may be placed in column 1. Any characters may be placed in columns 2 through 5. Except for comment lines, a continuation line may follow only an initial line or another continuation line.

In all cases, a statement may be continued indefinitely (extension of the standard).

SOURCE PROGRAM STATEMENTS AND LABELS

Source program statements and statement labels are written according to the following rules.

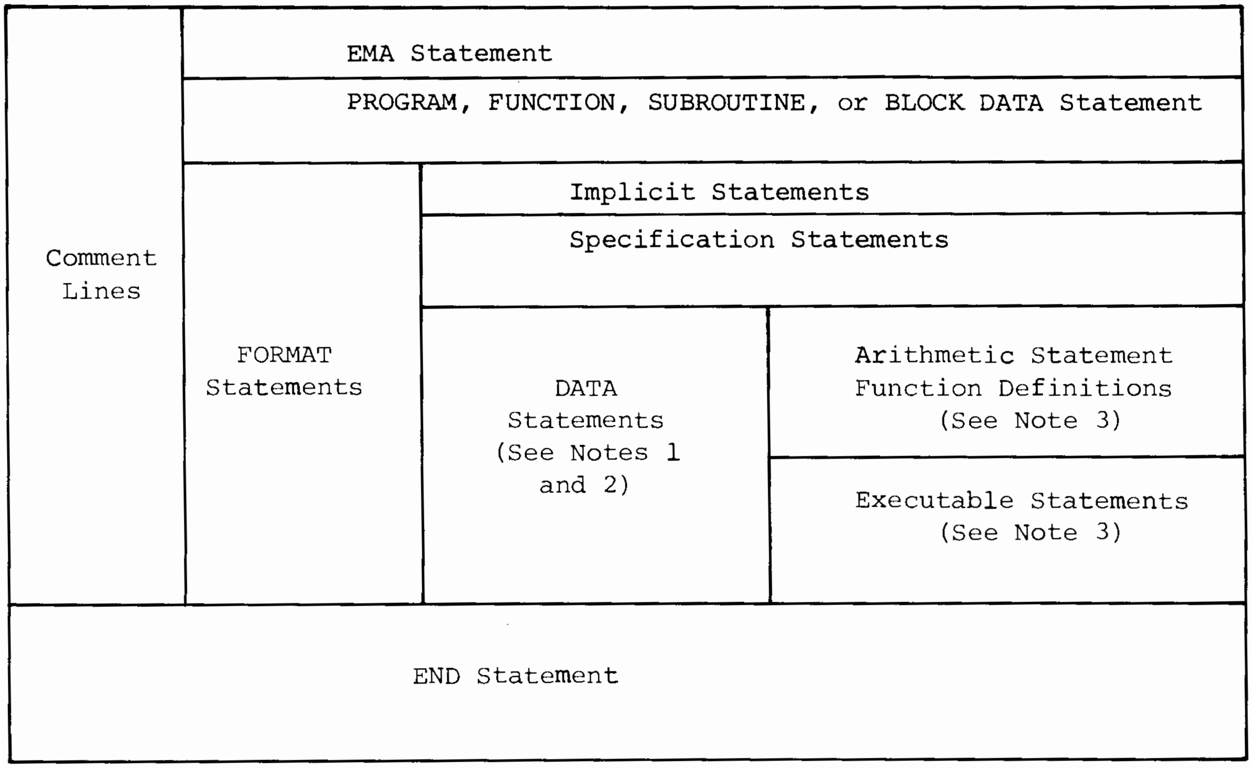
Statements: A statement consists of an initial line optionally followed by continuation lines. The statement is written in columns 7 through 72 of the lines. The order of the characters in the statement is columns 7 through 72 of the first continuation line, columns 7 through 72 of the next continuation line, etc.

Symbolic Names: A symbolic name consists of from one to six alphanumeric characters, the first of which must be alphabetic.

> External names (i.e., SUBROUTINE, FUNCTION, COMMON labels, and Main program names are shortened automatically to five characters by deletion of the fifth character. For example, the name PROG01 becomes PROG1.

ORDER OF STATEMENTS IN A SOURCE PROGRAM

The following diagram shows the source program statement ordering requirements for RTE FORTRAN IV main programs and subprograms. Statement types that must appear in a specific sequence are separated by the horizontal lines. For example, the PROGRAM statement must precede FORMAT statements, while Specification statements must precede DATA statements, and so forth. Statement types that may be interspersed with higher level statements are separated by the vertical lines. For example, Arithmetic statement function definitions and Executable statements may be interspersed with DATA statements, and so forth.



NOTES:

1. *Items in the DATA statement list are initialized at loading and not at every entrance to a program or subprogram.*

2. *Compile time is shortened if all DATA statements immediately follow the last specification statement (with no intervening arithmetic statement function definitions).*

3. *Arithmetic statement function definitions and executable statements are not allowed in block data subprograms.*

SECTION II

DATA, CONSTANTS, VARIABLES AND ARRAYS

There are six types of data in FORTRAN IV:

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

HOLLERITH

Each data type has a specific format in main memory and a unique mathematical significance and representation.

IDENTIFYING DATA TYPES

A symbolic name, called a data name, is used to reference or otherwise identify data of any type. The following rules are used when identifying data:

a. Data is named when it is identified, but not necessarily made available.

b. Data is defined when it has a value assigned to it.

c. Data is referenced when the current defined value of the data is made available during the execution of the statement that contains the data reference.

Data Type Association

The data name used to identify data carries the data type association, subject to the following restrictions:

a. A data item keeps the same data type throughout the program unit.

b. An explicit type specification overrides both the IMPLICIT specificatior (see section 4) and the default specification.

Establishing Data Names

There are different ways of establishing a data name for a data type, depending upon the type of data and how the data is used.

The form of a string representing a constant defines both the value and the type of the data. This definition is a function of how data is stored in main memory. The type of a constant is implicit in its name.

A data name that identifies a variable or an array may have its data type specified in a Type-specification. (See Section IV, "Specification Statements.") In the absence of an explicit declaration in a Type-specification, the data type is implied by the first character of the data name. The default type specifications are as follows:

```
I, J, K, L, M, or N = integer type data
any other letter    = real type data
```

This implied type specification may be changed using the IMPLICIT statement (see section IV).

Using Data Names

Data names are used to identify

VARIABLES

ARRAYS, or ARRAY ELEMENTS

FUNCTIONS (See Section IX.)

WRITING CONSTANTS, VARIABLES AND ARRAYS

The following pages describe how to write constants, variables and arrays in FORTRAN IV. See Appendix A "Formats of Data in Core Memory," for a description of how each data type is stored in main memory.

INTEGER CONSTANT

PURPOSE: An integer constant is written as a string of digits interpreted as a decimal number.

FORMAT:

$\pm$n

n

n = a decimal number with a range of -32,768 to 32,767

COMMENTS: An integer constant is signed when it is written immediately following a + or - sign. If it is unsigned, an integer constant is assumed to be positive.

EXAMPLES:

```
-32768
 32767
 0
-12
 329
+5557
```

REAL CONSTANT

PURPOSE: A real constant is written as a string of decimal digits containing an integer part, a decimal point, a decimal fraction and an exponent, in that order.

FORMAT:

$\pm$m . n Ex

m = an integer constant

. = a decimal point

n = a decimal constant representing a fraction

Ex = the character E followed by the exponent, a signed or unsigned integer

COMMENTS: The decimal exponent is a multiplier (applied to the constant written immediately before it) that is equal to the number 10, raised to the power indicated by the integer following the E.

Either m or n (but not both) may be omitted; and either the decimal point or the exponent (but not both) may be omitted from a real constant.

EXAMPLES:

1.29	0.18E+2
.00123	2E-3
-901.	1.E+15
256.177E2	-256.177E-2

DOUBLE PRECISION CONSTANT

PURPOSE: A double precision constant is written as a string of decimal digits containing an integer part, a decimal point, a decimal fraction and an exponent, in that order.

FORMAT:

$\pm$m . n Dx

m = an integer constant

. = a decimal point

n = a decimal constant representing a fraction

Dx = the character D followed by the exponent, a signed or unsigned integer

COMMENTS: The decimal exponent is a multiplier (applied to the constant written immediately before it) that is equal to the number 10, raised to the power indicated by the integer following the D.

Either m or n (but not both) can be omitted. A decimal point must separate m and n when both are specified. When m is present, both the decimal point and n can be omitted.

EXAMPLES:

```
 1.29D0
 .0123D-1
 256.17702D02
-256.17702D-2
 2D-3
```

COMPLEX CONSTANT

PURPOSE: A complex constant is composed of a real part and an imaginary part, and is written as an ordered pair of real constants, separated by a comma and enclosed in parentheses.

FORMAT:

$$(m_1 , m_2)$$

m_1 and m_2 are real constants, signed or unsigned

COMMENTS: The first real constant is the real part; the second, the imaginary part.

EXAMPLES:

```
(1.29, 256.177E-2)
(-901., 0.)
(-.123E+01, -12.3E-4)
(0., 0.)
```

LOGICAL CONSTANT

PURPOSE: A logical constant is a truth value, either true or false.

FORMAT:

```
.TRUE.
.FALSE.
```

COMMENTS: The periods must be used as shown.

EXAMPLES:

```
ITRUE = .TRUE.
```

When the above instruction is executed in an RTE Fortran IV program, the internal representation of logical true will be assigned to the variable ITRUE.

HOLLERITH CONSTANT

PURPOSE: A Hollerith constant is written as an integer constant followed by the letter H, followed by any ASCII character except carriage return.

FORMAT:

nHx

n = an integer constant

H = the Hollerith descriptor, which is the character H

x = one to n alphanumeric characters

COMMENTS: The character immediately following the H is placed in the left half of the computer word used to store the constant. The right half of the word contains the next character and so on. If n is odd, the last word will have a blank in its right half.

Hollerith constants are typed as follows:

n = 1 or 2	integer
3 or 4	real
5 or 6	double precision
7 or 8	complex
n > 8	legal only as a simple parameter in a CALL statement or a function reference, or in FORMAT statements.

EXAMPLES:

1H@	2HBB
1HA	2H$$
2H A	2H12
8HABCDEFGH	10HCALL STMT.

OCTAL CONSTANT

PURPOSE: An octal constant is written as a string of from one to six octal digits terminating with a B octal descriptor. An octal constant is an implied integer constant.

FORMAT:

$$\pm n_1 n_2 n_3 n_4 n_5 n_6 B$$

n_1 to n_6 = octal digits

B = the octal descriptor, the character B

COMMENTS: If an octal constant has more than six digits or if the leading digit in a six-digit constant is greater than one, an error diagnostic occurs.

Integers n_1 up to n_5 may be omitted if they equal 0. The octal constant may carry a sign.

EXAMPLES:

```
 21B
+00B
 0B
 177777B
-1705B
```

NOTE: The B suffix to indicate octal is an extension of the standard.

SIMPLE VARIABLE

PURPOSE: Is the symbolic name of a single value.

FORMAT:

One to six alphanumeric characters, the first of which must be a letter.

COMMENTS: If the variable has a first character of I, J, K, L, M or N, it is implicitly typed as an integer variable. All other first letters imply that the variable is real.

Implicit typing may be overridden for individual symbolic names by declaring them in a Type-specification. (See Section IV.)

EXAMPLES:

Integer	Real
I125	A125
JMAX	HMAX
MREAL	REAL
K	X

ARRAY

An array is an ordered set of data of one, two or three dimensions. An array is identified by a symbolic name called the array name. The size and number of dimensions of an array must be defined in a DIMENSION, COMMON or TYPE-statement.

ARRAY ELEMENT

An array element is a member of the array data set. The array element is identified by a subscript immediately following the array name.

An array element may be defined and referenced.

SUBSCRIPT EXPRESSIONS

A subscript expression may be any arithmetic expression allowed in FORTRAN IV. If the expression is of a data type other than integer, it is converted to integer before being used as a subscript. It must evaluate to an integer between 1 and 32767 inclusive.

In a program unit any appearance of a symbolic name that identifies an array must be immediately followed by a subscript, except in the following cases:

a. In the list of an input/output statement

b. In a list of dummy arguments

c. In the list of actual arguments in a function or subroutine reference

d. In a COMMON statement

e. In a TYPE- statement

f. In a DATA statement

SUBSCRIPT

A subscript is written as a parenthesized list of subscript expressions. Each subscript expression is separated by a comma from its successor, if there is a successor.

The number of subscript expressions must be less than or equal to the number of dimensions declared for the array name in a DIMENSION, COMMON or TYPE- statement. The value of a subscript is defined in Table 2-1, below. The value refers to the number of array elements (stored in column order) inclusively between the base entry and the one represented by the subscript.

TABLE 2-1
THE VALUE OF AN ARRAY SUBSCRIPT
(IN AN ARRAY)

ARRAY DIMENSION(S)	SUBSCRIPT DECLARATOR	SUBSCRIPT	SUBSCRIPT VALUE	*MINIMUM SUBSCRIPT VALUE	*MAXIMUM SUBSCRIPT VALUE
1	(A)	(a)	a	1	A
2	(A,B)	(a,b)	a+A*(b-1)	1	A*B
3	(A,B,C)	(a,b,c)	a+A*(b-1) +A*B*(c-1)	1	A*B*C

*Refer to warning on page 2-14.

Usage of an unsubscripted array name always denotes the first element of that array, except in an I/O statement or a DATA statement, where the entire array is referenced.

DEFINING VARIABLES AND ARRAY ELEMENTS

Variables and array elements become initially defined (before execution begins) if, and only if, their names are associated in a DATA statement with a constant of the same data type as the variable or array in question. Any entity not so defined is said to be "undefined" at the time the first executable statement in a main program is executed.

SUBSCRIPTED VARIABLE

PURPOSE: Refers to a particular element of an array of the same symbolic name as that of the subscripted variable.

FORMAT:

$$s\ (\ a_1,\ a_2,\ \ldots,\ a_n\ )$$

s = the symbolic name of the array

a = expression(s) which determine the values of the subscript(s) of the subscripted variable

n = 1, 2, or 3

COMMENTS: Subscripted variables must have their subscript bounds specified in a COMMON, DIMENSION, or TYPE- statement prior to their first appearance in an executable statement or in a DATA statement.

A subscript may be any arithmetic expression. If non-integer, the subscript is evaluated and converted to integer (by truncating) before being used as a subscript.

A subscripted variable is named and typed according to the same rules as a simple variable.

WARNING: *No check is made by the compiler to verify that array subscript values fall within declared DIMENSION bounds. Unpredictable results occur if references are made to dimensioned variables outside of the declared bounds of the array. Thus, array subscripts may not be less than one or greater than the declared array size.*

EXAMPLES:

```
A(3,5,2)          MAX (I,J)
I(10)             MIN (I-J,(I-J)*K/A,4)
ARRAY(2,5)
```

SECTION III
EXPRESSIONS

An expression is a constant, variable or function reference (see Section IX), or combination of these, separated by operators, commas or parentheses.

There are three types of expressions: arithmetic, logical and relational.

ARITHMETIC EXPRESSIONS

An arithmetic expression, formed with operators and elements, defines a numerical value. Both the expression and its elements identify integer, real, double precision or complex values.

Arithmetic Operators

The arithmetic operators are:

Symbol	Mathematic Function	Example
**	exponentiation	A**B
/	division	A/B
*	multiplication	A*B
-	subtraction (or negative value)	A-B or -A
+	addition (or positive value)	A+B or +A

Arithmetic Elements

The arithmetic elements are defined as:

PRIMARY: An arithmetic expression enclosed in parentheses, a constant, a variable reference, an array element reference or a function reference.

FACTOR: A primary, or a construct of the form:

PRIMARY**PRIMARY

TERM: A factor, or a construct of one of the forms:

TERM/FACTOR

TERM*TERM

SIGNED TERM: A term, immediately preceded by + or -

SIMPLE ARITHMETIC EXPRESSION: A term, or two simple arithmetic expressions separated by + or -.

ARITHMETIC EXPRESSION: A simple arithmetic expression or a signed term or either of the preceding forms immediately followed by + or -, followed by a simple arithmetic expression.

Combining Arithmetic Elements

When adding, subtracting, dividing or multiplying, the compiler combines arithmetic elements according to the rules shown in Table 3-1.

TABLE 3-1

RESULTS: COMBINING ARITHMETIC ELEMENTS (+,-,*,/)

FIRST ELEMENT TYPE	SECOND ELEMENT TYPE: INTEGER	REAL	DOUBLE PRECISION	COMPLEX
INTEGER	INTEGER	REAL	DOUBLE PRECISION	COMPLEX
REAL	REAL	REAL	DOUBLE PRECISION	COMPLEX
DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	COMPLEX
COMPLEX	COMPLEX	COMPLEX	COMPLEX	COMPLEX

CAUTION: Real or Integer Division by zero produces the following results:

:INTEGER/0 = ABS|INTEGER|

e.g.

K = -123/0 = 123

:REAL/0 = LARGEST REAL NUMBER

e.g.

A = 18.4/0. = .17014E+39

The overflow bit is set but does not affect the use of the result in succeeding FORTRAN statements.

NO DIAGNOSTIC WARNING OR ERROR MESSAGE IS DISPLAYED.

Exponentiation of Arithmetic Elements

Arithmetic elements can be exponentiated according to the rules shown in Table 3-2.

TABLE 3-2

RESULTS: EXPONENTIATION OF ARITHMETIC ELEMENTS (**)

	EXPONENT TYPE			
BASE TYPE	INTEGER	REAL	DOUBLE PRECISION	COMPLEX
INTEGER	INTEGER	NOT ALLOWED	NOT ALLOWED	NOT ALLOWED
REAL	REAL	REAL	DOUBLE PRECISION	NOT ALLOWED
DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE PRECISION	NOT ALLOWED
COMPLEX	COMPLEX	NOT ALLOWED	NOT ALLOWED	NOT ALLOWED

Evaluating Expressions

The compiler evaluates expressions from left to right, according to the following rules:

PRECEDENCE FROM HIGHEST TO LOWEST:

arithmetic	( )	parentheses, for grouping expressions
	**	exponentiation
	*,/	multiplication and division (whichever occurs first)
	-	unary minus
	+,-	addition and subtraction (whichever occurs first).
relational	.LT.,.LE.,.EQ.,.NE.,.GT.,.GE. (whichever occurs first).	
logical	.NOT.	
	.AND.	
	.OR.	

SEQUENCE: Evaluation begins with the subexpression most deeply nested within parentheses.

Within parentheses, subexpressions are evaluated from left to right in the order of precedence above.

Function references are evaluated from left to right as they occur.

No factor is evaluated that requires a negative valued primary to be raised to a real or double precision exponent. No factor is evaluated that requires raising a zero valued primary to a zero valued exponent. No element is evaluated if its value has not been mathematically defined. Integer overflow resulting from arithmetic operations is not detected at execution time.

LOGICAL EXPRESSIONS

A logical expression is a rule for computing a logical value. It is formed with logical operators and logical elements and has the value true or false.

Logical Operators

The logical operators and the logical result of their use in an expression are:

Symbol	Mathematic Function	Example
.OR.	LOGICAL DISJUNCTION	A .OR. B
.AND.	LOGICAL CONJUNCTION	A .AND. B
.NOT.	LOGICAL NEGATION	.NOT.A

Logical Expression (logical elements A and B)	LOGICAL RESULT IS TRUE	LOGICAL RESULT IS FALSE
A .OR. B	If either A or B is true	If both A and B are false
A .AND. B	If both A and B are true	If either A or B is false
.NOT. A	If A is false	If A is true

Logical Elements

The logical elements are defined as:

LOGICAL PRIMARY: A logical expression enclosed in parentheses, a relational expression, a logical constant, a logical variable reference, a logical array element reference, or a logical function reference.

LOGICAL FACTOR: A logical primary, or .NOT. followed by a logical primary.

LOGICAL TERM: A logical factor or a construct of the form:

LOGICAL TERM .AND. LOGICAL TERM

LOGICAL EXPRESSION: A logical term or a construct of the form:

LOGICAL EXPRESSION .OR. LOGICAL EXPRESSION

RELATIONAL EXPRESSIONS

A relational expression is a rule for computing a conditional logical expression. It consists of two arithmetic expressions separated by a relational operator. The relation has the value true or false as the relation is true or false. The operands of a relational operator must be of type integer, real, or double precision, except that the operators .EQ. and .NE. may have operands of type complex.

Relational Operators

The relational operators are:

Symbol	Mathematic Function	Example
.LT.	less than	A .LT. B
.LE.	less than or equal to	A .LE. B
.EQ.	equal to	A .EQ. B
.NE.	not equal to	A .NE. B
.GT.	greater than	A .GT. B
.GE.	greater than or equal to	A .GE. B

EXAMPLE: If A = 5 and B = 3, then

(A .LT. B) is false

((A .LE. B) .OR. (B .LE. A)) is true

CAUTION: The relational operators .LT., .LE., .GT., and .GE. may cause an integer overflow when executed. This will not be detected at execution time. If overflow is anticipated (i.e., the variables to be compared may be more than 32767 apart), they may still be correctly tested by FLOATing them prior to the test. For example, (I .LT. J) would become (FLOAT(I) .LT. FLOAT(J)).

The object code generated by this compiler for relational operators on integers is as follows:

I .LT. J	I .LE. J	I .EQ. J	I .NE. J	I .GT. J	I .GE. J
LDA J	LDA I	LDA I	LDA I	LDA I	LDA J
CMA,INA	CMA,INA	CPA J	CPA J	CMA,INA	CMA,INA
ADA I	ADA J	CCA,RSS	CLA,RSS	ADA J	ADA I
	CMA	CLA	CCA		CMA

SECTION IV
SPECIFICATION STATEMENTS

Specification statements are non-executable statements that specify variables, arrays and other storage information to the compiler. There are six specification statements in FORTRAN IV. It is recommended, but not required, that specification statements be used in the following order:

```
IMPLICIT
TYPE-
DIMENSION
COMMON
EQUIVALENCE
EXTERNAL
DATA
```

Refer to section I on Order of Statements in a Source Program for a complete explanation of the ordering requirements.

ARRAY DECLARATOR

DIMENSION, COMMON and TYPE- statements use array declarators to specify the arrays used in a program unit. An array declarator indicates the symbolic name of the array, the number of dimensions (one, two or three), and the size of each array dimension. An array declarator has the following format:

v (i)

v = the symbolic name of the array

i = one, two or three declarator subscripts (for one, two or three dimensional arrays). Each subscript must be an integer constant or a dummy integer variable name. (See Section IX.)

If a two or a three dimensional array is being specified, each declarator subscript is separated from its successor by a comma.

The values given for the declarator subscripts indicate the maximum value that the subscripts can attain in any array element name. The minimum value is always one; the maximum value is 32767.

EXTERNAL

PURPOSE: To declare external function or subroutine names that will be referenced in the program unit.

FORMAT:

EXTERNAL $v_1, v_2, \ldots, v_n$

v = any external function or subroutine name

COMMENTS: If an external function or subroutine name is used as an argument to another external function or subroutine, it must appear in an EXTERNAL statement in the program unit in which it is so used.

NOTE: EXTERNAL names are limited to five characters in length. Names of six characters are shortened automatically to five by deletion of the fifth character.

EXAMPLES:

```
EXTERNAL SIN, IS, FUN
```

TYPE-SPECIFICATION

PURPOSE: To declare the data type of variable names, array names, function names or array declarators used in a program unit.

FORMAT:

```
INTEGER
REAL
DOUBLE PRECISION     }   v1, v2, ..., vn
COMPLEX
LOGICAL
```

v = a variable, array, function, or array declarator.

COMMENTS: Subroutine names cannot appear in a Type-specification statement.

The same symbolic name may not appear in a second Type-specification statement with a different type.

A Type-specification statement can be used to override or confirm the implicit typing of integer or real data and must be used to declare the data type for double precision, complex or logical data.

A symbolic name in a Type-specification statement informs the compiler that it is of the specified data type for all appearances in the program unit.

EXAMPLES:

```
INTEGER I,A,ARRAY(3,5,2)
REAL MAX, UNREAL, R(5)
DOUBLE PRECISION D, DOUBLE(2), DARRAY(3,3)
COMPLEX C, CPLEX, CARRAY(2,3,4), CAREA
LOGICAL T, FALSE, L(4), J
```

DIMENSION

PURPOSE: To specify the symbolic names and dimension(s) of arrays used in a program unit.

FORMAT:

DIMENSION $v_1(i_1)$, $v_2(i_2)$, ..., $v_n(i_n)$

v(i) = an array declarator

COMMENTS: Every array in a program unit must be specified in a DIMENSION, TYPE or COMMON statement.

WARNING: *No check is made by the compiler to verify that array subscript values fall within declared DIMENSION bounds. Unpredictable results occur if references are made to dimensioned variable outside of the declared bounds of the array. Thus, array subscripts may not be less than one or greater than the declared array size.*

EXAMPLES:

```
DIMENSION MATRIX(3,3,3)
DIMENSION I(4), A(3,2)
```

COMMON

PURPOSE: To provide a means for sharing a common block of memory between a main program and its subprograms, or between subprograms. A block of common memory labeled by a name refers to block common. A block without a label refers to blank common.

FORMAT:

$COMMON/blockname_1/a_1,...,a_n ... /blockname_n/a_1,...,a_n$

$COMMON// a_1,...,a_n$

$COMMON\ a_1,...,a_n$

blockname = a symbolic common block name delimited with slash characters.

// = a blank common block.

a = a variable or array name, or an array declarator.

COMMENTS: A symbolic name in a COMMON statement must be a variable or array name, or an array declarator. Once declared in a COMMON statement, a name cannot be declared in another COMMON statement within the same program unit.

The size of a common block is the sum of the storage required for the elements introduced through COMMON and EQUIVALENCE statements in a program unit. Common entities are strung together in the order in which they are declared.

A blank common block is declared by specifying a null block name (//). If a blank common block is declared as the first block in a COMMON statement, the slashes can be omitted.

COMMENTS: (cont.)

Blank common is available to every module of a program. Each module must completely describe all entries in any common block that it references. In multiprogramming systems, blank common and/or block common may be available to more than one program.

By using named common blocks, the program may group together similar data constructs and set up the programs common area so that only the data of interest to a given module need be declared.

Named common blocks, except EMA common, must be described in a BLOCK DATA subprogram. Furthermore, the required BLOCK DATA subprogram may initialize named common blocks while blank common blocks cannot be initialized.

EXAMPLES:

```
COMMON I,CAREA(2,3),J(3)/HELLO/W,X(2,5),Z/BYE/A
COMMON/HELLO/KK(10)//Q,P
```

I, CAREA, and J are in blank common. W, X, and Z are in a common block named HELLO. A is in a common block named BYE. KK follows Z in a common block named HELLO. Q and P follow J in blank common.

For an example of HP implementation of named common, see Appendix F.

EXTENDED MEMORY AREA (EMA) DIRECTIVE

PURPOSE: To provide a means for the storage and manipulation of large amounts of data, up to the total amount of available physical memory. Available in RTE-IV only.

FORMAT:

$EMA (blockname,mseg)

where:

$ The dollar sign ($) must appear in column 1.

blockname is the symbolic name of a block common area to be further defined in one or more COMMON statements.

mseg is the size in pages of the RTE MSEG. If 0 or not specified, MSEG is the default size determined at load time (default MSEG = maximum logical address space - program size-1). For more information on MSEG refer to the RTE-IV Programmer's Reference Manual. The EMA directive is an extension to the ANSI standard.

COMMENTS: The EMA common is a memory access method that allows very quick referencing and manipulation of large amounts of data. The size of the EMA may be as large as all of available physical memory. Refer to the RTE-IV Programmer's Reference Manual.

The EMA directive must be the first non-comment statement in the module. The common blockname must not be initialized and the EMA directive is not allowed in a BLOCK DATA subprogram. Only one EMA directive per module is allowed, and must appear in each module that references in EMA variable. All variables specified in the common block will go into the EMA.

COMMENTS: (cont.)

An EMA variable is referenced within a main program like any other variable except when being passed to other subroutines or functions. When calling subroutines which do not expect EMA parameters, e.g. EXEC, the user must take care to pass EMA variables "by value". Call by value is indicated by enclosing the variable in an extra layer of parentheses, e.g. F((x)) or by passing the variable as part of an arithmetic expression, e.g. F(x+0.). The arguments of functions listed in Table G-2, Appendix G, and the arguments of statement functions are always passed by value regardless of parentheses.

The implication of "call by value" is that only the value of the variable is available to the subroutine. Therefore, the variable may not be modified by the subroutine. Also, an EMA array may not be passed as an argument to a subroutine or function.

For subroutines expecting EMA variables, arguments may be passed by reference. "Call by reference" implies that the variable itself with its value is available and can be altered by the subroutine. To modify EMA variables and/or pass EMA arrays, EMA variables may be referenced within a subroutine or function in one or both ways:

1. By declaring the EMA common inside the subroutine or function.

2. By declaring formal parameters to be type EMA (see EMA statement) and passing the actual arguments "by reference".

An additional restriction on EMA variables is that they may not be used as format specifiers in READ or WRITE statements. For example, if J is an EMA variable, the following code is illegal:

```
10    FORMAT (.......)
      ASSIGN 10 TO J
      WRITE  (1,J)
```

An EMA variable may be equivalenced the same as any other variable in a common block. The same restrictions apply. Refer to the EQUIVALENCE statement elsewhere in this section.

NOTES ON USAGE OF EMA

While any variable may be declared to be in EMA, it is recommended that the user restrict EMA usage to those arrays which require a large area. Since references to EMA variables take longer than references to local variables, this policy will speed the execution of programs.

EXAMPLE PROGRAM ILLUSTRATING THE USE OF EMA

```
FTN4,L
$EMA(XYZ,3)
      PROGRAM TEST
      COMMON /XYZ/A(100,200),C(3000,80),E(200,300)
      EQUIVALENCE (A(99,1000),B)
        :
      B=SIN(A(J,K))
C     CALL BY VALUE TO UFUN
      D=UFUN((A(J,K)))
        :
C     PASS SUBSCRIPTS FOR EMA ARRAYS TO SUBROUTINE ADD1
C     SUBR ADD1 HAS EMA ARRAYS DEFINED IN NAMED COMMON
      CALL ADD1 (J,K)
        :
C     PASS EMA ARRAY E BY REFERENCE WITH ITS
C     DIMENSIONS TO SUBROUTINE ADD2
      CALL ADD2 (E,200,300,SUM)
        :
      END
      FUNCTION UFUN(X)
C     SQUARE THE NUMBER
      UFUN = X * X
      RETURN
      END
$EMA(XYZ,3)
      SUBROUTINE ADD1(M,N)
C     M AND N ARE SUBSCRIPT PARAMETERS
      COMMON /XYZ/A(100,200),C(3000,80),E(200,300)
C     INCREMENT AN ELEMENT IN THE EMA ARRAY A
      A(M,N) = A(M,N) + 1
        :
      RETURN
      END
```

```
      SUBROUTINE ADD2(EPRIME,ME,NE,SUM)
C     EPRIME IS AN EMA ARRAY PASSED BY REFERENCE AND SUM IS NON-EMA
C     NOTE THAT SUBROUTINE ADD2 DOES NOT REQUIRE A
C     $EMA DIRECTIVE OR ANY EMA NAMED COMMON BLOCKS
      EMA EPRIME(ME,NE)
        :
      J=1
      DO 100 I=1,NE
      EPRIME(J,I) = EPRIME(J,I) + 2
100   CONTINUE
        :
      RETURN
      END
```

Arrays A, C, and E are in EMA common because they are in the block common named XYZ, which is declared in the EMA directive. B is in EMA it is equivalenced to A. EPRIME is a formal parameter declared to be in EMA by the EMA statement.

The call to SIN may use standard notation because SIN is in Table G-2. The call to UFUN must use "call by value" because its parameter is not declared in an EMA statement. This is indicated by enclosing its argument in an extra layer of parentheses as shown. An element in array A is incremented in Subroutine ADD1, which has declared the EMA common block. The array E is passed by reference to Subroutine ADD2, which has declared the formal parameter, EPRIME, to be in EMA.

EXTENDED MEMORY AREA (EMA) STATEMENT

PURPOSE: To declare that formal parameters are located in EMA and have been passed by reference. Available in RTE-IV only.

FORMAT:

EMA v1,v2,...,vn

v = a variable, array or array declarator which is a formal parameter.

The EMA statement is an extension to the ANSI standard.

COMMENTS: Since variables in EMA are accessed by a different mechanism than those not in EMA, it is necessary to specify which formal parameters are EMA parameters to the compiler. The default type for formal parameters is non-EMA. See the EMA directive for a discussion of call by value and call by reference.

WARNING: The addressing mode (EMA or non-EMA) of actual and formal parameters must match. If they do not, an incorrect address will be used. The effect will be similar to accessing an array with a subscript of unknown value. Therefore, do not pass a non-EMA variable to a subroutine expecting an EMA argument or vice versa.

EXAMPLE: EMA EARRAY(100,1000),EVAR,IARR(5000)

EQUIVALENCE

PURPOSE: Allows the sharing of memory locations by two or more entities.

FORMAT:

EQUIVALENCE (k_1), (k_2), ..., (k_n)

k = a list of two or more variable names, array names or array element names with integer constant subscripts.

COMMENTS: A symbolic name which appears in an EQUIVALENCE statement must be a variable, array, or array element name.

Equivalence can be established between different data types, but the EQUIVALENCE statement cannot be used to equate two or more entities mathematically.

> *CAUTION: RTE FORTRAN IV does not use the same amount of storage for INTEGER and REAL variables (see Appendix A). Therefore, mixed variable types should be equivalenced with caution.*

The EQUIVALENCE statement can associate a variable in COMMON with one or more variables not in COMMON, or may associate two or more variables none of which are in COMMON.

No equivalence grouping is allowed between two entities in COMMON. Dummy parameters may not appear in EQUIVALENCE statements. A variable not in COMMON, when equivalenced to a variable in COMMON, becomes a part of the COMMON area. A COMMON area, however, only can be lengthened by equivalence groupings. If an equivalence grouping causes an entity to be relocated before the first entity in COMMON, an error diagnostic occurs.

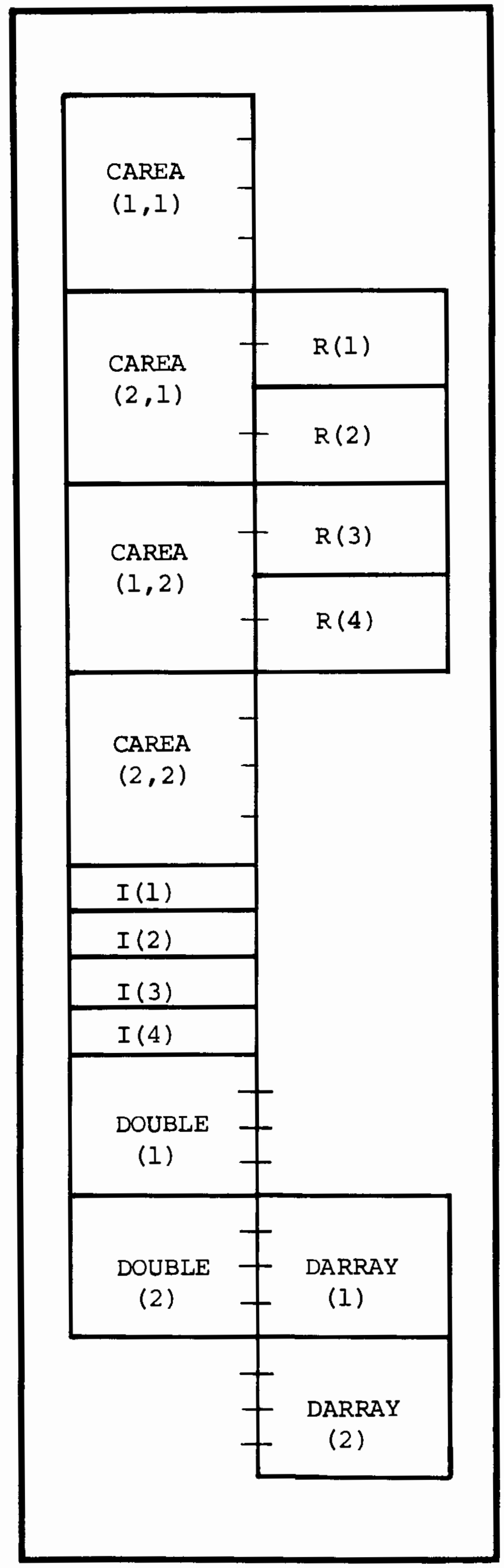
EXAMPLES:

See the following page for examples of correct equivalence grouping.

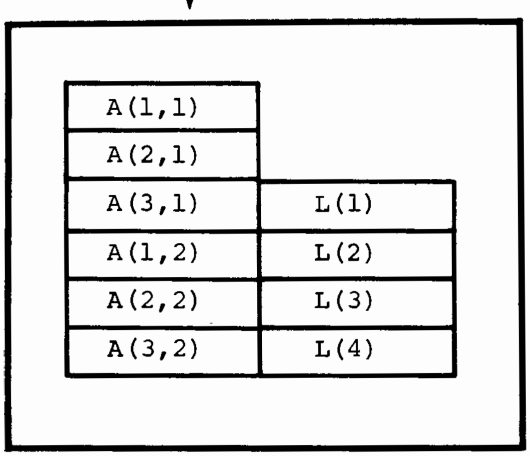
The following statements will result in the allocation of space for variables in COMMON and non-COMMON areas as shown. Double precision is assumed to be 4-word.

```
INTEGER I, A, ARRAY
REAL R(4)
COMPLEX CAREA
LOGICAL L
DOUBLE PRECISION DOUBLE(2), DARRAY
DIMENSION DARRAY(2)
DIMENSION I(4),A(3,2), L(4)
COMMON CAREA(2,2), I, DOUBLE
EQUIVALENCE (CAREA(2,1),R),(DOUBLE(2),DARRAY)
EQUIVALENCE (A (3,2), L(4) )
```

Results in this COMMON and equivalenced area of 32 words (28 words in original COMMON, 4 added by EQUIVALENCE).



Results in this non-COMMON equivalenced area of six words.



DATA

PURPOSE: To define the initial values of variables, single array elements, portions of arrays or entire arrays.

FORMAT:

DATA $k_1/d_1/, k_2/d_2/, \ldots, k_n/d_n/$

k = lists of names of variables, array elements or arrays

d = lists of constants (optionally signed) which can be immediately preceded by an integer constant (followed by an asterisk) identifying the number of times the constant is to be repeated.

/ = separators, used to bound each constant list

COMMENTS: Mixed mode assignments are not permitted. The DATA statement may only assign values that agree in mode to their identifiers. Hollerith data can be assigned to any variable provided that the data fits into that variables allocated storage space. Hollerith data is padded with blanks on the right to fill the allocated storage space.

If you use a DATA statement within a serially reusable program, the data may not be the same each time the program is reused because the DATA statement elements are not initialized upon re-entry into the program.

COMMENTS: (cont.)

If a list contains more than one entry, the entries must be separated by commas. An initially-defined variable, array element or array may not be in a common area, nor can it be a dummy argument, except that in a block data subprogram, all entries must be in a named common block.

DATA statements must come after all specification statements in the program.

> *NOTE: Unsubscripted array names are allowed in DATA statements. If the array has n elements, the next n constants from the list are used to initialize the array (in column order). If the remainder of the constant list has m<n elements in it, then only the first m elements of the array are initialized.*

EXAMPLES:

1)
```
DIMENSION IA(2,3),IB(3)
DATA IA/1,2,3,4,5,6/,X/1.9E-1/,IB/3*2/
```

The above data statement will assign values to the variables as follows:

```
IA(1,1)=1 IA(2,1)=2 IA(1,2)=3 IA(2,2)=4 IA(1,3)=5 IA(2,3)=6
X=.19
IB(1)=2 IB(2)=2 IB(3)=2
```

2)
```
DATA FALSE,ICHAR/.FALSE.,2HXY/,DBLE/-2.39D-01/
```

The above data statement will assign values to the variables as follows:

FALSE = <internal representation of boolean false>

ICHAR = <hollerith character string XY>

DBLE = -.239 represented as a double precision number.

IMPLICIT Statement

PURPOSE: To change or confirm the default implicit integer and real typing of variables.

FORMAT:

IMPLICIT *type*(*a*[,*a*]....)[,*type*(*a*[,*a*]....)]....

where:

type is one of INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL

a is either a single letter or a range of single letters in alphabeticl order. A range is denoted by the first and last letters of the range separated by a minus sign; e.g., a_1-a_2. A range will specify the default type of all identifiers beginning with letters in the interval a_1 to a_2, inclusive.

COMMENTS: An IMPLICIT statement specifies a type for all variables, arrays, and functions (except intrinsic functions) that begin with any letter that appears in the specification, either as a single letter or included in a range of letters. IMPLICIT statements do not change the type of any intrinsic functions. An IMPLICIT statement applies only to the program unit that contains it.

Type specification by an IMPLICIT statement may be overridden or confirmed for any particular variable, array, or function name by the appearance of that name in a type-statement. An explicit type specification in a FUNCTION statement overrides an IMPLICIT statement for the name of that function subprogram.

Within the specification statements of a program unit, IMPLICIT statements must precede all other specification statements. A program unit may contain any number of IMPLICIT statements.

The same letter must not appear as a single letter, or be included in a range of letters, more than once in all of the IMPLICIT statements in a program unit.

SECTION V
ASSIGNMENT STATEMENTS

Assignment statements are executable statements that assign values to variables and array elements. There are three types of assignment statements:

Arithmetic assignment statements
Logical assignment statements
ASSIGN TO statement

ARITHMETIC ASSIGNMENT STATEMENT

PURPOSE: Causes the value represented by an arithmetic expression to be assigned to a variable.

FORMAT:

v = e

v = a variable name or an array element name of any data type except logical

e = any arithmetic expression

COMMENTS: v is altered according to the rules expressed in Table 5-1, A variable must have a value assigned to it before it can be referenced.

EXAMPLES:

```
K = 2HAB
A(I,J,K)=SIN(X)*2.5-A(2,1,3)
I=1
```

Table 5-1.

RULES FOR ASSIGNING e to v

If v Type Is	And e Type Is	The Assignment Rule Is
Integer	Integer	Assign
Integer	Real	Fix & Assign
Integer	Double Precision	Fix & Assign
Integer	Complex	Fix Real Part & Assign
Real	Integer	Float & Assign
Real	Real	Assign
Real	Double Precision	DP Evaluate & Real Assign
Real	Complex	Assign Real Part
Double Precision	Integer	DP Float & Assign
Double Precision	Real	DP Evaluate & Assign
Double Precision	Double Precision	Assign
Double Precision	Complex	DP Evaluate Real Part & Assign
Complex	Integer	Convert & Assign as Real Part With Imaginary Part = 0
Complex	Real	Convert & Assign as Real Part With Imaginary Part = 0
Complex	Double Precision	Convert & Assign as Real Part With Imaginary Part = 0
Complex	Complex	Assign

NOTES:

1. Assign means transmit the resulting value, without change, to the entity.
2. Real Assign means transmit to the entity as much precision of the most significant part of the resulting value as a real datum can contain.
3. DP Evaluate means evaluate the expression then DP Float.
4. Fix means truncate any fractional part of the result and transform that value to the form of an integer datum.
5. Float means transform the value to the form of a real datum.
6. DP Float means transform the value to the form of a double precision datum, retaining in the process as much of the precision of the value as a double precision datum can contain.

LOGICAL ASSIGNMENT STATEMENT

PURPOSE: Causes the value represented by the logical expression to be assigned to a simple or subscripted variable.

FORMAT:

```
v = e
```

v = a logical variable name or a logical array element name

e = a logical expression

COMMENTS: A variable must have a value assigned to it before it can be referenced.

EXAMPLES:


```
T = .TRUE.
FALSE = .FALSE.
T = A.LT.B
```

ASSIGN TO STATEMENT

PURPOSE: Initializes an INTEGER Variable to a statement label.

FORMAT:

```
ASSIGN k TO i
```

k = a statement label

i = an integer variable name

COMMENTS: After the ASSIGN TO statement is executed, any subsequent execution of an assigned GO TO statement using the integer variable causes the statement identified by the assigned statement label to be executed next. The integer variable may also be used in a READ or WRITE statement as the format identifier.

STANDARD extension. The INTEGER variable may be used in a CALL statement or function reference and the dummy assigned its value may be used in an assigned GO TO, READ, or WRITE statement.

Once mentioned in an ASSIGN TO statement, an integer variable may not be referenced in any statement other than an assigned GO TO statement or as a format reference in a READ or WRITE statement until it has been redefined.

EXAMPLES:

```
      ASSIGN 1234 TO ILABEL
      .
      .
      .
      GO TO ILABEL, (100,1234,200)     (or, GO TO ILABEL)
      .
      .
      .
 1234 I = 1
```

SECTION VI
CONTROL STATEMENTS

Normally, a program begins with the execution of the first executable statement in the program. When the execution of that statement is completed, the next sequential executable statement is executed. This process continues until the program ends.

A subprogram, if referenced, starts with its first executable statement, then executes the next sequential executable statement, and so on, until it returns control to the program statement which referenced it.

Control statements are executable statements that alter the normal flow of a program or subprogram. There are twelve control statements in FORTRAN IV.

GO TO (Unconditional)
GO TO (Assigned)
GO TO (Computed)
IF (Arithmetic)
IF (Logical)
CALL
RETURN
CONTINUE
PAUSE
STOP
DO
END

GO TO

UNCONDITIONAL

PURPOSE: Causes the statement identified by the statement label to be executed next.

FORMAT:

```
GO TO k
```

k = a statement label

COMMENTS: The program continues to execute from the statement identified by k.

EXAMPLE:

```
GO TO 1234
```

GO TO

ASSIGNED

PURPOSE: Causes the statement identified by the current value of an integer variable reference to be executed next.

FORMAT:

```
GO TO i, (k1, k2, ..., kn)
GO TO i
```

i = an integer variable reference

k = a statement label

COMMENTS: The current value of i must have been assigned by a previous execution of an ASSIGN TO statement.

The compiler does not check if i contains one of the statement labels in the list. The list is for programmer's documentation purposes only. The values k_1, k_2, ..., k_n are checked to ensure that they are valid statement numbers.

EXAMPLE:

```
ASSIGN 1234 TO ILABEL
  .
  .
  .
GO TO ILABEL, (1234,200,100)     (or, GO TO ILABEL)
```

GO TO

COMPUTED

PURPOSE: Causes the statement identified by an indexed label from a list of labels to be executed next.

FORMAT:

```
GO TO (k1, k2, ..., kn), e
```

$GO\ TO\ (k_1, k_2, \ldots, k_n), e$

k = a statement label

e = an arithmetic expression

COMMENTS: The expression is evaluated, and converted to integer, if necessary.

If the expression value is less than one, statement k_1 is executed. If the expression value is greater than n, statement k_n is executed. If $1 \leq e \leq n$, statement k_e is executed.

EXAMPLE:

```
    GO TO (100,200,300), k
100 CONTINUE    (if k ≤ 1)
200 CONTINUE    (if k = 2)
300 CONTINUE    (if k ≥ 3)
```

IF

ARITHMETIC

PURPOSE: Causes one of two or three statements to be executed next, depending upon the value of an arithmetic expression.

FORMAT:

IF (e) k_1, k_2, k_3

IF (e) k_1, k_2

e = an arithmetic expression of type integer, real or double precision.

k = a statement label

COMMENTS: When the statement contains three statement labels, the statement identified by the label k_1, k_2, or k_3 is executed next if the value of e is less than zero, equal to zero, or greater than zero, respectively.

When the statement contains two statement labels, the statement identified by k_1 is executed next when the value of e is less than zero; k_2 is executed next when the value of e is equal to or greater than zero.

EXAMPLES:

```
IF (A - B) 100, 200, 300
IF (SIN(X) - A*B) 100,200
```

IF

LOGICAL

PURPOSE: Causes a statement to be executed next if a logical expression is true, or causes one of two statements to be executed, depending upon the value of the logical expression.

FORMAT:

```
IF (e) s
IF (e) k1, k2
```

s = an executable statement (except a DO or a logical IF)

e = a logical expression

k = a statement label

COMMENTS: If the logical expression is true (first format), statement s is executed. If s does not transfer control elsewhere, execution then continues with the statement following the IF. If e is false, the statement s is not executed, but the next sequential statement after the IF is executed.

If the logical expression is true (second format), statement k_1 is executed. If the logical expression is false, statement k_2 is executed.

Refer to the sections on logical expressions and relational expressions for a further explanation. Note particularly the caution on the use of the relational operators .LT., .LE., .GT., and .GE..

EXAMPLES:

```
IF (A .EQ. B) A = 1.0
IF (SIN(X) .LE. (A-B)) 100,200
```

CALL

PURPOSE: Causes a subroutine to be executed.

FORMAT:

```
CALL s
CALL s (a1, a2, ..., an)
```

CALL s (a_1, a_2, ..., a_n)

s = the name of a subroutine

a = an actual argument

COMMENTS: When the subroutine returns control to the main program, execution resumes at the statement following the CALL.

An actual argument is a constant, a variable name, an array name, an array element name, expression or subprogram name. Actual arguments in a CALL statement must agree in order, type and number with the corresponding dummy parameters in a subroutine. (See Section IX.)

EMA variables appearing as an actual argument must be passed using "call by value". Refer to the section on the EMA statement for more information.

EXAMPLES:

```
CALL MATRX              SUBROUTINE MATRX
  :                       :
CALL SUBR (I, J)        RETURN
                        END
                        SUBROUTINE SUBR (I,J)
                          :
                        RETURN
                        END
```

RETURN

PURPOSE Causes control to return to the current calling program unit, if it occurs in a function subprogram or a subroutine. Causes the program to stop if it occurs in a main program.

FORMAT:

```
RETURN
```

COMMENTS: When the RETURN statement occurs in a subroutine, control returns to the first executable statement following the CALL statement that referenced the subroutine.

When the RETURN statement appears in a function subprogram, control returns to the referencing statement. The value of the function is made available in the expression which referenced the function subprogram.

The END statement of a function subprogram or a subroutine is also interpreted as a RETURN statement, provided there is a path to the END statement.

EXAMPLES:

```
CALL MATRX
  .
  .
  .
I = MIX(L,M)/A*B
  .
  .
  .
RETURN
```

```
SUBROUTINE MATRX
  .
  .
  .
RETURN
END
INTEGER FUNCTION MIX(I,J)
  .
  .
  .
MIX = I + J
RETURN
END
```

CONTINUE

PURPOSE: Causes continuation of the program's normal execution sequence.

FORMAT:

```
CONTINUE
```

COMMENTS: The CONTINUE statement can be used as the terminal statement in a DO loop.

If used elsewhere, the CONTINUE statement acts as a dummy statement which causes no action on the execution of a program.

EXAMPLE:

```
      DO 5 I = 1, 5
      .
      .
      .
5      CONTINUE
```

STOP

PURPOSE: Causes the program to stop executing.

FORMAT:

```
STOP n
STOP
```

n = an octal digit string of one to four characters

COMMENTS: When this statement is executed, STOP is printed on the teleprinter output unit. If n is given, its value is also printed, after the word STOP.

EXAMPLES:

```
STOP 1234
STOP
```

PAUSE

PURPOSE: Causes the program to stop executing. Execution is resumable in sequence.

FORMAT:

```
PAUSE
PAUSE n
```

n = an octal digit string of one to four characters

COMMENTS: When this statement is executed, PAUSE is printed on the teleprinter output unit. If n is given, its value is also printed, after the word PAUSE.

The decision to resume processing is not under program control. To restart, a system directive must be issued by the system operator.

EXAMPLES:

```
PAUSE 1234
PAUSE
```

DO

PURPOSE: To initiate and control the sequence of instructions in a programmed loop.

FORMAT:

DO n [,] i = m_1, m_2, m_3

DO n [,] i = m_1, m_2

n = the statement label of an executable statement (called the terminal statement)

[,] = an optional comma

i = a simple integer variable name (called the control variable)

m_1 = an arithmetic expression (called the initial parameter)

m_2 = an arithmetic expression (called the terminal parameter)

m_3 = an arithmetic expression (called the step-size parameter)

COMMENTS: The terminal statement must physically follow and be in the same program unit as the DO statement. The terminal statement may not be any form of a GO TO, an arithmetic IF, a two-branch logical IF, a RETURN, STOP, PAUSE, DO or a logical IF statement containing any of these statements.

The initial, terminal and step-size parameters can be any arithmetic expressions. However, if these expressions are not of type integer, they are converted to integer (by truncation) after they are evaluated.

CAUTION: The maximum allowable difference between the initial parameter and the terminal parameter is 32,767 (2^{15}-1). If more iterations are desired, two or more DO loops can be nested to achieve this (see Example d following).

If the step-size parameter is omitted (format 2), a value of +1 is implied for the step size.

NOTE: The step-size may be positive or negative, allowing either incrementing or decrementing to the terminal parameter value.

COMMENTS: (cont.) The range of a DO statement is from (and including) the first executable statement following the DO to (and including) the terminal statement of the DO.

When the range of one DO statement contains another DO statement, the range of the contained DO must be a subset of the range of the containing DO.

Succeeding executions of the DO loop do not cause re-evaluation of the initial, terminal or step-size parameters if they are expressions. Therefore, any changes made within the DO loop to the values of variables occurring in these expressions do not affect the control of the loop's execution. Only changes to the control variable itself or to step-size parameters (if they are unsigned simple integer variables) affect the loop's execution.

> *NOTE: A DO statement is executed at least once regardless of the relationship of the initial parameter to the terminal parameter.*

If a subprogram reference occurs in the range of a DO, the actions of that subprogram are considered to be temporarily within that range.

When a statement terminates more than one DO loop, the label of that statement may be used only in a GO TO or arithmetic IF statement that occurs in the range of the most deeply nested DO that ends with that terminal statement. Other control flows can be achieved by having separate terminal statements for DO loops.

EXAMPLES:

a)
```
    DO 5I=1,5
    :
  5 CONTINUE
```

b)
```
    DO 20 I=1,10,2
    :
    DO 20 J=1,5
    :
 20 CONTINUE
```

c)
```
    DO 20 I=1,10,2
    :
    DO 15 J=2,5
    :
 15 CONTINUE
    :
 20 CONTINUE
```

d)
```
    DO 100 I=1,200
    DO  50 J=1,250
    A(I,J)=A(I,J)+1
 50 CONTINUE
100 CONTINUE
```

Array A declared to be in EMA.

The following occurs when a DO statement is executed:

a. The control variable is assigned the value represented by the initial parameter. The DO loop is executed at least once regardless of the relationship of the initial parameter to the terminal parameter value.

b. The range of the DO is executed.

c. If control reaches the terminal statement, then after execution of the terminal statement, the control variable of the most recently executed DO statement associated with the terminal statement is modified by the value represented by the associated step-size parameter.

d. If the value of the control variable (after modification by the step-size parameter) has not gone past the value represented by the associated terminal parameter, then the action described starting as step b. is repeated, with the understanding that the range is that of the DO whose control variable has been most recently modified. If the value of the control variable has gone past the value represented by its associated terminal parameter, then the DO is said to have been satisfied.

e. At this point, if there were one or more other DO statements referring to the terminal statement in question, the control variable of the next most recently executed DO statement is modified by the value represented by its associated step-size parameter and the action in step d. is repeated until all DO statements referring to the particular terminal statement are satisfied, at which time the first executable statement following the terminal statement is executed.

f. Upon exiting from the range of a DO by the execution of a GO TO or an arithmetic IF statement (that is, by exiting other than by satisfying the DO), the control variable of the DO is defined and is equal to the most recent value attained as defined in steps a. through e.

END

PURPOSE: Indicates to the compiler that this is the last statement in a program unit.

FORMAT:

END

COMMENTS: Every program unit must terminate with an END statement.

The characters E, N and D (once each and in that order in columns 7 through 72) can be preceded by, interspersed with, or followed by blank characters; column 6 must contain a blank character. Columns 1 through 5 may contain either a statement label or blank characters. Undefined source program statement numbers are printed when the END statement is encountered. External names shortened from six characters to five characters are reported as well as any user supplied names that conflict with implicit library names.

EXAMPLES:

```
^^^^^^END

^^^^^^E^^N^^D

^^100^END
```

SECTION VII
INPUT/OUTPUT STATEMENTS

Input/output statements are executable statements which allow the transfer of data records to and from external files and memory, and the positioning and demarcation of external files. The FORTRAN IV input/output statements are:

READ (Formatted Records)
WRITE (Formatted Records)
READ (Unformatted Records)
WRITE (Unformatted Records)
REWIND
BACKSPACE
ENDFILE

> *NOTE: All external files must be sequential files.*

IDENTIFYING INPUT/OUTPUT UNITS

An input or output unit is identified by a logical unit number assigned to it by the operating system. (See the RTE Operating System Reference Manuals for a description of logical units.) The logical unit reference may be an integer constant or an integer variable whose value identifies the unit. Any variable used to identify an input/output unit must be defined at the time of its use.

IDENTIFYING ARRAY NAMES OR FORMAT STATEMENTS

The format specifier for a record or records may be an array name or the statement label of a FORMAT statement (see Section VIII). If the format specifier is an array name, the first part of the information contained in the array must constitute a valid FORMAT specification: a normal FORMAT statement less the statement number and the word "FORMAT."

If the format specifier is a FORMAT statement label, the identified statement must appear in the same unit as the input or output statement.

INPUT/OUTPUT LISTS

An input list specifies the names of the variables, arrays and array elements to which values are assigned on input. An output list specifies the references to variables, arrays, array elements and constants whose values are transmitted on output. Input and output lists have the same form, except that a constant is a permissable output list element. List elements consist of variable names, array names, array element names and constants (output only), separated by commas. The order in which the elements appear in the list is the sequence of transmission.

There are two types of input/output lists in FORTRAN IV: simple lists and DO-implied lists.

Simple Lists

A simple list, n, is a variable name, an array name, an array element name, a constant (output only) or two simple lists separated by a comma. It has the form:

n

n,n

DO-Implied Lists

A DO-implied list contains a simple list followed by a comma and a DO-implied specification, all enclosed by parentheses. It has the form:

$(n,\ i = m_1,\ m_2,\ m_3)$

where

n = a simple list

i = a control variable (a simple integer variable)

m_1 = the initial parameter (an integer arithmetic expression)

m_2 = the terminal parameter (an integer arithmetic expression)

m_3 = the step-size parameter (an integer arithmetic expression)

> The parameters m_1, m_2, and m_3 may be any arithmetic expression. However, if these expressions are not Type-INTEGER, they are converted to Type-INTEGER by truncation following evaluation. Functions may be referenced only if they do not execute, or cause to be executed, any other READ or WRITE statements, or other I/O operations.

Data defined by the list elements is transmitted starting at the value of m_1, in increments of m_3, until m_2 is exceeded. If m_3 is omitted, the step-size is assumed to be +1.

The step-size parameter may be positive or negative, allowing incrementing or decrementing to the terminal parameter value.

The elements of a DO-implied list are specified for each cycle of the implied DO loop.

EXAMPLES:

Simple List	DO-Implied List
A,B,C	((ARRAY(I,J),J=1,5),I=1,5)
READ(5,10)A,B,C	READ(5,10)((ARRAY(I,J),J=1,5),I=1,5)

Note: For output lists, signed or unsigned constants are permitted as list elements.

FORMATTED AND UNFORMATTED RECORDS

A formatted record consists of a string of the characters that are permissible in Hollerith constants. The transfer of such a record requires that a format specification be referenced to supply the necessary positioning and conversion specifications. The number of records transferred by the execution of a formatted READ or WRITE statement is dependent upon the list and referenced format specification.

An unformatted record consists of binary values.

READ

FORMATTED

PURPOSE: To read formatted records from an external device into main memory or to provide data conversion from ASCII data to numeric data.

FORMAT:

```
READ (u,f) k
READ (u,*) k
READ (u,f)
```

u = an input unit
f = an array name or a FORMAT statement label or an integer variable defined in an ASSIGN statement (must not be in EMA)
k = an input list
* = specification for free-field input (no format statement)

COMMENTS: The format statement or specification (in an array) can be anywhere in the program unit.

If free-field input is specified, the formatting is directed by special characters in the input records; a FORMAT statement or specification is not required.

If data conversion is to be made, a call to the relocatable subroutine CODE must precede the READ instruction.

The Fortran IV Formatter supports the transfer of data records containing a maximum of 132 characters within a formatted READ operation. In some systems the user may extend this size by supplying an alternate buffer. Refer to the explanation of the LGBUF subroutine in the DOS/RTE Relocatable Library Reference Manual.

EXAMPLES:

```
READ (5,100) (A(I), I = 1, 20)
READ (5,200) A,L,X
READ (5,*) (A(J), J=1, 10)
READ (5,ARRAY)
READ (5,100) ((A(I,J),I=1,5),J=1,20)
ASSIGN 100 to K
READ (5,K) ((A(I,J),I=1,5), J=1,20)
```

The following performs a data conversion of the ASCII buffer IN and stores the numeric equivalents in variables A,L,X:

```
CALL CODE
READ (IN,200) A,L,X
```

In this case any required statement labels must be on the CALL CODE statement and it must not be the terminal statement of a DO loop. Caution: IN should not be subscripted.

WRITE

FORMATTED

PURPOSE: To write formatted records from main memory to an external device or to provide data conversion from numeric data to ASCII data.

FORMAT:

```
WRITE (u,f) k
WRITE (u,f)
```

u = an output unit
f = an array name or a FORMAT statement label or an integer variable defined in an ASSIGN statement (must not be in EMA)
k = an output list

COMMENTS: The FORMAT statement or specification (in an array) can be anywhere in the program unit.

If data conversion is to be performed, a call to the relocatable subroutine CODE must precede the WRITE instruction.

The Fortran IV Formatter supports the transfer of data records containing a maximum of 132 characters within a formatted WRITE operation. In some systems the user may extend this size by supplying an alternate buffer. Refer to the explanation of the LGBUF subroutine in the DOS/RTE Relocatable Library Reference Manual.

EXAMPLES:

```
WRITE (2,200) A, L, X
WRITE (2, ARRAY)
```

The following performs a data conversion of variables A,L,X and stores the ASCII equivalents in buffer TU:

```
CALL CODE
WRITE (TU,200) A,L,X
```

In this case any required statement labels must be on the CALL CODE statement and it must not be the terminal statement of a DO loop. Caution: TU should not be subscripted.

READ

UNFORMATTED

PURPOSE: To read one unformatted record from an external file.

FORMAT:

```
READ (u) k
READ (u)
```

u = an input unit

k = an input list

COMMENTS: The sequence of values required by the list may not exceed the sequence of values from the unformatted record.

READ (u) causes a record to be skipped.

The Fortran IV Formatter supports the transfer of data records containing a maximum of 60 words within an unformatted (binary) READ operation. In some systems the user may employ the LGBUF subroutine to extend this limit. Refer to the explanation of LGBUF in the DOS/RTE Relocatable Library Reference Manual.

EXAMPLES:

```
READ (5) A, L, X
READ (5)
```

WRITE

UNFORMATTED

PURPOSE: To write one unformatted record from main memory to an external file.

FORMAT:

WRITE (u) k

u = an output unit

k = an output list

COMMENTS: This statement transfers the next binary record from main memory to unit u from the sequence of values represented by the list k.

The Fortran IV Formatter supports the transfer of data records containing a maximum of 60 words within an unformatted (binary) WRITE operation. In some systems the user may employ the LGBUF subroutine to extend this limit. Refer to the explanation of LGBUF in the DOS/RTE Relocatable Library Reference Manual.

EXAMPLES: WRITE (2) A, L, X

REWIND, BACKSPACE, ENDFILE

PURPOSE: These statements are used for magnetic tape files. REWIND is used to rewind a tape to the beginning of tape. BACKSPACE is used to backspace a tape file one record. ENDFILE is used to write an end-of-file record on a tape file.

FORMAT:

```
REWIND u
BACKSPACE u
ENDFILE u
```

u = an input/output unit

COMMENTS: If the magnetic tape unit is at beginning of tape when a REWIND or a BACKSPACE statement is executed, the statement has no effect.

EXAMPLES:

```
BACKSPACE 2
ENDFILE I
REWIND 5
```

FREE FIELD INPUT

By following certain conventions in the preparation of his input data, a FORTRAN IV programmer can write programs without using an input FORMAT statement. The programmer uses special characters included within input data items to direct the formatting of records.

Data records composed this way are called free field input records, and can be used for numeric input data only. Free field input is indicated in a formatted READ statement by using an asterisk (*) instead of an array name or a FORMAT statement label.

The special characters used to direct the formatting of free field input records are:

space or ,	data item delimiters
/	record terminator
+ or -	sign of item
. E + -	floating point number
@	octal integer
"..."	comments

Data Item Delimiters

A space or a comma is used to delimit a contiguous string of numeric and special formatting characters (called a data item), whose value corresponds to a list element. A data item must occur between two commas, a comma and a space or between two spaces. (A string of consecutive spaces is equivalent to one space.) Two consecutive commas indicate that no data item is supplied for the corresponding list element, i.e., the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

EXAMPLES:

```
100 READ (5,*) I, J, K, L
```

Input data items:

1720,1966,1980,1492

Result:

I = 1720

J = 1966

K = 1980

L = 1492

```
200 READ (5,*) I, J, K, L
```

Input data items:

,,1794,2000

Result:

I = 1720

J = 1966

K = 1794

L = 2000

Record Terminator

A slash within a record causes the next record to be read immediately; the remainder of the current record is skipped.

EXAMPLE:

```
READ (5,*) I, J, K, L, M
```

Input data items:

987,654,321,123/DESCENDING

456

Result:

I = 987

J = 654

K = 321

L = 123

M = 456

> *NOTE: If the input list requires more than one external input record, a slash (/) is required to terminate each of the input records except the last one.*

Sign of Data Item

Data items may be signed. If they are not signed, they are assumed to be positive.

Floating Point Number Data Item

A floating point data item is represented in the same form as E-TYPE conversion of an external real number on input. (See Section VIII.) If the decimal point is not present, it is assumed to follow the last digit of the number.

Octal Data Item

The symbol @ is used to indicate an octal data item. List elements corresponding to the octal items must be type integer.

EXAMPLE:

```
READ (5,*) I, J, K

Input Data Items:
@177777, @0, @5555

Result:
I = 177777B
J = 0
K = 5555B
```

Comment Delimiters

Quotation marks ("...") are used to bound comments; characters appearing between quotation marks are ignored.

EXAMPLE:

```
READ (5,*) I, J, K, L

Input Data Items:
123, 456, "ASCENDING"123, 456

Result:
I = 123
J = 456
K = 123
L = 456
```

SECTION VIII
THE FORMAT STATEMENT

There are three ways a user can transfer data records to and from memory using READ and WRITE statements (described in Section VII).

a. As "free field input" when the input data itself contains special characters that direct the formatting of the records in memory. (See "Free Field Input.")

b. As unformatted input or output records containing strings of binary values. (See "READ (Unformatted)" and "WRITE (Unformatted).")

c. As formatted input or output records. (See "READ (Formatted)" and "WRITE (Formatted).")

When a formatted READ or WRITE statement is executed, the actual number of records transferred depends upon:

a. The elements of an input/output list (if present), which specify the data items involved in the transfer, and

b. A format specification for the list element(s), which defines the positioning and conversion codes used for the string of characters in a record.

A format specification for a formatted READ or a formatted WRITE list element can be defined in either:

a. A FORMAT statement, or

b. An array, the first elements of which contain a valid format specification constructed according to the rules of a FORMAT statement (minus the FORMAT statement label and the "FORMAT").

The FORMAT statement and its components are described in the following pages.

FORMAT

PURPOSE: The FORMAT statement is a non-executable statement that provides format control for data records being transferred to and from core memory by defining a format specification for each record.

FORMAT:

label FORMAT $(q_1 t_1 z_1 \; t_2 z_2 \; \ldots \; t_n z_n \; t_{n+1} q_2)$

label = a statement label.

q = a series of slashes (optional)

t = a field descriptor, or a group of field descriptors

z = a field separator

COMMENTS: A FORMAT statement must be labeled.

When a formatted READ statement is executed, one record is read when format control is initiated; thereafter, additional records are read only as the format specification(s) demand. When a formatted WRITE statement is executed, one record is written each time a format specification demands that a new record be started.

EXAMPLES:

```
    READ(5,100)A,B,C
    :
100 FORMAT (2F5.1, F6.2)
```

```
    WRITE(2,200)A,L,X
    :
200 FORMAT (F5.1, I10, F6.4)
```

The components of a format specification (field separators, field descriptors, scale factor, repeat specification and conversion codes) are described in the following pages.

FIELD DESCRIPTOR

PURPOSE: To provide the elements that define the type, magnitude and method of conversion and editing between input and output.

FORMAT: One of the following conversion and editing codes:

Integer data:	rIw	Octal data:	r@w
Real data:	srEw.d		rKw
	srFw.d		rOw
	srGw.d	Hollerith data:	rAw
Double precision data:	srDw.d		rRw
Logical data:	rLw		$wHh_1h_2 \ldots h_w$
Column positioning:	wX,Tw,TLw,TRw		$r("h_1h_2 \ldots h_w")$
Complex data:	sEw.d,Ew.d		$r('h_1h_a \ldots h_w')$

w = a positive integer constant, representing the length of the field in the external character string.

s = a scale factor designator (optional for real and double precision type conversions).

r = a repeat specification, an optional positive integer constant indicating the number of times to repeat the succeeding field descriptor or group of field descriptors.

h = any character in the FORTRAN character set.

d = an non-negative integer constant representing the number of digits in the fractional part of the external character string (except for G-type conversion codes).

. = a decimal point.

The characters F, E, G, I, @, K, O, L, A, R, H, ", ', T, TL, TR and X indicate the manner of conversion and editing between the internal and external character representations, and are called the conversion codes.

COMMENTS: For all field descriptors, except "h_1h_2 ... h_w" and 'h_1h_a ... h_w', the field length (w) must be specified, and must be greater than or equal to d.

For field descriptors of the form w.d, the d must be specified, even if it is zero.

A basic field descriptor is a field descriptor unmodified by the scale factor (s) or the repeat specification (r).

The internal representation of external fields corresponds to the internal representation of the corresponding data type constants.

A numeric input field of all blanks is treated as the number zero.

The use of a decimal point in the input data field overrides the d portion of a floating point conversion format.

Negative numbers are output with a minus sign.

If the output field is larger than that required by the datum being written, the datum is right-justified in the output field.

The number of characters produced by an output conversion must not exceed the field width (w). If the characters produced do exceed the field width, the field is filled with the currency symbol $.

EXAMPLES:

2I10	2@2
E20.10	2K2
F5.1	2O2
G20.10	2A2
D10.2	2R2
E10.4, E10.4	2HAB
2X	"ABCD"

REPEAT SPECIFICATION

PURPOSE: Allows repetition of field descriptors through the use of a repeat count preceding the descriptor. The specified conversion is interpreted repetitively, up to the specified number of times.

FORMAT:

r (basic field descriptor)

r = an integer constant, called the group repeat count.

COMMENTS: All basic field descriptors may have group repeat counts, except these codes: wH or wX.

A further grouping may be formed by enclosing field descriptors, field separators, or basic groups within parentheses, and by specifying a group repeat count for the group. The depth of this grouping is limited to the fourth level.

The parentheses enclosing the format specification are not group delineating parentheses.

EXAMPLES:

```
2I10
6E14.6
4(E10.4, E10.4)
3/
```

I-TYPE CONVERSION

INTEGER NUMBERS

PURPOSE: Provides conversion between an internal integer number and an external integer number.

FORMAT:

r I w

r = a repeat specification (optional)

w = length of external field

COMMENTS:

Input: The external input field contains a character string in the form of an integer constant or a signed integer constant. Blank characters are treated as zeros.

Output: The external output field consists of blanks, if necessary, a minus (if the value of the internal datum is negative), and the magnitude of the internal value converted to an integer constant, right-justified in the field.

If the output field is too short, the field is filled with the currency symbol $.

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Number
-^123	I5	-123
12003	I5	12003
^102	I4	102
3	I1	3

OUTPUT:

Internal Number	Format	External Field
-1234	I5	-1234
+12345	I5	12345
+12345	I4	$$$$
+12345	I6	^12345

SCALE FACTOR

PURPOSE: Provides a means of normalizing the number and exponent parts of real or double precision numbers specified in a FORMAT statement.

FORMAT:

nP

n = an integer constant or a minus sign followed by an integer constant.

P = the scale factor indicator, the character P

COMMENTS: When format control is initialized, a scale factor of zero is established. Once a scale factor has been established, it applies to all subsequent real and double precision conversions until another scale factor is encountered.

Input: When there is no exponent in the external field, the relationship between the externally represented number (E) and the internally represented number (I) is this:

$$I = E * 10^{-n}$$

When there is an exponent in the external field, the scale factor has no effect.

Output: For E- and D- type output, the basic real constant part (I) of the output quantity is multiplied by 10^n and the exponent is reduced by n. For G-type output, the effect of the scale factor is suspended unless the magnitude of the datum to be converted is outside the range that permits effective F-type conversion.

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Number
528.6	1PF10.3	52.86
.5286E+03	1PG10.3	528.6
528.6	-2PD10.3	52860.

OUTPUT:

Internal Number	Format	External Field
528.6	1PF8.2	^5286.00
.5286	2PE10.4	52.860E-02
5.286	-1PD10.4	^.0529D+02
52.86	1PG10.3	^^52.9^^^^
-5286.	1PG10.3	-5.286E+03

E-TYPE CONVERSION

REAL NUMBERS

PURPOSE: Provides conversion between an internal real number and an external floating-point number.

FORMAT:

s r E w. d

s = a scale factor (optional)

r = a repeat specification (optional)

w = the length of the external field

. = the decimal point

d = the total number of digits to the right of the decimal point in the external field.

COMMENTS:

Input: The external input field may contain an optional sign, followed by a string of digits optionally containing a decimal point, followed by an exponent, in one of the following forms: a signed integer constant; or E followed by an integer constant or a signed integer constant.

Output: The external output field may contain a minus sign (or a blank, if the number is positive), a zero, a decimal point, the most significant rounded digits of the internal value, the letter E and a decimal exponent (which is signed if it is negative).

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Number
123.456E6	E9.3	123456000
.456E6	E6.5	456000
.456	E4.3	.456
123E6	E5.0	123000000
123	E3.1	12.3
E6	E9.3	0
^	E9.3	0

OUTPUT:

Internal Number	Format	External Field
+12.34	E10.3	^^.123E+02
-12.34	E10.3	^-.123E+02
+12.34	E12.4	^^^.1234E+02
-12.34	E12.4	^^-.1234E+02
+12.34	E7.3	.12E+02
+12.34	E5.1	$$$$$

F-TYPE CONVERSION

REAL NUMBERS

PURPOSE: Provides conversion between an internal real number and an external fixed-point number.

FORMAT:

s r F w . d

s = a scale factor (optional)

r = a repeat specification (optional)

w = the length of the external field

. = the decimal point

d = the total number of digits to the right of the decimal point in the external field

COMMENTS:

Input: The external input field is the same as for E-TYPE conversion.

Output: The external output field may contain blanks, a minus (if the internal value is negative), a string of digits containing a decimal point (as modified by the scale factor) rounded to d fractional digits.

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT: Same as in E-TYPE conversion, except "F" replaces "E" in the format specification.

OUTPUT:

Internal Number	Format	External Field
+12.34	F10.3	^^^^12.340
-12.34	F10.3	^^^-12.340
+12.34	F12.3	^^^^^^12.340
-12.34	F12.3	^^^^^-12.340
+12.34	F4.3	12.3
+12345.12	F4.3	$$$$

G-TYPE CONVERSION

REAL NUMBERS

PURPOSE: Provides conversion between an internal real number and an external floating-point or fixed-point number.

FORMAT:

s r G w . d

s = a scale factor (optional)

r = a repeat specification (optional)

w = the length of the external field

. = the decimal point

d = the total number of digits to the right of the decimal point in the external field.

COMMENTS:

Input: The external input field is the same as for E-TYPE conversion.

Output: The external output field depends upon the magnitude of the real data being converted, and follows these rules:

Magnitude Of Data	Equivalent Conversion
$0.1 \leq N < 1$	F(w-4).d,4X
$1 \leq N < 10$	F(w-4).(d-1),4X
⋮	⋮
$10^{d-2} \leq N < 10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N < 10^{d}$	F(w-4).0,4X
otherwise	SEw.d

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT: Same as for E-TYPE conversion, except that "G" replaces "E" in the format specification.

OUTPUT:

Format	Internal Number	External Field
G10.3	.05234	^^.523E-01
	.5234	^^.523^^^^
	52.34	^^52.3^^^^
	523.4	^^523.^^^^
	5234.	^^.523E+04

D-TYPE CONVERSION

DOUBLE PRECISION NUMBERS

PURPOSE: Provides conversion between an internal double precision number and an external floating-point number.

FORMAT:

s r D w . d

s = a scale factor (optional)

r = a repeat specification (optional)

w = the length of the external field

. = the decimal point

d = the total number of digits to the right of the decimal point in the external field.

COMMENTS:

Input: The external input field is the same as for E-TYPE conversion.

Output: The external output field is the same as for E-TYPE conversion, except that the character D replaces the character E in the exponent.

EXAMPLES:

INPUT: Same as in E-TYPE conversion except "D" replaces "E."

OUTPUT: Same as in E-TYPE conversion except "D" replaces "E."

COMPLEX CONVERSION

COMPLEX NUMBERS

PURPOSE: Provides conversion between an internal ordered pair of real numbers and an external complex number.

FORMAT:

A complex datum consists of a pair of separate real data. The total conversion is specified by two real field descriptors, interpreted successively. The first descriptor supplies the real part; the second, the imaginary part.

COMMENTS:

Input: Same as for any pair of real data.

Output: Same as for any pair of real data.

EXAMPLES:

See E-, F- and G-TYPE conversions.

L-TYPE CONVERSION

LOGICAL NUMBERS

PURPOSE: Provides conversion between an external field representing a logical value and an internal logical datum.

FORMAT:

L w

w = the length of the external field.

COMMENTS:

Input: The external input field consists of optional blanks followed by a T or an F followed by optional characters, representing the values true or false, respectively.

Output: The external output field consists of w - 1 blanks followed by a T or an F as the value of the internal logical datum is true or false, respectively.

EXAMPLES:

INPUT:

External Field	Format	Internal Number
^TRUE	L5	100000B
^^^^^F	L6	0

OUTPUT:

Internal Number	Format	External Field
0 (or positive)	L3	^^F
(negative)	L1	T

@ -TYPE, K-TYPE AND O-TYPE CONVERSIONS

OCTAL NUMBERS

PURPOSE: Provides conversion between an external octal number and an internal octal datum.

FORMAT:

r @ w

r K w

r O w

r = a repeat specification (optional)

w = the width of the external field in octal digits.

COMMENTS: List elements must be of type integer.

Input: If $w \geq 6$, up to six octal digits are stored; non-octal digits are ignored. If the value of the octal digits within the field is greater than 177777, results are unpredictable. If $w < 6$ or if less than six octal digits are encountered in the field, the number is right-justified with zeros to the left.

Output: If $w \geq 6$, six octal digits are written right-justified in the field with blanks to the left. If $w < 6$, the w least significant octal digits are written.

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Number
123456	@6	123456
-123456	O7	123456
2342342342	2K5	023423 and 042342
,396E-05	2@4	000036 and 000005

OUTPUT:

Internal Number	Format	External Field
99	K6	^^^143
99	O2	43
-1	@8	^^177777
32767	@6	^77777

A-TYPE CONVERSION

HOLLERITH INFORMATION

PURPOSE: Allows a specified number of Hollerith characters to be read into, or written from, a specified list element.

FORMAT:

r A w

r = a repeat specification, (optional)

w = the length of the Hollerith character string.

COMMENTS: Input: Assume "n" to be the size of the list element in characters. If w ≥ n, the rightmost n characters are taken from the external input field. If w <n, the characters appear left-justified in the list element, with w-n trailing blanks.

Output: If w > n, the external output field consists of w - n blanks, followed by n characters from the internal representation. If w = <n, the characters in the left part of the list element is written.

EXAMPLES:

See the next page.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Value
XYZ	A2	XY
VWXYZ	A5	WXYZ (Real variable)
X	A1	X^

OUTPUT:

Internal Value	Format	External Field
XY	A2	XY
WXYZ	A6	^^WXYZ (Real variable)
XY	A1	X

R-TYPE CONVERSION

HOLLERITH INFORMATION

PURPOSE: Allows a specified number of Hollerith characters to be read into, or written from, a specified list element.

FORMAT:

r R w

r = a repeat specification (optional)

w = the length of the Hollerith character string.

COMMENTS: Assume "n" to be the size of the list element in characters. The Rw descriptor is equivalent to the Aw descriptor, except that characters are right-justified in the word with leading binary zeros (on input); and on output, if w = 1, the characters in the right part of the list element is written.

NOTE: The HP FORTRAN conversion Aw is replaced by the FORTRAN IV conversion Rw.

EXAMPLES: See the next page.

NOTE: The FORTRAN IV program can be modified at run-time to interpret A as in HP FORTRAN if the user calls the OLDIO entry point:

CALL OLDIO

To change back to a FORTRAN IV A conversion, the user calls the NEWIO entry point:

CALL NEWIO

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Value
XYZ	R2	XY
VWXYZ	R5	WXYZ (Real variable)
X	R1	OX

OUTPUT:

Internal Value	Format	External Field
XY	R2	XY
WXYZ	R6	^^WXYZ (Real variable)
XY	R1	Y

wH EDITING

HOLLERITH INFORMATION

PURPOSE: Allows Hollerith information to be read into, or written from, the characters following the wH descriptor in a format specification.

FORMAT:

$$wH\ h_1\ h_2\ \ldots\ h_w$$

w = a nonzero positive integer constant equal to the total number of h's

h = any character in the HP ASCII character set.

COMMENTS:

Input: The characters in the external field (h_1 to h_w) replace the characters in the field specification.

Output: The characters in the field specification are written to an output file.

EXAMPLES:

INPUT:

External Field	Format	Resulting Internal Value of Formatted Item
PACKARD	7HHEWLETT	7HPACKARD

OUTPUT:

Format	External Field
7HPACKARD	PACKARD

'...' AND "..." EDITING

HOLLERITH INFORMATION

PURPOSE: Allows Hollerith information to be written from the characters enclosed by the quotation marks in a format specification.

FORMAT:

r "$h_1h_2 \ldots h_w$" or r '$h_1h_2 \ldots h_w$'

h = any character in the FORTRAN character set, except the quote mark being used.

r = a repeat count.

COMMENTS: Input: The number of characters within the quotation marks is skipped (equivalent to wX).

Output: Is equivalent to wH, with a repeat specification capability added.

EXAMPLES:

OUTPUT:

Format	External Field
"ABZ"	ABZ
"^^^"	^^^
2 '***'	******

X,T,TL,TR-TYPE CONVERSION

SKIP OR BLANKS

PURPOSE: Sets the next column at which conversion will start.

FORMAT:

w X, Tw, TLw or TRw

w = a positive integer constant

COMMENTS:

T: Move to column w.

TL: Move left w columns.

X,TR: Move right w columns.

On output, if the new position is to the right of the previous rightmost position, the intervening positions are blank-filled.

EXAMPLES:

14X

2X

T5

TL3

TR72

FIELD SEPARATOR

PURPOSE: To separate each field descriptor, or group of field descriptors in a FORMAT statement.

FORMAT:

/ or ,

COMMENTS: A repeat count can be specified immediately preceding the slash (/) field separator. Each slash terminates a record. A series of slashes causes records to be skipped on input, or lines to be skipped on an output listing.

EXAMPLES:

READ (5,100)A,B 100 FORMAT (F5.1,F7.3)	Causes A and B to be read from one record.
READ (5,101)A,B 101 FORMAT (F5.1/F7.3)	Causes A and B to be read from two consecutive records.
READ (5,102)A,B 102 FORMAT(//F5.1///F7.3/)	Causes two records to be skipped, A to be read from the third record, two more records to be skipped, B to be read from the sixth record and one additional record to be skipped.
WRITE (6,100)A,B	Causes A and B to be printed on the same line.
WRITE (6,101)A,B	Causes A and B to be printed on two consecutive lines.
WRITE (6,102)A,B	Causes two lines to be skipped, A to be printed on the third line, two more lines to be skipped, B to be printed on the sixth line and one more additional line to be skipped.

CARRIAGE CONTROL

PURPOSE: To indicate the line spacing used when printing an output record on a line printer or a teleprinter.

FORMAT:

```
              ^  ┐
              0  │
              1  │ as the first character in the record
              *  │
any other character ┘

^ = single space (print on every line).
0 = double space (print on every other line).
1 = eject page
* = suppress spacing (overprint current line).
any other character = single space (print on every line).
```

EXAMPLES:

When these records are printed...	they look like this:
100 FORMAT ("^PRINT ON EVERY LINE")	PRINT ON EVERY LINE
120 FORMAT ("OPRINT ON EVERY OTHER LINE")	PRINT ON EVERY OTHER LINE
140 FORMAT ("1")	(a page is ejected, then a line is skipped)
160 FORMAT ("*PRINT ON CURRENT LINE")	(an overprint of current line)
180 FORMAT ("PRINT ON EVERY LINE")	RINT ON EVERY LINE
999 FORMAT (1H1, E16.8, I5)	(a page is ejected, and a floating point number and an integer are then printed.)

SECTION IX

PROGRAMS, FUNCTIONS, SUBROUTINES, AND BLOCK DATA SUBPROGRAMS

PROGRAM STATEMENT

PURPOSE: The PROGRAM statement names the main program and assigns parameters to it which are passed to the binary record and hence to the loader loading the relocatable object code. Similarly, a comment line can be passed to the loader.

Refer to the FORTRAN IV Operations Section of this manual for additional information.

FORMAT:

PROGRAM name (p_1,p_2,...,p_8), comment

or,

PROGRAM name ,p_1,p_2,...,p_8, comment

name = the name assigned to the program.

p_1-p_8 = up to eight integer parameters to be passed to the loader. See the appropriate operating system documentation for the meaning attached to these parameters. If not specified, the defaults are:

p_1 = 3	disc-based, background (ignored by RTE-M)
p_2 = 99	priority
p_3-p_8 = 0	time values

comment = a comment line to be passed to the loader. All characters after the comma (,) including blanks are passed. The comment is limited to 84 characters in length.

COMMENTS: In the first format shown above, one or more of the parameters may be omitted while still retaining the comment. In the second format, all parameters must be accounted for at least by the presence of a comma. Data after the program name is optional. The PROGRAM statement, if present, must be the first non-comment statement in the module.

EXAMPLES:

```
PROGRAM XY(),THIS PROGRAM HAS NO PARAMETERS
PROGRAM XY,,,,,,,,,COMMAS MUST BE PRESENT TO FIND THIS COMMENT
PROGRAM XY
PROGRAM XY(1,10),HELP! 770105
```

NOTE: All information following the program name within the PROGRAM statement is an extension of the standard.

An executable FORTRAN IV program consists of one main program with or without subprograms. Subprograms, which are either functions, subroutines, or block data subprograms, are sets of statements that may be written and compiled separately from the main program.

A main program calls or references subprograms; subprograms can call or reference other subprograms as long as the calls are non-recursive. That is, if subprogram A calls subprogram B, subprogram B may not call subprogram A. Furthermore, a program or subprogram may not call itself. A calling program is a main program or subprogram that refers to another subprogram.

Main programs and subprograms communicate by means of arguments (parameters). The arguments appearing in a call or a reference are called actual arguments. The corresponding parameters appearing within the called or referenced definition are called dummy arguments.

FUNCTIONS

If the value of one quantity depends on the value of another quantity, then it is a function of that quantity. Quantities that determine the value of the function are called the actual arguments of the function.

In FORTRAN IV, there are three types of functions (collectively called function procedures); they supply a value to be used at the point of reference.

a. A statement function is defined and referenced internally in a program unit.

b. A FORTRAN IV library function is processor-defined external to the program unit that references it. The FORTRAN IV functions are stored on an external disc or tape file.

c. A function subprogram is user-defined external to the program unit that references it. The user compiles function subprograms, loads them with his calling program unit and references them the same way he references FORTRAN IV library functions.

SUBROUTINES

The RTE FORTRAN IV user can compile a program unit and store the resultant object program in an external file. If the program unit begins with a SUBROUTINE statement and contains a RETURN statement, it can be called as a subroutine by another program unit.

Data Types For Functions and Subroutines

All functions are identified by symbolic names.

A symbolic name that identifies a statement function may have its data type declared in a Type-specification statement. In the absence of an explicit declaration in a Type-specification statement, the type is implied by the first character of the name, as follows:

I, J, K, L, M, or N = integer type data
any other character = real type data

A symbolic name that identifies a FORTRAN IV function has a predefined data type associated with it, as explained in Table 9-1.

A symbolic name that identifies a function subprogram may have its data type declared in the FUNCTION statement that begins the subprogram or in a subsequent Type-specification statement. In the absence of an explicit declaration in the FUNCTION statement or a Type-specification statement, the data type is implied by the first character of the name, as for statement functions. A function subprogram which has been explicitly typed must also have its name identically typed (in a Type-specification statement) in each program unit which calls it. Otherwise, unpredictable results may occur.

The symbolic names which identify subroutines are not associated with any data type.

DUMMY ARGUMENTS

Dummy arguments are identified by symbolic name. They are used in functions and subroutines to identify variables, arrays, other subroutines or other function subprograms. The dummy arguments indicate the type, order and number of the actual arguments upon which the value of the function depends.

When a variable or an array reference is specified by symbolic name, a dummy argument can be used, providing a value of the same type is made available through argument association.

When a subroutine reference is specified by the symbolic name, a dummy argument can be used if a subroutine name is associated with that dummy argument.

When a function subprogram reference is specified by symbolic name, a dummy argument can be used if a function subprogram name is associated with that dummy argument.

BLOCK DATA SUBPROGRAMS

Block data subprograms are used to define and, optionally, to initialize named common blocks. A block data subprogram begins with a BLOCK DATA statement followed by specification statements describing variables in various named common blocks, optional DATA statements to initialize these variables, and an END statement.

STATEMENT FUNCTION

PURPOSE: To define a user-specified function in a program unit for later reference in that program unit.

FORMAT:

$$f\ (\ a_1,\ a_2,\ \ldots,\ a_n\ )\ =\ e$$

f = the user-specified function name, a symbolic name

a = a distinct variable name (the dummy arguments of the function)

e = an arithmetic or logical expression

COMMENTS: The statement function is referenced by using its symbolic name, with an actual argument list, in an arithmetic or logical expression.

In a given program unit, all statement function definitions must precede the first executable statement of the program unit and must follow any specification statements used in the program unit.

The name of a statement function must not be a variable name or an array name in the same program unit.

EXAMPLES:

```
ISUM(I,J,K) = I+J+K
 ∫∫
ROOT1(A,B,C) = (-B+SQRT(B**2-4.0*A*C))/(2.0*A)
L = ISUM(M**2,1,M-1)
 ∫∫
R = ROOT1 (X,Y,Z)
```

Defining Statement Functions

The names of dummy arguments may be identical to variable names of the same type that appear elsewhere in the program unit, since they bear no relation to the variable names.

The dummy arguments must be simple variables; they represent the values passed to the statement function. These values are used in an expression to evaluate the user-specified function. Dummy arguments cannot be used to represent array elements or function subprograms.

Aside from the dummy arguments, the expression may contain only these values:

- Constants
- Variable references (both simple and subscripted)
- FORTRAN IV library function references
- External function references
- References to previously-defined statement functions in the same program

Referencing Statement Functions

When referenced, the symbolic name of the statement function must be immediately followed by an actual argument list.

The actual arguments constituting the argument list must agree in order, number and type with the corresponding dummy arguments. An actual argument in a statement function reference may be an expression of the same type as the corresponding dummy argument.

When a statement function reference is executed, the actual argument values are associated with the corresponding dummy arguments in the statement function definition and the expression is evaluated. Following this, the resultant value is made available to the expression that contained the statement function reference.

FORTRAN IV LIBRARY FUNCTION

PURPOSE: To reference a processor-defined function by specifying its symbolic name in an arithmetic or logical expression. The value is made available at the point of reference.

FORMAT:

An arithmetic or logical expression that contains the symbolic name of the FORTRAN IV function (together with an actual argument list) as a primary.

COMMENTS: Table 9-1 contains the FORTRAN IV library functions available with the FORTRAN IV Compiler. The trigonometric functions listed in this table use radians measure.

If the symbolic name for the function appears in a TYPE-specification statement which defines the name as a data type different from that specified for the function in Table 9-1, the function becomes "external". The user must then supply his own version of the FORTRAN IV library function.

NOTE: Some "intrinsic" functions are accessed by FORTRAN IV using different names and/or calling sequences than for "external" functions. Care should be taken when using names of intrinsic functions for user-specified subroutines.

EXAMPLES:

```
X = SIN(Y)
I = IFIX(X)
```

TABLE 9-1

FORTRAN IV LIBRARY FUNCTIONS

FORTRAN IV Function	Definition	Number of Arguments	Symbolic Name	Type of: Argument	Function
Absolute Value	$\lvert a \rvert$	1	ABS	Real	Real+
			IABS	Integer	Integer+
			DABS	Double	Double
Truncation	Sign of a times largest integer $\leq \lvert a \rvert$	1	AINT	Real	Real+
			INT	Real	Integer+
			IDINT	Double	Integer
Remaindering*	$a_1 \pmod{a_2}$	2	AMOD	Real	Real*
			MOD	Integer	Integer*
Choosing Largest Value	Max $(a_1, a_2, \ldots)$	≥ 2	AMAXØ	Integer	Real
			AMAX1	Real	Real
			MAXØ	Integer	Integer
			MAX1	Real	Integer
			DMAX1	Double	Double
Choosing Smallest Value	Min $(a_1, a_2, \ldots)$	≥ 2	AMINØ	Integer	Real
			AMIN1	Real	Real
			MINØ	Integer	Integer
			MIN1	Real	Integer
			DMIN1	Double	Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real+
Fix	Conversion from real to integer	1	IFIX	Real	Integer+
Transfer of Sign	Sign of a_2 times $\lvert a_1 \rvert$	2	SIGN	Real	Real+
			ISIGN	Integer	Integer+
			DSIGN	Double	Double
Positive Difference	a_1 - Min (a_1, a_2)	2	DIM	Real	Real
			IDIM	Integer	Integer
Obtain Most Significant Part of Double Precision Argument		1	SNGL	Double	Real
Obtain Real Part of Complex Argument		1	REAL	Complex	Real
Obtain Imaginary Part of Complex Argument		1	AIMAG	Complex	Real
Express Single Precision Argument in Double Precision Form		1	DBLE	Real	Double

TABLE 9-1 (cont.)

FORTRAN IV LIBRARY FUNCTIONS

FORTRAN IV Function	Definition	Number of Arguements	Symbolic Name	Type of: Argument	Function
Express Two Real Arguments in Complex Form	$a_1 + a_2 \cdot \sqrt{-1}$	2	CMPLX	Real	Complex
Obtain Conjugate of a Complex Argument		1	CONJG	Complex	Complex
Exponential	e^a	1	EXP	Real	Real+
		1	DEXP	Double	Double+
		1	CEXP	Complex	Complex+
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real+
		1	DLOG	Double	Double+
		1	CLOG	Complex	Complex+
Common Logarithm	$\log_{10}(a)$	1	ALOGT	Real	Real+
			DLOGT	Double	Double+
Trigonometric Sine	sin(a)	1	SIN	Real	Real+
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex+
Trigonometric Cosine	cos(a)	1	COS	Real	Real+
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex+
Trigonometric Tangent	tan(a)	1	TAN	Real	Real+
			DTAN	Double	Double+
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real+
			DTANH	Double	Double+
Square Root	$(a)^{1/2}$	1	SQRT	Real	Real+
		1	DSQRT	Double	Double+
		1	CSQRT	Complex	Complex
Arctangent	arctan(a)	1	ATAN	Real	Real+
		1	DATAN	Double	Double
	$\arctan(a_1/a_2)$	2	ATAN2	Real	Real
		2	DATN2	Double	Double
Remaindering*	$a_1 \pmod{a_2}$	2	DMOD	Double	Double*
Modulus		1	CABS	Complex	Real
Logical Product	i.j	2	IAND	Integer	Integer+
Logical Sum	i+j	2	IOR	Integer	Integer+
Exclusive OR		2	IXOR	Integer	Integer
Complement	i	1	NOT	Integer	Integer+
Sense Switch Register Switch (n)		1	ISSW	Integer	Integer+

* The functions MOD, AMOD and DMOD are defined as $a_1-[a_1/a_2]a_2$ where [X] is the largest integer whose magnitude does not exceed the magnitude of X and whose sign is the same as the sign of X.

+ These FORTRAN IV functions have different entry points when called by value and called by name. See the DOS/RTE Relocatable Library Reference Manual for a complete description of each entry point.

Double precision functions have different entry points for 3-word and 4-word double precision. The names used to call these functions within a FORTRAN program are the same for both sizes of double precision.

FUNCTION SUBPROGRAM

PURPOSE: To define a user-specified subprogram that supplies a function value when its symbolic name is used as a reference.

FORMAT:

t FUNCTION f (a_1, a_2, ..., a_n), comment

t = omitted, or one of the following data type identifiers

- REAL
- INTEGER
- DOUBLE PRECISION
- COMPLEX
- LOGICAL

f = the symbolic name of the function

a = a dummy argument.

comment = up to 50 character comment

COMMENTS: The FUNCTION statement must be the first statement of a function subprogram. A function subprogram is referenced by using its symbolic name (together with an actual argument list) as a primary in an arithmetic or logical expression in another program unit. The comment and its preceeding comma are optional. If present it is passed to the loader via the relocatable object code.

EXAMPLES:

```
VAR = USER1 (X,Y,Z)**USER2(X,Y)    REAL FUNCTION USER1(A,B,C)
                                   .
                                   .
                                   .
                                   USER1 = A+B/C
                                   RETURN
                                   END
                                   REAL FUNCTION USER2 (VARR1, VARR2)
                                   .
                                   .
                                   .
                                   USER2 = VARR1-VARR2
                                   RETURN
                                   END
```

NOTE: The ",comment" in the FUNCTION statement is an extension of the standard.

Defining Function Subprograms

The symbolic name of the function subprogram must also appear as a variable name in the defining subprogram. During every execution of the subprogram, this variable must be defined, and, once defined, may be referenced or redefined. The value of the variable at the time of execution of any RETURN statement in this subprogram is called the value of the function.

The symbolic name of the function subprogram must not appear in any non-executable statement in this program unit, except as a symbolic name of the function subprogram in the FUNCTION statement or in a Type-specification statement.

The symbolic names of the dummy arguments may not appear in an EQUIVALENCE, COMMON or DATA statement in the function subprogram.

A dummy parameter can be used to dimension in array name, which also appears as a dummy parameter of the function. An array which is declared with dummy dimensions in a function must correspond to an array which is declared with constant dimensions (through some sequence of argument association) in a calling program unit. An array declared with dummy dimensions may not be in common.

The symbolic name of a dummy argument may represent a variable, array, a subroutine or another function subprogram.

The function subprogram may contain any statements except PROGRAM, SUBROUTINE, BLOCK DATA, another FUNCTION statement, or any statement that directly or indirectly references the function being defined.

The function subprogram may define or redefine one or more of its arguments to return results as well as the value of the function. Therefore, the user must be aware of this when writing his programs. For example, a function subprogram that defines the value of GAMMA as well as finding the value of ZETA could be coded:

```
FUNCTION ZETA (BETA, DELTA, GAMMA)
A = BETA**2 - DELTA**3
GAMMA = A*5.2
ZETA = GAMMA**2
RETURN
END
```

Then, a program referencing the function could be:

```
GAMMB = 5.0
RSLT = GAMMB+7.5 + ZETA (.2,.3,GAMMB)
```

which results in the following calculation:

```
RSLT = 5.0 + 7.5 + ZETA, where ZETA is determined as:

        A = .2**2 - .3**3 = .04 - .027 = .013
    GAMMA = .013*5.2 = .0676 (GAMMB is not altered)
     ZETA = .0676**2 = .00456976
     RSLT = 5.0 + 7.5 + .0046976 = 12.50456976
```

However, the program:

```
GAMMB = 5.0
RSLT = ZETA (.2,.3,GAMMB) + 7.5 + GAMMB
```

would result in the following calculations for ZETA and GAMMB:

```
        A = .2**2 - .3**3 = .04 - .027 = .013
    GAMMA = .013*5.2 = .0676 = GAMMB
     ZETA = .0676**2 = .00456976
     RSLT = .00456976 + 7.5 + .0676 = 7.57216976
```

Referencing Function Subprograms

The actual arguments of a function subprogram reference argument list must agree in order, number and type with the corresponding dummy arguments in the function subprogram.

When referenced, the symbolic name of the function subprogram must be immediately followed by an actual argument list, except when used in a Type-specification or EXTERNAL statement, or as an actual argument to another subprogram.

An actual argument in a function subprogram reference may be one of the following:

- A constant
- A variable name
- An array element name
- An array name
- Any other expression
- The name of a FORTRAN IV library function
- The name of a user-defined FUNCTION or SUBROUTINE subprogram.

If an actual argument is a function subprogram name or a subroutine name, the corresponding dummy argument must be used as a function subprogram name or a subroutine name, respectively.

If an actual argument corresponds to a dummy argument defined or redefined in the referenced function subprogram, the actual argument must be a variable name, an array element name, or an array name.

Execution of a function subprogram reference results in an association of actual arguments with all appearances of dummy arguments in executable statements and adjustable dimensions in the defining subprogram. If the actual argument is an expression, this association is by value rather than by name. Following these associations, the first executable statement of the defining subprogram is executed.

An actual argument which is an array name containing variables in the subscript could, in every case, be replaced by the same argument with a constant subscript containing the same values as would be derived by computing the variable subscript just before the association of arguments takes place.

If a dummy argument of a function subprogram is an array name, the corresponding actual argument must be an array name or an array element name.

SUBROUTINE

PURPOSE: **To define a user-specified subroutine, which may be compiled independently from a program unit which references it.**

FORMAT:

SUBROUTINE s, comment

SUBROUTINE s (a_1, a_2, ..., a_n), comment

s = the symbolic name of the subroutine

a = dummy argument

comment = up to 84 character comment

COMMENTS: To reference a subroutine, a program unit uses a CALL statement.

The SUBROUTINE statement must be the first statement in a subroutine subprogram.

The SUBROUTINE statement cannot be used in a function subprogram. The comment and its preceeding comma is optional. If present it is passed to the loader via the relocatable object code.

EXAMPLES:

CALL MATRX	SUBROUTINE MATRX, INVERSE- DATE 19 OCT
ʃʃ	ʃʃ
CALL SUBR(I,J)	RETURN
	END
	SUBROUTINE SUBR (K,L), DATE 30 OCT 76
	ʃʃ
	RETURN
	END

NOTE: The ",comment" in the SUBROUTINE statement is an extension of the standard.

Defining Subroutines

The symbolic name of the subroutine must not appear in any statement except as the symbolic name of the subroutine in the SUBROUTINE statement itself.

The symbolic names of the dummy arguments may not appear in an EQUIVALENCE, COMMON, or a DATA statement in the subroutine.

A dummy parameter can be used to dimension an array name, which also appears as a dummy parameter of the subroutine. An array which is declared with dummy dimensions in a subroutine must correspond to an array which is declared with constant dimensions (through some sequence of argument association) in a calling program unit. If a parameter array is declared with values (instead of dummy dimensions) in a subroutine, the actual values must be specified for the first (N-1) dimensions. An array declared with dummy dimensions may not be in common.

The symbolic name of a dummy argument may be used to represent a variable, array, another subroutine or a function subprogram.

The subroutine defines or redefines one or more of its arguments to return results.

The subroutine may contain any statements except a FUNCTION statement, BLOCK DATA statement, PROGRAM statement, another SUBROUTINE statement, or any statement that directly or indirectly references the subroutine being defined.

Referencing Subroutines

The actual arguments which constitute the argument list must agree in order, number and type with the corresponding dummy arguments in the defining subroutine.

An actual argument in a subroutine reference may be one of the following:

- A constant
- A variable name
- An array element name
- An array name
- Any other expression
- A FORTRAN IV library function name
- A user-defined function or subroutine subprogram name

If an actual argument is a function subprogram name or a subroutine name, the corresponding dummy argument must be used as a function subprogram name or a subroutine name, respectively.

If an actual argument corresponds to a dummy argument defined or redefined in the referenced subroutine, the actual argument must be a variable name, an array element name, or an array name.

Execution of a subroutine reference results in an association of actual arguments with all appearances of dummy arguments in executable statements and adjustable dimensions in the defining subroutine. If the actual argument is an expression, this association is by value rather than by name. Following these associations, the first executable statement of the defining subroutine is executed.

An actual argument which is an array name containing variables in the subscript could, in every case, be replaced by the same argument with a constant subscript just before the association of arguments takes place.

If a dummy argument of a subroutine is an array name, the corresponding actual argument must be an array name or an array element name.

BLOCK DATA SUBPROGRAMS

PURPOSE: To define a block data subprogram, which may be compiled independently from a program unit which references it.

FORMAT:

```
BLOCK DATA name, comment
name    = an optional name
comment = up to 84-character comment
```

COMMENTS: The block data subprogram is used to:

1. Define the size of and generate subprograms which reserve space for each named common block, except EMA common.

2. Optionally to initialize the variables in one (or more) named common block.

The BLOCK DATA statement must be the first non-comment statement in a block data subprogram.

The name specified in the BLOCK DATA statement is used only in the heading produced for the listing. Each different named common block within a block data subprogram will produce a separate subprogram module which will have the common block name. The comment string will be passed to the loader with each named common subprogram produced.

Each named common block, except EMA common, referenced in an executable FORTRAN program must be defined in a block data subprogram. This is necessary to reserve room for the named common block.

EXAMPLES:

```
BLOCK DATA XYZ,DATE=770707
COMMON/XYZ/A(10),B(200),KKK
COMMON/BITS/IB(16)
DATA IB/1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,100000B
```

NOTE: The ",comment" parameter in the BLOCK DATA statement is an extension of the standard.

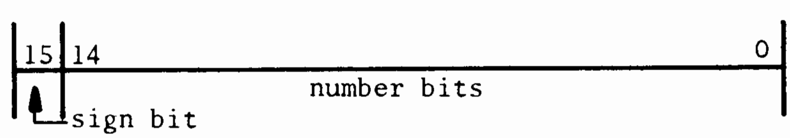
APPENDIX A
DATA FORMAT IN MEMORY

The six types of data used in FORTRAN IV (integer, real, double precision, complex, logical, and Hollerith) have the following format when stored in memory.

INTEGER FORMAT

PURPOSE: An integer datum is always an exact representation of a positive, negative or zero valued integer, occupies one 16-bit word and has a range of -2^{15} to $2^{15}-1$.

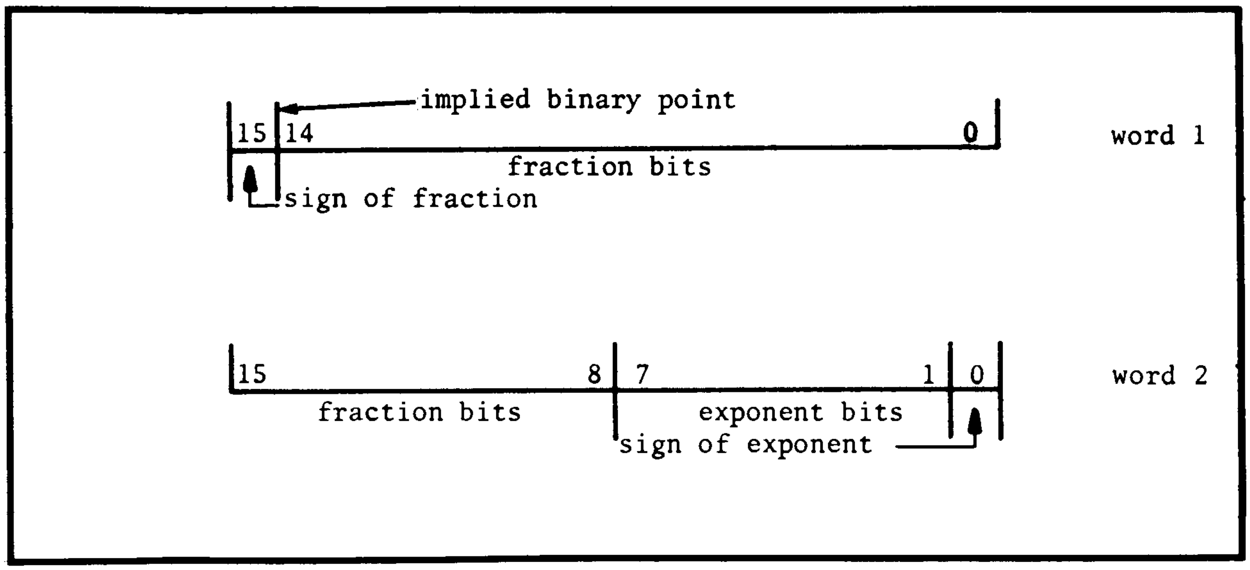
FORMAT:



REAL FORMAT

PURPOSE: A real datum is a processor approximation to the positive, negative or zero valued real number, occupies two consecutive 16-bit words in memory and has an approximate range of 10^{-38} to 10^{38}.

FORMAT:



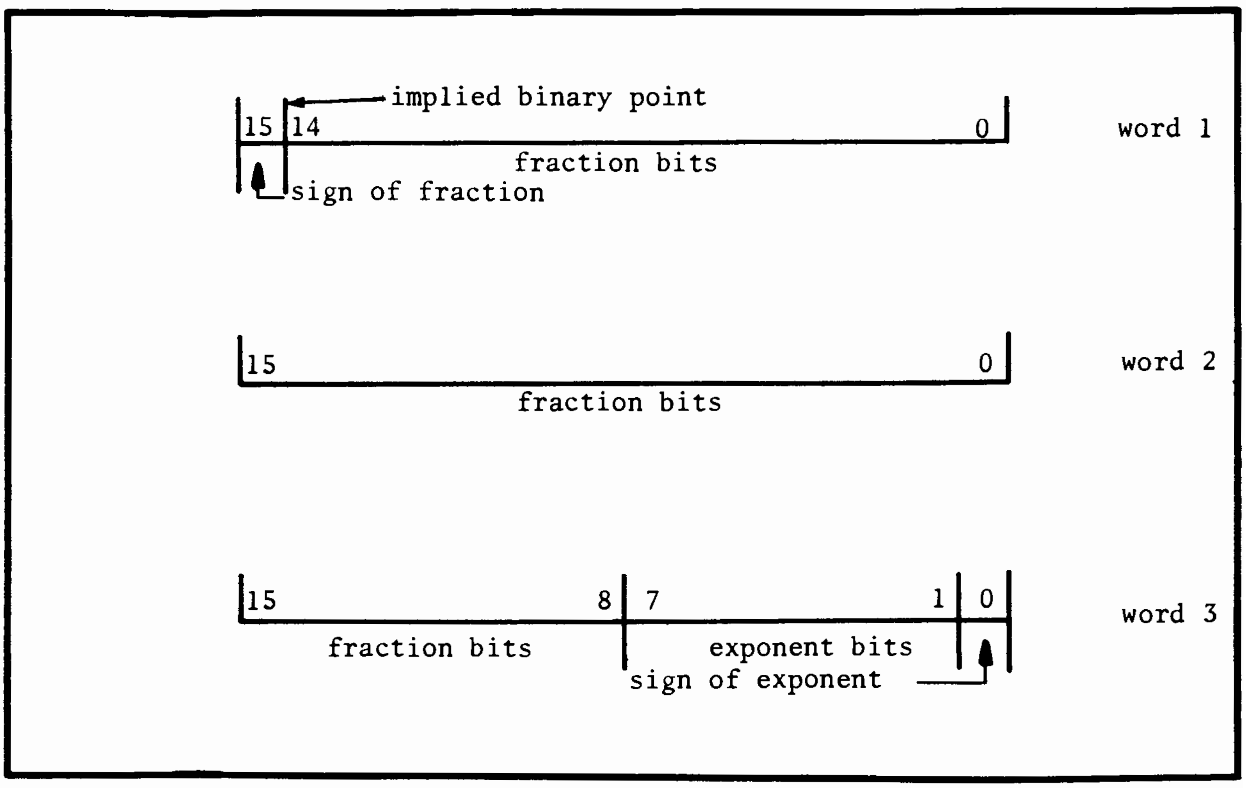
COMMENTS: A real number has a 23-bit fraction and a 7-bit exponent.

Significance (to the user) is to six or seven decimal digits, depending upon the magnitude of the leading digit in the faction.

3 WORD DOUBLE PRECISION FORMAT

PURPOSE: A double precision datum is a processor approximation to a positive, negative or zero valued double precision number, occupies three consecutive 16-bit words in memory and has an approximate range of 10^{-38} to 10^{38}.

FORMAT:



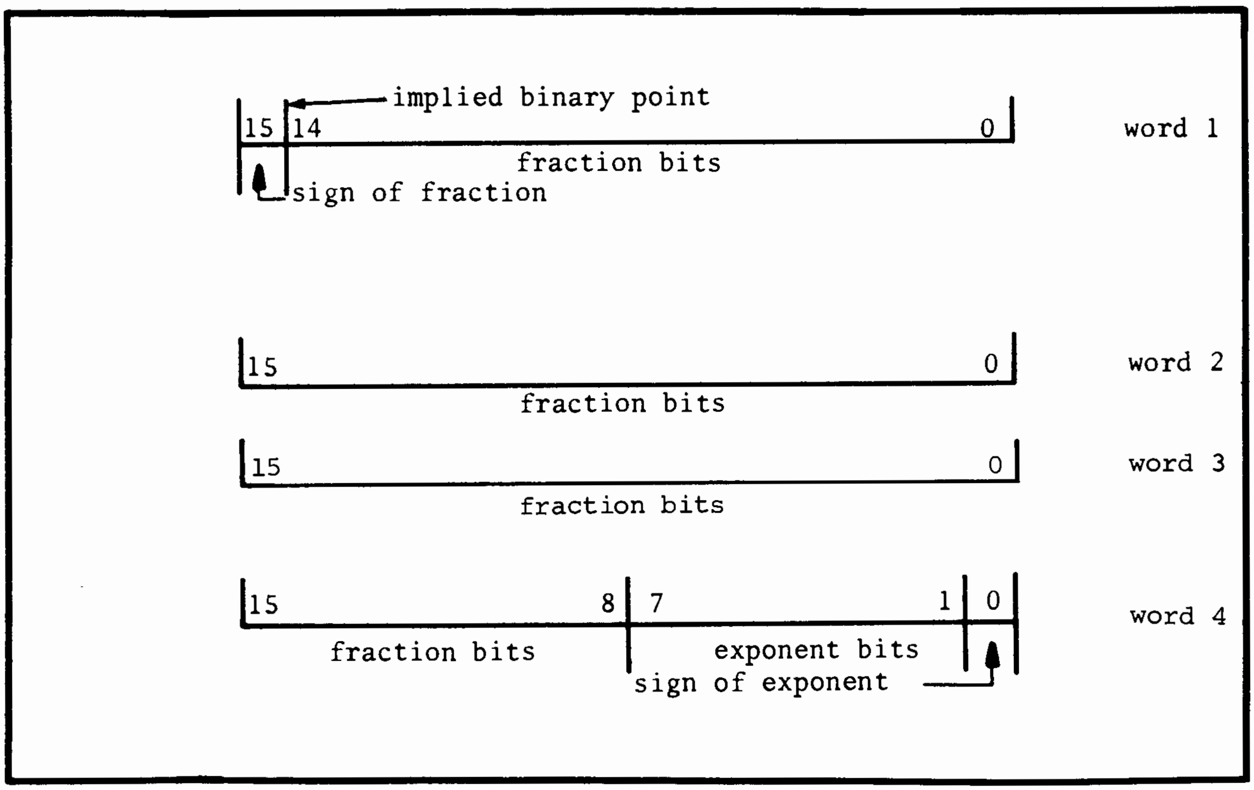
COMMENTS: A double precision number has a 39-bit fraction and a 7-bit exponent.

Significance (to the user) is from 11.44 to 11.74 decimal digits, depending upon the magnitude of the leading bit in the fraction.

4-WORD DOUBLE PRECISION FORMAT

PURPOSE: A double precision datum is a processor approximation to a positive, negative or zero valued double precision number, occupies four consecutive 16-bit words in memory and has an approximate range of 10^{-38} to 10^{38}.

FORMAT:



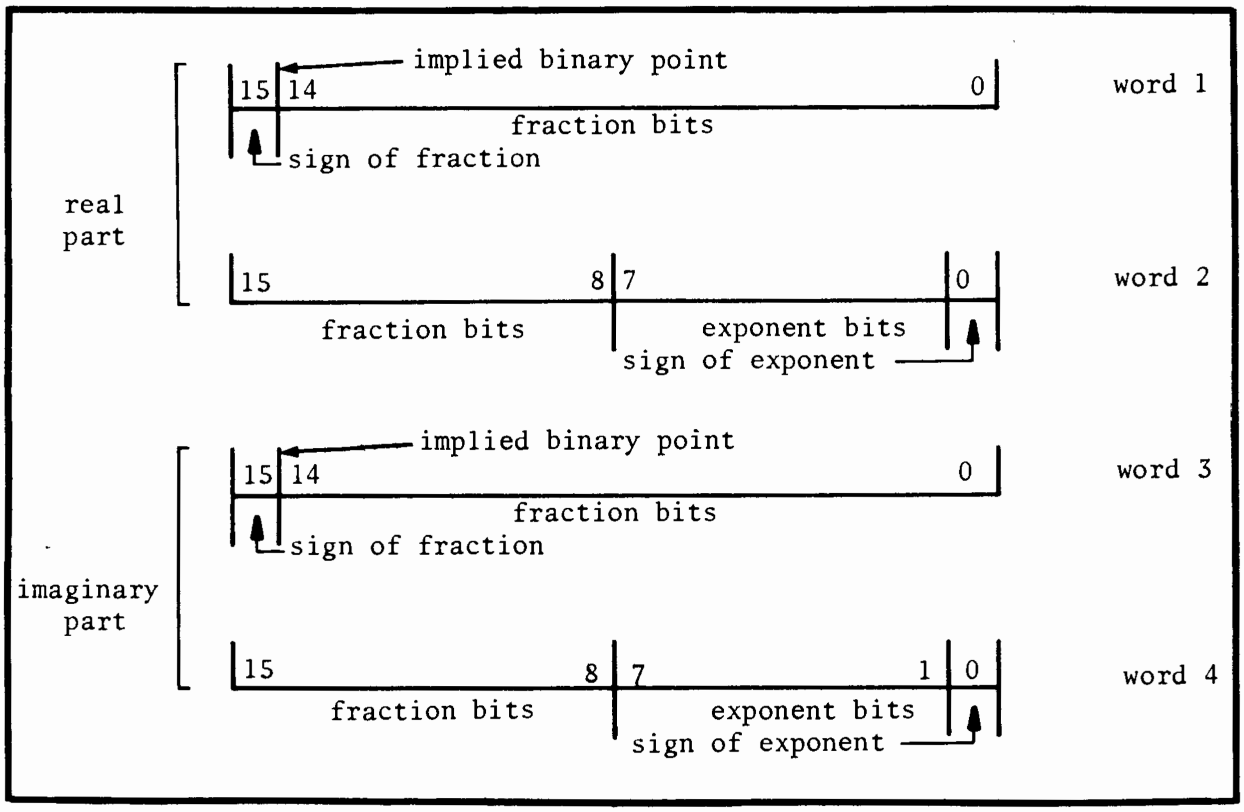
COMMENTS: A double precision number has a 55-bit fraction and a 7-bit exponent.

Significance (to the user) is from 16.26 to 16.56 decimal digits, depending upon the magnitude of the leading digit in the fraction.

COMPLEX FORMAT

PURPOSE: A complex datum is a processor approximation to the value of a complex number and occupies four consecutive 16-bit words in memory. Both the real and imaginary parts have an approximate range of 10^{-38} to 10^{38}.

FORMAT:

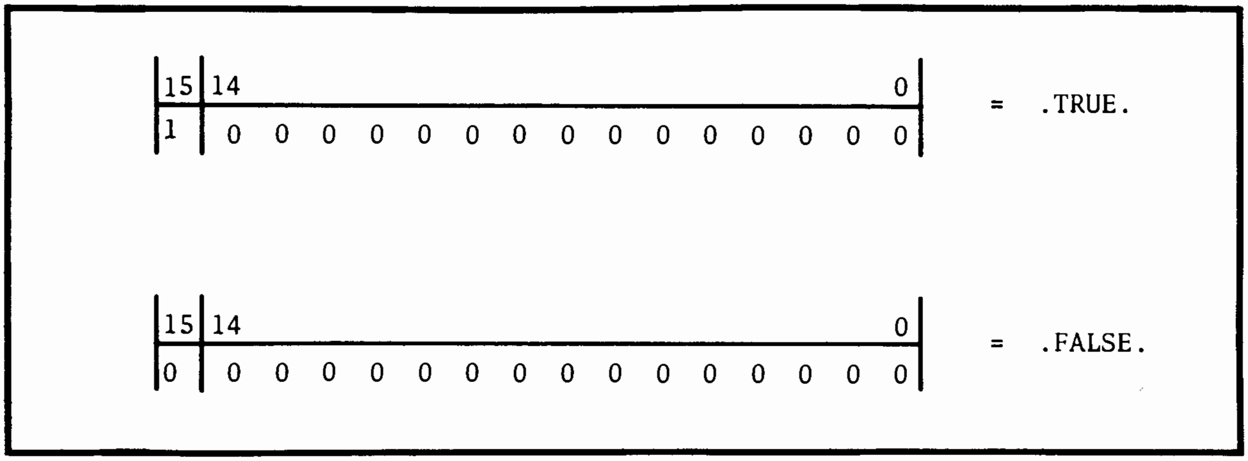


COMMENTS: Both the real part and the imaginary part have 23-bit fractions and 7-bit exponents; both have the same significance as a real number.

LOGICAL FORMAT

PURPOSE: A logical datum occupies one 16-bit word in memory. The sign bit determines the truth value: 1 = true, 0 = false.

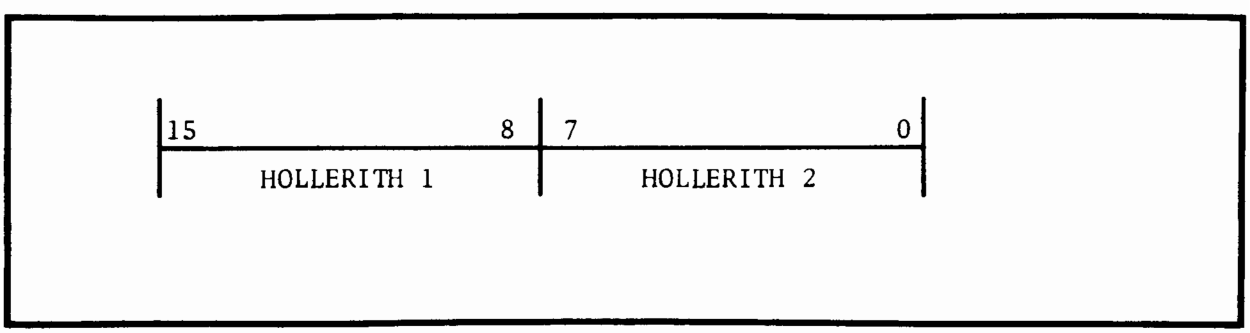
FORMAT:



HOLLERITH FORMAT

PURPOSE: A Hollerith datum is a one or two character string taken from the HP ASCII character set; it occupies one 16-bit word in memory.

FORMAT:



APPENDIX B
COMPOSING AN RTE FORTRAN IV JOB DECK

After a source program has been written, it is submitted as a FORTRAN IV job deck. A job deck is input in the form of a disc file, punched cards, a source paper tape or through a teleprinter. The job deck has the following form:

```
FORTRAN CONTROL STATEMENT
MAIN PROGRAM
     ∫∫
END STATEMENT
SUBPROGRAM(1)
     ∫∫
END STATEMENT
     .
     .
     .
SUBPROGRAM(n)
     ∫∫
END STATEMENT
FORTRAN END JOB STATEMENT
```

FORTRAN CONTROL STATEMENT

The FORTRAN CONTROL STATEMENT specifies the type of output to be produced by the compiler. The CONTROL STATEMENT parameters within a job deck may be overridden using the *options* parameter when the FORTRAN IV Compiler is invoked. Refer to the FORTRAN IV Operations Section in this manual for more information.

FORTRAN END JOB STATEMENT

A FORTRAN end job statement is a source statement that contains the currency symbol ($) in column one or END$ in columns 7-72.

APPENDIX C

SUMMARY OF COMPATIBILITY WITH ANSI FORTRAN IV

The RTE FORTRAN IV compiler conforms to the American National Standards Institute FORTRAN IV specifications as described in the ASA publication X3.9-1966, with the following exceptions and extensions.

EXCEPTIONS TO STANDARD

Program, subprogram, and external names are limited to five characters. Six character symbols are accepted but are shortened to five characters by deletion of the fifth character. For example, the program name JOHN01 is changed to JOHN1 by the RTE FORTRAN IV compiler.

Intrinsic functions are treated as external functions.

Integer values occupy one word less than real values.

RTE FORTRAN IV requires that each named common block be described in a block data subprogram even if no variables are to be initialized.

The FORTRAN IV Formatter supports the transfer of data records containing a maximum of 132 characters within a formatted READ or WRITE operation, or a maximum of 60 words within an unformatted (binary) READ or WRITE operation.

The Formatter processes READ or WRITE requests for the transfer of records larger than these limits by dividing the original record into records sized to match the limits. This process affects the file positioning operations.

For example, assume that a READ request is issued for a 1000-word binary record. The Formatter divides this record into 16 records of 60 words each and 1 record of 20 words. In order to backspace and re-read from the beginning of the original record, 17 backspace operations must be performed prior to the request to re-read the data.

EXTENSIONS OF STANDARD

A subscript expression may be any arithmetic expression allowed in RTE FORTRAN IV. However, if an expression is of a type other than INTEGER, it is converted to Type-INTEGER after it has been evaluated.

The initial, terminal, and step-size parameters of a DO statement or an implied DO list may be any arithmetic expression. If the expressions are not of Type-INTEGER, they are converted to Type-INTEGER after they have been evaluated. The step-size parameter may be either positive or negative, thereby allowing either incrementing or decrementing the terminal parameter value. (Implied DO lists may use only integer arithmetic expressions which do not reference functions that perform I/O operations or execute READ/WRITE statements.)

Comment lines may appear anywhere including within statements continued on additional lines.

Strings may appear in PROGRAM, FUNCTION, SUBROUTINE, and BLOCK DATA statements.

Specification of a comma as a statement separator is allowed in a DO statement.

For all statements, there is no limit to the number of continuation lines.

The integer variable reference in a computed GO TO can be replaced by any arithmetic expression. Non-integer expressions are converted to type integer before the GO TO statement is executed. If the value of the expression is less than one, the first statement in the computed GO TO list is executed. If the value is greater than the number of statements listed in the GO TO, the last statement in the computed GO TO list is executed.

The Hollerith constant $nHc_1c_2...c_n$ (for $n<9$) may be used in any arithmetic expression where a constant or an expression of type implied by n (see page 2-9) is permitted. Note, however, the n=0 is not permitted and that if n is odd the c_n is stored in the left half of the computer word, with a blank character in the right half.

Any two arithmetic types may be mixed in any relational or arithmetic operation except exponentiation.

Additional types of exponentiation are permitted. (See Table 3-2.)

An unsubscripted array name is an admissible list element in a DATA statement. In this case, the correspondence with constant values is as follows: If the array has n elements, then the next m constants from the list are used to initialize the array in the order in which it is stored (column order). If the remainder of the constant list (at the time the array name is encountered) has $m < n$ elements in it, then only the first m elements of the array are initialized.

ASSIGN statements may be used with FORMAT statement numbers and the integer variable then can be used in READ and WRITE statements.

Integer variables defined by the ASSIGN statement may be passed to functions and subfunctions.

APPENDIX D

COMPATIBILITY BETWEEN HP FORTRAN AND RTE FORTRAN IV

RTE FORTRAN IV contains some language extensions to provide compatibility with HP FORTRAN. These features are:

Special characters included with ASCII input data can direct its formatting (free field input); a FORMAT statement need not be specified in the source program.

Alphanumeric data can be written without giving the character count by specifying heading and editing information in the FORMAT statement through "..." entries.

The Aw conversion code of HP FORTRAN is equivalent to the Rw conversion code in RTE FORTRAN IV. A single character stored in a word under R format control is placed in the right half of the word with zeros in the left half. On output, using the Rw format, the right half of the word is written. A HP FORTRAN program using an A1 FORMAT specification may have to be changed to use the R1 specification. The user may also use calls to OLDIO. (See the Relocatable Subroutines manual.)

The END statement is interpreted as a RETURN statement (in a subprogram) or as a STOP statement (in a main program). A RETURN statement in a main program is interpreted as a STOP statement.

The HP FORTRAN External Functions which perform masking (Boolean) operations (IAND, IOR, NOT) and test the sense switches (ISSW) are retained as RTE FORTRAN IV library functions.

The two-branch arithmetic IF statement (IF (e) n_1, n_2) is retained in RTE FORTRAN IV.

Octal constants are valid in RTE FORTRAN IV.

Using an unsubscripted array name always denotes the first element of that array, except in an I/O statement or a DATA statement, where the entire array is referenced. A single subscript, i, with a multiply-dimensioned array, denotes the ith element of the array as it is stored (in column order).

The PROGRAM statement syntax for HP FORTRAN differs between the RTE-II/III and the RTE-M Operating Systems. The difference is in the handling of the optional parameter string and the inclusion of a comment in the PROGRAM statement. Refer to the RTE-II, RTE-III, and RTE-M Programming and Operating Manuals for specific details.

In the previous HP FORTRAN IV compiler, FORMAT statement code was generated in line within the program code produced by the compiler. This required use of a jump operation to avoid execution of the FORMAT statement. The FORMAT statement number was associated with the jump operation which allowed the FORMAT statement number to be used to control the flow of the program (that is, in GO TO, IF, or DO statements).

Because the ANSI standard for FORTRAN IV dictates that statement labels used in program control statements must be associated only with executable statements within the same program unit, RTE FORTRAN IV does not allow the FORMAT statement number to be used in this manner. The RTE FORTRAN IV compiler generates FORMAT statement code in the data area following the program code. The jump operation is not generated and the statement number is associated directly with the FORMAT statement. This allows usage of the ASSIGN statement with FORMAT statements but precludes the use of a FORMAT statement number in program control statements such as GO TO, IF, or DO statements.

An additional difference between the previous HP FORTRAN IV compiler and the RTE FORTRAN IV compiler exists in the handling of array addresses. The HP FORTRAN IV compiler generated the address of each array mentioned in the specification statements prior to generation of any executable code. Usually, the array addresses immediately preceded the actual array.

The RTE FORTRAN IV compiler generates array addresses only if they are needed. If generated, the addresses usually appear in the data area following the program code while the actual array precedes the program code.

APPENDIX E
CROSS REFERENCE SYMBOL TABLE

The RTE FORTRAN IV Compiler provides the option of producing a cross reference listing of symbols and labels used in the source program. The sample program listing shown in Appendix F contains a cross reference symbol table as the last item listed. If requested, the cross reference symbol table is always the last listing produced for each compiled program unit.

REQUESTING A CROSS REFERENCE SYMBOL TABLE LISTING

The optional parameter C is used in the FORTRAN Control Statement to request a cross reference symbol table. Appendix J describes the format and parameters of the FORTRAN Control Statement.

CHARACTERISTICS OF TABLE

Each symbol is printed followed by the line numbers in which the symbol appears. Multiple references in one line to the same symbol are noted. Statement labels are preceded by the @ character.

Up to eight line numbers are printed per line of the cross reference symbol table. The line numbers are listed in ascending order except when they occur in an EQUIVALENCE statement. For example,

```
0099   COMMON N
0100   EQUIVALENCE (N(1), M(1))
0101   DIMENSION N(50), M(50)
0102   N(1)=1
```

produces, for the symbol N, the following cross reference information:

```
N       0099  0101  0100  0102
```

ERROR CONDITIONS

The cross reference symbol table is not complete for lines which contain compilation errors, since compilation is terminated at the point in the line where the error is detected. Also for programs with a large number of EQUIVALENCE statements some references in the EQUIVALENCE statements may be absent from the cross reference.

APPENDIX F

SAMPLE LISTING OF RTE FORTRAN IV PROGRAM

```
PAGE 0001  FTN.   3:54 PM  FRI., 17  JUNE, 1977

FTN4,L,M
      BLOCK DATA X,TEST BLOCK DATA 770107
      COMMON /NAME1/BITS
      DIMENSION BITS(16)
      INTEGER BITS
      COMMON /NAME2/ B,A,C
      DIMENSION A(5),B(5,5),C(5,5,5)
C
C
      DATA BITS/1,2,4,8,16,32,64,128,256,512,1024,2048,4096,
     C 8192,16384,100000B/,A/5*5./
      END

FTN4 COMPILER: HP92060-16092 REV. 1726

**  NO WARNINGS **  NO ERRORS **
```

```
PAGE 0002  NAME1  3:54 PM  FRI., 17  JUNE, 1977

            BLOCK DATA X,TEST BLOCK DATA 770107
            COMMON /NAME1/BITS
            DIMENSION BITS(16)
            INTEGER BITS
            COMMON /NAME2/ B,A,C
            DIMENSION A(5),B(5,5),C(5,5,5)
            DATA BITS/1,2,4,8,16,32,64,128,256,512,1024,2048,4096,
     00000  000001          OCT 000001
     00001  000002          OCT 000002
     00002  000004          OCT 000004
     00003  000010          OCT 000010
     00004  000020          OCT 000020
     00005  000040          OCT 000040
     00006  000100          OCT 000100
     00007  000200          OCT 000200
     00010  000400          OCT 000400
     00011  001000          OCT 001000
     00012  002000          OCT 002000
     00013  004000          OCT 004000
     00014  010000          OCT 010000
           C 8192,16384,100000B/,A/5*5./
     00015  020000          OCT 020000
     00016  040000          OCT 040000
     00017  100000          OCT 100000
            END

BLOCK COMMON NAME1  SIZE =  00016
```

```
 0002          BLOCK DATA X,TEST BLOCK DATA 770107
 0003          COMMON /NAME1/BITS
 0004          DIMENSION BITS(16)
 0005          INTEGER BITS
 0006          COMMON /NAME2/ B,A,C
 0007          DIMENSION A(5),B(5,5),C(5,5,5)
 0010          DATA BITS/1,2,4,8,16,32,64,128,256,512,1024,2048,4096,
 0011         C 8192,16384,100000B/,A/5*5./
                                    BSS 00042B
        00062  050000               OCT 050000
        00063  000006               OCT 000006
        00064  050000               OCT 050000
        00065  000006               OCT 000006
        00066  050000               OCT 050000
        00067  000006               OCT 000006
        00070  050000               OCT 050000
        00071  000006               OCT 000006
        00072  050000               OCT 050000
        00073  000006               OCT 000006
 0012          END
                                    BSS 00372B

BLOCK COMMON NAME2  SIZE =  00310
```

SYMBOL TABLE

NAME	ADDRESS	USAGE	TYPE	LOCATION	
A	000062+	ARRAY(*)	REAL	L COMMON	NAME2
B	000000+	ARRAY(*,*)	REAL	L COMMON	NAME2
BITS	000000+	ARRAY(*)	INTEGER	L COMMON	NAME1
C	000074+	ARRAY(*,*,*)	REAL	L COMMON	NAME2
NAME1	000020R	COMMON LABEL	INTEGER	LOCAL	
NAME2	000466R	COMMON LABEL	INTEGER	LOCAL	

```
              PROGRAM(),MAIN NAMED COMMON
              COMMON /NAME1/BITS(16)/NAME2/B(5,5),A(5),C(5,5,5)
              INTEGER BITS
              WRITE(6,100)BITS,(A(JJ),JJ=1,5),((B(KK,KKK),KK=1,5),KKK=1,5),C
        100   FORMAT(X,16K7,/,31(5(XF10.2,2X)/))
              DO6I=1,5
              A(I)=I*I
              DO7J=1,5
              B(I,J)=I*I+J*J
              DO8K=1,5
              C(I,J,K)=I*I+J*J+K*K
        8     CONTINUE
        7     CONTINUE
        6     CONTINUE
              WRITE(6,100)BITS,A,B,C
              CONTINUE
              END


 FTN4 COMPILER: HP92060-16092 REV. 1726


 **  NO WARNINGS **  NO ERRORS **   PROGRAM = 00191      COMMON = 00000
```

```
0013          PROGRAM(),MAIN NAMED COMMON
0014          COMMON /NAME1/BITS(16)/NAME2/B(5,5),A(5),C(5,5,5)
0015          INTEGER BITS
0016          WRITE(6,100)BITS,(A(JJ),JJ=1,5),((B(KK,KKK),KK=1,5),KKK=1,5),C
       00000  000000          NOP
       00001  000001X         JSB CLRIO
       00002  000003R         DEF *-2+00003B
       00003  000242R         LDA 00242B
       00004  006400          CLB
       00005  000002X         JSB .DIO.
       00006  000261R         DEF @100
       00007  000072R         DEF 00072B
       00010  000003X         JSB .IAY.
       00011  000004X         DEF NAME1
              000000+
       00012  000020          OCT 000020
                              BSS 00002B
       00015  000243R         LDA JJ
       00016  001000          ALS
       00017  000244R         ADA 00244B
       00020  000245R         STA A.001
       00021  000005X         JSB .RIO.
       00022  100245R         DEF A.001,I
                              ORG 00013B
       00013  000246R         LDA 00246B
       00014  000243R         STA JJ
                              BSS 00006B
       00023  000243R         LDA JJ
       00024  000246R         ADA 00246B
       00025  000243R         STA JJ
       00026  003004          CMA,INA
       00027  000241R         ADA 00241B
       00030  002021          SSA,RSS
       00031  000015R         JMP 00015B
                              BSS 00004B
       00036  000252R         LDB 00252B
       00037  002400          CLA
       00040  000006X         JSB ..MAP
       00041  000253R         DEF 00253B
       00042  000250R         DEF KK
       00043  000251R         DEF KKK
       00044  000241R         DEF 00241B
       00045  000245R         STA A.001
       00046  000005X         JSB .RIO.
       00047  100245R         DEF A.001,I
                              ORG 00034B
       00034  000246R         LDA 00246B
       00035  000250R         STA KK
                              BSS 00012B
       00050  000250R         LDA KK
       00051  000246R         ADA 00246B
       00052  000250R         STA KK
       00053  003004          CMA,INA
       00054  000241R         ADA 00241B
       00055  002021          SSA,RSS
```

```
        00056  000036R             JMP 00036B
                                   ORG 00032B
        00032  000246R             LDA 00246B
        00033  000251R             STA KKK
                                   BSS 00023B
        00057  000251R             LDA KKK
        00060  000246R             ADA 00246B
        00061  000251R             STA KKK
        00062  003004              CMA,INA
        00063  000241R             ADA 00241B
        00064  002021              SSA,RSS
        00065  000034R             JMP 00034B
0017    100  FORMAT(X,16K7,/,31(5(XF10.2,2X)/))
        00066  000007X             JSB .RAY.
        00067  000010X             DEF NAME2+00074B
               000074+
        00070  000175              OCT 000175
        00071  000011X             JSB .DTA.
0018         DO6I=1,5
                                   BSS 00167B
        00261  024130  @100        ASC 1,(X
        00262  026061              ASC 1,,1
        00263  033113              ASC 1,6K
        00264  033454              ASC 1,7,
        00265  027454              ASC 1,/,
        00266  031461              ASC 1,31
        00267  024065              ASC 1,(5
        00270  024130              ASC 1,(X
        00271  043061              ASC 1,F1
        00272  030056              ASC 1,0.
        00273  031054              ASC 1,2,
        00274  031130              ASC 1,2X
        00275  024457              ASC 1,)/
        00276  024451              ASC 1,))
                                   ORG 00072B
        00072  000246R             LDA 00246B
        00073  000254R             STA I
0019         A(I)=I*I
        00074  000254R             LDA I
        00075  001000              ALS
        00076  000244R             ADA 00244B
0020         DO7J=1,5
        00077  000245R             STA A.001
        00100  000254R             LDA I
        00101  000012X             JSB .MPY
        00102  000254R             DEF I
        00103  000013X             JSB FLOAT
        00104  000014X             JSB .DST
        00105  100245R             DEF A.001,I
        00106  000246R             LDA 00246B
        00107  000255R             STA J
0021         B(I,J)=I*I+J*J
        00110  000252R             LDB 00252B
        00111  002400              CLA
        00112  000006X             JSB ..MAP
```

```
        00113  000253R         DEF 00253B
        00114  000254R         DEF I
        00115  000255R         DEF J
        00116  000241R         DEF 00241B
        00117  000245R         STA A.001
        00120  000254R         LDA I
        00121  000012X         JSB .MPY
        00122  000254R         DEF I
 0022          DO8K=1,5
        00123  000247R         STA I.001
        00124  000255R         LDA J
        00125  000012X         JSB .MPY
        00126  000255R         DEF J
        00127  000247R         ADA I.001
        00130  000013X         JSB FLOAT
        00131  000014X         JSB .DST
        00132  100245R         DEF A.001,I
        00133  000246R         LDA 00246B
        00134  000256R         STA K
 0023          C(I,J,K)=I*I+J*J+K*K
        00135  000252R         LDB 00252B
        00136  002404          CLA,INA
        00137  000006X         JSB ..MAP
        00140  000257R         DEF 00257B
        00141  000254R         DEF I
        00142  000255R         DEF J
        00143  000256R         DEF K
        00144  000241R         DEF 00241B
        00145  000241R         DEF 00241B
        00146  000245R         STA A.001
        00147  000254R         LDA I
        00150  000012X         JSB .MPY
        00151  000254R         DEF I
        00152  000247R         STA I.001
        00153  000255R         LDA J
        00154  000012X         JSB .MPY
        00155  000255R         DEF J
        00156  000247R         ADA I.001
 0024   8      CONTINUE
        00157  000260R         STA I.002
        00160  000256R         LDA K
        00161  000012X         JSB .MPY
        00162  000256R         DEF K
        00163  000260R         ADA I.002
        00164  000013X         JSB FLOAT
        00165  000014X         JSB .DST
        00166  100245R         DEF A.001,I
 0025   7      CONTINUE
        00167  000256R @8      LDA K
        00170  000246R         ADA 00246B
        00171  000256R         STA K
        00172  003004          CMA,INA
        00173  000241R         ADA 00241B
        00174  002021          SSA,RSS
        00175  000135R         JMP 00135B
```

```
0026    6     CONTINUE
        00176  000255R @7       LDA J
        00177  000246R          ADA 00246B
        00200  000255R          STA J
        00201  003004           CMA,INA
        00202  000241R          ADA 00241B
        00203  002021           SSA,RSS
        00204  000110R          JMP 00110B
0027          WRITE(6,100)BITS,A,B,C
        00205  000254R @6       LDA I
        00206  000246R          ADA 00246B
        00207  000254R          STA I
        00210  003004           CMA,INA
        00211  000241R          ADA 00241B
        00212  002021           SSA,RSS
        00213  000074R          JMP 00074B
        00214  000242R          LDA 00242B
        00215  006400           CLB
        00216  000002X          JSB .DIO.
        00217  000261R          DEF @100
        00220  000236R          DEF 00236B
        00221  000003X          JSB .IAY.
        00222  000004X          DEF NAME1
               000000+
        00223  000020           OCT 000020
        00224  000007X          JSB .RAY.
        00225  000010X          DEF NAME2+00062B
               000062+
        00226  000005           OCT 000005
        00227  000007X          JSB .RAY.
        00230  000010X          DEF NAME2
               000000+
        00231  000031           OCT 000031
0028          CONTINUE
        00232  000007X          JSB .RAY.
        00233  000010X          DEF NAME2+00074B
               000074+
        00234  000175           OCT 000175
        00235  000011X          JSB .DTA.
0029          END
        00236  000015X          JSB EXEC
        00237  000241R          DEF 00072B+00147B
        00240  000242R          DEF 00242B
        00241  000005           OCT 000005
        00242  000006           OCT 000006
                        JJ      BSS 00001B
        00244  000010X          DEF NAME2+00060B
               000060+
                        A.001   BSS 00001B
        00246  000001           OCT 000001
                        I.001   BSS 00003B
        00252  000002           OCT 000002
        00253  000010X          DEF NAME2
               000000+
                        I       BSS 00003B
```

```
     00257  000010X        DEF NAME2+00074B
            000074+
                    I.002  BSS 00017B
```

SYMBOL TABLE

NAME	ADDRESS	USAGE	TYPE	LOCATION	
@100	000261R	STATEMENT NUMBER			
@6	000205R	STATEMENT NUMBER			
@7	000176R	STATEMENT NUMBER			
@8	000167R	STATEMENT NUMBER			
A	000062+	ARRAY(*)	REAL	L COMMON	NAME2
B	000000+	ARRAY(*,*)	REAL	L COMMON	NAME2
BITS	000000+	ARRAY(*)	INTEGER	L COMMON	NAME1
C	000074+	ARRAY(*,*,*)	REAL	L COMMON	NAME2
CLRIO	000001X	SUBPROGRAM	REAL	EXTERNAL	
EXEC	000015X	SUBPROGRAM	REAL	EXTERNAL	
FLOAT	000013X	SUBPROGRAM	REAL	EXTERNAL	
I	000254R	VARIABLE	INTEGER	LOCAL	
J	000255R	VARIABLE	INTEGER	LOCAL	
JJ	000243R	VARIABLE	INTEGER	LOCAL	
K	000256R	VARIABLE	INTEGER	LOCAL	
KK	000250R	VARIABLE	INTEGER	LOCAL	
KKK	000251R	VARIABLE	INTEGER	LOCAL	
NAME1	000004X	COMMON LABEL	INTEGER	EXTERNAL	
NAME2	000010X	COMMON LABEL	INTEGER	EXTERNAL	

```
0030  $
```

APPENDIX G

RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

TYPES OF COMPILER DIAGNOSTICS

There are three types of RTE FORTRAN IV compiler diagnostics:

WARNING: The compiler continues to process the statement, but the object code may be erroneous. The program should be recompiled.

ERROR: The compiler ignores the remainder of the erroneous source statement, including any continuation lines. The object code is incomplete, and the program must be recompiled.

DISASTR: The compiler ignored the remainder of the FORTRAN IV job. The error must be corrected before compilation can proceed.

> *NOTE: If an error occurs in a program, the object code will contain a reference to the non-system external name .BAD. This prevents loading of the object tape, unless forced by the user. It is strongly recommended that a program with compilation errors not be executed. This reference is not produced for warnings.*

FORMAT OF COMPILER DIAGNOSTICS

When an error is detected in a source statement, the source statement number is printed, followed by the statement text. A question mark (?) is printed after the erroneous column. Then, a message is printed in the format:

```
                          ( WARNING )
** program name **        { ERROR   }  nn DETECTED AT COLUMN cc
                          ( DISASTR )
```

program name = the name of the program being compiled

nn = the diagnostic error number

cc = the column number of the source line being scanned when the error was detected

> *NOTE: If cc=01, the error is in the source line preceding the last line printed. If cc=00, there is an error in an EQUIVALENCE group, and the group (or a portion of the group) is printed before the error message.*

When the END statement is encountered by the compiler and undefined source program statement numbers still exist, an error message is printed of the form:

@ *nnnnn* UNDEFINED

nnnnn is the statement number that did not appear in columns 1 through 5 of any of the initial lines of the program just compiled.

At this point, a report is printed of any six-character names that will be shortened to five characters.

Following the listing of the source program, a summary line is listed of the form:

** *nn* ERRORS ** ** *mm* WARNINGS PROGRAM = *xxxxx* COMMON = *yyyyy*

nn is the number of errors detected (nn-NO, if no errors were detected).

mm is the number of warnings detected (mm=NO if no warnings were detected).

xxxxx is the decimal number of main memory locations required for the program object code.

yyyyy is the decimal number of main memory locations required for the blank common block. (The size of named common blocks is printed immediately following the listing of the block data subprogram which defines each block.)

When compilation is completed, a summary message is displayed at the system console. This message reports the number of disaster, error, or warning conditions encountered during compilation. The RTE FORTRAN IV compiler returns this information via the parameter return subroutine PRTN (see the appropriate Operating System Programming and Operating Manual for a description of this subroutine). The message appears in the form:

$END FTN4: *nn* DISASTRS *nn* ERRORS *nn* WARNINGS

nn is the total number of DISASTR, ERROR, or WARNING conditions encountered during compilation of all programs in the job deck (*nn* = NO, if none were encountered).

The parameters returned via PRTN are:

parameter 1 - the total value of parameters 2 thru 4.

parameter 2 - the number of disasters encountered.

parameter 3 - the number of errors encountered.

parameter 4 - the number of warnings encountered.

parameter 5 - the revision level of the compiler.

TABLE G-1

RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
01	COMPILER CONTROL STATEMENT MISSING There is no FTN or FTN4 directive preceding the FORTRAN IV job.	Compilation terminated	
02	ERROR IN COMPILER CONTROL STATEMENT Incorrect syntax or illegal parameter in FTN or FTN4 directive.	Compilation terminated	
03	SYMBOL TABLE OVERFLOW Insufficient memory exists for continuing compilation.	Compilation terminated	Reduce number of symbols (constants, variable names and statement numbers) in program and shorten lengths of variable names and statement numbers.
04	LABELED COMMON Name too long or "/" missing or name already used for variable.	Statement terminated	
05	IMPLICIT statement used to define default type for some character more than once. The last defined type is used.	Warning	
06	END OF FILE OCCURRED BEFORE "$" Source input file ended before the "$" or END$ statement ending the FORTRAN IV job was encountered.	Compilation terminated	Example: no "$" or END$ statement at end of source file
07	RETURN IN MAIN PROGRAM A RETURN statement occurs in a main program. It is interpreted as a STOP statement.	Comment	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
08	ILLEGAL COMPLEX NUMBER A complex number does not conform to the syntax: (± real constant, ± real constant)	Warning	Example: non-real constant as part of complex number: (1.0,2)
09	MISMATCHED OR MISSING PARENTHESIS An unbalanced parenthesis exists in a statement or an expected parenthesis is missing.	Statement terminated	
10	ILLEGAL STATEMENT The statement in question cannot be identified.	Statement terminated	Examples: The first 72 columns of a statement do not contain one of the following: (a) the '=' sign if it is a statement function or an assignment statement, (b) the ',' following the initial parameter if it is a DO statement, (c) 'IF(' for an IF statement or (d) the first four characters of the statement keyword for all other statements (e.g. DIME, WRIT). A statement keyword may also be misspelled in the first four characters (e.g. RAED).
11	ILLEGAL DECIMAL EXPONENT Non-integer constant exponent in floating point constant.	Statement terminated	
12	INTEGER CONSTANT EXCEEDS MAXIMUM INTEGER SIZE An integer constant is not in the range of -32768 to 32767.	Statement terminated	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
13	HOLLERITH STRING NOT TERMINATED In the use of 'nH', less than n characters follow the H before the end of the statement occurs. In a FORMAT statement, an odd number of quotation marks surround literals.	Statement terminated	
14	CONSTANT OVERFLOW OR UNDERFLOW The binary exponent of a floating point constant exceeds the maximum, i.e., $\lvert exponent \rvert > 38$. If underflow, the value is set to 0.	Warning	
15	ILLEGAL SIGN IN LOGICAL EXPRESSION An arithmetic operator precedes a logical constant.	Warning	Examples: -.FALSE., +.TRUE.
16	ILLEGAL OCTAL NUMBER An octal number has more than six digits, is greater than 177777B or is non-integer.	Statement terminated	Examples: 0000012B, 277777B, .1234B
17	MISSING OPERAND - UNEXPECTED DELIMITER Missing subscript in an array declarator in a DIMENSION statement or missing name in an EQUIVALENCE group.	Statement terminated	Examples: DIMENSION A(2,4,) EQUIVALENCE (B(2))
18	ILLEGAL CONSTANT USAGE A constant is used where a symbolic name is expected. Some illegal usages are when a constant is used as a subprogram or statement function name, as a parameter or a subprogram or statement function, as an element of an EQUIVALENCE group, or as the blockname in a $EMA directive.	Warning	Examples: SUBROUTINE 1234 FUNCTION NAME(X,12,A) EQUIVALENCE (I,5) $EMA (1234,0)

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
19	INTEGER CONSTANT REQUIRED An integer variable is used where an integer constant is required.	Statement terminated	Examples: A non-dummy integer variable is used in an array declarator or an integer variable is used as a subscript in an EQUIVALENCE group.
20	EMPTY HOLLERITH STRING In an 'nH' specification, n=0.	Statement terminated	
21	NON-OCTAL DIGIT IN OCTAL CONSTANT A digit > 7 occurs in an octal constant.	Warning	Example: 1289B
22	ILLEGAL USAGE OF NAME A variable is used as a subprogram name or an array name is used as a DO statement index variable.	Statement terminated	
23	DO TERMINATOR DEFINED PREVIOUS TO DO STATEMENT The terminating statement of a DO loop comes before the DO statement or is the DO statement itself.	Statement terminated	Example: 10 DO 10 I=1,5
24	ILLEGAL CONSTANT A variable name is expected but a constant appears.	Statement terminated	
25	ILLEGAL SUBPROGRAM NAME USAGE A subprogram name is used where a variable name or constant is expected.	Statement terminated	Examples: A subprogram name occurs on the left-hand side of an assignment statement. A FUNCTION or statement function name occurs as an operand in an expression but no argument list is given.

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
26	INTEGER VARIABLE OR CONSTANT REQUIRED Non-integer value is used where an integer quantity is required.	Statement terminated	Examples: A subscript in an EQUIVALENCE group element is a non-integer constant. A READ or WRITE statement has a non-integer logical unit reference.
27	STATEMENT NUMBER PREVIOUSLY DEFINED The same statement number appears on two statements.	Statement terminated	
28	UNEXPECTED CHARACTER Syntax of statement is incorrect.	Statement terminated	
29	ONLY STATEMENT NUMBER ON SOURCE LINE Some source code must appear within the first 72 columns of a numbered statement.	Warning	
30	IMPROPER DO NESTING OR ILLEGAL DO TERMINATING STATEMENT The ranges of nested DO loops overlap or a statement such as a GO TO, IF, RETURN or END terminated a DO loop.	Statement terminated	
31	STATEMENT NUMBER STARTS WITH NON-DIGIT A statement number must be a 1-5 digit integer.	Statement terminated	Example: Statement source code appears in columns 1-5 of first line of a statement.

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
32	INVALID STATEMENT NUMBER OR ILLEGAL USAGE OF A STATEMENT NUMBER A statement number has more than five digits, or it contains a non-digit character, or it is undefined. A statement number is of a wrong statement type.	Statement terminated	Examples: GOTO 100 100 FORMAT(-) WRITE(6,10) 10 J=1
33	VARIABLE NAME USED AS SUBROUTINE NAME A name which has been previously used as a variable is now used in a subprogram reference.	Statement	Example: A=SIN B=SIN(X)
34	STATEMENT OUT OF ORDER Source statements must be in the order 1. Specification, 2. DATA, 3. Statement Functions, and 4. Executable statements.	Statement terminated	Examples: A subprogram name occurring, with an argument list, on the left-hand side of an assignment statement may also generate this error message.
35	NO PATH TO THIS STATEMENT OR UNNUMBERED FORMAT STATEMENT The statement can never be executed since it is not numbered and it follows a transfer of control statement. A FORMAT statement is not numbered and therefore it cannot be used by the program.	Comment	
36	DOUBLY DEFINED COMMON NAME A name occurs more than once in a COMMON block.	Statement terminated	
37	ILLEGAL USE OF DUMMY VARIABLE A subprogram parameter occurs in a COMMON statement or dummy variables are equivalenced.	Statement terminated	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
38	MORE SUBSCRIPTS THAN DIMENSIONS An array name is referenced using more subscripts than dimensions declared for it.	Statement terminated	
39	ADJUSTABLE DIMENSION IS NOT A DUMMY PARAMETER The variable dimension used with a dummy array name must also be a dummy parameter.	Statement terminated	
40	IMPOSSIBLE EQUIVALENCE GROUP Two entries in COMMON appear in an EQUIVALENCE group or two EQUIVALENCE groups conflict. Further EQUIVALENCE groups are ignored.	Statement terminated	
41	ILLEGAL COMMON BLOCK EXTENSION An EQUIVALENCE group requires the COMMON block base to be altered. Further EQUIVALENCE groups are ignored.	Statement terminated	
42	FUNCTION HAS NO PARAMETERS OR ARRAY HAS EMPTY DECLARATOR LIST A function must have at least one parameter. There is insufficient information to dimension an array name.	Statement terminated	
43	PROGRAM, FUNCTION OR SUBROUTINE OR BLOCK DATA NOT FIRST STATEMENT A PROGRAM statement, if present, must come first. A FUNCTION or BLOCK DATA or SUBROUTINE statement is required for subprograms.	Statement terminated	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
44	NAME IN CONSTANT LIST IN DATA STATEMENT A constant list in a DATA statement contains a non-constant.	Statement terminated	
45	ILLEGAL EXPONENTIATION Exponentiation is not permitted with data types used.	Statement terminated	
46	FUNCTION NAME UNUSED OR SUBROUTINE NAME USED In a FUNCTION subprogram, the name of the FUNCTION is not defined or a SUBROUTINE name is used within the subroutine.	Warning	
47	FORMAT SPECIFICATION NOT A LOCAL ARRAY NAME, STATEMENT NUMBER OR * OR IT IS AN EMA REFERENCE The FORMAT reference in an I/O statement is invalid.	Statement terminated	
48	ILLEGAL USE OF EMA A variably dimensioned EMA array has its dimension(s) in EMA or was mentioned without subscripts in an I/O list.	Statement terminated	Example: EMA X(I),I
49	IMPROPER USE OF NAME A variable is used as a subprogram name.	Statement terminated	
50	DO STATEMENT IN LOGICAL IF A DO statement is illegal as the "true" branch of a logical IF.	Warning	

ERROR CODE	EXPLANATION	EFFECT	ACTION
51	CONTROL VARIABLE REPEATED IN DO NEST A variable occurs as the index of two DO loops or implied DO's or a combination of these which are nested.	Statement terminated	
52	LOGICAL IF WITHIN LOGICAL IF A logical IF statement is illegal as the "true" branch of another logical IF.	Statement terminated	
53	ILLEGAL EXPRESSION OR ILLEGAL DELIMITER Arithmetic or logical expression has invalid syntax or a delimiter is invalid in statement syntax.	Statement terminated	Examples: The expression contains an illegal operator or delimiter, has a missing operator (adjacent operands) or a missing operand (adjacent operators). A READ or WRITE statement list has a delimiter syntax error.
54	DOUBLY DEFINED ARRAY NAME An array name has dimensions defined for it twice.	Statement terminated	
55	LOGICAL CONVERSION ILLEGAL Conversion of logical data to arithmetic or arithmetic to logical is not defined.	Statement terminated	
56	OPERATOR REQUIRES LOGICAL OPERANDS An operand of type INTEGER, REAL, DOUBLE PRECISION or COMPLEX has been used with .AND., .OR., .NOT.	Statement terminated	

ERROR CODE	EXPLANATION	EFFECT	ACTION
57	OPERATOR REQUIRES ARITHMETIC OPERANDS A logical operand has been used in an arithmetic operation, i.e. +, -, *, /, **, or a relational operator.	Statement terminated	
58	COMPLEX ILLEGAL One of the relational operators .LT., .LE., .GT. or .GE. has a COMPLEX operand or an IF statement has a COMPLEX expression.	Statement terminated	
59	INCORRECT NUMBER OF ARGUMENTS FOR SUBPROGRAM One of the library routines SIGN, ISIGN, IAND or IOR is called with the number of arguments less or greater than two or a library routine which is called by value is called with more than one argument.	Statement terminated	
60	ARGUMENT MODE ERROR A library routine which is called by value is called with an argument that is DOUBLE PRECISION, COMPLEX or LOGICAL.	Statement terminated	
61	LOGICAL IF WITH THREE BRANCHES The expression in an IF statement is of type logical and there are three statement numbers specified in the IF statement.	Warning	
62	ARITHMETIC IF WITH NO BRANCHES No statement numbers in an arithmetic IF statement.	Warning	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
63	REQUIRED I/O LIST MISSING The I/O list required for a free field input or unformatted output statement has not been specified.	Statement terminated	
64	FREE FIELD OUTPUT ILLEGAL An '*' in place of a format designation is illegal in a WRITE statement.	Statement terminated	
65	HOLLERITH constant with count greater than 8 used in other than FORMAT or subprogram reference.	ERROR	
66	PROGRAM UNIT HAS NO BODY or BLOCK DATA SUBPROGRAM HAS A BODY A main program, SUBROUTINE or FUNCTION requires an object program, or a BLOCK DATA subprogram has a function statement or executable statements.	Warning	
67	SOURCE FILE OPEN OR ACCESS PROBLEM OR EOF or END$ or $ occurs before END statement or open or read error occurs while attempting to access the source file.	Compilation terminated	Example: END statement contains syntax error or it is missing.
68	EXTERNAL NAME HAS MORE THAN FIVE CHARACTERS The name of a PROGRAM, SUBROUTINE or FUNCTION has more than five characters. The fifth character is deleted.	Warning	
69	OCTAL STRING IN STOP OR PAUSE STATEMENT IS TOO LONG In the statement STOP n or PAUSE n, n has more than four digits.	Warning	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
70	EQUIVALENCE GROUP SYNTAX An EQUIVALENCE group does not start with a left parenthesis. All further groups are ignored.	Statement terminated	
71	DUMMY VARIABLE IN DATA LIST Dummy parameters of a subprogram cannot be initialized in a DATA statement.	Statement terminated	
72	COMMON VARIABLE IN DATA LIST or in BLOCK DATA SUBPROGRAM VARIABLE IN DATA LIST NOT IN BLOCK COMMON. Entities of a COMMON block cannot be initialized with a DATA statement. Similarly, in block data subprograms, only entities in a named common block may be initialized.	Statement terminated	
73	MIXED MODE IN DATA STATEMENT A name and its corresponding constant in a DATA statement do not agree in type.	Statement terminated	
74	ILLEGAL USE OF STATEMENT FUNCTION NAME The name of a statement function also occurs in its dummy parameter list.	Warning	
75	RECURSION ILLEGAL The current program unit name has been used in a CALL statement.	Statement terminated	
76	DOUBLY DEFINED DUMMY VARIABLE The same dummy variable name occurs twice in a subprogram or statement function parameter list.	Warning	

ERROR CODE	EXPLANATION	EFFECT	ACTION
77	STATEMENT NUMBER IGNORED A statement number on a specification, DATA statement, continuation line, or on a statement function is ignored.	Warning	
78	PROGRAM UNIT HAS NO EXECUTABLE STATEMENTS A program unit has only specification or DATA statements or statement functions.	Warning	
79	FORMAT DOES NOT START WITH LEFT PARENTHESIS	Warning	
80	FORMAT DOES NOT END WITH RIGHT PARENTHESIS	Warning	
81	ILLEGAL EQUIVALENCE GROUP SEPARATOR EQUIVALENCE groups are not separated by a comma or a non-array name has subscripts in an EQUIVALENCE group. All further EQUIVALENCE groups are ignored.	Statement terminated	
82	ILLEGAL USE OF ARRAY NAME IN AN EQUIVALENCE GROUP An array name in an EQUIVALENCE group is not followed by '(', ',' or ')'. All further EQUIVALENCE groups are ignored.	Statement terminated	
83	SUBPROGRAM NAME RETYPED The type declared for a subprogram name within its body does not agree with the type established in the SUBROUTINE or FUNCTION statement.	Warning	

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
84	OBJECT CODE MEMORY OVERFLOW Object program size is greater than 32K.	Compiler terminated	
85	POSSIBLE RECURSION MAY RESULT The use of one of the library names, enumerated in Table G-2 as the name of a program, sub-program, or common block may produce recursion if the body of the subprogram so named required an implicit call to one of these names.	Warning	The user is advised to change the name of the subprogram or to make certain that no mixed mode exists in the program and that no library subprogram used requires a call to ERRØ.
86	DUMMY VARIABLE IN STATEMENT FUNCTION CANNOT BE SUBSCRIPTED A dummy variable in a statement function cannot represent an array or a subprogram name.	Warning	Example: ASF(A)=A(1,1)+A(2,2)
87	NOT CURRENTLY USED.		
88	END OR FORMAT STATEMENT IN LOGICAL IF An END or FORMAT statement is illegal as the "true" branch of a logical IF.	Statement terminated	Specify a branch that is not an END or FORMAT statement.
89	CONTINUE STATEMENT OR NO BRANCH IN LOGICAL IF Specifying no branch or a CONTINUE statement as a branch in a logical IF is logically equivalent to a NOP (No Operation). The statement is assembled as stated.	Warning	Specify a valid branch or delete statement.
90	FIRST RECORD OF SUBPROGRAM IS A CONTINUATION LINE The first statement is incomplete if it contains a continuation code.	Statement termination	Statements are missing or out of order in source program.

TABLE G-1 (Cont.) RTE FORTRAN IV COMPILER ERROR DIAGNOSTICS

ERROR CODE	EXPLANATION	EFFECT	ACTION
91	RESULT OF RENAME DUPLICATES EXISTING EXTERNAL NAME	ERROR	Use a name that does not duplicate an existing external name.
92	RESULT OF RENAME DUPLICATES REQUIRED INTRINSIC	ERROR	Use a name that does not duplicate the intrinsic name.
93	DATA STATEMENT attempts to initialize EMA variable	ERROR	Delete DATA STATEMENT or remove variable from EMA.
94	NAME IN EMA STATEMENT IS NOT FORMAL PARAMETER OR APPEARS TWICE IN THE STATEMENT	ERROR	
96	A BREAK WAS DETECTED Operator break causes the compiler to be terminated.	DISASTR	
97	OPEN OR WRITE ERROR ON BINARY FILE File does not exist or improper security code given or there is no room.	DISASTR	
98	READ ACCESS ERROR ON SCRATCH FILE Scratch file access failed.	DISASTR	
99	WRITE ACCESS ERROR ON SCRATCH FILE Scratch file access failed (OPEN, WRITE, REWIND).	DISASTR	

TABLE G-2. LIBRARY ROUTINE INTRINSIC LIST

The use of these names as program, subprogram, or common block names may result in a recursive operation if the program, subprogram, or common block contains an implicit call to a name that duplicates its own name (see Table G-1, Error number 85).

ABS	CSGRT	DMAX1	IAND	TANH
AINT	CSIN	DMIN1	IFIX	
ALOG	DABS	DMOD	INT	
ALOG10*	DATAN	DSIGN	IOR	
ALOGT	DATAN2*	DSIN	ISIGN	
ATAN	DATN2	DSQRT	ISSW	
CCOS	DBLE	DTAN	NOT	
CEXP	DCOS	DTANH	REAL	
CLOG	DDINT	ERRO	SIGN	
CLRIO	DEXP	EXEC	SIN	
CMPLX	DLOG	EXP	SNGL	
CONJG	DLOG10*	FLOAT	SQRT	
COS	DLOGT	IABS	TAN	

* The five-character equivalent for these names: ALOG0
DATA2
DLOG0

Arguments to these functions (except EXEC) are always passed by value even without extra parentheses.

APPENDIX H
OBJECT PROGRAM DIAGNOSTIC MESSAGES

During execution of programs referencing Relocatable Library Subroutines, error messages may be generated. Error messages are listed together with the subroutine involved.

Mathematical Subroutines

Error messages are printed in the form:

program name nn xx

program name — is the name of the user program where the error was encountered.

nn — is a number in the range 02 through 14 which identifies the subroutine involved in the error condition.

xx — is the error type, as follows:

OF = Integer or Floating Point Overflow
OR = Out of Range
UN = Floating Point Undefined

These error messages can occur when system intrinsics are called or during an exponentiation operation. Suppose X and Y are real values and I and J are integers. Then, the following relocatable subroutines are called for these computations:

X**Y	.RTOR	(real to real)
X**I	.RTOI	(real to integer)
I**J	.ITOI	(integer to integer)

The following is a summary of possible error messages:

Error Message	Issuing Subroutine	Where Used	Error Condition
02-UN	ALOG	ALOG	$X \leq 0$
		ALOGT	$X \leq 0$
		CLOG	$X = 0$
03-UN	SQRT	SQRT DSQRT	$X < 0$

Error Message	Issuing Subroutine	Where Used	Error Condition
04-UN	.RTOR	.RTOR	$X = 0, Y \le 0$ $X < 0, Y \ne 0$
05-OR	SIN	SIN CSNCS CEXP COS	$\frac{1}{2}\left\lvert \frac{X}{\pi} + \frac{1}{2} \right\rvert > 2^{14}$
06-UN	.RTOI	.RTOI	$X = 0, Y \le 0$
07-OF	EXP	EXP	$X * \log_2 e \ge 124$
		CEXP	$X_1 * \log_2 e \ge 124$
		.RTOR	$\lvert X * ALOG(X) \rvert \ge 124$
		CSNCS	$X_2 * \log_2 e \ge 124$
08-UN	.ITOI	.ITOI	$I = 0, J \le 0$
08-OF	.ITOI	.ITOI	$I^J \ge 2^{15}$ or $I^J < -2^{15}$
09-OR	TAN	TAN	$X > 2^{14}$
10-OF	DEXP	DEXP	$e^X > (1-2^{-39})\ 2^{127}$
		.DTOD .DTOR .RTOD	$X > (1-2^{-39})\ 2^{127}$
11-UN	DLOG	DLOG DLOGT	$X \le 0$ $X < 0$
12-UN	.DTOI	.DTOI	$X = 0, I \le 0$
13-UN	.DTOD	.DTOD .DTOR .RTOD	$X = 0, Y \le 0$ $X < 0, Y \ne 0$
14-UN	.CTOI	.CTOI	$X = 0, I \le 0$
15-UN	DATN2	DATN2	$X = Y = 0$

Utility Subroutines

Subroutine	Error
MAGTP	Returns on an illegal call.
.SWCH	Returns if element is out of range.

During execution of the object program error messages may be printed on the output unit by the input/output system supplied for FORTRAN programs. The error message is printed in the form:

FMT ERR *nn* program name

nn is the error code.

program name is the name of the user program.

The following is a summary of the FMT error codes:

Error Code	Explanation	Action
01	FORMAT ERROR: a) w or d field does not contain proper digits. b) No decimal point after w field. c) w - d <= 4 for E-specification.	Irrecoverable error; program must be recompiled.
02	a) FORMAT specifications are nested more than one level deep. b) A FORMAT statement contains more right parentheses than left parentheses.	Irrecoverable error; program must be recompiled.
03	a) Illegal character in FORMAT statement. b) Format repetition factor of zero. c) FORMAT statement defines more character positions than possible for device. d) List items remain and no conversion items are accessible in FORMAT statement.	Irrecoverable error; program must be recompiled.
04	Illegal character in fixed field input item or number not right-justified in field.	Verify data.
05	A number has an illegal form (e.g., two Es, two decimal points, two signs, etc.).	Verify data.

APPENDIX I

HP CHARACTER SET FOR COMPUTER SYSTEMS

Effect of Control key *

000-037B | 040-077B | 100-137B | 140-177B

b_7 b_6 b_5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
BITS				COLUMN →	0	1	2	3	4	5	6	7
b_4	b_3	b_2	b_1	ROW ↓								
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

32 CONTROL CODES

Upshifted Lower Case

64 CHARACTER SET

96 CHARACTER SET

128 CHARACTER SET

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
BINARY	1	0	0	1	0	1	1
OCTAL	1	1			3		

* Depressing the Control key while typing an upper case letter produces the corresponding control code on most terminals. For example, Control-H is a backspace.

9206- 1A

HEWLETT-PACKARD CHARACTER SET FOR COMPUTER SYSTEMS

This table shows HP's implementation of ANS X3.4-1968 (USASCII) and ANS X3.32-1973. Some devices may substitute alternate characters from those shown in this chart (for example, Line Drawing Set or Scandanavian font). Consult the manual for your device.

The left and right byte columns show the octal patterns in a 16 bit word when the character occupies bits 8 to 14 (left byte) or 0 to 6 (right byte) and the rest of the bits are zero. To find the pattern of two characters in the same word, add the two values. For example, AB produces the octal pattern 040502. (The parity bits are zero in this chart.)

The octal values 0 through 37 and 177 are control codes. The octal values 40 through 176 are character codes.

Decimal Value	Octal Values		Mnemonic	Graphic[1]	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	NU	Null
1	000400	000001	SOH	SH	Start of Heading
2	001000	000002	STX	SX	Start of Text
3	001400	000003	ETX	EX	End of Text
4	002000	000004	EOT	ET	End of Transmission
5	002400	000005	ENQ	EQ	Enquiry
6	003000	000006	ACK	AK	Acknowledge
7	003400	000007	BEL	⍾	Bell, Attention Signal
8	004000	000010	BS	BS	Backspace
9	004400	000011	HT	HT	Horizontal Tabulation
10	005000	000012	LF	LF	Line Feed
11	005400	000013	VT	VT	Vertical Tabulation
12	006000	000014	FF	FF	Form Feed
13	006400	000015	CR	CR	Carriage Return
14	007000	000016	SO	SO	Shift Out } Alternate Character Set
15	007400	000017	SI	SI	Shift In } Alternate Character Set
16	010000	000020	DLE	DL	Data Link Escape
17	010400	000021	DC1	D1	Device Control 1 (X-ON)
18	011000	000022	DC2	D2	Device Control 2 (TAPE)
19	011400	000023	DC3	D3	Device Control 3 (X-OFF)
20	012000	000024	DC4	D4	Device Control 4 ($\overline{\text{TAPE}}$)
21	012400	000025	NAK	NK	Negative Acknowledge
22	013000	000026	SYN	SY	Synchronous Idle
23	013400	000027	ETB	EB	End of Transmission Block
24	014000	000030	CAN	CN	Cancel
25	014400	000031	EM	EM	End of Medium
26	015000	000032	SUB	SB	Substitute
27	015400	000033	ESC	EC	Escape[2]
28	016000	000034	FS	FS	File Separator
29	016400	000035	GS	GS	Group Separator
30	017000	000036	RS	RS	Record Separator
31	017400	000037	US	US	Unit Separator
127	077400	000177	DEL	█	Delete, Rubout[3]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
32	020000	000040		Space, Blank
33	020400	000041	!	Exclamation Point
34	021000	000042	"	Quotation Mark
35	021400	000043	#	Number Sign, Pound Sign
36	022000	000044	$	Dollar Sign
37	022400	000045	%	Percent
38	023000	000046	&	Ampersand, And Sign
39	023400	000047	'	Apostrophe, Acute Accent
40	024000	000050	(	Left (opening) Parenthesis
41	024400	000051	)	Right (closing) Parenthesis
42	025000	000052	*	Asterisk, Star
43	025400	000053	+	Plus
44	026000	000054	,	Comma, Cedilla
45	026400	000055	-	Hyphen, Minus, Dash
46	027000	000056	.	Period, Decimal Point
47	027400	000057	/	Slash, Slant
48	030000	000060	0	} Digits, Numbers
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	
57	034400	000071	9	
58	035000	000072	:	Colon
59	035400	000073	;	Semicolon
60	036000	000074	<	Less Than
61	036400	000075	=	Equals
62	037000	000076	>	Greater Than
63	037400	000077	?	Question Mark

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	
66	041000	000102	B	
67	041400	000103	C	
68	042000	000104	D	
69	042400	000105	E	
70	043000	000106	F	
71	043400	000107	G	
72	044000	000110	H	
73	044400	000111	I	
74	045000	000112	J	
75	045400	000113	K	
76	046000	000114	L	
77	046400	000115	M	
78	047000	000116	N	Upper Case Alphabet.
79	047400	000117	O	Capital Letters
80	050000	000120	P	
81	050400	000121	Q	
82	051000	000122	R	
83	051400	000123	S	
84	052000	000124	T	
85	052400	000125	U	
86	053000	000126	V	
87	053400	000127	W	
88	054000	000130	X	
89	054400	000131	Y	
90	055000	000132	Z	
91	055400	000133	[	Left (opening) Bracket
92	056000	000134	\	Backslash. Reverse Slant
93	056400	000135	]	Right (closing) Bracket
94	057000	000136	^ ↑	Caret. Circumflex. Up Arrow[4]
95	057400	000137	_ ←	Underline. Back Arrow[4]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent[5]
97	060400	000141	a	
98	061000	000142	b	
99	061400	000143	c	
100	062000	000144	d	
101	062400	000145	e	
102	063000	000146	f	
103	063400	000147	g	
104	064000	000150	h	
105	064400	000151	i	
106	065000	000152	j	
107	065400	000153	k	
108	066000	000154	l	
109	066400	000155	m	
110	067000	000156	n	Lower Case Letters[5]
111	067400	000157	o	
112	070000	000160	p	
113	070400	000161	q	
114	071000	000162	r	
115	071400	000163	s	
116	072000	000164	t	
117	072400	000165	u	
118	073000	000166	v	
119	073400	000167	w	
120	074000	000170	x	
121	074400	000171	y	
122	075000	000172	z	
123	075400	000173	{	Left (opening) Brace[5]
124	076000	000174	¦	Vertical Line[5]
125	076400	000175	}	Right (closing) Brace[5]
126	077000	000176	~	Tilde. Overline[5]

Notes [1]This is the standard display representation. The software and hardware in your system determine if the control code is displayed, executed, or ignored. Some devices display all control codes as " ", "@", or space

[2]Escape is the first character of a special control sequence. For example, ESC followed by "J" clears the display on a 2640 terminal

[3]Delete may be displayed as "_", "@", or space

[4]Normally, the caret and underline are displayed. Some devices substitute the up arrow and back arrow

[5]Some devices upshift lower case letters and symbols (` through ~) to the corresponding upper case character (@ through ^). For example, the left brace would be converted to a left bracket

9206-1C

RTE SPECIAL CHARACTERS

Mnemonic	Octal Value	Use
SOH (Control A)	1	Backspace (TTY)
EM (Control Y)	31	Backspace (2600)
BS (Control H)	10	Backspace (TTY, 2615, 2640, 2644, 2645)
EOT (Control D)	4	End-of-file (TTY 2615, 2640, 2644, 2645)

9206-1D

APPENDIX J
RTE FORTRAN IV OPERATIONS

INTRODUCTION

This Appendix contains information pertinent to RTE FORTRAN IV operations in an RTE operating system. This information explains the on-line loading of the compiler; the capabilities and invocation procedures of the compiler; and possible error messages to the operator that may arise during compiler operations.

RTE FORTRAN IV is a problem-oriented programming language that is translated by a compiler into relocatable object code. Source programs are accepted from either an input device or an FMGR file. Error messages, list output, and relocatable object code are stored in FMGR files or output to devices. The object code produced by the compiler can be loaded by the RTE Relocating Loader and then executed using the RU command. When an RTE FORTRAN IV program has been completely debugged, the RTE Relocating Loader can make it a permanent part of the RTE system if desired.

The RTE FORTRAN IV compiler is a segmented program that executes in the background under control of RTE-IV. It consists of a main program and overlay segments, and normally resides in the protected area of the disc which has been reserved for such programs during the generation process.

LOADING THE RTE FORTRAN-IV COMPILER ON-LINE

The compiler can be loaded on-line using the RTE-IV Relocating Loader. The page size of the program should be increased to give the compiler room for its symbol table. The minimum recommended size is thirteen pages, with fourteen or more preferred. The following example presents a typical RTE-IV on-line load of the RTE FORTRAN-IV compiler. %CLIB need be searched only if it is not in the system library or if it contains modules that should be used instead of system library modules.

```
:RU,LOADR
  /LOADR:   SZ,14
  /LOADR:   RE,%FTN4          *main
  /LOADR:   RE,%FFTN4         *helper module
  /LOADR:   SE,%CLIB          *search compiler library
  /LOADR:   RE,%0FTN4         *first segment
  /LOADR:   SE,%CLIB
  /LOADR:   RE,%1FTN4         *second segment
  /LOADR:   SE,%CLIB
  /LOADR:   RE,%2FTN4         *third segment

       continue similarly for all segments (c--5)

  /LOADR:   SE,%CLIB          *last search of library
  /LOADR:   EN                *end LOADR operations
```

The following example presents a typical on-line load of the FORTRAN-IV compiler for an RTE-II or RTE-III system.

```
:LG,10
:MR,%FTN4
:MR,%FFTN4
:MR,%0FTN4
:MR,%1FTN4
:MR,%2FTN4
```

continue similarly for all segments

```
:RU,LOADR,99,6,,1
```

FORMAT OF AN RTE FORTRAN IV PROGRAM

Several statements pertinent to the RTE implementation of FORTRAN are described in the following pages. These statements define compiler options and give other information necessary for the compiler's operation.

Fortran Control Statement

PURPOSE: To describe the output to be produced by the RTE FORTRAN IV compiler.

FORMAT: FTN4,p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8,p_9

p_1-p_9 are optional parameters, in any order, chosen from the parameters in Table J-1.

TABLE J-1 RTE FORTRAN IV OPERATIONS

Parameter	Meaning
L	List output. A listing of the source language program is output to the list *namr* as the source program is read.
A	Assembly listing. A listing of the object program in assembly language is output to the list *namr*.
T	Symbol table listing. A listing of the symbol table for each main or subprogram is output to the list device. If a U follows the address of a variable, that variable is undefined. An A or M specification also produces a symbol table listing.
M	Mixed listing. A listing of both the source and object program is produced. Each source line is included with the object code it generated in the compilation process. This listing is produced during both the source code and the intermediate code in order for this parameter to be used. If both M and A are specified, M is used. Source code lines are passed to the inter-pass file as they are encountered. This means that in the mixed listing, object code will not necessarily immediately follow the source code that produced it (see the sample listing in Appendix F).
C	Cross reference symbol table listing. A cross reference listing of symbols and labels used in the source program is produced.

TABLE J-1 (cont.) RTE FORTRAN IV OPERATIONS

Parameter	Meaning
F	Perform page eject. Usually, terminal driver software will replace a page eject function code with two line space function codes to keep text displayed on a CRT terminal screen. If you are using another type of terminal (such as a teleprinter), you may specify F to perform a normal page eject. If the output *namr* is a line printer, this parameter is ignored and normal page ejects are done.
D	Compile debug lines. The character D specified in column position 1 of a source program line will cause such a line to be treated as a comment by the compiler. To cause compilation of these lines, specify D as a control statement parameter.
n	Error routine *n* supplied. *n* is a decimal digit (1-9) which specifies an error routine, *ERRn*. The error routine is called when an error occurs in the ALOG,SQRT,.RTOT,SIN, COS,.RTOI,EXP,.ITOI, or TAN. "The ERRn routine must be written in Assembler. The calling sequence for ERRn must be the same as ERR0, as listed in the DOS/RTE Relocatable Library Reference Manual (24998-90001)." If this option does not appear, the standard library error routine, ERR0 is used.
B	This option is ignored. See the FORTRAN invocation command sequence for information about producing binary output files.
X	Double precision is three words (default).
Y	Double precision is four words. The default setting may be changed to four words at generation time.
Q	Includes the approximate relocatable address of each statement on a listing. Each line of the listing becomes 6 characters longer. If the Q option is specified, the L option is implied.

RTE FORTRAN IV PROGRAM STATEMENT

The program statement is a source code statement defining the name and optionally the type, priority, and time values of the module in which it appears.

The program statement must be the first non-comment statement in a module without the extended memory area (EMA). In a module with EMA, the EMA directive must be the first non-comment statement, and the program statement must be the second non-comment statement.

In the absence of a PROGRAM statement, the program name defaults to FTN., and the type, priority, and time parameters default as specified below.

FORMAT:

```
PROGRAM name,(type,pri,res,mult,hr,min,sec,msec)
```

where:

name is the name of the program (and its entry point).

type is the program type (set to 3 for main program, or 7 for subroutines, if not given).

0 = system program
1 = memory-resident
2 = real-time disc-resident
3 = background disc-resident
4 = background disc-resident without Table Area II access
5 = segment
6 = illegal
7 = library, utility subroutines (appended to calling program)
8 = if program is a main, it is deleted from the system

-or-

8 = if program is a subroutine, then it is used to satisfy any external references during generation. However, it is not loaded in the relocatable library area of the disc.

NOTE: In some cases the primary type code (i.e. types 1 through 8) may be expanded by adding 8,16, or 24 to the number. These expanded types allow features such as access to real-time COMMON by background programs, and access to SSGA.

pri is the priority (1-32767, set to 99 if not given)

res is the resolution code

mult is the execution multiple

hr is hours

min is minutes

sec is seconds

msec is tens of milliseconds

COMMENTS: The parameters type through msec must appear in the order shown. Even though the parameters are optional, if any one parameter is given, those preceding it must appear also. For example:

```
PROGRAM name(,90)
```

COMMENTS: (cont.) is illegal and will be rejected by the system. The only method of legally defaulting the parameters is shown below:

```
PROGRAM name
PROGRAM name(3,90)
```

All parameters are set to 0 if not specified with the following two exceptions:

a. The priority parameter pri is set to 99, the lowest priority recognized by FORTRAN.

b. The program type parameter type is set to 3 for a main program, or 7 for a subroutine.

RTE FORTRAN IV can also pass a comment line to the loader, via binary record. The following format should be used:

PROGRAM name (p_1,... ,p_8),comment

or:

PROGRAM name,p_1,p_2,... ,p_8,comment

where:

name and p_1-p_8 are as defined above

comment = a comment line to be passed to the loader. All characters after the comma (,) including blanks are passed. The comment is limited to 84 characters in length.

In the first format shown above, one or more of the parameters may be omitted while still retaining the comment. In the second format, all parameters must be accounted for at least by the presence of a comma. Data after the program name is optional.

COMPILER INVOCATION

PURPOSE: To schedule the RTE FORTRAN IV compiler.

FORMAT:

$\left\{\begin{matrix} \text{*ON} \\ \text{*RU} \\ \text{:RU} \end{matrix}\right\}$,FTN4,*source input*[,*list output*[,*binary output* [,*line count*[,*options*]]]]

source input Name of an FMGR file or a logical unit number of the device containing the FORTRAN source code; this entry must conform to the format required by the FMGR *namr* parameter.

source input (cont.)

If an interactive device is specified, FTN4 will print a right bracket (]) on the device as a prompt. It will then accept input a line at a time and issue another prompt until an END statement is entered.

list output

Choose one of the following:

- (minus symbol)
FMGR file name
logical unit number
null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand, the ampersand is replaced with an apostrophe and the remaining source file name characters are used for the list file name. The list file is forced to reside on the same cartridge (cartridge reference code) as the source file. For example:

&FIL1	source file name
'FIL1	list file name

If an FMGR file name is specified, it must conform to the format required by the FMGR *namr* parameter. The list file is created if it does not exist. If the file does exist, the first character in the file name must be an apostrophe; otherwise, an error results.

If a logical unit number is specified, the listed output is directed to that logical device.

If this parameter is omitted, the user's terminal is assumed. Further, if subsequent parameters are specified, the comma must be used as a parameter placeholder.

binary output

Choose one of the following:

- (minus symbol)
FMGR file name
logical unit number
null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand, the ampersand is replaced with a percent symbol and the remaining source file name characters are used for the binary file name. This binary file is forced to reside on the same cartridge (cartridge reference code) as the source file. For example:

&FIL1	source file name
%FIL1	binary file name

binary output (cont.)	If an FMGR file name is specified, it must conform to the format required by the FMGR *namr* parameter. The binary file is created if it does not exist. If the file exists, it is necessary that: a. the first character of the file's name be a percent sign (%). b. the existing file be of the type specified in the *namr* parameter (if the file type is not declared in *namr*, the file's type must be Type 5, relocatable binary). If the above conditions are not met, a compiler error will result. If a logical unit number is specified, the binary output is directed to that logical device. If this parameter is omitted, no binary relocatable code is generated. Further, if the subsequent parameter is specified, the comma must be used as a parameter placeholder.
line count	A decimal number which defines the number of lines per page for the list device. Specification of this parameter is optional. If it is omitted, 56 lines per page are assumed. If a number less than 10 is specified, the compiler treats it as effectively infinite. The line count must be in the range 10 $\leq$ line count $\leq$ 999.
options	Up to seven characters that select control function options. No commas are allowed within the option string. These characters are: A, C, D, F, L, M, T and Q. If specified, these options replace (override) the character options declared in the FTN4 control statement (see Appendix B). These options do not override the FTN4 control statement numeric options. Characters other than the above are ignored, except that any option specified in this parameter position negates all character options declared in the FTN4 control statement. NOTE: The X and Y options are intentionally omitted, as they have no meaning in this option list.

EXAMPLES:

*RU,FTN4,&PROGA,-,-

Schedules RTE FTN4 to compile source file &PROGA. Listed output is directed to list file ´PROGA and binary relocatable code is directed to binary file %PROGA. The number of lines per list file page defaults to 56.

:RU,FTN4,&FIL1,´LIST

Schedules RTE FTN4 to compile source file &FIL1. Listed output is directed to list file ´LIST. No binary relocatable code is generated. The number of lines per list file page defaults to 56.

:RU,FTN4,&ABCD

Schedules RTE FTN4 to compile source file &ABCD. Listed output defaults to the user's terminal. No binary relocatable code is generated. The number of lines per list file page defaults to 56.

:RU,FTN4,&AAAA,-,-,28

Schedules RTE FTN4 to compile source file &AAAA. Listed output is directed to list file ´AAAA. Binary relocatable code is directed to binary file %AAAA. The number of lines per list file page is 28.

:RU,FTN4,&SFIL,-,-,,MTD

Schedules RTE FTN4 to compile source file &SFIL. Listed output is directed to list file ´SFIL. Binary relocatable code is directed to binary file %SFIL. The number of lines per list file page defaults to 56. A mixed listing and a symbol table will be produced, and debug lines will be compiled.

:RU,FTN4,&SFIL,-,-,,X

This command string results in the same action as the previous example, except that only errors will be listed and debug lines will not be compiled. The character X in the options parameter position is ignored, but it does negate any character options that may have been declared in program's FTN4 control statement.

RTE-M OPERATING SYSTEM

RTE FORTRAN IV invocation command syntax for RTE-M:

$$\left\{\begin{matrix}\text{*ON}\\ \text{*RU}\end{matrix}\right\}\text{,FTN4 [,}\mathit{fi,le,nm}\text{] [,}\mathit{line\ count}\text{]}$$

or,

$$\left\{\begin{matrix}\text{*ON}\\ \text{*RU}\end{matrix}\right\}\text{,FTN4 [,}\mathit{lu\ number}\text{] [,,,}\mathit{line\ count}\text{]}$$

fi,le,nm — The name of a file containing the input, output, and list file responses for the compiler. This file name is specified as parameters 1, 2, and 3 with two file name characters per parameter. If these parameters are omitted, the file responses are assumed to be from the session console.

line count — A decimal number which defines the number of lines per page for the list file. This entry is specified as parameter 4. In the alternate syntax shown above, the three leading commas are required as parameter position placeholders. If this parameter is omitted, 56 lines per page are assumed. The line count must be in the range $10 \le$ line count ≤ 999.

lu number — The logical unit number of a device from which the input, output, and list file responses to the compiler will be entered. This value is specified as parameter 1. If this parameter is omitted, the file responses are assumed to be from the session console.

When the RTE FORTRAN IV compiler is executed, it expects to obtain the input, output, and list file information from a named file, a logical device, or (by default) from the session console depending on the parameters passed in the invocation command. If these file responses are expected from the session console (or other keyboard/display device), the compiler will display separate requests in the form:

```
INPUT  ?
OUTPUT ?
LIST   ?
```

Enter a FMGR *namr* in response to each request. Parameters beyond the cartridge reference number are ignored.

EXAMPLES:

```
*RU,FTN4
```

Schedules RTE FTN4 to compile a program for which the input, output, and list file names will be entered from the session console. The list file will be formatted to 56 lines per page.

```
*ON,FTN4,7,,,28
```

Schedules RTE FTN4 to compile a program for which the input, output, and list file names will be entered from the device associated with logical unit number 7. List file output will be formatted with 28 lines per page. The commas appearing between the logical unit number and the list file line count are placeholders for null parameters.

EXAMPLES: (Cont.)

```
*RU,FTN4,RE,SP,NS
```

Schedules RTE FTN4 to compile a program for which the input, output, and list file names will be obtained from a file named RESPNS. The list file will be formatted with 56 lines per page.

MESSAGES TO OPERATOR

More than one source tape can be compiled into one FORTRAN program by leaving off the $END statement on all but the last source tape. When the end of each source tape is encountered (end of tape or EOT condition), RTE driver DVR00 can interpret it in two ways. An EOT can set the tape reader down (make it inactive), or not set it down. The action depends on how DVR00 subchannels were configured during generation. In any case, an EOT does not suspend the FORTRAN compiler. Therefore, it is recommended that when compiling multiple tapes, DVR00 be configured to set the tape reader down on EOT (see the LU command). For more information, refer to the DVR00 manual (29029-95001).

If an end of tape causes the tape reader to be set down, the RTE system will output a message to the operator:

```
I/O ET L lu E eqt S sub
```

The operator must place the next source tape into the tape reader and set the tape reader up with the UP command.

```
UP,eqt
```

where eqt is the number reported in the above message.

If an EQT does not cause the tape reader to be set down, the RTE system does not output ant message and the compiler is not suspended.

RTE FORTRAN IV MESSAGES

At the end of the compilation (when the compiler detects the $END statement), the following message is printed:

```
$END FTN4:  nn DISASTRS nn ERRORS nn WARNINGS
```

where "nn" will be the number of occurrences of each problem type or "NO" if there are no occurrences of a particular type.

All error messages are output to the list output file or device unless there is an error in the list output specification itself. There are two possibilities:

If the operator incorrectly specified the list destination. The following message will appear on the log list device:

```
/FTN4:  ACCESS FAILED ON LIST
```

If the operator incorrectly specified both the source input and list output parameters, the following message will appear on the log list device:

```
/FTN4:  ACCESS FAILED ON LIST AND SOURCE
```

EXAMPLE RTE FORTRAN IV PROGRAM

```
FTN4,L,T
      PROGRAM PROGA,3,90
      WRITE (1,100)
100   FORMAT(1X,"HELLO")
      END
      END$
```

If the above source code were stored into a FMGR file name &PROGA, it could be compiled with the following command (among others):

```
*RU,FTN4,&PROGA,6,%PROGA
```

This command would compile the source code in file &PROGA. Error messages, a program listing, and a symbol table listing would be output to logical unit 6 since L and T were specified in the control statement of the source program. The relocatable object code would be stored in the FMGR file %PROGA.

SPECIAL USAGE NOTE

In the event that a FORTRAN source file is compiled under a RTE Operating System (e.g. RTE-IVB) that supports the four-word (Y) compiler option for double precision data, transportation of the relocatable file to a HP 2100 with FFP for execution is not allowed unless the software versions of .DFER and .XFER are loaded.

INDEX

A

B

C

D

V

W

X

HEWLETT PACKARD

MANUAL PART NO. 92060-90023
Printed in U.S.A. July 1980

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014