HEWLETT hp PACKARD

FORTRAN IV

reference manual

FORTRAN IV

reference manual

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

PART NO. 5951-1321

Printed in U.S.A. 12/75

PREFACE

The Hewlett-Packard FORTRAN IV Reference Manual describes the language elements used to code source programs in the HP FORTRAN IV programming language. This manual may be used with any of the following compilers: HP 24170 FORTRAN IV, HP 24177 FORTRAN IV, or HP 18967 FORTRAN IV.

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 formats of data and expressions used in HP FORTRAN IV. Sections IV through IX describe the language elements used to code a source program, including the formats and uses of HP FORTRAN IV statements. The Appendices describe the format of data in memory, the form of HP FORTRAN IV jobs, departures from and extensions of ANSI FORTRAN IV specifications, features included in HP FORTRAN IV for compatibility with HP FORTRAN and HP FORTRAN IV Compiler error diagnostics.

> *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.

CONTENTS

INTRODUCTION

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

The FORTRAN IV Compiler can read source input from paper tape, punched cards, magnetic tape, or from a file (or files) in the User Area of the disc. The Compiler outputs the resultant object program on a standard punch device or to the Job Binary Area or Core Image Buffer Area of the disc in a format acceptable to the Relocating Loader.

HP FORTRAN IV is a two pass compiler. A pass is defined as a processing cycle of the source program. In the first pass, the source program is processed, a symbol table is constructed, and a set of intermediate machine code is generated. During the second pass, 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 outputs the object program to the standard punch device and/or to the Job Binary Area of the disc. Source and object listings may be produced, if specified in the FTN control statement.

The HP FORTRAN IV Compiler is available in these HP operating systems: Real-Time Executive (RTE-II/III), Test Oriented Disc System (TODS-C), and the DOS-III Disc Operating System. The hardware configurations required for compiling and executing HP FORTRAN IV programs under the control of these systems are the same as the minimum requirements for the systems, as described in these manuals:

Real-Time Executive-II Software System (HP 92001-93001)
Real-Time Executive-III Software System (HP 92060-90004)
HP 24307 DOS-III Disc Operating System (HP 24307-90006)
Test Oriented Disc System (HP 09500-90234)

The libraries of relocatable subroutines available with HP FORTRAN IV are described in the Relocatable Subroutines manual (HP 02116-91780).

NOTE: HP FORTRAN IV source programs cannot be compiled under the control of the Basic Control System (BCS). However, object programs produced by the HP FORTRAN IV Compiler can be loaded and executed under BCS control if the computer has 8,192 words of memory and no program segments are used.

SECTION I
THE FORM OF A FORTRAN IV PROGRAM

The HP 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 holds a sample source 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 or a SUBROUTINE statement, beginning with a PROGRAM statement and ending with an END statement.
Subprogram:	A set of statements and comments containing a FUNCTION or a SUBROUTINE 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. Subprograms can also be written in HP FORTRAN, HP ALGOL, or HP Assembly Language.
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.

*Segments cannot be included in programs to be run in BCS environment.

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.

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

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. 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: The letter C in column 1 of a line designates that line as a comment line. A comment line must be immediately followed by an initial line, another comment line, or an end line. A comment line does not affect the program in any way, and is available as a convenience for the user.

Program Line: The first statement of a main program may be the following:

PROGRAM name (P_1,P_2,...P_8)

name = An alphanumeric identifier of up to five characters.

(P_1,P_2,...P_8) are optional parameters. The acceptable list of parameters depends upon the operating system used.

Basic Control System. No parameters are allowed.

DOS-III Disc Operating System. Only P_1 is allowed. P_1 defines the program type according to the following.

3=main program
5=segment
6=library
7=library

The program type is set to 3 if not specified.

Real-Time Executive. Parameters P_1 through P_8 are allowed depending on whether the RTE system is disc-based (RTE-E) or core-based (RTE-C). P_1 defines the program type according to the following:

0=system program
1=real-time, core-resident program
2=real-time, disc-resident program (RTE-E only)
3=background, disc-resident program (RTE-E only)
4=background, core-resident program (RTE-E only)
5=background program segment (RTE-E only)
7=library utility program
8=subroutine used to satisfy generation requirement only

The program type is set to 3 if not specified. P_2 through P_8 are real-time parameters. For a full description of parameters refer to the applicable Real-Time Executive Software System manual (listed in the introduction).

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. Column 1 through 5 may contain a statement label or the character blank.

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. A continuation line may only follow an initial line or another continuation line. A maximum of 19 continuation lines can be used after one initial line.

End Line: An end line is a line with the character blank in columns 1 through 6, the characters E, N and D (preceded by, interspersed with, or followed by blank characters) in columns 7 through 72. The end line indicates to the compiler the end of the written description of a program unit. Every program unit must terminate with an end line.

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 (except that external names, i.e., main program, SUBROUTINE and FUNCTION names are limited to five characters), the first of which must be alphabetic.

ORDER OF STATEMENTS IN A SOURCE PROGRAM

When the source program is a main program:

PROGRAM LINE
SPECIFICATION STATEMENTS
DATA STATEMENTS
ARITHMETIC STATEMENT FUNCTIONS
EXECUTABLE STATEMENTS
END STATEMENT

When the source program is a subprogram:

FUNCTION or SUBROUTINE STATEMENT
SPECIFICATION STATEMENTS
DATA STATEMENTS (See Note 2.)
ARITHMETIC STATEMENT FUNCTIONS
EXECUTABLE STATEMENTS
END STATEMENT

NOTE: 1. *FORMAT Statements can appear anywhere in a source program, as long as they appear after the PROGRAM LINE (main program) or FUNCTION or SUBROUTINE statement (subprogram).*

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

SECTION II
DATA, CONSTANTS, VARIABLES AND ARRAYS

There are six types of data in HP FORTRAN IV:

INTEGER
REAL
DOUBLE PRECISION
COMPLEX
LOGICAL
HOLLERITH

Each data type has a specific format in core 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. If a TYPE- statement is used to establish a data type association (for integer, real, double precision, complex or logical data), it overrides the implied association which occurs in integer and real data types in variables and arrays. (See "Establishing Data Names," below.)

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 core 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- statement. (See Section IV, "Specification Statements.") In the absence of an explicit declaration in a TYPE- statement, the data type is implied by the first character of the data name, as follows:

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

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 HP FORTRAN IV. See Appendix A "Formats of Data in Core Memory," for a description of how each data type is stored in core memory.

INTEGER CONSTANT

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

FORMAT:

$$\begin{matrix} \pm n \\ n \end{matrix}$$

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, ir 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:

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

HOLLERITH CONSTANT

PURPOSE: A Hollerith constant is written as an integer constant followed by the letter H, followed by one or two characters from the FORTRAN character set.

FORMAT:

n H x

n = an integer constant (either 1 or 2)
H = the Hollerith descriptor, which is the character H
x = one or two alphanumeric characters

COMMENTS: If n = 1, 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 a blank character.

If n = 2, the first character after the H is put in the left half of the word, the next character in the right half.

An error diagnostic occurs if n = 0 or n >2.
Hollerith constants are typed as integer.

EXAMPLES:

1H@	2HBB
1HA	2H$$
2H A	2H12

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
```

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- statement. (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.

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 sc 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 (including system crashes) 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. Expressions are evaluated by the compiler.

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

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 Arithmetic Expressions

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

PRECEDENCE:

( )	parentheses, for grouping expressions, then
**	exponentiation, then
*,/	multiplication and division (whichever occurs first) then
-	unary minus, then
+,-	addition and subtraction (whichever occurs first).

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	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

NOTE: Integer overflow resulting from arithmetic operations is not detected at execution time. Care must be taken when the relational operators .LT., .LE., .GT., and .GE. are used with integer operands. The object codes generated by this compiler for relational operators on integers are 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 I
CMA,INA	CMA,INA	CPA J	CPA J	CMA,INA	CMA,INA
ADA I	ADA J	CCA,RSS	CLA,RSS	ADA J	ADA J
	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 HP FORTRAN IV. IT is recommended that specification statements be used in the following order:

TYPE-
DIMENSION
COMMON
EQUIVALENCE
EXTERNAL
DATA

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.

EXTERNAL

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

FORMAT:

EXTERNAL v_1, v_2, ..., 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.

EXAMPLES:

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

TYPE-

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
COMPLEX
LOGICAL

} $v_1, v_2, \ldots, v_n$

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

COMMENTS: Subroutine names cannot appear in a TYPE- statement.

If the same symbolic name appears in more than one TYPE- statement, the last use of the name states the data type.

A TYPE- 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- 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 bs 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 (including system crashes) 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 core memory between a main program and its subprograms, or for sharing core memory between subprograms.

FORMAT:

```
COMMON a
```

a = a list of variable names, array names or array declarators.

COMMENTS: A symbolic name that appears in a COMMON statement must be a variable name, an array name or an array declarator. Once these names are used in a COMMON statement, they cannot be used in another COMMON statement in the same program unit.

All entities in the COMMON statement are declared to be in unlabeled (blank) common.

The size of a common block is the sum of the storage required for the elements introduced through COMMON and EQUIVALENCE statement in a program unit. Entities are strung together in the order of appearance.

NOTE: Named common blocks are not permitted in HP FORTRAN IV.

EXAMPLES:

```
COMMON I, CAREA(2,3), J(3)
```

EQUIVALENCE

PURPOSE: Allows the sharing of core 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.

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.

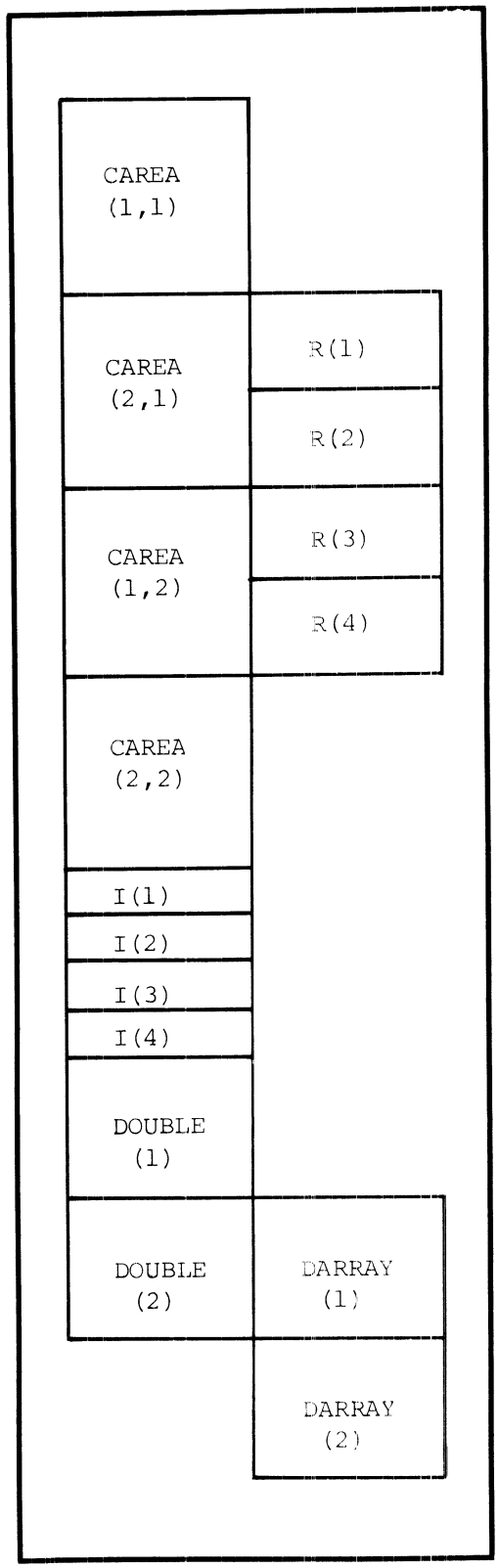
A variable not in COMMON, when equivalenced to a variable in COMMON, becomes a part of the COMMON area. A COMMON area, however, only can be lengthened by equivalence groupings. If an equivalence grouping causes an entity to be relocated before the first entity in COMMON, an error diagnostic occurs.

EXAMPLES:

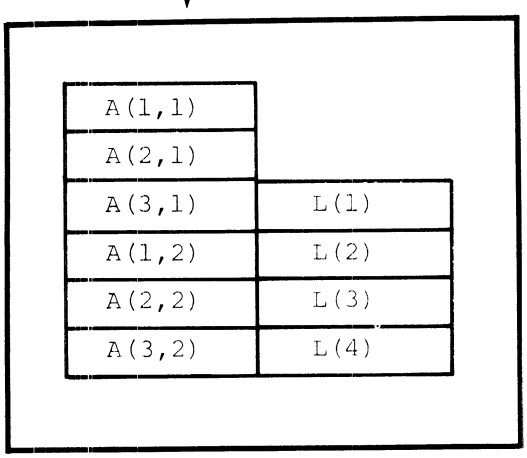
See the following page for examples of correct equivalence grouping.

```
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 29 words (26 words in original COMMON, 3 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 only to integer type variables or arrays.

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 common, nor can it be a dummy argument.

DATA statements must come after all other 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:

```
DATA A,CARRAY(2,3,1)/6*0, (1.0,-2.39E-1)/
DATA FALSE,ARRAY/.FALSE., 2HIA/,D/-2.39D-01/
```

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	
Complex	Double Precision	
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 assigned GO TO statement variable reference by storing in it the location of 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 statement label must refer to an executable statement in the same program unit in which the ASSIGN TO statement occurs.

Once mentioned in an ASSIGN TO statement, an integer variable may not be referenced in any statement other than an assigned GO TO 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 HP 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:

```
G 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, (k_1, k_2, ..., k_n)

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.

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 (k_1, k_2, ..., 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 tc be executed, depending upon the value of the logical expression.

FORMAT:

IF (e) s

IF (e) k_1, k_2

s = an executable statement (except a DC 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.

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 (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.)

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.

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)

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.

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 not be used in any GO TO or arithmetic IF statement that occurs anywhere but in the range of the most deeply nested DO that ends with that terminal statement. Integer overflow resulting from arithmetic operations is not detected at execution time.

EXAMPLES:

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

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

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

   20 CONTINUE
```

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.

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 core memory, and the positioning and demarcation of external files. The HP 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, DOS-M, or DOS-III manual 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 HP 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)

} May not reference any kind of function

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 or the label of a buffer holding an ASCII character string.
f = an array name or a FORMAT statement label
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.

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)
```

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
```

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 or the label of a buffer to receive the ASCII data string.

f = an array name or a FORMAT statement label

k = an output list

COMMENTS: The format statement or specification (in an array) can be anywhere in the program unit. The maximum record size is 67 elements.

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

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
```

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.

EXAMPLES:

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

WRITE

UNFORMATTED

PURPOSE: To write one unformatted record from core 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 core memory to unit u from the sequence of values represented by the list k. The maximum record size is 60 words.

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 HP 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 + -	flcating 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 core 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 core 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                    WRITE(2,200)A,L,X
      :                                   :
  100 FORMAT (2F5.1, F6.2)            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 rKw rOw
Real data:	srEw.d srFw.d srGw.d	Hollerith data:	rAw rRw
Double precision data:	srDw.d		$wHh_1h_2 \ldots h_w$
Logical data:	rLw		$r("h_1h_2 \ldots h_w")$
Blank data:	wX		
Complex data:	sEw.d,Ew.d		

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, ", 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 "$h_1h_2 \ldots h_w$" 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 deliniating 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: If $w \geq 2$, the rightmost two characters are taken from the external input field. If w = 1, the character appears left-justified in the word, with a trailing blank.

Output: If $w > 2$, the external output field consists of w - 2 blanks, followed by two characters from the internal representation. If w = 1, the character in the left half of the word is written.

EXAMPLES:

See the next page.

NOTE: Input/output of A-format elements must be to/from type integer variables or arrays.

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Value
XYZ	A2	XY
XYZ	A3	YZ
X	A1	X^

OUTPUT:

Internal Value	Format	External Field
XY	A2	XY
XY	A4	^^XY
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: The Rw descriptor is equivalent to the Aw descriptor, except that single characters are right-justified in the word with leading binary zeros (on input); and on output, if w = 1, the character in the right half of the word is written.

NOTE: The HP FORTRAN conversion Aw is replaced by the HP FORTRAN IV conversion Rw: a single character stored in a word under R format control is placed in the right half of the word with zeroes to the left half. On output, using the Rw format, the right half of the word is written.

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 HP FORTRAN IV A conversion, the user calls the NEWIO entry point:

CALL NEWIO

EXAMPLES: (Cont.)

INPUT:

External Field	Format	Internal Value
XYZ	R2	XY
XYZ	R3	YZ
X	R1	0X

OUTPUT:

Internal Value	Format	External Field
XY	R2	XY
XY	R4	^^XY
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

"..." EDITING

HOLLERITH INFORMATION

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

FORMAT:

$$r("h_1h_2 \ldots h_w")$$

h = any character in the FORTRAN character set, except "

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-TYPE CONVERSION

SKIP OR BLANKS

PURPOSE: Allows a specified number of characters to be skipped (input) or allows a specified number of blanks to be inserted (output).

FORMAT:

w X

w = a positive integer constant

COMMENTS:

Input: In the external input field, w characters are skipped.

Output: In the external output field, w blanks are inserted.

EXAMPLES:

14X

2X

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 ("0PRINT 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")	PRINT 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
FUNCTIONS AND SUBROUTINES

An executable FORTRAN IV program consists of one main program with or without subprograms. Subprograms, which are either functions or subroutines, 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 HP 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 externally 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 externally 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 HP 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 specified in a TYPE- statement. In the absence of an explicit declaration in a TYPE- statement, the type is implied by the first character of the name:

I, J, K, L, M or N = integer type data
any other letter = 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 specified in the FUNCTION statement that begins the subprogram. In the absence of an explicit declaration in the FUNCTION 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 in its FUNCTION statement must also have its name identically typed in each program unit which calls it.

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.

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 HP FORTRAN IV Compiler.

The symbolic name for the function cannot appear in a TYPE- statement which defines the name as a data type different from that specified for the function in Table 9-1 unless the user supplies his own version of the FORTRAN IV library function.

NOTE: HP FORTRAN IV makes no distinction between "intrinsic" and "external" functions.

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	$\|a\|$	1	ABS	Real	Real+
			IABS	Integer	Integer+
			DABS	Double	Double
Truncation	Sign of a times largest integer $\leq \|a\|$	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 $\|a_1\|$	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+
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real+
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	AMOD	Double	Double*
Modulus		1	CABS	Complex	Real
Logical Product	i.j	2	IAND	Integer	Integer+
Logical Sum	i+j	2	IOR	Integer	Integer+
Complement	$\overline{i}$	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 Relocatable Subroutines manual for a complete description of each entry point.

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)

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.

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. A function subprogram may not be called recursively.

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
```

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 re-defined. 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.

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 an 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, 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- 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
SUBROUTINE s  (a1, a2, ..., an)
```

s = the symbolic name of the subroutine

a = dummy argument

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.

EXAMPLES:

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

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. 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, 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. (A Hollerith constant must correspond to an integer type dummy argument.)

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.

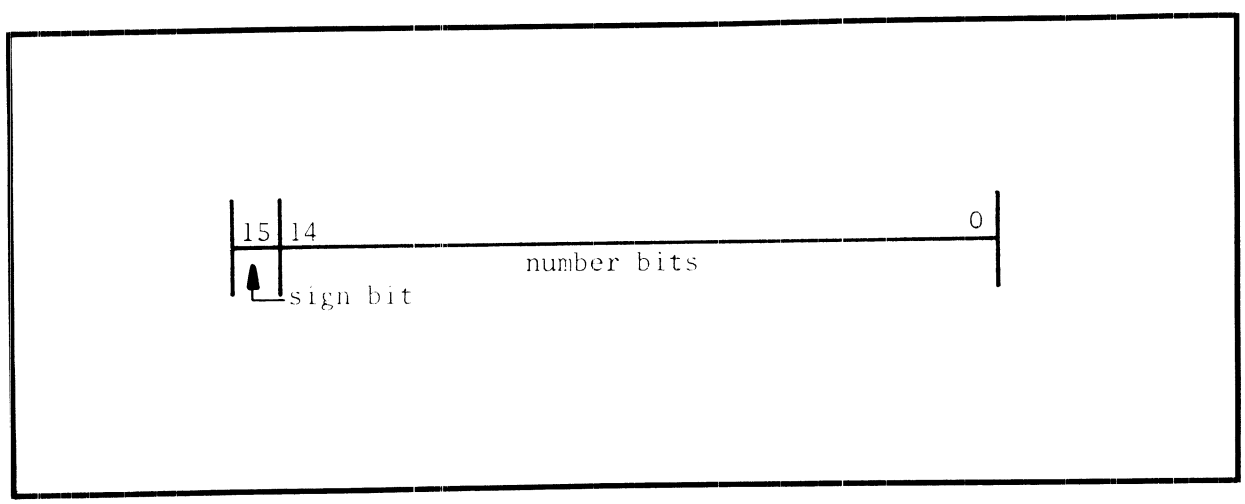
APPENDIX A
DATA FORMAT IN MEMORY

The six types of data used in HP 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 $2^{15}-1$.

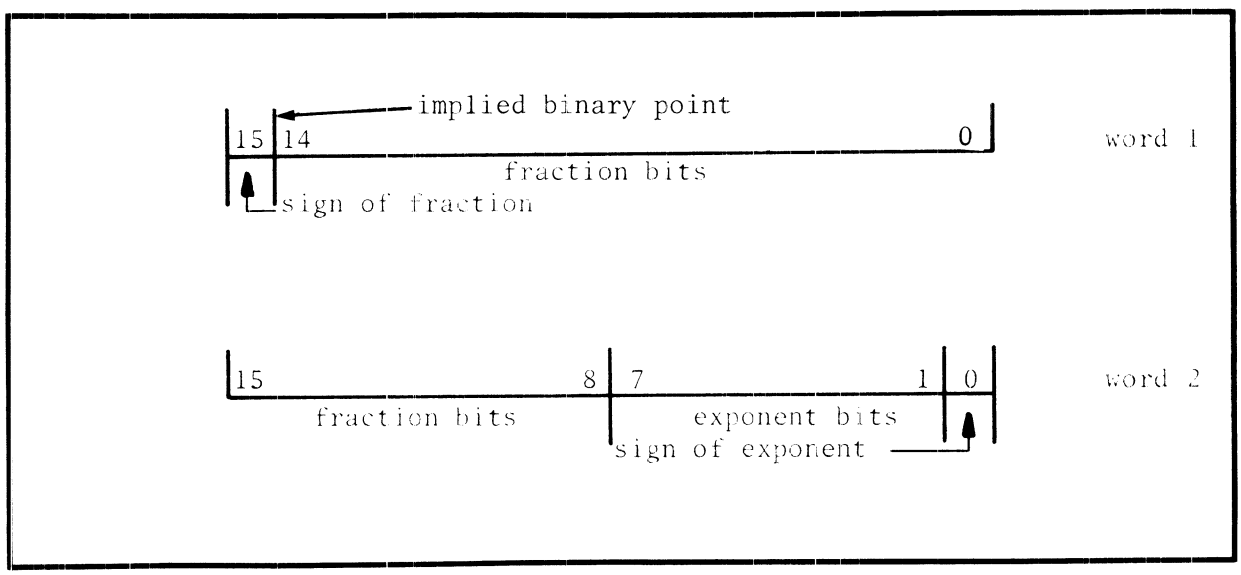
FORMAT:



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:



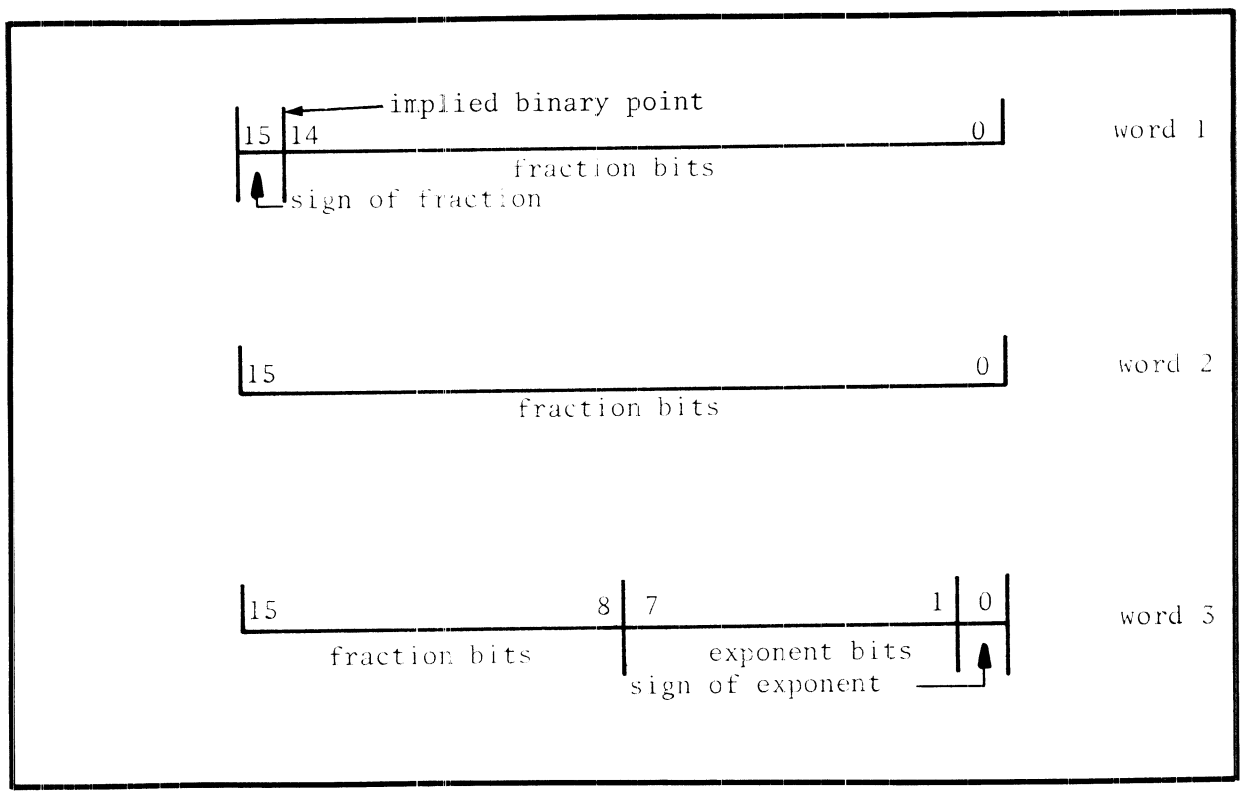
COMMENTS: A real number has a 23-bit fraction and a 7-bit exponent.

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

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:



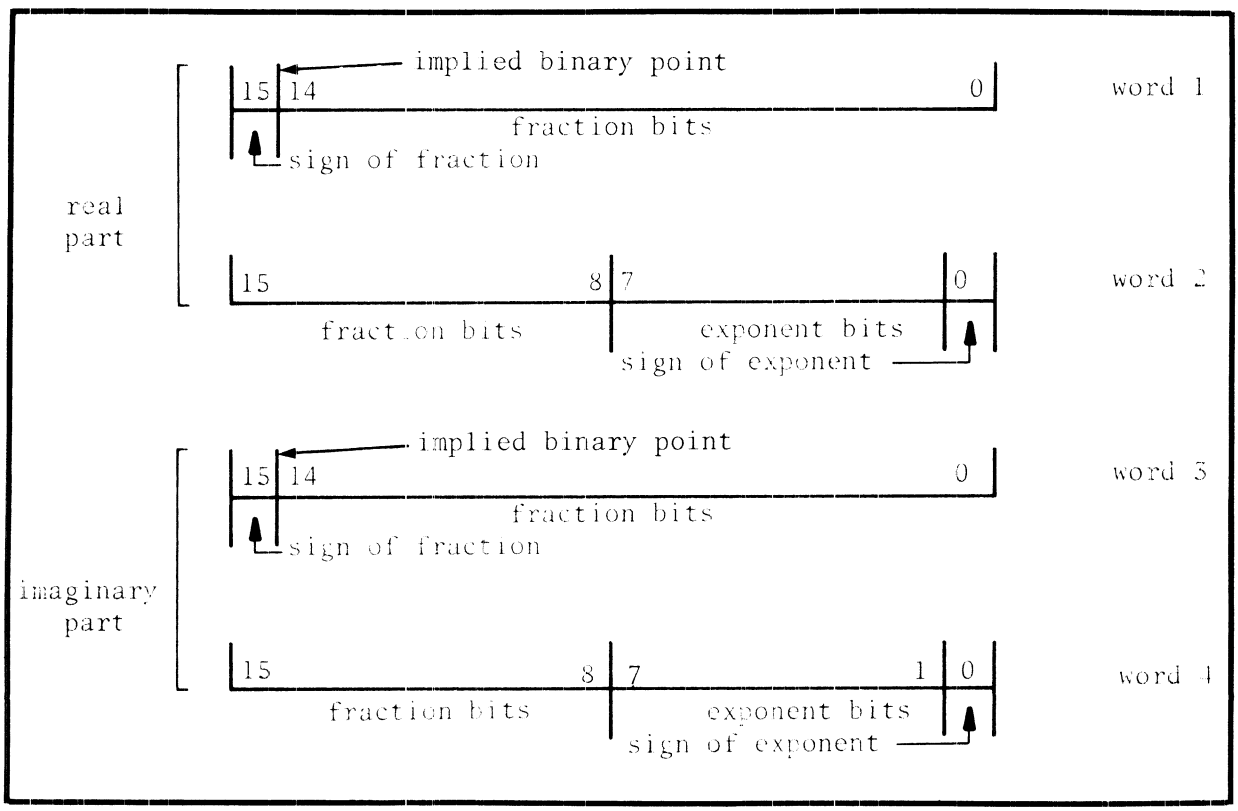
COMMENTS: A double precision number has a 39-bit fraction and a 7-bit exponent.

Significance (to the user) is to eleven or twelve 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:

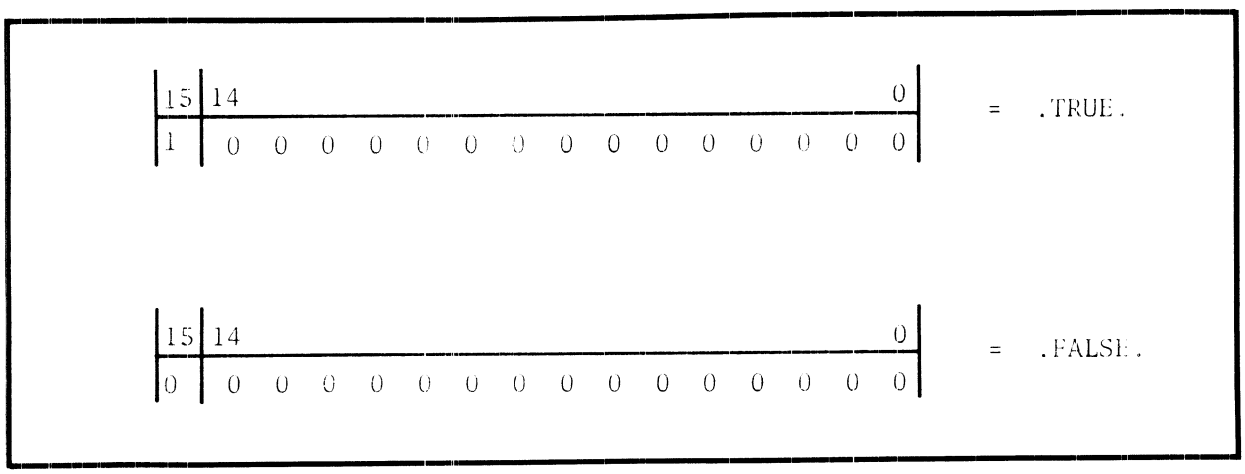


COMMENTS: Both the real part and the imaginary part have 23-bit fractions and 7-bit exponents; both have the same significance as a real number.

LOGICAL FORMAT

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

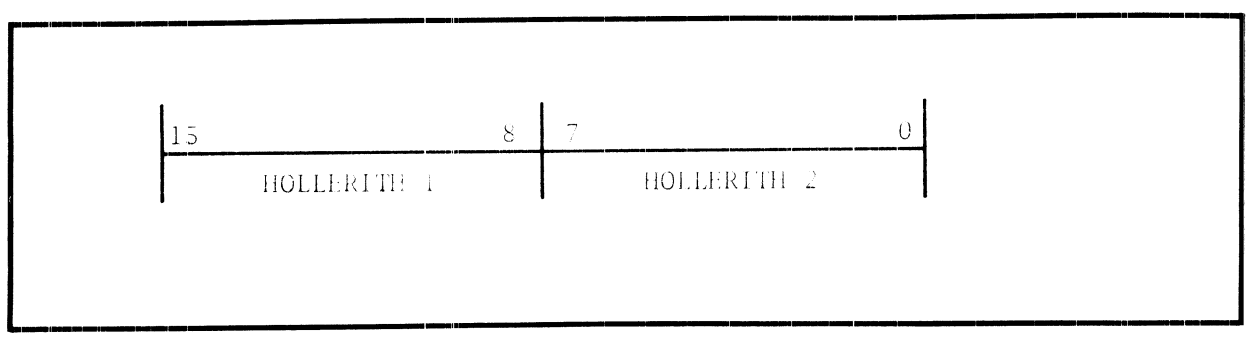
FORMAT:



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 A 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 punched cards or 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 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.

The FORTRAN control statement is described on the following page.

FORTRAN CONTROL STATEMENT

PURPOSE: To describe the type of output to be produced by the compiler.

FORMAT:

FTN[4], p_1, p_2, p_3, p_4, p_5

FTN[4] = Signals a FORTRAN Control Statement; the "4" is optional.

p_1 - p_5 = optional parameters, in any order, chosen from the following set:

B = Binary Output. An object program is to be punched in relocatable binary format suitable for loading by any of the operating system loaders.

L = List Output. A listing of the source language program is to be produced as the source program is read in.

A = Assembly Listing. A listing of the object program in assembly level language is to be produced in the second pass.

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 the second pass, and therefore it is necessary to store the source language program on the disc when it is read in during the first pass. (Sufficient disc space must be available for storing both the source and intermediate code in order for this parameter to be used.)

T = Table Listing. A listing of the symbol table for each main or subprogram is produced during the second pass.

FORMAT:
(cont.)

C = Cross Reference Symbol Table Listing. A cross reference listing of symbols and labels used in the source program is produced. (See Appendix E for a description.)

n = Error Routine *n* Supplied. *n* is a decimal digit (1-9) which specifies an error routine, ERR*n*. The error routine is called when an error occurs in ALOG, SQRT, .RTOR, SIN, COS, .TROI, EXP, ITOI, or TAN. If this option does not appear, the standard library error routine, ERR0, is used.

COMMENTS: Undefined source program statement numbers are printed when an END Statement is encountered.

If both M and A are specified, M is used. Both A and M will generate the symbol table listings automatically.

TODS-C users do not use 'C' to produce a cross-reference; the system asks if a cross-reference is desired at compilation time.

APPENDIX C
SUMMARY OF CHANGES TO ANSI FORTRAN IV

The HP 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.

Named COMMON blocks are not allowed.

BLOCK DATA subprograms are not allowed. (With the elimination of named COMMON blocks, BLOCK DATA subprograms have no function.)

Intrinsic functions are treated as external functions.

EXTENSIONS OF STANDARD

A subscript expression may be any arithmetic expression allowed in HP 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 may be any arithmetic expressions. 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 to the terminal parameter value. (Implied DO lists may use only integer arithmetic expressions which do not reference functions.)

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 $nHc_1c_2...c_n$ may be used in any arithmetic expression where an integer constant or an integer-valued expression is permitted. Note, however, that if $n > 2$, only the first two characters in the constant are used, that $n = 0$ is not permitted, and that if $n = 1$, the character C is stored in the left half of the computer word, with a blank character in the right half. Characters are stored in a single word in ASCII form.

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.

APPENDIX D
COMPATIBILITY OF HP FORTRAN AND FORTRAN IV

HP 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 HP 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 Al FORMAT specification may have to be changed to use the Rl 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 FORTRAN IV library functions.

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

Octal constants are valid in 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).

APPENDIX E

CROSS REFERENCE SYMBOL TABLE

The HP 24177 RTE/DOS 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. Section VII 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.

If more than 2000 unique symbols are used in the compiled program unit, the following message is printed instead of the cross reference symbol table:

```
PROGRAM TOO LARGE — CROSS REFERENCE ABORTED
```

APPENDIX F

SAMPLE LISTING OF FORTRAN IV PROGRAM

```
0001  FTN4,L,T,C
0002        PROGRAM CDUMP(2,90)
0003        DIMENSION IPARM(5), IBLOCK(64), IWORD(8), IASCII(8)
0004        EQUIVALENCE (IPARM(3),IBASE), (IPARM(4),IFWA), (IPARM(5),ILWA), (I
0005       1ASCII(1),NOF)
0006        DATA IN/1/, LIS/6/, NO,NOF/2*0/, J/1H  /
0007        IOCTAL(IA0)=IA0/10000*4096+MIN0(MOD(IA0,10000)/1000*512,3584)+MIN0
0008       1(MOD(IA0,1000)/100*64,448)+MIN0(MOD(IA0,100)/10*8,56)+MIN0(MOD(IA0
0009       2,10),7)
0010        CALL RMPAR (IPARM)
0011        IF (IPARM(1).GT.0) IN=IPARM(1)
0012        IF (IPARM(2).GT.0) LIS=IPARM(2)
0013        IBASE=IOCTAL(IBASE)
0014        IFWA=IOCTAL(IFWA)
0015        ILWA=IOCTAL(ILWA)
0016        IF (IBASE+IFWA+ILWA.GT.0)3,4
0017      4 WRITE (IN,5)
0018      5 FORMAT(/,"/CDUMP: @BASE,@FWA,@LWA: ←")
0019        READ (IN,*) IBASE,IFWA,ILWA
0020      3 IF (ILWA.EQ.0) ILWA=IFWA
0021        IF (IBASE.GT.-1.AND.IBASE.LT.077671B.AND.IFWA.LE.ILWA.AND.IBASE+IF
0022       1WA.GT.000001B.AND.IBASE+ILWA.LT.077700B)1,13
0023     13 DO 12 I=3,5,1
0024     12 IPARM(I)=0
0025        GO TO 4
0026      1 IFWA=IAND(IBASE+IFWA,077770B)
0027        ILWA=IBASE+ILWA
0028        IF (IBASE.GT.000000B) J=1HR
0029        N=IAND(IFWA-IBASE,000077B)
0030        WRITE (LIS,6) IBASE
0031      6 FORMAT(" BASE RELOCATION ADDRESS: ",O5,"B",/)
0032        DO 150 L=1,(ILWA-IFWA)/64+1,1
0033        IF (NO.GT.1) WRITE (LIS,400) NO
0034    400 FORMAT(3X,2(2X,5("*")),3X,I3," LINES SAME AS ABOVE  ",2(2X,5("*"))
0035       1," *")
0036        IF (NO.NE.1)30,25
0037     25 I=IAND(N-000010B,000077B)
0038        WRITE (LIS,200) I,IWORD,IASCII
0039    200 FORMAT(1X,O2,":",8(1X,O6),"*",8A2)
0040     30 NO=0
0041        DO 35 M=1,64,1
0042        IBLOCK(M)=IGET(IFWA)
0043     35 IFWA=IFWA+1
0044        I=IFWA-1
0045        IF (ILWA.LT.I) I=IAND(ILWA+000010B,077770B)-1
0046        M=J
0047        K=IFWA-IBASE-64
0048        IF (K.GT.-1)7,8
0049      8 M=1H
0050        K=K+IBASE
0051      7 I=I-IBASE
0052        WRITE (LIS,100) K,M,I,J
0053    100 FORMAT(2/,5X,"LOCATIONS: ",O5,A1," THRU ",O5,A1,/)
0054        DO 150 M=1,MIN0((ILWA-IFWA+64)/8*8+1,57),8
0055        IF (NOF.EQ.0)9,110
0056    110 DO 125 I=1,8,1
```

```
0057          IF (IWORD(I).NE.IBLOCK(I+M-1))130,125
0058      125 CONTINUE
0059          NO=NO+1
0060          GO TO 150
0061      130 IF (NO.GT.1) WRITE (LIS,400) NO
0062          IF (NO.NE.1)10,11
0063       11 I=IAND(N-0000108,000077B)
0064          WRITE (LIS,200) I,IWORD,IASCII
0065       10 NO=0
0066        9 DO 148 I=1,8,1
0067          IWORD(I)=IBLOCK(I+M-1)
0068          K=IWORD(I)
0069          IF (IAND(K,177400B).LT.020000B.OR.IAND(K,177400B).GT.057400B) K=IA
0070         1NO(K,000377B)+020000B
0071          IF (IAND(K,000377B).LT.000040B.OR.IAND(K,000377B).GT.000137B) K=IA
0072         1NO(K,177400B)+000040B
0073      148 IASCII(I)=K
0074          WRITE (LIS,200) N,IWORD,IASCII
0075      150 N=IAND(N+0000108,000077B)
0076          IF (NO.GT.1) WRITE (LIS,400) NO
0077          N=IAND(N-0000108,000077B)
0078          IF (NO.EQ.1) WRITE (LIS,200) N,IWORD,IASCII
0079          CALL EXEC (3,LIS+001100B,-1)
0080          WRITE (IN,2)
0081        2 FORMAT("/CDUMP: $END")
0082          END
```

** NO ERRORS** PROGRAM = 00902 COMMON = 00000

SYMBOL TABLE

NAME	ADDRESS	USAGE	TYPE	LOCATION
*1	000467R	STATEMENT NUMBER		
*10	001234R	STATEMENT NUMBER		
*100	001043R	STATEMENT NUMBER		
*11	001206R	STATEMENT NUMBER		
*110	001124R	STATEMENT NUMBER		
*12	000451R	STATEMENT NUMBER		
*125	001150R	STATEMENT NUMBER		
*13	000447R	STATEMENT NUMBER		
*130	001163R	STATEMENT NUMBER		
*140	001335R	STATEMENT NUMBER		
*150	001370R	STATEMENT NUMBER		
*2	001504R	STATEMENT NUMBER		
*200	000704R	STATEMENT NUMBER		
*25	000656R	STATEMENT NUMBER		
*3	000401R	STATEMENT NUMBER		
*30	000723R	STATEMENT NUMBER		
*35	000737R	STATEMENT NUMBER		
*4	000337R	STATEMENT NUMBER		
*400	000606R	STATEMENT NUMBER		
*5	000345R	STATEMENT NUMBER		
*6	000530R	STATEMENT NUMBER		
*7	001021R	STATEMENT NUMBER		
*8	001014R	STATEMENT NUMBER		
*9	001236R	STATEMENT NUMBER		
CLRIO	000001X	SUBPROGRAM	REAL	EXTERNAL
EXEC	000016X	SUBPROGRAM	REAL	EXTERNAL
I	001557R	VARIABLE	INTEGER	LOCAL
IAND	000013X	SUBPROGRAM	INTEGER	EXTERNAL
IASCII	000012R	ARRAY(*)	INTEGER	LOCAL
IBASE	000006R	VARIABLE	INTEGER	LOCAL
IBLOCK	000024R	ARRAY(*)	INTEGER	LOCAL
IFWA	000007R	VARIABLE	INTEGER	LOCAL
IGET	000015X	SUBPROGRAM	INTEGER	EXTERNAL
ILWA	000010R	VARIABLE	INTEGER	LOCAL
IN	000135R	VARIABLE	INTEGER	LOCAL
IOCTAL	000142R	STATEMENT FUNCTION	INTEGER	LOCAL
IPARM	000004R	ARRAY(*)	INTEGER	LOCAL
IWORD	000125R	ARRAY(*)	INTEGER	LOCAL
J	000140R	VARIABLE	INTEGER	LOCAL
K	001572R	VARIABLE	INTEGER	LOCAL
L	001556R	VARIABLE	INTEGER	LOCAL
LIS	000136R	VARIABLE	INTEGER	LOCAL
M	001570R	VARIABLE	INTEGER	LOCAL
MIN0	000006X	SUBPROGRAM	INTEGER	EXTERNAL
MOD	000005X	SUBPROGRAM	INTEGER	EXTERNAL
N	001564R	VARIABLE	INTEGER	LOCAL
NO	000137R	VARIABLE	INTEGER	LOCAL
NOF	000012R	VARIABLE	INTEGER	LOCAL
RMPAR	000007X	SUBPROGRAM	REAL	EXTERNAL

CROSS-REFERENCE LIST

SYMBOL	REFERENCES				
@1	0022	0026			
@10	0062	0065			
@100	0052	0053			
@11	0052	0063			
@110	0055	0056			
@12	0023	0024			
@125	0056	0057	0058		
@13	0022	0023			
@130	0057	0061			
@148	0056	0073			
@150	0032	0054	0060	0075	
@2	0080	0081			
@200	0038	0039	0064	0074	0078
@25	0036	0037			
@3	0016	0020			
@30	0036	0040			
@35	0041	0043			
@4	0016	0017	0025		
@400	0033	0034	0061	0076	
@5	0017	0018			
@6	0030	0031			
@7	0048	0051			
@8	0048	0049			
@9	0055	0066			
EXEC	0079				

CROSS-REFERENCE LIST

SYMBOL	REFERENCES							
I	0023	0024	0037	0038	0044	0045	0045	0051
	0051	0052	0056	0057	0057	0063	0064	0066
	0067	0067	0068	0073				
IA0	0007	0007	0007	0008	0008	0008		
IAND	0026	0029	0037	0045	0063	0069	0069	0069
	0071	0071	0071	0075	0077			
IASCII	0003	0004	0038	0064	0073	0074	0078	
IBASE	0004	0013	0013	0016	0019	0021	0021	0021
	0022	0026	0027	0028	0029	0030	0047	0050
	0051							
IBLOCK	0003	0042	0057	0067				
IFWA	0004	0014	0014	0016	0019	0020	0021	0021
	0026	0026	0029	0032	0042	0043	0043	0044
	0047	0054						
IGET	0042							
ILWA	0004	0015	0015	0016	0019	0020	0020	0021
	0022	0027	0027	0032	0045	0045	0054	
IN	0006	0011	0017	0019	0080			
IOCTAL	0007	0013	0014	0015				
IPARM	0003	0004	0004	0004	0010	0011	0011	0012
	0012	0024						
IWORD	0003	0038	0057	0064	0067	0068	0074	0078
J	0006	0028	0046	0052				
K	0047	0048	0050	0050	0052	0068	0069	0069
	0069	0070	0071	0071	0071	0072	0073	
L	0032							
LIS	0006	0012	0030	0033	0038	0052	0061	0064
	0074	0076	0078	0079				
M	0041	0042	0046	0049	0052	0054	0057	0067
MIN0	0007	0007	0008	0008	0054			
MOD	0007	0008	0008	0008				

CROSS-REFERENCE LIST

SYMBOL	REFERENCES							
N	0029	0037	0063	0074	0075	0075	0077	0077
	0078							
NO	0006	0033	0033	0036	0040	0059	0059	0061
	0061	0062	0065	0076	0076	0078		
NOF	0005	0006	0055					
RMPAR	0010							

APPENDIX G

FORTRAN IV COMPILER ERROR DIAGNOSTICS

TYPES OF COMPILER DIAGNOSTICS

There are four types of FORTRAN IV compiler diagnostics:

COMMENT: The compiler continues to process the source statement containing the error. Executable object code is produced, even though the program's logic may be faulty.

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

STATEMENT TERMINATED: 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.

COMPILATION TERMINATED: The compiler ignores 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.

FORMAT OF COMPILER DIAGNOSTICS

When an error is detected, the erroneous source statement is printed, followed by a message in this format:

```
** pname ** ERROR nn DETECTED AT COLUMN cc
```

pname = the program name
nn = the diagnostic error number
cc = column number of source line being scanned when error detected.

> *NOTE: If cc = 01, the error is in the source line preceding the last one 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:

```
@ nnn UNDEFINED
```

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

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

```
** nn ERRORS ** PROGRAM = xxxxx COMMON = yyyyy
```

where *nn* is the number of errors detected (*nn* = NO, if no errors 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 common block.

TABLE G-1

HP 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 core 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 NOT ALLOWED Only unlabeled (blank) COMMON is allowed in HP FORTRAN IV.	Statement terminated	Convert labeled COMMON blocks to blank COMMON.
05	NO DISC SOURCE FILE ASSIGNED The logical unit for input of the FORTRAN IV source program is 2, but the address of source file on disc has not been assigned.	Compilation terminated	Precede compilation by a :JFILE (DOS) or LS (RTE) directive to operating system
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.) HP 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.) HP 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., |exponent| > 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 as a subprogram or statement function name, as a parameter of a subprogram or statement function, or as an element of an EQUIVALENCE group.	Warning	Examples: SUBROUTINE 1234 FUNCTION NAME(X,12,A) EQUIVALENCE (I,5)

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

ERROR CODE	EXPLANATION	EFFECT	ACTION
32	INVALID STATEMENT NUMBER A statement Number has more than five digits or it contains a non-digit character.	Statement terminated	
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 terminated	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.	Statement terminated	

TABLE G-1 (Cont.) HP 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 case 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 NOT FIRST STATEMENT A PROGRAM statement, if present, must come first. A FUNCTION or SUBROUTINE statement is required for subprograms.	Statement terminated	

TABLE G-1 (Cont.) HP 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 AN ARRAY NAME, STATEMENT NUMBER OR * The FORMAT reference in an I/O statement is invalid.	Statement terminated	
48	DO MISSPELLED Keyword DO misspelled.	Comment	Example: DØ
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	

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

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	

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

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.) HP 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 COUNT GREATER THAN 2 In an 'nH' specification, n > 2.	Comment	Only the first two characters after the H are used.
66	PROGRAM UNIT HAS NO BODY A main program, SUBROUTINE or FUNCTION requires no object program.	Warning	
67	END$ OR $ OCCURS BEFORE END STATEMENT The end of the FORTRAN job was encountered before the END statement terminating the current program unit.	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 first five characters are used.	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.) HP 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 Entities of a COMMON block cannot be initialized with a DATA statement.	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	

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

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.) HP 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 REAL, SNGL, DBLE, CMPLX, FLOAT, CLRIO, IFIX, ERRO or EXEC as the name of a PROGRAM, may produce recursion if the body of the subprogram so named requires an implicit call to one of these names.	Comment	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	TOO MANY CONTINUATION LINES More than 19 continuation lines for a statement.	Compilation terminated	
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.

APPENDIX H
OBJECT PROGRAM DIAGNOSTIC MESSAGES

During execution of the object program, diagnostic messages may be printed on the output unit by the input/output system supplied for FORTRAN programs. When a halt occurs, the A-register contains a code which further defines the nature of the error:

Message	A-register	Explanation	Action
*FMT	000001	FORMAT error: a) w or d field does not contain proper digits. b) No decimal point after w field. c) w - d $\leq$4 for E specification.	Irrecoverable error; program must be recompiled.
*FMT	000002	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.
*FMT	000003	a) Illegal character in FORMAT statement. b) Format repetition factor of zero. c) FORMAT statement defines more character positions than possible for device.	Irrecoverable error; program must be recompiled.
*FMT	000004	Illegal character in fixed field input item or number not right-justified in field.	Verify data.
*FMT	000005	A number has an illegal form (e.g., two Es, two decimal points, two signs, etc.).	Verify data.

INDEX

F

G

H

I

J

K

L

M

READER COMMENT SHEET

FORTRAN IV
Reference Manual
5951-1321 Dec 1975

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate?

Is this manual complete?

Is this manual easy to read and use?

Other comments?

FROM:

Name _____

Company _____

Address _____

_____

_____

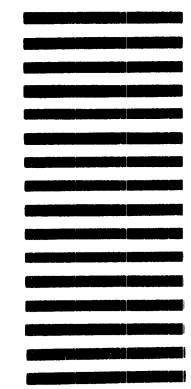
FOLD FOLD

FIRST CLASS
PERMIT NO. 141
CUPERTINO
CALIFORNIA

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States Postage will be paid by

Manager, Technical Publications
Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014



FOLD FOLD

PART NO. 5951-1321
Printed in U.S.A. 12/75

HEWLETT hp PACKARD

Sales and service from 172 offices in 65 countries.
11000 Wolfe Road, Cupertino, California 95014