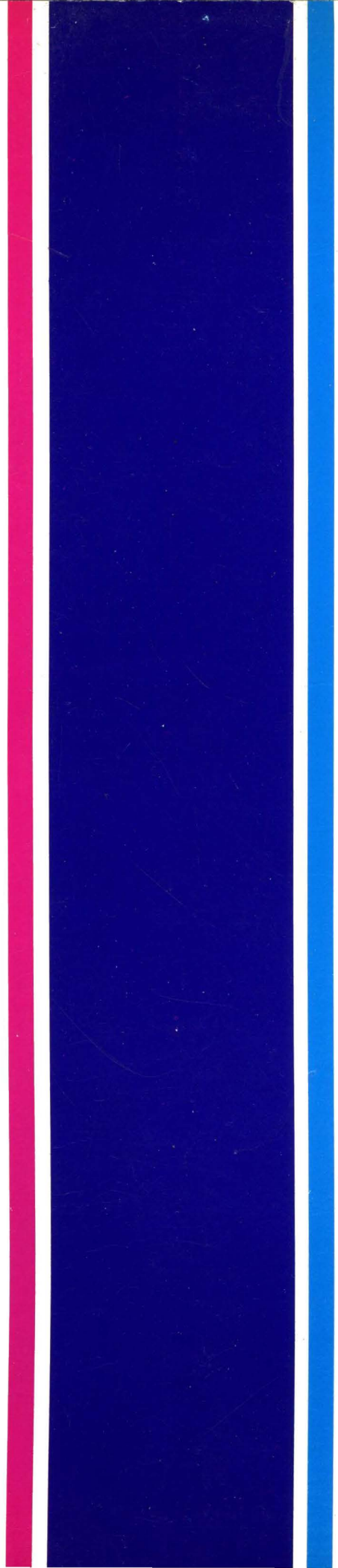


HEWLETT hp PACKARD

H P FORTRAN

HP FORTRAN Programmer's Reference Manual

HEWLETT hp PACKARD

11000 Wolfe Road
Cupertino, Calif. 95014

HP 02116-9015

April, 1970

First Edition, Feb. 1968
Revised, April 1970

Printed in the U.S.A.

Second Edition

Printed in the U.S.A.

PREFACE

This publication is a reference manual for the programmer using the HP FORTRAN Compiler. It includes both the elements of the language and the information required to operate the Compiler on the computer.

The programmer should also refer to:

Basic Control System's Programmers Manual (02116-9017)

Program Library Subroutine Manual (02116-9032)

Magnetic Tape System (02116-91752)

Prepare Tape System (02116-91751)

NEW AND CHANGED INFORMATION

For this printing, all known errors in the HP FORTRAN book have been corrected. Changes are shown in the text by a horizontal line in the margin. In addition, the information on instrument formats in the Appendices has been eliminated.

TABLE OF CONTENTS

INTRODUCTION

The FORTRAN compiler system accepts as input, a source program written according to American Standard Basic FORTRAN specifications; it produces as output, a relocatable binary object program which can be loaded and executed under control of the HP Basic Control System.

In addition to the ASA Basic FORTRAN language, HP FORTRAN provides a number of features which expand the flexibility of the system. Included are:

Free Field Input: Special characters included with ASCII input data direct its formatting; a FORMAT statement need not be specified in the source program.

Specification of heading and editing information in the FORMAT statement through use of the "..." notation; permits alphanumeric data to be read or written without giving the character count.

Array declaration within a COMMON statement.

Redefinition of its arguments and common areas by a function subprogram.

Interpretation of an END statement as a RETURN statement.

Basic External Functions which perform masking (Boolean) operations.

Two-branch IF statement.

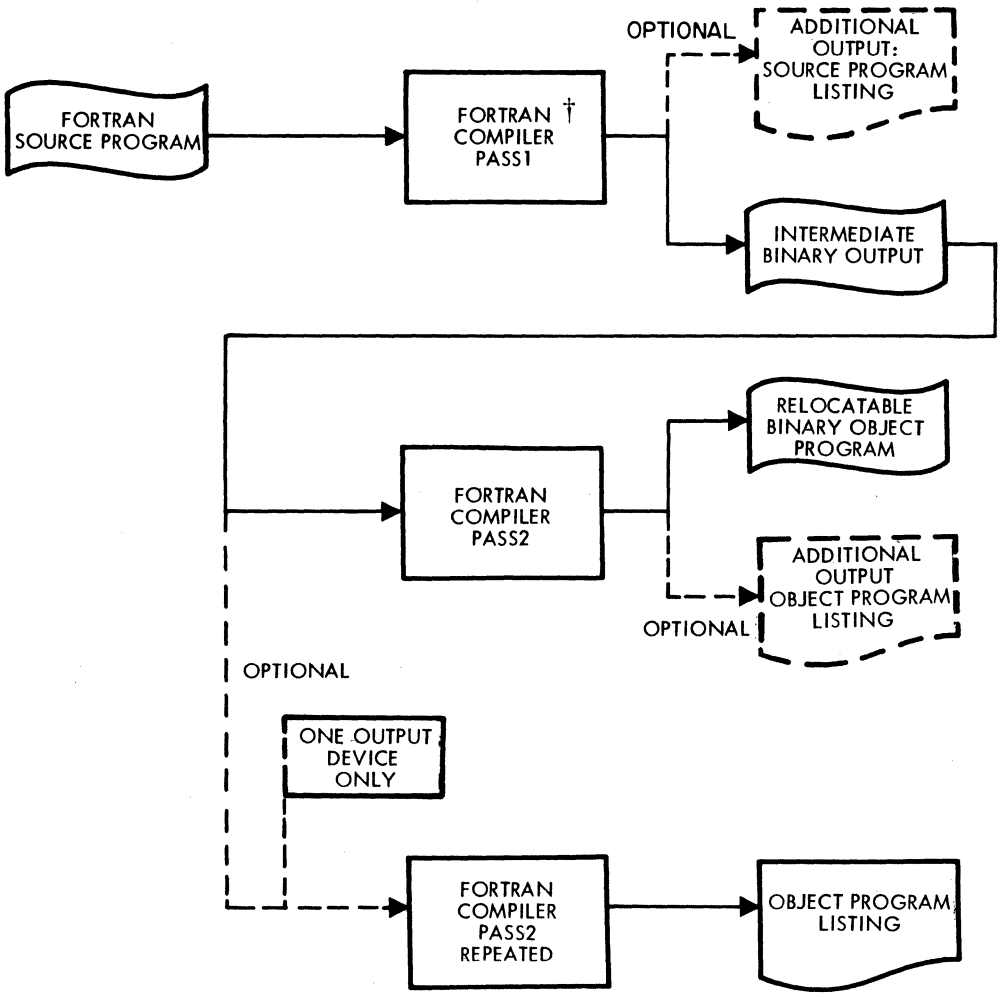
Octal constants.

The paper tape version of the compiler operates in two or four passes depending on the size of memory. For an 8K system, the compiler produces a source listing and an intermediate binary tape in the first pass. This intermediate tape serves as input to the second pass. The second pass produces the relocatable object program tape and a listing of the program in assembly level language. If only one output device is available the last pass is repeated to produce the listing. For the 4K system, two additional passes are introduced before the pass producing the relocatable program tape. For these passes the intermediate binary output of the previous pass becomes the input for the current pass.

When magnetic tape is available, the compiler uses the third file for storage of intermediate binary code. Pass 1 of FORTRAN writes the intermediate program. At the end of Pass 1, FORTRAN calls the Inter-Pass Loader; it searches for and loads Pass 2 of FORTRAN. (When not in MTS, Pass 2 must be hand-loaded using BBL.) Pass 2 spaces forward to the third file, processes the intermediate code and produces output on the punch and list devices as requested.

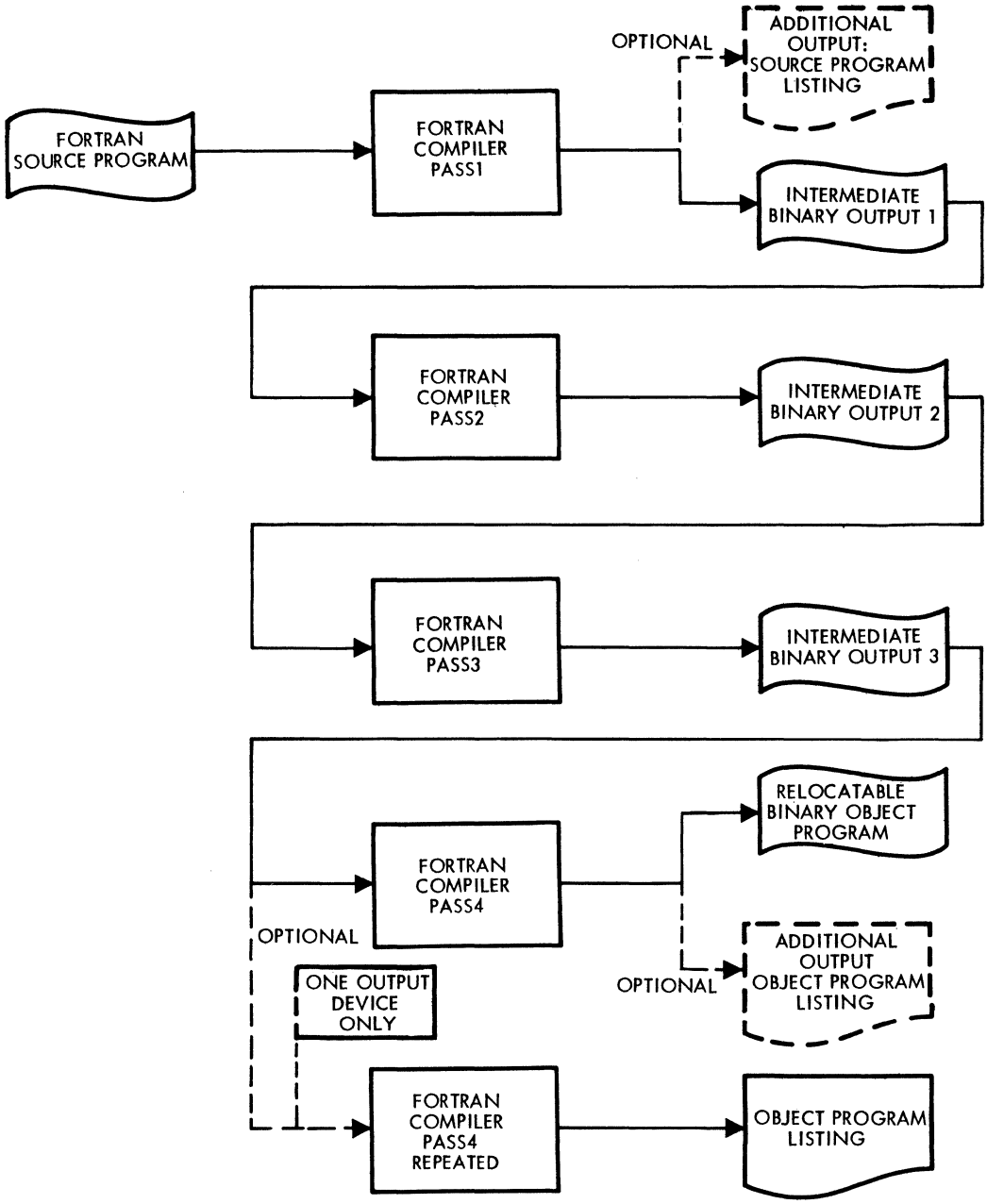
The minimum equipment configuration required to compile a program on the Computer is as follows:

2116A, 2115A, or 2114 Computer with 4K memory
2752A Teleprinter



†When compiling with the magnetic tape system, operator intervention ceases after Pass 1 has been loaded.

8K MEMORY
FORTRAN COMPILATION PROCESS



4K MEMORY
FORTRAN COMPILATION PROCESS

PROGRAM FORM 1

A FORTRAN program is constructed of characters grouped into lines and statements.

1.1 CHARACTER SET

The program is written using the following characters:

Alphabetic:	A through Z
Numeric:	0 through 9
Special:	
	Space
=	Equals
+	Plus
-	Minus
*	Asterisk
/	Slash
(	Left Parenthesis
)	Right Parenthesis
,	Comma
.	Decimal Point
$	Dollar Sign
"	Quotation mark

Spaces may be used anywhere in the program to improve appearance; they are significant only within heading data of FORMAT statements and, in lieu of other information, in the first six positions of a line.

In addition to the above set which is used to construct source language statements, certain characters have special significance when appearing with ASCII input data. They are the following:

space,	Data item delimiters
/	Record terminator
+ -	Sign of item
.E + -	Floating point number
@	Octal integer
"..."	Comments
←	Suppress CR-LF (output)

Details on the input data character set are given in Chapter 7.

1.2 LINES

A line is a string of up to 72 characters. On paper tape, each line is terminated by a return, (CR) , followed by a line feed, (LF) . This terminator may be in any position following the statement information or comment contained in the line.

Statements

A statement may be written in an initial line and up to five continuation lines. The statement may occupy positions 7 through 72 of these lines. The initial line contains a zero or blank in position 6. A continuation line contains any character other than zero or space in position 6 and may not contain a C in position 1.

Statement Labels

A statement may be labeled so that it may be referred to in other statements. A label consists of one to four numeric digits placed in any of the first five positions of a line. The number is unsigned and in the range of 1 through 9999. Imbedded spaces and leading zeros are ignored. If no label is used, the first five positions of the statement line must be blank. The statement label or blank follows the (CR) (LF) terminator of the previous line.

Comments

Lines containing comments may be included with the statement lines; the comments are printed along with the source program listing. A comment line requires a C in position 1 and may occupy positions 2 through 72. If more than one line is used, each line requires a C indicator. Each comment line is terminated with a (CR) and (LF) .

Control Statement

The first statement of a program is the control statement; it defines the output to be produced by the FORTRAN compiler. The following options are available:

Relocatable binary – The program is to be loaded by the loader of the Basic Control System.

Source Listing output – A listing of the source program is produced during the first pass of compiler operation.

Object Listing output – A list of the object program is produced during the last pass of compiler operation.

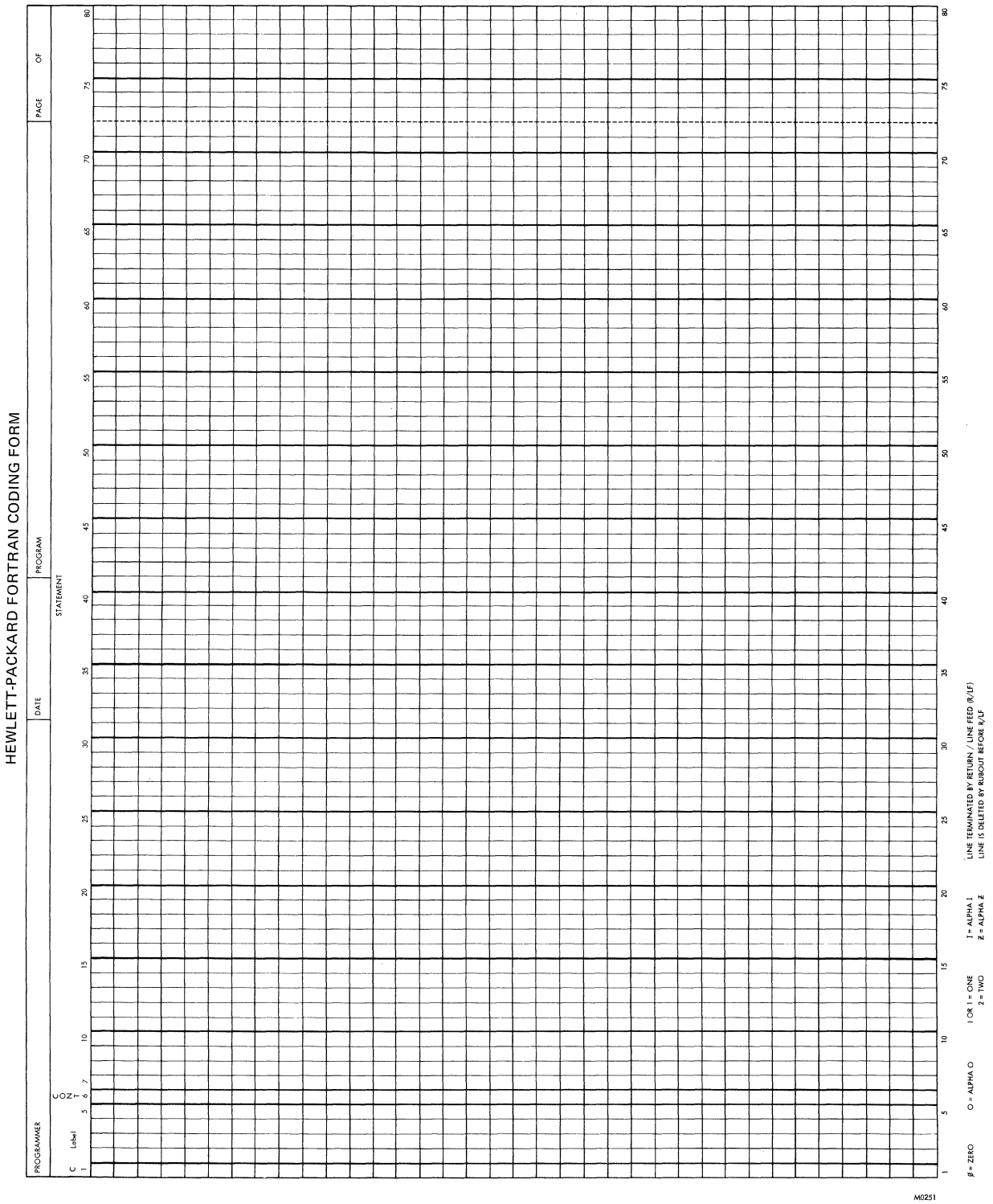
The control statement must be followed by the (CR) (LF) terminator.

End Line

Each subprogram is terminated with an end line which consists of blanks in positions 1 through 6 and the letters E, N, and D located in any of the positions 7 through 72. The special end line, END$, signifies the end of five or less programs being compiled at one time. The end line is terminated by (CR) (LF)

1.3 CODING FORM

The FORTRAN coding form is shown below. Columns 73-80 may be used to indicate a sequence number for a line; they must not be punched on paper tape. All other columns of the form conform with line positions for paper tape.



SAMPLE CODING FORM
(Actual Size 11 x 13-1/2)

ELEMENTS OF HP FORTRAN 2

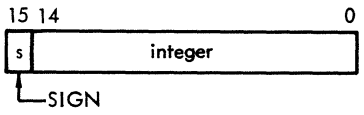
HP FORTRAN processes two types of data. They differ in mathematical significance, constant format, and symbolic representation. The two types are real and integer quantities.

2.1 DATA TYPE PROPERTIES

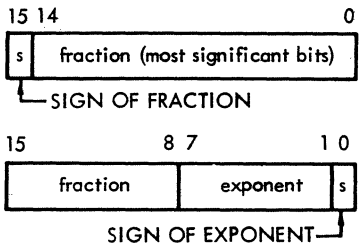
Integer and real data quantities have different ranges of values.

An integer quantity has an assumed fixed decimal point. It is represented by a 16-bit computer word with the most significant bit as the sign and the assumed decimal point on the right of the least significant bit.

An integer quantity has a range of -2^{15} to $2^{15}-1$.



A real quantity has a floating decimal point; it consists of a fractional part and an exponent part. It is represented by two 16-bit computer words; the exponent and its sign are eight bits; the fraction and its sign are twenty-four bits.



It has a range in magnitude of approximately 10^{-38} to 10^{38} and may assume positive, negative, or zero values. If the fraction is negative, the number is in two's complement form. A zero

value is stored as all zero bits. Precision is approximately seven decimal digits.

2.2 CONSTANTS

A constant is a value that is always defined during execution and may not be redefined. Three types of constants are used in HP FORTRAN: integer, octal (treated as integer), and real. The type of constant is determined by its form and content.

Integer

An integer constant consists of a string of up to five decimal digits. If the range -32768 to 32767 (-2^{15} to $2^{15}-1$) is exceeded, a diagnostic is provided by the compiler.

Examples:

8364	5932
1720	9
1872	31254
125	1
3653	30000

Octal

Octal constants consist of up to six octal digits followed by the letter B. The form is:

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

n_1 is 0 or 1

$n_2 - n_6$ are 0 through 7

If the constant exceeds six digits, or if a non-octal digit appears, the constant is treated as zero and a compiler diagnostic is provided.

Examples:

7677B	7631B
3270B	5B
3520B	75026B
175B	177776B
567B	177777B

Real

Real constants may be expressed as an integer part, a decimal point, and a decimal fraction part. The constant may include an exponent, representing a power of ten, to be applied to the preceding quantity. The forms of real constants are:

n.n n. .n n.nE±e n.E±e .nE±e

n is the number and e is the exponent to the base ten. The plus sign may be omitted for a positive exponent. The range of e is 0 through 38. When the exponent indicator E is followed by a + or - sign, then all digits between the sign and the next operator or delimiter are assumed to be part of the exponent expression, e.

If the range of the real constant is exceeded, the constant is treated as zero and a compiler diagnostic message occurs.

Examples:

4.512	4.5E2
4.	.45E+3
.512	4.5E-5
4.0	0.5
4.E-10	.5E+37
1.	10000.0

2.3 VARIABLES

A variable is a quantity that may change during execution; it is identified by a symbolic name. Simple and subscripted variables are recognized. A simple variable represents a single quantity; a subscripted variable represents a single quantity (element) within an array of quantities. Variables are identified by one to five alphanumeric characters; the first character must be alphabetic.

The type of variable is determined by the first letter of the name. The letters I, J, K, L, M, and N, indicate an integer (fixed point) variable; any other letter indicates a real (floating point) variable. Spaces imbedded in variable names are ignored.

Simple Variable

A simple variable defines the location in which values can be stored. The value specified by the name is always the current value stored in that location.

Examples:

Integer	Real
I	ALPHA
JAIME	G13
K9	DOG
MIL	XP2
NIT	GAMMA

Subscripted Variable

A subscripted variable defines an element of an array; it consists of an alphanumeric identifier with one or two associated subscripts enclosed in parentheses. The identifier names the array; the subscripts point to the particular element. If more than two subscripts appear, a compiler diagnostic message is given.

Subscripts may be integer constants, variables, or expressions; they may have the form (exp_1, exp_2), where exp_i is one of the following:

c*v+k	v-k
c*v-k	v
c*v	k
v+k	

where c and k are integer constants and v is a simple integer variable.

Examples:

Integer	Real
I(J, K)	A(J)
LAD(3, 3)	BACK(M+5, 9)
MAJOR (24*K, I+5)	OP45(4*I)
NU (K+2)	RADI (IDEG)
NEXT (N*5)	VOLTI (I, J)

2.4 ARRAYS

An array is an ordered set of data of one or two dimensions; it occupies a block of successive memory locations. It is identified by a symbolic name which may be used to refer to the entire array. An array and its dimensions must be declared at the beginning of the program in a DIMENSION or COMMON statement. The type of an array is determined by the first letter of the array name. The letters I, J, K, L, M, and N, indicate an integer array; any other letter indicates a real array.

Each element of an array may be referred to by the array name and the subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array, however, no diagnostic messages are issued.

Array Structure

Elements of arrays are stored by columns in ascending order of storage locations. An array declared as SAM(3, 3), would be structured as:

	Columns		
Rows	SAM(1, 1)	SAM(1, 2)	SAM(1, 3)
	SAM(2, 1)	SAM(2, 2)	SAM(2, 3)
	SAM(3, 1)	SAM(3, 2)	SAM(3, 3)

and would be stored as:

m	SAM(1, 1)
m+1	SAM(2, 1)
m+2	SAM(3, 1)
m+3	SAM(1, 2)
m+4	SAM(2, 2)
m+5	SAM(3, 2)
m+6	SAM(1, 3)
m+7	SAM(2, 3)
m+8	SAM(3, 3)

The location of an array element with respect to the first element is a function of the subscripts, the first dimension, and the type of the array. Addresses are computed modulo 2^{15} .

Given DIMENSION A(L, M), the memory location of A(i, j) with respect to the first element, A, of the array, is given by the equation:

$$\ell = A + \left| i - 1 + L(j - 1) \right| * s$$

The quantity in braces is the expanded subscript expression. The element size, s, is the number of storage words required for each element of the array: for integer arrays, s = 1; for real arrays, s = 2.

Array Notation

The following subscript notations are permitted for array elements:

For a two-dimensional array, $A(d_1, d_2)$:

A(I, J)	implies A(I, J)
A(I)	implies A(I, 1)
A	implies A(1, 1)†

For a single-dimension array, A(d)

A(I)	implies A(I)
A	implies A(1)

The elements of a single-dimension array, A(d), however, may not be referred to as A(I, J). A diagnostic message is given by the compiler if this is attempted.

2.5 EXPRESSIONS

An expression is a constant, variable, function or a combination of these separated by operators and parentheses, written to comply with the rules for constructing the particular type of instruction. An arithmetic expression has numerical value; its type is determined by the type of the operands.

† In an Input/Output list, the name of a dimensioned array implies the entire array rather than the first element.

Examples:

A+B-C	.4+SIN(ALPHA)
X*COS(Y)	A/B+C-D*F
RALPH-ALPH	4+2*IABS(LITE)

2.6 STATEMENTS

Statements are the basic functional units of the language. Executable statements specify actions; non-executable statements describe the characteristics and arrangement of data, editing information, statement functions, and classification of program units.

A statement may be given a numeric label of up to four digits (1 to 9999); a label allows other statements to refer to a statement. Each statement label used must be unique within the program.

ARITHMETIC EXPRESSIONS AND ASSIGNMENT STATEMENTS 3

3.1 ARITHMETIC EXPRESSIONS

An arithmetic expression may be a constant, a simple or subscripted variable, or a function. Arithmetic expressions may be combined by arithmetic operators to form complex expressions.

Arithmetic operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

If α is an expression, (α) is an expression.
If α and β are arithmetic expressions, then the following are expressions:

$\alpha + \beta$ $\quad \alpha - \beta$ $\quad \alpha/\beta$
$\alpha * \beta$ $\quad +\alpha$ $\quad -\alpha$
$\alpha ** \beta$

An arithmetic expression may not contain adjoining arithmetic operators, α op op β.

Expressions of the form $\alpha**\beta$ and $\alpha**(-\beta)$ are valid; $\alpha**\beta**\gamma$ is not valid.

Examples:

PROGRAMMER | DATE | PROGRAM

STATEMENT

C | Label | CONT | 1 5 6 7 10 15 20 25 30 35 40 45 50

```
      Z
      L*533+2**15-I
      ABLE-3.14*HOUSE**32.E-2
      5*JACK(K,L+5)-LOUD
```

Order or Evaluation

In general, the hierachy of arithmetic operation is:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
-	subtraction	class 3
+	addition	

In an expression with no parentheses or within a pair of parentheses, evaluation basically proceeds from left to right, or in the above order if adjacent operators are in a different class. †

Expressions enclosed in parentheses and function references are evaluated as they are encountered from left to right.

Examples:

In the examples below, s_1, s_2, ..., s_n indicate intermediate results during the evaluation of the expression; the symbol → can be interpreted as "goes to".

a) Evaluation of class 1 precedes class 3

A+B**C-D
B**C → s_1
s_1+A → s_2
s_2-D → s_3 s_3 is the evaluated expression

b) Evaluation of class 2 precedes class 3

A*B*C/D+E*F-G/H
A*B → s_1
s_1*C → s_2
s_2/D → s_3
E*F → s_4
s_4 + s_3 → s_5
G/H → s_6
-s_6 → s_7
s_7 + s_5 → s_8 s_8 is the evaluated expression

† When writing an integer expression it is important to remember not only the left to right scanning process, but also that dividing an integer quantity by an integer quantity yields a truncated result; thus 11/3 = 3 . The expression I*J/K may yield a different result than the expression J/K*I . For example, 4*3/2 = 6 ; but 3/2*4 = 4 .

c) Evaluation of an expression including a function is performed.

A+B**C+D+COS(E)
B**C → s_1
A+s_1 → s_2
s_2 + D → s_3
COS(E) → s_4
s_4 + s_3 → s_5 s_5 is the evaluated expression

d) Parentheses can control the order of evaluation

A*B/C+D
A*B → s_1
s_1 /C → s_2
s_2 +D → s_3 s_3 is the evaluated expression

A*B/(C+D)
A*B → s_1
C+D → s_2
s_1 /s_2 → s_3 s_3 is the evaluated expression

e) If more than one pair of parentheses or if an exponential expression appears, evaluation is performed left to right.

A+B**C-(D*E+F)+(G-H*P)
B**C → s_1
s_1 + A → s_2
D*E → s_3
s_3 +F → s_4
-s_4 → s_5
s_5 + s_2 → s_6
H*P → s_7
-s_7 → s_8
s_8 + G → s_9
s_9 + s_6 → s_{10} s_{10} is the evaluated expression

Type of Expression

With the exception of exponentiation and function arguments, all operands within an expression must be of the same type. An expression is either real or integer depending on the type of all of its constituent elements.

If either an integer or real operand is exponentiated by an integer operand, the resultant element is of the same type as that of the operand being exponentiated. If both operands are real, the resultant element is real.

Examples:

```
J**I    integer
A**I    real
A**B    real
```

An integer exponentiated by a real operand is not valid.

3.2 ASSIGNMENT STATEMENTS

An arithmetic assignment statement is of the form:

```
v = e
```

The variable, v , may be simple or subscripted; e is an expression. Execution of this statement causes the evaluation of the expression, e , and the assignment of the value to the variable.

Type of Statement

The processing of the evaluated expression is performed according to the following table:

Type of v	Type of e	Assignment rule
Integer	Integer	Transmit e to v without change.
Integer	Real	Truncate and transfer as integer to v.
Real	Integer	Transform integer form of e to floating decimal and transfer to v.
Real	Real	Transmit e to v without change.

Examples:

```
A=B**C+D+COS(E)                    Transmit without change
SAM(6)=R-S(6,2)*(T/U)              Transmit without change
N=W+3.*(X**Y-Z)                    Truncate
BAKER=I*J+K*(L-M/N)                Convert to real
N=IZZY+LAKE/MOD                    Transmit without change
```

3.3 MASKING OPERATIONS

In HP FORTRAN, masking operations may be performed using the Basic External Functions IAND, IOR, and NOT (see Chapter 6). These functions are as follows:

IAND	Form the bit-by-bit logical product of two operands
IOR	Form the bit-by-bit logical sum of two operands
NOT	Complement the operand

The operations are described by the following table:

Value of Arguments		Value of Function		
a_1	a_2	IAND (a_1, a_2)	IOR (a_1, a_2)	NOT (a_1)
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Examples:

```
IA = 72507B
IB = 71550B
```

IAND (IA, IB) is 70500B
IOR (IA, IB) is 73557B
NOT (IA) is 105270B

SPECIFICATIONS STATEMENTS 4

The Specifications statements, which include DIMENSION, COMMON, and EQUIVALENCE, define characteristics and arrangement of the data to be processed. These statements are non-executable; they do not produce machine instructions in the object program. The statements must all appear before the first executable statement in the following order: DIMENSION, COMMON, and EQUIVALENCE.

4.1 DIMENSION

The DIMENSION statement reserves storage for one or more arrays.

$$\text{DIMENSION } v_1 (i_1),\ v_2 (i_2), \ldots,\ v_n (i_n)$$

An array declarator, $v_j(i_j)$; defines the name of an array, v_j , and its associated dimensions, (i_j) . The declarator subscript, i , may be an integer constant or two integer constants separated by a comma. The magnitude of the values given for the subscripts indicates the maximum value that the subscript may attain in any reference to the array.

The number of computer words reserved for a given array is determined by the product of the subscripts and the type of the array name. For integer arrays, the number of words equals the number of elements in the array. For real arrays, two words are used for each element; the storage area is twice the product of the subscripts.

A diagnostic message is printed if an array size exceeds 2^{15} -1 locations.

Examples:

```
DIMENSION SAM (5, 10), ROGER (10, 10), NILE (5,20)

Area reserved for SAM      5*10*2 = 100 words
Area reserved for ROGER   10*10*2 = 200 words
Area reserved for NILE     5*20*1 = 100 words
```

4.2 COMMON

The COMMON statement reserves a block of storage that can be referenced by the main program and one or more subprograms. The areas of common information are specified by the statement form:

COMMON $a_1, a_2, \ldots, a_n$

Each area element, a_i, identifies a segment of the block for the subprogram in which the COMMON statement appears. The area elements may be simple variable identifiers, array names, or array declarators (dimensioned array names).

If dimensions for an array appear both in a COMMON statement and a DIMENSION statement, those in the DIMENSION statement will be used.

Any number of COMMON statements may appear in a subprogram section (preceding the first executable statement). The order of the arrays in common storage is determined by the order of the COMMON statements and the order of the area elements within the statements. All elements are stored contiguously in one block.

At the beginning of program execution, the contents of the common block are undefined; the data may be stored in the block by input/output or assignment statements.

Examples:

```
COMMON I (5), A (6), B (4)

Area reserved for I  =  5 words
Area reserved for A  = 12 words
Area reserved for B  =  8 words
                       -------
Common area            25 words

                Common
                Block
Origin          I (1)
                I (2)
                I (3)
                I (4)
                I (5)
                A (1)
                A (1)
```

A (2)
A (2)
A (3)
A (3)
A (4)
A (4)
A (5)
A (5)
A (6)
A (6)
B (1)
B (1)
B (2)
B (2)
B (3)
B (3)
B (4)
B (4)

Correspondence of Common Blocks

Each subprogram that uses the common block must include a COMMON statement. Each subprogram may assign different variable and array names, and different array dimensions, however, if corresponding quantities are to agree, the types should be the same for corresponding positions in the block.

Examples:

MAIN PROG COMMON I (5), A (6), B (4)

⋮

SUBPROG1 COMMON J (3), K (2), C (5), D (5)

MAIN PROG reference	Common Block	SUBPROG1 reference
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	K (1)
I (5)	integer 5	K (2)
A (1)	real 1	C (1)
A (1)	real 1	C (1)

MAIN PROG reference	Common Block	SUBPROG1 reference
A (2)	real 2	C (2)
A (2)	real 2	C (2)
A (3)	real 3	C (3)
A (3)	real 3	C (3)
A (4)	real 4	C (4)
A (4)	real 4	C (4)
A (5)	real 5	C (5)
A (5)	real 5	C (5)
A (6)	real 6	D (1)
A (6)	real 6	D (1)
B (1)	real 7	D (2)
B (1)	real 7	D (2)
B (2)	real 8	D (3)
B (2)	real 8	D (3)
B (3)	real 9	D (4)
B (3)	real 9	D (4)
B (4)	real 10	D (5)
B (4)	real 10	D (5)

If portions of a common block are not referred to by a particular subprogram, dummy variables may be used to provide correspondence in reserved areas.

Examples:

MAIN PROG COMMON I (5), A (6), B (4)

⋮

SUBPROG2 COMMON J (17), B (4)

MAIN PROG reference	Common Block	SUBPROG2 reference
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	J (4)
I (5)	integer 5	J (5)

MAIN PROG reference	Common block	SUBPROG2 reference	
A (1)	real 1	J (6)	
A (1)	real 1	J (7)	J (17) is a dummy array. It is not referenced in SUBPROG 2 but provides proper correspondence in reserved areas so that SUBPROG 2 can refer to array B.
A (2)	real 2	J (8)	
A (2)	real 2	J (9)	
A (3)	real 3	J (10)	
A (3)	real 3	J (11)	
A (4)	real 4	J (12)	
A (4)	real 4	J (13)	
A (5)	real 5	J (14)	
A (5)	real 5	J (15)	
A (6)	real 6	J (16)	
A (6)	real 6	J (17)	
B (1)	real 7	B (1)	
B (1)	real 7	B (1)	
B (2)	real 8	B (2)	
B (2)	real 8	B (2)	
B (3)	real 9	B (3)	
B (3)	real 9	B (3)	
B (4)	real 10	B (4)	
B (4)	real 10	B (4)	

The length of the common block may differ in different subprograms, however, the subprogram (or main program) with the longest common block must be the first to be loaded at execution time.

4.3 EQUIVALENCE

The EQUIVALENCE statement permits sharing of storage by two or more entities. The statement has the form:

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

in which each k is a list of the form:

a_1, a_2, . . . , a_m

Each a is either a variable name or a subscripted variable; the subscript of which contains only constants. The number of subscripts must correspond to the number of subscripts for the related array declarator.

All names in the list may be used to represent the same location. If an equivalence is established between elements of two or more arrays, there is a corresponding equivalence between other elements of the arrays; the arrays share some storage locations. The lengths may be different or equal.

Examples:

```
DIMENSION A (5), B (4)

EQUIVALENCE (A (4), B (2))
```

Array 1 Name	Array 2 Name	Quantity Element
A (1)		real 1
		real 1
A (2)		real 2
		real 2
A (3)	B (1)	real 3
		real 3
A (4)	B (2)	real 4
		real 4
A (5)	B (3)	real 5
		real 5
	B (4)	real 6
		real 6

The EQUIVALENCE statement establishes that the names A (4) and B (2) identify the fourth real quantity. The statements also establish a similar correspondence between A (3) and B (1), and A (5) and B (3).

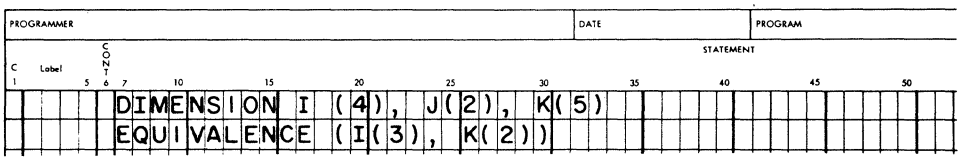
An integer array/or variable may be made equivalent to a real array or variable; equivalence may be established between different types. The variables may be with or without subscripts.

The effect of an EQUIVALENCE statement depends on whether or not the variables are assigned to the common block. When two variables or array elements share storage, the symbolic names of the variables or arrays may not both appear in COMMON statements in the same subprogram. The assignment of storage to variables and arrays declared in a COMMON statement is determined on the basis of their type and the array

declarator. Entities so declared are always contiguous according to the order in the COMMON statement. The EQUIVALENCE statement must not alter the origin of the common block, but arrays may be defined so that the length of the common block is increased.

Examples:

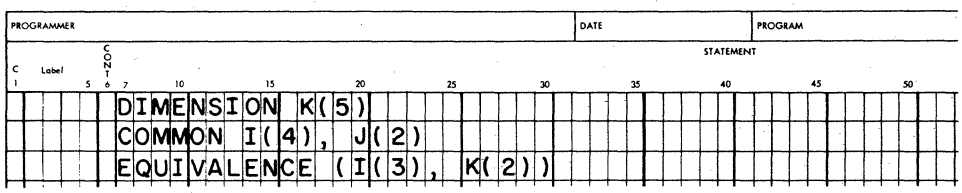
a) Effect of EQUIVALENCE, variables not in common block:



storage is assigned as follows:

Arrays	Quantities
I (1)	integer 1
I (2) K (1)	integer 2
I (3) K (2)	integer 3
I (4) K (3)	integer 4
K (4)	integer 5
K (5)	integer 6
J (1)	integer 7
J (2)	integer 8

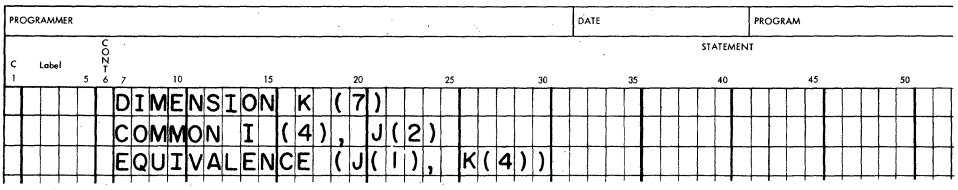
b) Effect of EQUIVALENCE, some variables in common block:



storage is assigned as follows:

Arrays	Quantities	
I (1)	integer 1	common block
I (2) K (1)	integer 2	
I (3) K (2)	integer 3	
I (4) K (3)	integer 4	
J (1) K (4)	integer 5	
J (2) K (5)	integer 6	

c) Effect of EQUIVALENCE on the length of the common block:



storage is assigned as follows:

Arrays	Quantities	
I (1)	integer 1	common block
I (2) K (1)	integer 2	
I (3) K (2)	integer 3	
I (4) K (3)	integer 4	
J (1) K (4)	integer 5	
J (2) K (5)	integer 6	
K (6)	integer 7	
K (7)	integer 8	

The value of the subscripts for an array being made equivalent to another array should not be such that the origin of the common block is changed (for example, EQUIVALENCE (I (3), K(4)).

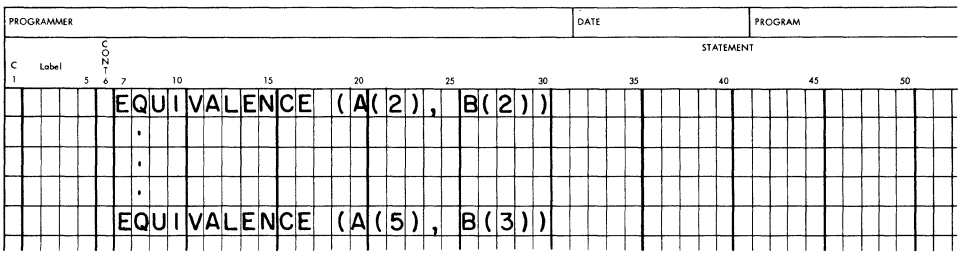
	Arrays			Quantities
		K (1)	← origin changed	integer 1
origin →	I (1)	K (2)		integer 2
	I (2)	K (3)		integer 3
	I (3)	K (4)		integer 4

Arrays		Quantities
I (4)	K (5)	integer 5
J (1)	K (6)	integer 6
J (2)	K (7)	integer 7

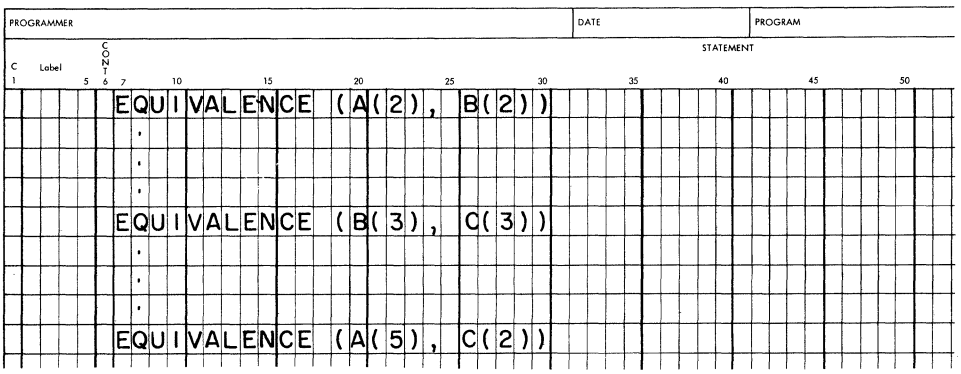
If contradictory EQUIVALENCE relationships are specified, a diagnostic message is printed.

Example:

a)



b)



CONTROL STATEMENTS 5

Program execution normally proceeds from statement to statement as they appear in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section. Control may be transferred to an executable statement only; a transfer to a non-executable statement will result in a program error which is usually recognized during compilation as a transfer to an undefined label.† With the DO statement, a predetermined sequence of instructions can be repeated a number of times with the stepping of a simple integer variable after each iteration.

Statements are labelled by unsigned numbers, 1 through 9999, which can be referred to from other sections of the program. A label up to four digits long precedes the FORTRAN statement and is separated from it by at least one blank or a zero. Imbedded blanks and leading zeros in the label are ignored: 1, 01, 0 1, 0001 are identical.

5.1 GO TO STATEMENTS

GO TO statements provide transfer of control.

GO TO k

This statement, an unconditional GO TO, causes the transfer of control to the statement labelled k .

GO TO (k_1, k_2, ... , k_n), i

This statement, a computed GO TO, acts as a many-branched transfer. The k's are statement labels and i is a simple integer variable. Execution of this statement causes the statement identified by the label k_j to be executed next, where j

† A transfer to a FORMAT statement is not detectable during compilation; if such an error occurs, no diagnostic message is produced.

is the value of i at the time of execution, and $1 \leq j \leq n$. If $i < 1$, a transfer to k_1 occurs; if $i > n$, a transfer to k_n occurs.

Examples:

```
PROGRAMMER                                  DATE        PROGRAM
                                                        STATEMENT
C  Label  CONT
1      5  6  7    10        15        20        25        30        35        40        45        50
      10   GO TO 500
           ISWCH = 2
      35   A = X**Y
             .
             .
      40   GO TO (5,10,15,20), ISWCH
     500   JSWCH = ISWCH + 1
     540   GO TO (25,30,35,40), JSWCH
```

At statement 40, control transfers to statement 10, which is an unconditional transfer to statement 500. At 540 control transfers to statement 35.

5.2 IF STATEMENTS

The arithmetic IF statement provides conditional transfer of control

IF (e) k_1, k_2, k_3

The e is an arithmetic expression and the k's are statement labels. The arithmetic IF is a three-way branch. Execution of this statement causes evaluation of the expression and transfer of control depending on the following conditions:

e < 0, go to k_1
e = 0, go to k_2
e > 0, go to k_3

Examples:

```
PROGRAMMER                                  DATE        PROGRAM
                                                        STATEMENT
C  Label  CONT
1      5  6  7    10        15        20        25        30        35        40        45        50
           IF (A)5, 10, 15
           IF (X**Y*COS(Z)+W)5,35,15
```

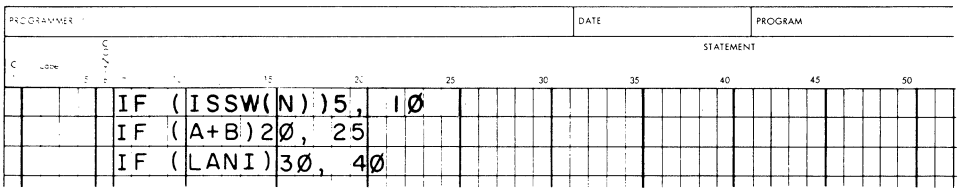
The logical IF statement provides conditional transfer of control to either of two statements:

IF (e) k_1, k_2

The e is an arithmetic expression that may yield a negative or non-negative (positive or zero) value. Execution of this statement causes evaluation of the expression and transfer of control under the following conditions:

e < 0, go to k_1
e ≥ 0, go to k_2

Examples:



5.3 DO STATEMENTS

A DO statement makes it possible to repeat a group of statements.

DO n i = m_1, m_2, m_3

or

DO n i = m_1, m_2

The n is the label of an executable statement which ends the group of statements. The statement, called the terminal statement, must physically follow the DO statement in the source program. It may not be a GO TO of any form, IF, RETURN, STOP, PAUSE, or DO statement.

The i is the control variable; it may be a simple integer variable.

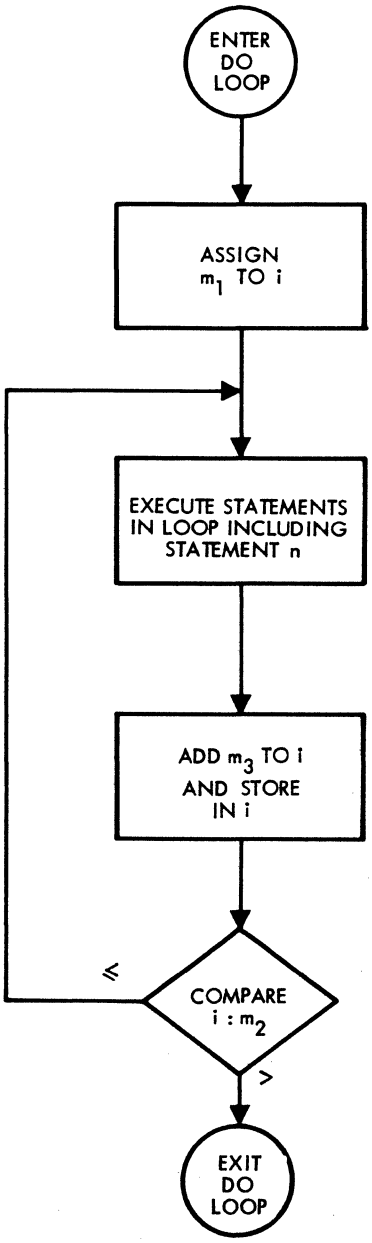
The m's are indexing parameters: m_1 is the initial parameter; m_2, the terminal parameter; and m_3, the incrementation parameter. They may be unsigned integer constants or

simple integer variables. At time of execution, they all must be greater than zero. If m_3 does not appear (second form), the incrementation value is assumed to be 1.

A DO statement defines a loop. Associated with each DO statemeant is a range that is defined to be those executable statements following the DO, to and including the terminal statement associated with the DO. At time of execution, the following steps occur:

1. The control variable is assigned the value of the initial parameter.
2. The range of the DO is executed.
3. The terminal statement is executed and the control variable is increased by the value of the incrementation parameter.
4. The control variable is compared with the terminal parameter. If less than or equal to the terminal parameter, the sequence is repeated starting at step 2. If the control variable exceeds the terminal parameter, the DO loop is satisfied and control transfers to the statement following n . The control variable becomes undefined.

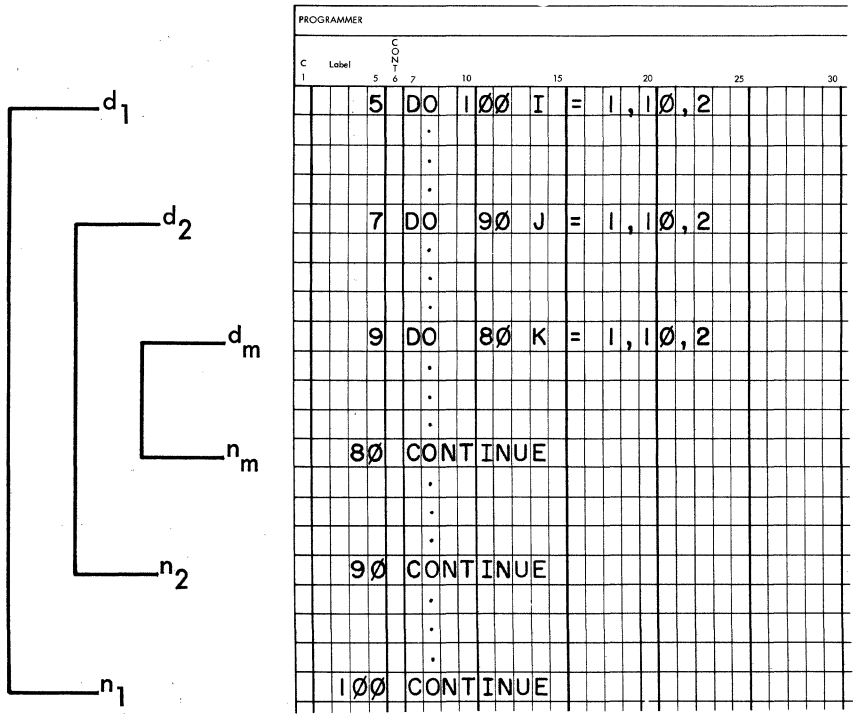
Should m_1 exceed m_2 on the initial entry to the loop, the range of the DO is executed and control passes to the statement after n . If a transfer out of the DO loop occurs before the DO is satisfied, the current value of the control variable is preserved. The control variable, initial parameters, terminal parameter, and incrementation parameters may not be redefined during the execution of the range of the DO loop.

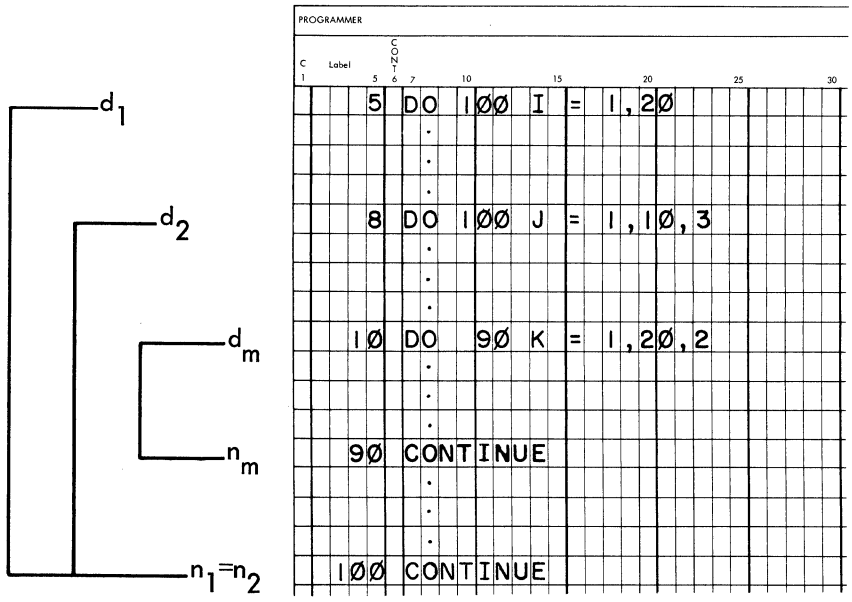

ENTER DO LOOP
ASSIGN m_1 TO i
EXECUTE STATEMENTS IN LOOP INCLUDING STATEMENT n
ADD m_3 TO i AND STORE IN i
COMPARE i : m_2
≤
>
EXIT DO LOOP

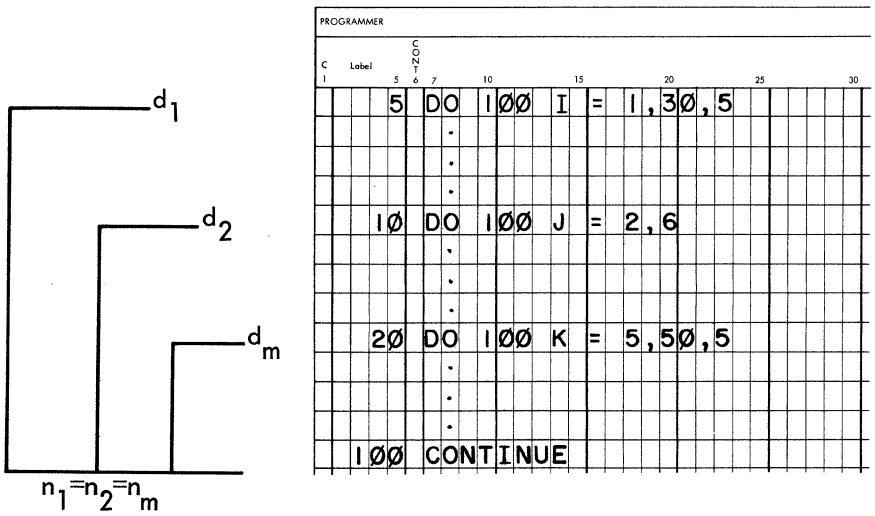
DO Nests

When the range of a DO loop contains another DO loop, the latter is said to be nested. DO loops may be nested 10 deep. The last statement of a nested DO loop must be the same as the last statement of the outer loop or occur before it. If $d_1, d_2, \ldots, d_n$ are DO statements, which appear in the order indicated by the subscripts; and if $n_1, n_2, \ldots, n_m$ are the respective terminal statements, then n_m must appear before or be the same as n_{m-1}, n_{m-1} must appear before or be the the same as n_2, and n_2 must appear before or be the same as n_1.

Examples:



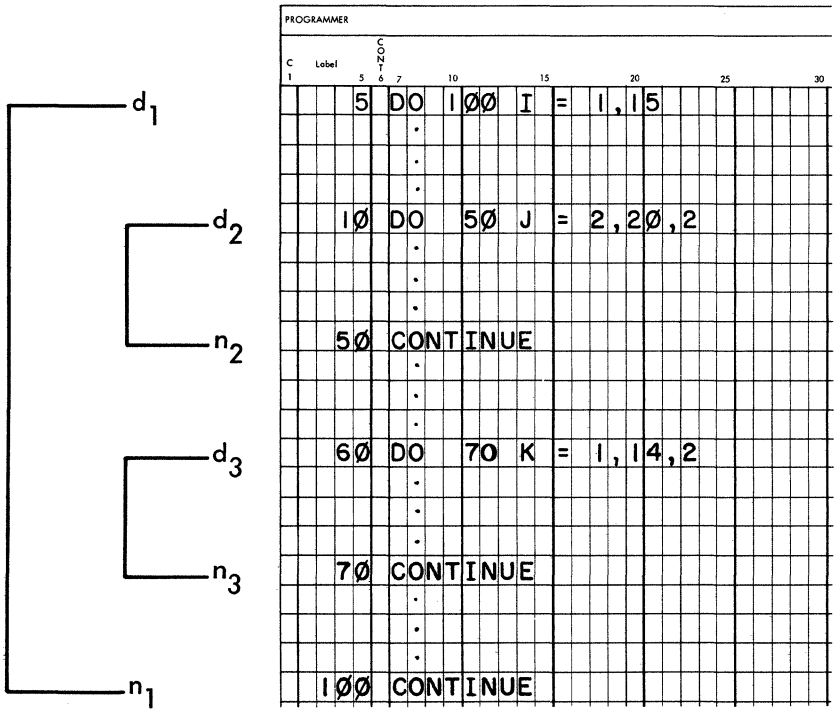


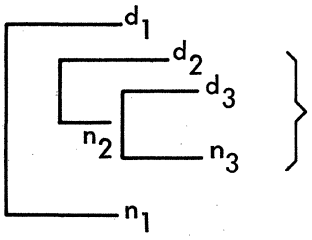


If one or more nested loops have the same terminal statement, when the inner DO is satisfied, the control variable for the next outer loop is incremented and tested against its associated terminal parameter. Control transfers to the statement following the terminal statement only when all related loops are satisfied.

DO loops may be nested in common with other loops as long as their ranges do not overlap.

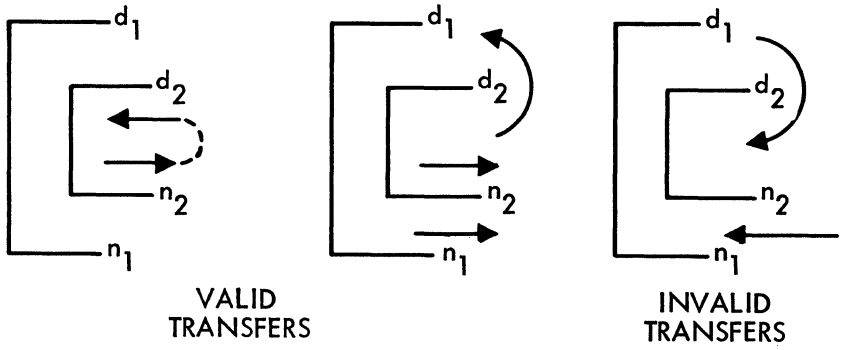
Examples:





Invalid, ranges overlap

In a DO nest, a transfer may be made from an inner loop into an outer loop, and transfer is permissible outside of the loop. It is illegal, however, for a GO TO or IF to initiate a transfer of control from outside of the range of a DO into its range.



5.4 CONTINUE

This statement acts as no-operation instruction.

```
CONTINUE
```

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement.

5.5 PAUSE

This statement provides a temporary program halt.

```
PAUSE n
   or
PAUSE
```

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types PAUSE on the Standard Teleprinter Output unit. The value of n , if given is displayed in the A-Register. When the RUN button is pressed, program execution resumes at the next statement.

5.6 STOP

The STOP statement terminates the execution of the program.

```
STOP n
   or
STOP
```

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types STOP on the Standard Teleprinter Output unit. The value of n , if given, is displayed in the A-Register. If the RUN button is pressed, the halt operation is repeated.

5.7 END

The END statement indicates the physical end of a program or subprogram. It has the form:

END name

The END statement is required for every program or subprogram. The name of the program can be included, but it is ignored by the compiler. The END statement is executable in the sense that it will effect return from a subprogram in the absence of a RETURN statement. An END statement may be labeled and may serve as a junction point.

5.8 END$

The END$ statement indicates the physical end of five or less programs or subprograms that are to be compiled at one time. If there are four or less programs, the statement is printed on the source program listing. If there are exactly five, the statement is not printed. If more than five programs are on the same tape, the END$ may be omitted after the fifth program; the compiler stops accepting input after the fifth is processed.

MAIN PROGRAM, FUNCTIONS, AND SUBROUTINES 6

A FORTRAN program consists of a 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.

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

In addition to multi-statement function subprograms, a function may be defined by a single statement in the program (statement function) or it may be defined as part of the FORTRAN Library (basic external function). A statement function definition may appear in a main program or subprogram body and is available only to the main program or subprogram containing it. A statement function may contain references to function subprograms, basic external functions, or other previously defined statement functions in the same subprogram. Basic external function references may appear in the main program, subprogram, and statement functions.

Main programs, subprograms, statement functions, and basic external functions communicate by means of arguments (parameters). The arguments appearing in a subroutine call or function reference are actual arguments. The corresponding entities appearing with the subprogram, statement function, or basic external function definition are the dummy arguments.

6.1 ARGUMENT CHARACTERISTICS

Actual and dummy arguments must agree in order, type, and number. If they do not agree in type, errors may result in the program execution, since no conversion takes place and no diagnostic messages are produced.

Within subprograms, dummy arguments may be array names or simple variables; for statement functions, they may be variables only. Dummy arguments are local to the subprogram or statement function containing them and, therefore, may be the same as names appearing elsewhere in the program. A maximum of 63 dummy arguments may be used in a function or subroutine.

No element of a dummy argument list may appear in a COMMON or EQUIVALENCE statement within the subprogram. If it does, a compiler diagnostic results. When a dummy argument represents an array, it should be declared in a DIMENSION statement within the subprogram. If it is not declared, only the first element of the array will be available to the subprogram and the array name must appear in the subprogram without subscripts.

Actual arguments appearing in subroutine calls and function references may be any of the following:

A constant
A variable name
An array element name
An array name
Any other arithmetic expression

6.2 MAIN PROGRAM

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

PROGRAM name

The name is an alphanumeric identifier of up to five characters. If the PROGRAM statement is omitted, the compiler assigns the name "FTN."

6.3 SUBROUTINE SUBPROGRAM

An external subroutine is a computational procedure which may return none, one, or more than one value through its arguments or through common storage. No value or type is associated with the name of a subroutine.

The first statement of a subroutine subprogram gives its name and, if relevant, its dummy arguments.

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

or

SUBROUTINE s

The symbolic name, s, is an alphanumeric identifier of up to five characters by which the subroutine is called. If the subroutine is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the subroutine.

The name of the subroutine must not appear in any other statement within the subprogram.

The subroutine may define or redefine one or more of its arguments and areas in common so as to effectively return results. It may contain any statements except FUNCTION, another SUBROUTINE statement, or any statement that directly or indirectly references the subroutine being defined. It must have at least one RETURN or END statement which returns control to the calling program.

Examples:

```
PROGRAMMER
C  Label  CONT
1      5   6  7        10        15        20        25
           SUBROUTINE JIV(P,W,H)
           Z=5. *W+P**3
           H=Z-3.
           RETURN
           END


           SUBROUTINE MUL(K)
           COMMON MAT(10),PROD(10)
           DO 5 I=1, 10
         5 PROD(I)=MAT(I)*K
           RETURN
           END
```

P, W and H are the dummy parameters. Actual values supplied by a calling program are to be substituted for P and W. The variable name supplied for H would contain the result on return to the calling program.
MUL multiplies the array supplied for MAT by the single value supplied for K to produce values to be stored in array PROD.

6.4 SUBROUTINE CALL

The executable statement in the calling program for referring to a subroutine is:

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

or

CALL s

The symbolic name, s, identifies the subroutine being called; the a's define the actual arguments. The name may not appear in any specification statements in the calling program.

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

The CALL statement transfers control to the subroutine. Execution of the subroutine results in an association of actual arguments with all appearances of dummy arguments in executable statement and function definition statements. If the actual argument is an expression, the association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement is encountered, control is returned to the next executable statement following the CALL in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until satisfied.

Examples:

PROGRAMMER

C Label CONT

```
      CALL JIV (15.,12.,ABLE)
      COMMON N(10), Q(10)
            .
            .
            .
      CALL MUL(I(5,3))
```

These calls provide actual arguments for the subroutines defined in the previous example. In subroutine JIV, 15. is substituted for P; 12., for W; and ABLE, for H.
For subroutine MUL, the data is passed via COMMON. The value supplied for the dummy argument K is element (5,3) of matrix I of the calling program.

6.5 FUNCTION SUBPROGRAM

A function subprogram is a computational procedure which returns a single value associated with the function name. The type of the function is determined by the name; an integer quantity is returned if the name begins with I, J, K, L, M, or N, otherwise it will be a real quantity.

The first statement of a function subprogram must have the following form:

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

The symbolic name, f, is an alphanumeric identifier of up to five characters by which the function is referenced. If the function is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the function.

The name of the function must not appear in any non-executable statement in the subprogram. It must be used in the subprogram, however, at least once as any of the following:

The left-hand identifier of an assignment statement
An element of an input list
An actual parameter of a subprogram reference

The value of name at the time of execution of a RETURN or END statement in the subprogram is called the value of the function.

The function subprogram may define or redefine one or more of its arguments and areas in common so as to effectively return results in addition to the value of the function. If the subprogram redefines variables contained in the same expression as the function reference, the evaluation sequence of the expression must be taken into account. Variables in the portion of the expression that is evaluated before the function reference is encountered and the values of variable subscripts are not affected by the execution of the function subprogram. Variables that appear following the function reference are modified according to the subprogram processing.

Examples:

a)

```
FUNCTION IDIV(I,J)
IDIV=I/J
RETURN
END
```

The function IDIV calculates the value of I divided by J. On return to the calling program the result provided is the value of IDIV.

b)

```
FUNCTION IREAD (IUNT)
      .
      .
      .
READ(IUNT,*)IREAD
      .
      .
      .
RETURN
END
```

The function IREAD reads a value from the unit IUNT (specified as an actual parameter in the calling program.) IREAD has this value on return to the calling program.

c)

```
FUNCTION SCALL(A,B,C)
      .
      .
      .
CALL SUBF(SCALL,A,B,C)
      .
      .
      .
RETURN
END
```

SCALL is both the function name and an actual parameter of a subroutine call. The value of SCALL is provided by SUBF and returned to the calling program.

d)

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

The function defines the value of GAMMA as well as finding the value of ZETA.

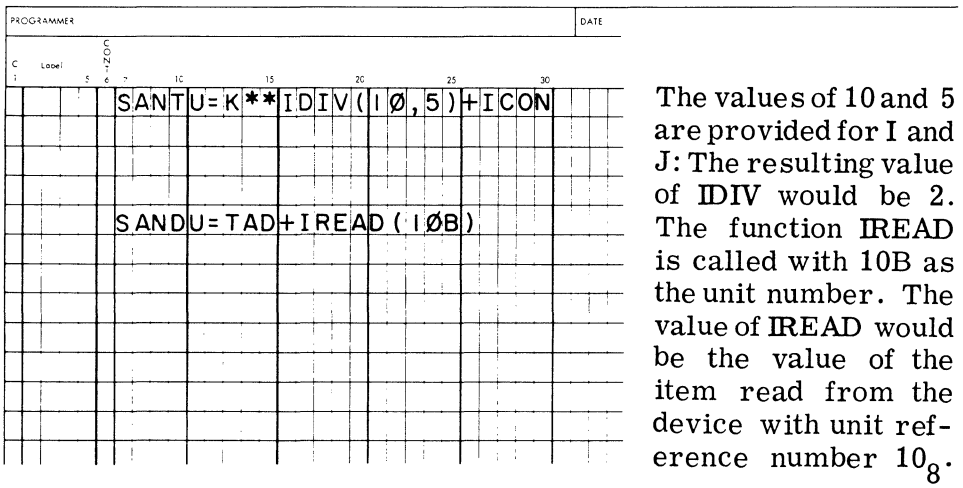
6.6 FUNCTION REFERENCE

A function subprogram is referenced by using the name and arguments in an arithmetic expression:

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

The type of function depends on the first letter of the name of the function referenced; the a's are the actual arguments. The reference may appear any place in an expression as an operand. The evaluated function will have a single value associated with the function name. When a function reference is encountered in an expression, control is transferred to the function indicated. Execution of the function results in an association of actual arguments with all appearances of dummy arguments in executable statements and function definition statements. If the actual argument is an expression, this association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement in the function subprogram is encountered, control returns to the statement containing the function reference. During execution the function also may define or redefine one or more of its arguments and areas in common.

Example:



The values of 10 and 5 are provided for I and J: The resulting value of IDIV would be 2. The function IREAD is called with 10B as the unit number. The value of IREAD would be the value of the item read from the device with unit reference number 10_8.

c)

```
PROGRAMMER                                        DATE
C  Label  CONT
1      5  6  7   10        15        20        25        30
           ALPH=BETA*SCALL(10.,9.,8.)
```

The actual parameters SCALL are 10., 9., and 8. The value of SCALL would depend on the value supplied by the subroutine SUBF.

d) The program,

```
PROGRAMMER                          DATE          PROGRAM
                                                  STATEMENT
C  Label  CONT
1      5  6  7   10        15        20        25        30        35        40        45        50
           GAMMB=5.0
           RSLT=GAMMB+7.5+ZETA(.2,.3,GAMMB)
```

would result in the following calculation:

RSLT = 5.0 + 7.5 + ZETA

where ZETA would be 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 + .00456976
     = 12.50456976
```

But, the program,

```
PROGRAMMER                          DATE          PROGRAM
                                                  STATEMENT
C  Label  CONT
1      5  6  7   10        15        20        25        30        35        40        45        50
           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
```

When referring to a function which redefines an argument which appears as a variable elsewhere in the same expression, the order of evaluation (i.e., the order in which the expression is stated) is significant.

6.7 STATEMENT FUNCTION

A statement function is defined internally to the program or subprogram in which it is referenced and must precede the first executable statement. The definition is a single statement similar in form to an arithmetic assignment statement.

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

The name of the statement function, f , is an alphanumeric identifier; a single value is associated with the name. The dummy arguments, a's , must be simple variables. One to ten arguments may be used. The expression, e , may be an arithmetic expression and may contain references to basic external functions, previously defined statement functions, or function subprograms. The dummy arguments must appear in the expression. Other variables appearing in the expression have the same values as they have outside the statement function.

The statement function name must not appear in any specification statements in the program or subprogram containing it.

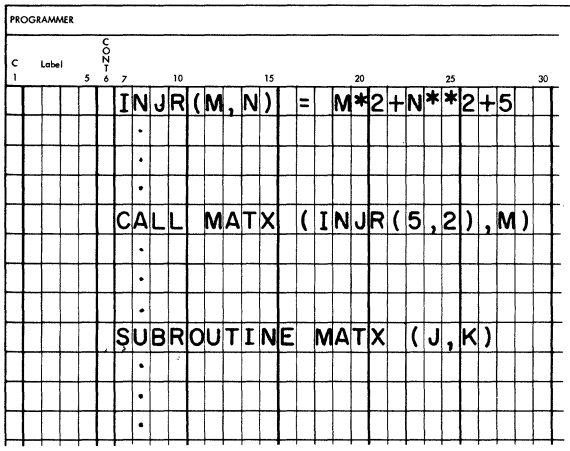
Statement functions must precede the first executable statement of the program or subprogram, but they must follow all specification statements.

A statement function reference has the form:

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

f is the function name and the a's are the actual arguments. A function reference with its appropriate actual arguments may be used to define the value of an actual argument in a subroutine call or function subprogram reference.

Example:



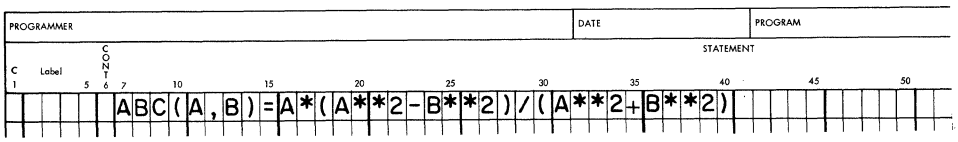
Statement function definition.

Subroutine call using statement function reference.

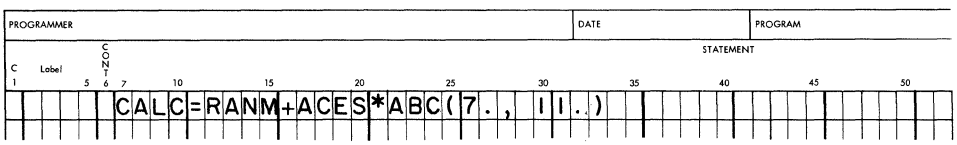
Execution of a statement function reference results in an association of actual argument values with the corresponding dummy arguments in the expression of the function definition, and evaluation of the expression. Following this, the resultant value is made available to the expression that contained the function reference and control is returned to that statement.

Example:

Statement function:



Function reference:



6.8 BASIC EXTERNAL FUNCTIONS

Certain basic functions are defined as part of the 2116A FORTRAN Library. When one of these appears as an operand in an expression, the compiler generates the appropriate calling sequence within the object program.

The types of these functions and their arguments are defined. The compiler recognizes the basic function and associates the type with the results. The actual arguments must correspond to the type required for the function; if not, a diagnostic message is issued. The functions available are shown below:

Function Name	Definition	Symbolic Name	No. of Arguments	Type of	
				Argument	Function
Absolute Value	$\|a\|$	ABS	1	Real	Real
		IABS	1	Integer	Integer
Float	Conversion from integer to real	FLOAT	1	Integer	Real
Fix	Conversion from real to integer	IFIX	1	Real	Integer
Transfer sign	Sign of a_2 times $\|a_1\|$	SIGN	2	Real	Real
		ISIGN	2	Integer	Integer
Exponential	e^a	EXP	1	Real	Real
Natural Logarithm	$\log_e$ (a)	ALOG	1	Real	Real
Trigonometric Sine	sine (a)†	SIN	1	Real	Real
Trigonometric Cosine	cos (a)†	COS	1	Real	Real
Trigonometric Tangent	tan (a)†	TAN	1	Real	Real
Hyperbolic Tangent	tanh (a)	TANH	1	Real	Real
Square Root	$(a)^{1/2}$	SQRT	1	Real	Real
Arctangent	arctan(a)	ATAN	1	Real	Real
And (Boolean)	$a_1 \wedge a_2$	IAND	2	Integer	Integer
Or (Boolean)	$a_1 \vee a_2$	IOR	2	Integer	Integer
Not (Boolean)	$\neg a$	NOT	1	Integer	Integer
Sense Switch	Sense Switch Register switch (n)	ISSW	1	Integer	Integer

†a is in radians

Examples:

PROGRAMMER | DATE | PROGRAM

C | Label | CONT | STATEMENT

```
      SIGND=A+B*C/D-E
      SIGNN=ABS(SIGND)
      Y=FLOAT(NEWT)
      ISGND=I+J*K/L-M
      ISGNN=IABS(ISGND)
      IAL   =JACK*KEN*LARRY
      ISAL  =ISIGN(IAL,ISGNN)
      POWR  =EXP(X)
      ANTLG=ALOG(Y)
      OOHYP=SIN(AGL)
      AOHYP=COS(AGL)
      OOAH  =TANH(AGLH)
      HFPR  =SQRT(Z)
      ARC   =ATAN(S)
      LPROD=IAND(M,N)
      LSUM  =IOR(M,N)
      LCLMT=NOT(M)
```

6.9 RETURN AND END

A subprogram normally contains a RETURN statement that indicates the end of logic flow within the subprogram and returns control to the calling program. It must always contain an END statement.

In function subprograms, control returns to the statement containing the function reference. In subroutine subprograms, control returns to the next executable statement following the CALL. A RETURN statement in the main program is interpreted as a STOP statement.

The END statement marks the physical end of a program, subroutine subprogram, or function subprogram. If the RETURN statement is omitted, END causes a return to the calling program. The END$ is required in addition to END statements when five or less subprograms are being compiled at one time.

INPUT/OUTPUT LISTS AND FORMAT CONTROL 7

Data transmission between internal storage and external equipment requires an input/output statement and, for ASCII character strings, either a FORMAT statement or format control symbols with the input data. The input/output statement specifies the input/output process, such as READ or WRITE; the unit of equipment on which the process is performed; and the list of data items to be moved. The FORMAT statements or control symbols provide conversion and editing information between the internal representation and the external character strings. If the data is in the form of strings of binary values, format control is unnecessary.

7.1 INPUT/OUTPUT LISTS

The input list specifies the names of the variables and array elements to which values are assigned on input. The output list specifies the references to the variables, array elements, and constants whose values are transmitted. The input and output lists are of the same form. The list elements consist of variable names, array elements, and array names separated by commas. The order in which the elements appear in the list is the sequence of transmission. If FORMAT statements are used, the order of the list elements must correspond to the order of the format descriptions for the data items. In array elements buffer length is limited to a maximum output of 60 computer words.

Subscripts in an input/output list may be of the form (exp_1, exp_2), where exp_i is one of the following:

```
c*v+k    v-k
c*v-k    v
c*v      k
v+k
```

where c and k are integer constants and v is a simple integer variable previously defined or defined within an implied DO loop.

DO-Implied Lists

A DO-implied list consists of one or more list elements and indexing parameters. The general form is

$$(\ldots(\text{list},\ i = m_1,\ m_2,\ m_3)\ldots)$$

list	Any series of arrays, array elements, or variables separated by commas
i	Control variable
m's	Index parameters in the form of unsigned integer constants or predefined integer variables

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 it is assumed to be one.

An implied DO loop may be used to transmit a simple variable or a sequence of variables more than one time.

Two-dimensional arrays may appear in the list with values specified for the range of the subscripts in an implied DO loop. The general form for an array is:

$$((a(d_1, d_2), i_1 = m_1,\ m_2,\ m_3),\ i_2 = n_1,\ n_2,\ n_3)$$

where,

a	An array name
d_1, d_2	Subscripts of the array in one of the preceding forms
i_1, i_2	Control variables representing either of the variable subscripts d_1 and d_2
m's, n's	Index parameters in the form of unsigned integer constants or predefined integer variables. If m_3 or n_3 is omitted, it is construed as 1.

The input/output list may contain nested implied DO loops. During execution, the control variables are assigned the values of the initial parameters ($i_1 = m_1$, $i_2 = n_1$). The first control variable defined in the list is incremented first. When the first control variable reaches the maximum value, it is reset; the next control variable to the right is incremented and the process is repeated until the last control variable has been incremented.

If the name of a dimensioned array appears in a list without subscripts, the entire array is transmitted.

Examples:

a) The DO-implied list:
((A(I,J), I=1, 20, 2), J=1, 50,5)
replaces the following:
DO x J=1, 50, 5
DO x I=1, 20, 2
transmit A (I, J)
x CONTINUE

b) Other implied DO loops might be:
((ABLE(5*KID-3, 100*LID), KID=1, 100), LID=1, 10)
((A(I, J), I=1, 5), J=1, 5) Transmit elements by column
((A(I, J), J=1, 5), I=1, 5) Transmit elements by row.

c) Nested implied DO loops:
((((A(I, J), B(K, L), K=1, 10), L=1, 15), I=1, 20), J=1, 25)
(((A(I, J), B(K), K=1, 10), I=20, 100, 10), K=9, 90, 10)

d) Simple variable transmission:
(A, K=1, 10) Transmits 10 values of A.

e) Dimensioned array transmission:
DIMENSION A(50, 20)

:

... A ... list element
is equivalent to:
DO x I = 1, 20
DO x J = 1, 50
transmit A(J, I)
x CONTINUE

7.2 FORMAT STATEMENT

ASCII input/output statements may refer to a FORMAT statement which contains the specifications relating to the internal-external structure of the corresponding input/output list elements.

FORMAT ($spec_1, \ldots,$ r($spec_m, \ldots$), $spec_n, \ldots$)

The spec's are format specifications and r is an optional repetition factor which must be an unsigned integer constant. FORMAT specifications may be nested to a depth of one level. The FORMAT statement is non-executable and may appear anywhere in the program.

7.3 FORMAT STATEMENT CONVERSION SPECIFICATIONS

The data elements in the input/output lists may be converted from external to internal and from internal to external representation according to FORMAT conversion specifications.† FORMAT statements may also contain editing codes.

Conversion Specifications

rEw.d	Real number with exponent
rFw.d	Real number without exponent
rIw	Decimal integer
r@w rKw	Octal integer
rAw	Alphanumeric character

Editing Specification

nX	Blank field descriptor
$nHh_1\ h_2 \ldots h_n$ $r''h_1\ h_2 \ldots h_n''$	Heading and labeling descriptors
r/	Begin new record

† If the type of a variable in the input/output list does not correspond to the type specified in the FORMAT statement, the compiler insures that the proper conversion from one type to the other will take place.

Both w and n are nonzero integer constants representing the width of the field in the external character string; n may be omitted if the width is one. d is an integer constant representing the number of digits in the fractional part of the string. r , the repeat count, is an optional nonzero integer constant indicating the number of times to repeat the succeeding basic field descriptor. Each h is one character.

Ew.d Output

The E specification converts numbers in storage to character form for output. The field occupies w positions in the output record; the number appears in floating point form right justified in the field as:

$$\triangleq .x_1 \ldots x_d \; E \pm ee^{\dagger}$$

$x_1 \ldots x_d$ are the most significant digits of the value of the data to be output. ee are the digits in the exponent. Field w must be wide enough to contain significant digits, signs, decimal point, E , and exponent. Generally, w should be greater than or equal to d + 4.

If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still to large for the field, dollar signs ($) are inserted in the entire field. If the field is longer than the output value, the quantity is right-justified with spaces to the left.

Examples:

PROGRAMMER

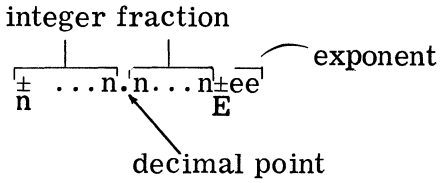
C	Label	CONT		
			WRITE(4,5)A	A contains +12.34 or -12.34
	5		FORMAT(E10.3)	Result is ^^.123E+02 or ^-.123E+02
			WRITE(4,5)A	A contains +12.34 or -12.34
	5		FORMAT(E12.3)	Result is ^^^^.123E+02 or .^^^-.123E+02
			WRITE(4,5)A	A contains +12.34 or -12.34
	5		FORMAT(E7.3)	Result is .12E+02 or -.1E+02
			WRITE(4,5)A	A contains +12.34
	5		FORMAT(E5.1)	Result is $$$$$

†The caret symbol, ^ , indicates the presence of a space.

Ew.d Input

The E specification converts the number in the input field (specified by w) to a real number and stores it in the appropriate storage locations.

The input field may consist of integer, fraction, and exponent subfields:



The integer subfield begins with a + or - sign, or a digit and may contain a string of digits terminated by a decimal point, an E , + , - , or the end of the input field.

The fraction subfield begins with a decimal point and may contain a string of digits terminated by an E , + , - , or the end of the input field.

The exponent field may begin with a sign or an E and contains a string of digits. When it begins with E , the + is optional between E and the string. The value of the string of digits should not exceed 38. The number may appear in any positions within the field; spaces in the field are ignored.

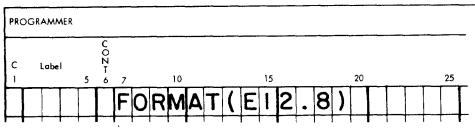
Examples:

```
+1.2345E2
123.456+9
-0.1234-6
.12345E-3
1234
+12345
+1234E6
```

When no decimal point is present in the input quantity, d acts as a negative power of ten scaling factor. The internal representation of the input quantity will be:

(integer subfield) $x10^{-d}$ $x10^{(\text{exponent subfield})}$

Example:



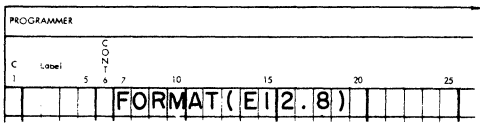
Input quantity = ^^^1234+5^^

Conversion performed: $1234x10^{-8}x10^{5}$

Result: 1.234

If a d value in the specification conflicts with the a decimal point appearing in an input field, the actual decimal point takes precedence.

Example:

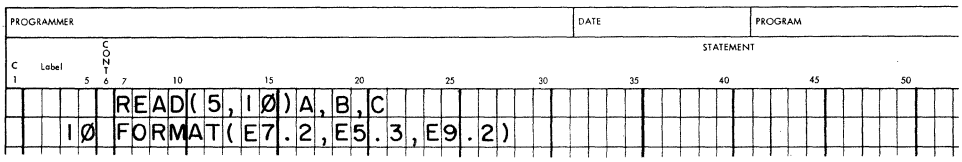


Input quantity = ^^^^^1.234+5

Quantity stored: $1.234x10^{5}$

The field width specified by w should always be the same as the width of the input field. When it is not, incorrect data may be read, converted and stored. The value of w should include positions for signs, the decimal point, the letter E , as well as the digits of the subfields:

Example:



Assuming input data in contiguous fields:

-12.3E1+1234123.46E-3
|← 7 →*← 5 →* 9 →|

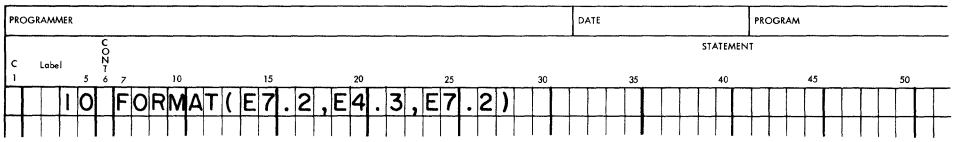
The fields read would be:

-12.3E1
+1234
123.46E-3

and converted as:

-123.
1.234
.12346

However, if specifications were:



The fields read would be:

-12.3E1
+123
4123.46

and converted as:

-123
.123
4123.46

The effects of possible FORMAT specification errors such as the above may not be detected by the system.

Examples:

FORMAT Specification	Input Field	Converted Value
E9.2	+1.2345E2	123.45
E9.4	-0.1234-6	-.0000001234
E4.2	1234	12.34

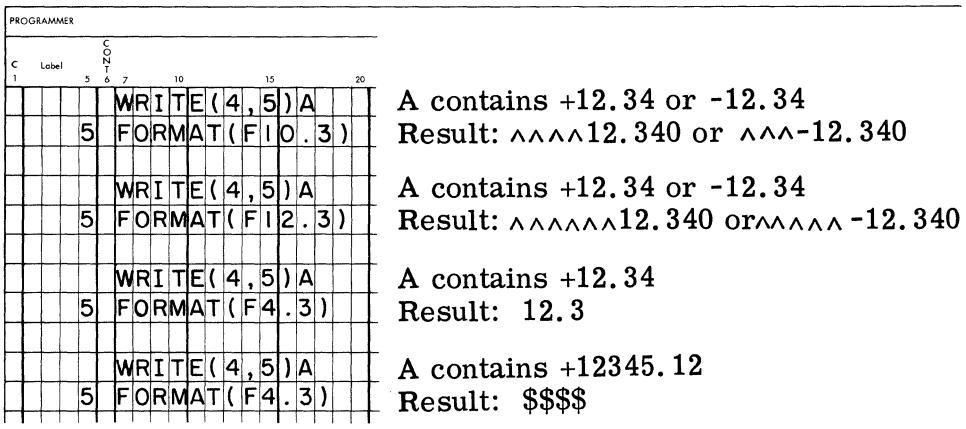
Fw.d Output

The F specification converts real numbers in storage to character form for output. The field occupies w positions and will appear as a decimal number, right justified in the field.

$\underline{\wedge}$x...x.x...x

The x's are the most significant digits. The number of decimal places to the right of the decimal point is specified by d . If d is zero, no digits appear to the right of the decimal point. The field must be wide enough to contain the significant digits, sign, and decimal point. If the number is positive, the + sign is suppressed. If the field is not long enough to contain the output value, an attempt is made to adjust the value of d (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs ($) are inserted in the entire field. If the field is longer than the output value, the number is right-justified with spaces occupying the excess positions on the left.

Examples:



Fw.d Input

The F specification input is identical to the E specification input. Although the fields are generally assumed to contain only a sign, integer, decimal point, and fraction; they may also contain an exponent subfield. All restrictions for Ew.d input apply.

Iw

The Iw specification converts internal values to output character strings, or input character strings to internal numbers. The output external field occupies w record positions and appears right justified (spaces on left) as:

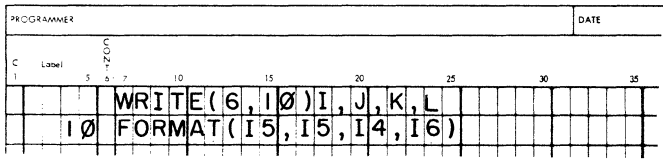
$$\pm x_1 \ldots x_d$$

During input conversion, if a value is less than -32768_{10}, the value is converted to a positive 32767.

The x's represent the decimal digits (maximum of 5) of the integer. When the integer is positive on output, the sign is suppressed. If an output field is too short, dollar signs ($) will be placed in the output record.

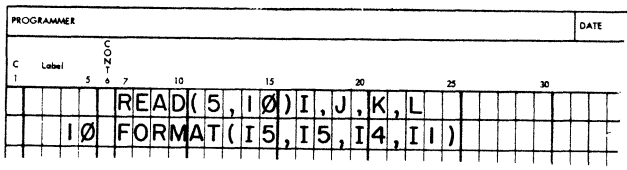
The Iw specification, when used for input, is identical to an Fw.0 specification.

Examples:



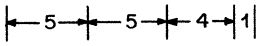
I contains -1234
J contains +12345
K contains +12345
L contains +12345

Result: -123412345$$$$∧12345



Input contains:

-∧12312∧∧3∧1∧23



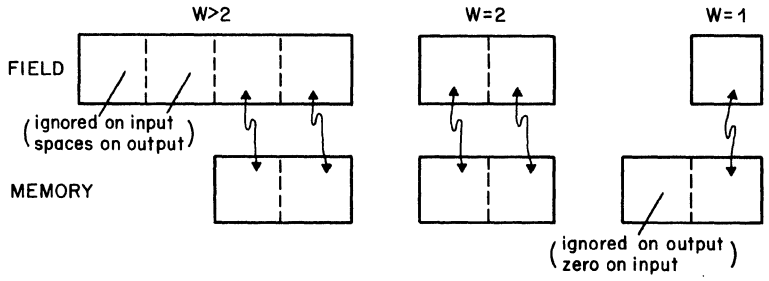
I contains -0123
J contains 12003
K contains 0102
L contains 3

Aw

This specification (not available in the 4K version of FORTRAN) causes alphanumeric data on an external medium to be translated to or from ASCII form in memory. The associated list element must be of type integer.

On input, if the field, as indicated by w, is greater than 2, the first w-2 characters are ignored; only the last two characters are read. When w equals 2, the two characters are read. If w equals 1, one character is read and stored in the right half of a computer word; zero is entered in the left half.

On output, if the field is greater than 2, two characters are written with right justification in the field; the leading positions are filled with spaces. If w equals 2, the two characters are written. If w equals 1, the character in the right half of the computer word is written.



Example:

```
      Input data: AZZ213-ABCXABC137 - ZZ9 (CR) (LF)

      DIMENSION ID (5)
      READ (5, 1Ø) I2, I1, ID
10    FORMAT (A1Ø, A1, 5A2)

      Result: I2 BC
              I1 ØX
              ID AB
                 C1
                 37
                 -Z
                 Z9
```

r @ w
r K w

Octal integer values are converted under either the @ or the K specification. The field is w octal digits in length; the corresponding list element must be of type integer. (Not available in the 4K version of FORTRAN.)

On input, if w is greater than or equal to 6, up to six octal digits are stored; non-octal digits appearing within the field are ignored. If the value of the octal digits within the field is greater than 177777, the results are unpredictable. If w is less than 6, or if less than six octal digits are encountered in the field, the number is right justified in the computer word with zero fill on the left.

On output, if the field is greater than 6, six octal digits are written with right justification in the field; the leading positions are filled with spaces. If w equals 6, the six octal digits are written. If w is less than 6, the w least significant octal digits are written.

Example:

```
      Input data: 123456-1234562342342342, 396E-Ø5  CR   LF

      DIMENSION ID(2), IE(2)
      READ (5, 10) IB, IC, ID, IE
10    FORMAT (@6, @7,  2@5, 2@4)
```

Result: IB 123456
IC 123456
ID Ø23423
Ø42342
IE ØØØØ36
ØØØØØ5

nX

The X specification may be used to include n blanks in an output record or to skip n characters on input to permit spacing of input/output quantities. In the specifications list, the comma following X is optional. ^X is interpreted as 1X. 0X is not permitted.

Examples:

```
PROGRAMMER                    DATE        PROGRAM
                                   STATEMENT
C  Label   CONT  1....5....10....15....20....25....30....35....40....45....50
           WRITE(6,1Ø)A,B,I
     1Ø    FORMAT(E8.3,5X,F6.2,5X,I4)
```

A contains +123.4
B contains -12.34
I contains -123

Result: ^.1234E2^^^^^-12.34^^^^^-123

Input:

WEIGHT^^10^^PRICE^^$1.98^^TOTAL^^$19.80

```
PROGRAMMER                    DATE        PROGRAM
                                   STATEMENT
C  Label   CONT  1....5....10....15....20....25....30....35....40....45....50
           READ(5,1Ø)I,A,B
     1Ø    FORMAT(8X,I2,1ØXF4.2,1ØXF5.2)
```

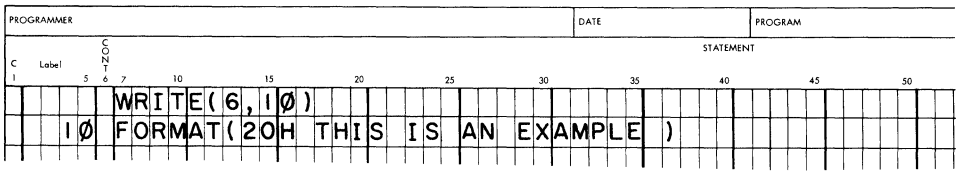
Result: I contains 10
A contains 1.98
B contains 19.80

nH$h_1h_2...h_n$

The H specification provides for the transfer of any combination of 8-bit ASCII characters, including blanks. n is an unsigned integer specifying the number of characters to the right of the H that are to be transmitted. The comma following the H specification is optional. ^H is interpreted as 1H. 0H is not permitted.

On output, the ASCII data in the FORMAT statement is written on the unit in the form of comments, titles, and headings.

Example:



Result: THIS IS AN EXAMPLE

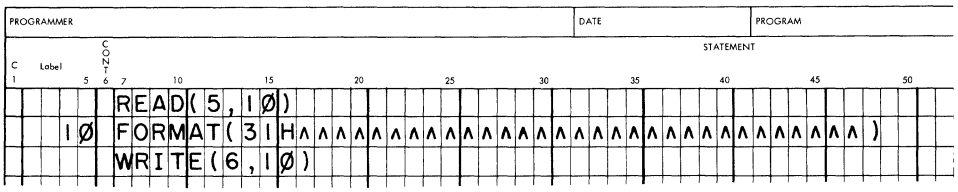
```
      WRITE(6,10)I,A,B
   10 FORMAT(8HWEIGHT  I2,10H  PRICE  $,F4.2,
     C10H  TOTAL  $,F5.2)
```

I contains 10
A contains 1.98
B contains 19.80

Result: WEIGHT 10 PRICE $1.98 TOTAL $19.80

On input, the data is transmitted from the unit to the FORMAT statement. A subsequent output statement transfers the new data to the output record.

Examples:



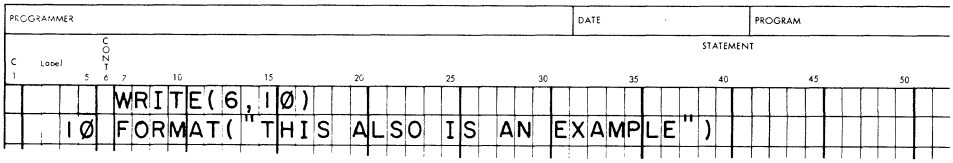
Input: H INPUT ALLOWS VARIABLE HEADERS

Result: H INPUT ALLOWS VARIABLE HEADERS

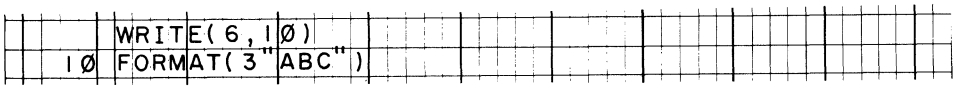
r"$h_1h_2...h_n$"

This specification also provides for the transfer of any combination of ASCII characters (except the quotation marks). The number of characters transmitted is the number of positions between the two quotation marks; field length is not specified. If r , an optional repeat count, is present, the character string within the quotation marks is repeated that number of times. Commas preceding the initial quotation mark and following the closing quotation are optional.

Examples:



Result: THIS ALSO IS AN EXAMPLE



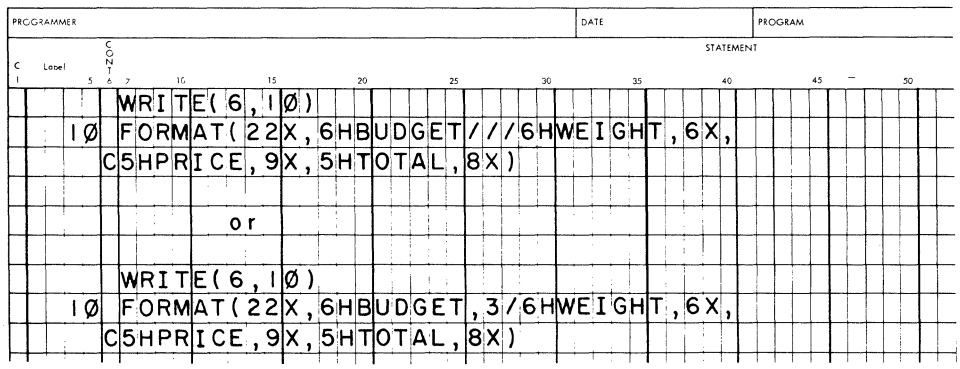
Result: ABCABCABC

On input, the number of characters within the quotation marks is skipped on the input field.

New Record

The slash, / , terminates the current record and signals the beginning of a new record of formatted data. It may occur anywhere in the specifications list and need not be separated from the other list elements by commas. Several records may be skipped by indicating consecutive slashes or by preceding the slash with a repetition factor; r-1 records are skipped for r/. On output the slash is used to skip lines, cards, or tape records; on input, it specifies that control passes to the next record or card.

Examples:



Result:

line 1 ^^^^^^^^ ^^^^^^^^^^^^ ^^^^ BUDGET

line 2

line 3

line 4 WEIGHT^^^^^^ PRICE ^^^^^^^^^^ TOTAL^^^^^^

Repeat Specifications

Repetition of the field descriptors (except nH) is accomplished by preceding the descriptor with a repeat count, r . If the input/output list warrants, the conversion is interpreted repetitively up to the specified number of times.

Repetition of a group of field descriptors, including nH is accomplished by enclosing the group in parentheses and preceding the left parenthesis with a group repeat count. If no group repeat count is specified, a value of one is assumed. Grouped field descriptors may be nested to a depth of one level.

Examples:

```
      WRITE(4,10)I,J,K
10    FORMAT(I5,I5,I5)
```

can be written as

```
      WRITE(4,10)I,J,K
10    FORMAT(3I5)

      WRITE(4,10)A,B,I,C,D,J
10    FORMAT(E8.3,5X,F6.2,5X,I4,E8.3,5X,
     CF6.2,5X,I4)
```

can be written as

```
      WRITE(4,10)A,B,I,C,D,J
10    FORMAT(2(E8.3,5X,F6.2,5X,I4))
```

A nested repetition specification would be:

```
      FORMAT(E8.3,5X,5(F6.2,5X,I4))
```

The group F6.2, 5X, I4 would be written five times, and the entire group, once.

Unlimited Groups

FORMAT specifications may be repeated without use of the repetition factor. If list elements remain after all specifications in a FORMAT statement are processed, the rightmost group of repeated (enclosed in parentheses) specifications is used. If there is no repeated group, processing resumes with the first specification in the statement. On output, each time the rightmost parenthesis in the statement, or in the unlimited group, is reached, the current record is terminated.

7.4 FREE FIELD INPUT

By following certain conventions in the preparation of the input data, a 2116A FORTRAN program may be written without use of FORMAT statements. Special symbols included with the ASCII input data items direct the formatting:

space or,	Data item delimiters
/	Record terminator
+ -	Sign of item
. E + -	Floating point number
@	Octal integer
"..."	Comments

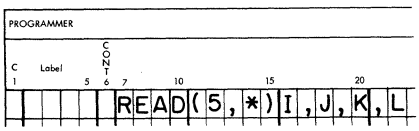
All other ASCII non-numeric characters are treated as spaces (and delimiters). Free field input may be used for numeric data only. Free field input is indicated in the FORTRAN READ statement by using an asterisk rather than a number of a FORMAT statement.

Data Item Delimiters

Any contiguous string of numeric and special formatting characters occurring between two commas, a comma and a space, or two spaces, is a data item whose value corresponds to a list element. 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; the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

Example:

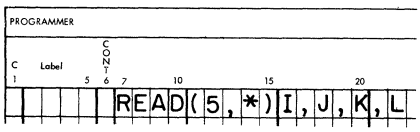
1)



Input data: 1720, 1966
1980 1492

Result: I contains 1720
J contains 1966
K contains 1980
L contains 1492

2)

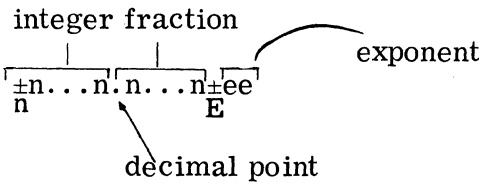


Input data: 1266,,1794,2000

Result: I contains 1266
J contains 1966
K contains 1794
L contains 2000

Floating Point Input

The symbols used to indicate a floating point data item are the same as those used in representing floating point data for FORMAT statement directed input:



If the decimal point is not present, it is assumed to follow the last digit.

Examples:

```
PROGRAMMER                                  DATE        PROGRAM
                                                STATEMENT
C  Label  CONT
1        5  6  7        10        15        20        25        30        35        40        45        50
           READ(5,*)A,B,C,D,E
```

Input Data: 3.14, 314E-2, 3140-3, .0314+2, .314E1

All are equivalent to 3.14

Octal Input

An octal input item has the following format:

$$@\, x_1 \cdots x_d$$

The symbol @ defines an octal integer. The x's are octal digits each in the range of 0 through 7. List elements corresponding to the octal data items must be type integer.

Record Terminator

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

Example:

```
PROGRAMMER                                  DATE        PROGRAM
                                                STATEMENT
C  Label  CONT
1        5  6  7        10        15        20        25        30        35        40        45        50
           READ(5,*)I,I,J,J,K,K,L,L,M,M
```

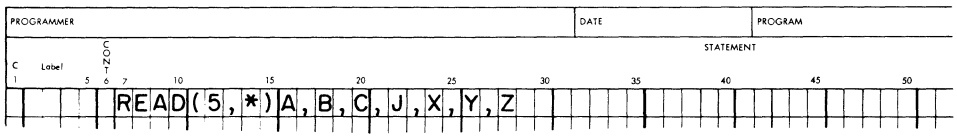
Input data: 987, 654, 321, 123/DESCENDING (CR) (LF)
456

Result: II contains 987
JJ contains 654
KK contains 321
LL contains 123
MM contains 456

List Terminator

If a line terminates (with a (CR) (LF)) and a slash has not been encountered, the input operation terminates even though all list elements may not have been processed. The current values of remaining elements are unchanged.

Examples:



Input Data:

A=7.987 B=5E2 C=4.6859E-3 (CR) (LF)
J=3456 (CR) (LF)

Result: A contains 7.987
B contains 5E2
C contains 4.6859E-3

J, X, Y, Z are unchanged.

Comments

All characters appearing between a pair of quotation marks in the same line are considered to be comments and are ignored.

Examples:

"6.7321"	is a comment and ignored
6.7321	is a real number

INPUT/OUTPUT STATEMENTS 8

Input/output statements transfer information between memory and an external unit. The unit is specified as an integer variable that is defined elsewhere in the program or an integer constant.

Each statement may include a list of names of variables, arrays, and array elements. The named elements are assigned values on input and have their values transferred on output.

Records may be formatted or unformatted. A formatted record consists of a string of ASCII characters. The transfer of such a record requires the specification of a FORMAT statement or free field input data. An unformatted record consists of a string of binary values.

8.1 UNIT-REFERENCE

The integer specified for an input/output unit is a number which represents a Standard unit assignment or an installation unit assignment. The physical device referenced depends on tables established within the Basic Control System.

The Standard unit numbers are as follows:

Number	Name	Usual Equipment Type
1	Keyboard Input	2752A Teleprinter†
2	Teleprinter Output	2752A Teleprinter
3	Program Library	2737A Punched Tape Reader
4	Punch Output	2753A Tape Punch
5	Input	2737A Punched Tape Reader
6	List Output	2752A Teleprinter

†If data is to be printed on the Teleprinter as it is read, Unit-Reference number 1 must be used; printing occurs with no other number.

Installation unit numbers may be in the range $7\text{-}74_8$ with the largest value being determined by the number of units of equipment available at the installation. Each Standard unit may be a separate device, or a single device may be accessed by several Standard unit numbers as well as an installation unit number.†

8.2 FORMATTED READ, WRITE

A formatted READ statement is one of the forms:

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

Execution of this statement causes the input of the next ASCII records from unit u. The information is scanned and converted according to the FORMAT specification statement, f, and assigned to the elements of list k. If the input is free field, an asterisk is specified in the READ statement rather than the label of a FORMAT statement. If the list is absent, the FORMAT statement should contain editing specifications only.

A formatted WRITE statement may have one of the following forms:

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

This statement transfers ASCII information from locations given by names in the list k to output unit u. The values are converted and positioned as specified by the FORMAT statement f. If the list is absent, the FORMAT statement should contain editing specifications only.

†For complete details, see Basic Control System Programmer's Reference Manual.

8.3 UNFORMATTED READ, WRITE

An unformatted READ statement has one of the forms:

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

This statement transfers the next binary input record from the unit u to the elements of list k. The sequence of values required by the list may not exceed the sequence of values from the record. If no list is specified, READ (u) skips the next record.

An unformatted WRITE statement has the form:

```
WRITE (u)k
```

Execution of this statement creates the next record on unit u from the sequence of values represented by the list k.

8.4 AUXILIARY INPUT/OUTPUT STATEMENTS

There are three types of auxiliary input/output statements:

```
REWIND
BACKSPACE
ENDFILE
```

A REWIND statement has the form:

```
REWIND u
```

This statement causes the unit u to be positioned at its initial point. If the unit is currently at this position, the statement acts as a CONTINUE.

A BACKSPACE statement is as follows:

```
BACKSPACE u
```

BACKSPACE positions the unit u so that what had been the preceding record becomes the next record. If the unit is currently at its initial point, the statement acts as a CONTINUE.

An ENDFILE statement is of the form:

```
ENDFILE u
```

Execution of this statement causes the recording of an end-of-file record on the output unit u. If given for an input unit, the statement acts as a CONTINUE.

In addition to the three auxiliary input output statements, a subroutine may be called to perform file and record spacing on magnetic tape. The subroutine call is as follows:

```
CALL PTAPTE (u, f, r)
```

u Unit-Reference number of tape device

f File spacing:

A positive integer specifying the number of files to be spaced forward.

A negative integer specifying the number of files to be backspaced.

r Record spacing:

A positive integer specifying the number of records to be spaced forward.

A negative integer specifying the number of records to be backspaced.

Both file and record spacing may be specified in the same call (e.g., space forward 5 files, then backward 2 records). If file or record spacing is not to be performed, a zero is supplied as the parameter.

If backspacing would result in spacing beyond the Start-of-Tape mark, the spacing operation is terminated and program execution resumes. If forward spacing results in spacing beyond the End-of-Tape marker, the message "*EOT" is printed on the Standard Teleprinter Output unit. When the operator presses RUN, program execution resumes.

COMPILER INPUT AND OUTPUT 9

The FORTRAN Compiler accepts as input, paper tape containing a control statement and a source language program. The output produced by the Compiler may include a punched paper tape containing the object program; a listing of the source language program with diagnostic messages, if any; and a listing of the object program in assembly level language.

9.1 CONTROL STATEMENT

The control statement must be the first statement of the source program; it directs the compiler.

FTN, p_1, p_2, p_3

FTN is a free field control statement. Following the comma are one to three parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, (CR) (LF). Spaces embedded in the statement are ignored.

The parameters may be a combination of the following:

B Binary output: A program is to be punched in relocatable binary format suitable for loading by the Basic Control System loader.

L List output: A listing of the source language program is to be produced during Pass One.

A Assembly listing: A listing of the object program in assembly level language is to be produced in the last pass.

T Symbol table only: A listing of the symbol table only is produced; if both T and A are specified, only the last used will be decisive.

9.2 SOURCE PROGRAM

The source program follows the control statement. Each statement is followed by the end-of-statement mark, (CR) (LF). Specifications statements must precede executable statements. The last statement in each program submitted for compilation must be an END statement. Up to five source programs may be compiled at one time. The last program must be followed by and END$ statement, if less than six programs are to be compiled.

The control statement, each of the five programs, and the END$ terminator may be submitted on a single tape or on separate tapes. If more than five programs are contained on a tape, the compiler processes the first five and halts with the T-Register containing 102077 (end of Pass 1). The remaining programs must be compiled separately.

9.3 BINARY OUTPUT

The punch output produced by the compiler is a relocatable binary program. It does not include system subroutines introduced by the compiler, or library subroutines referred to in the program.

9.4 LIST OUTPUT

If the List Output parameter is specified, the first 72 characters of each line of the source program is printed on the List Output device. The END$ is the last statement printed. If exactly five programs are compiled, however, the END$ is omitted from the list.

If the Assembly listing parameter is specified, the program is printed in assembly level language on the List Output device. The program listing is followed by a Symbol Table for the assembly level program.

The format for the assembly level listing is as follows:

Columns	Content
1-5	Zero-relative location (octal) of the instruction
6-7	Blank

Columns	Content
8-13	Object code word in octal
14	Relocation or external symbol indicator
15	Blank
16-18	Mnemonic operation code
19	Blank
20-25	Operand address in octal or external symbol name.
26-27	The indicator ",I" if indirect addressing is used.

The Symbol Table listing has the following format:

Columns	Content
1-5	Symbol, statement label, or numeric symbol assigned by the compiler.
6	Blank
7	Relocation indicator
8	Blank
9-14	The zero-relative value of the symbol

The characters that designate an external symbol or type of relocation for the operand address or a symbol in the Symbol Table are:

Character	Relocation Base
Blank	Absolute
R	Program relocatable
X	External symbol
C	Common relocatable

9.5 OPERATING INSTRUCTIONS

The exact operating procedures for a compilation depend on the available hardware configuration.

One possible allocation of equipment might be as follows:

Compiler Input/Output	Standard Unit Designation	Physical Unit Assignment
Binary output	Punch Output	Tape Punch
List output	List Output	Teleprinter, Line Printer
Control Statement	Input	Teleprinter
Source tape(s)	Input	Punched Tape Reader, Marked Sense Cards

If there are two output devices as shown above, there are two passes (8K memory) or four passes (4K memory). The list output and an intermediate binary tape are both produced during the first pass; the assembly listing and the binary output are both produced during the last pass.

If one output device is available list output and intermediate binary output are written on the same tape during the first pass (the Compiler ignores the list output when reading the binary data during the second pass). The binary output is then produced in the next to the last pass; and the assembly listing, in the last pass.

The compiler determines whether a magnetic tape unit is available by checking location 107. See the Magnetic Tape System manual for operating procedures in a magnetic tape environment (non-4K only).

The following procedures indicate the sequence of steps for compilation of a source program on paper tape:

A. Set Teleprinter to LINE and check that all equipment to be used is operable. If the Teleprinter is the only output device, turn ON punch unit.

B. Load FORTRAN Pass 1 using the Basic Binary Loader:

   1. Place FORTRAN binary tape in the device serving as the Standard Input unit (e.g., Punched Tape Reader).

   2. Set Switch Register to starting address of Basic Binary Loader.

3. Press LOAD ADDRESS.

4. Set Loader switch to ENABLED.

5. Press PRESET.

6. Press RUN.

7. When the computer halts and indicates that the FORTRAN tape is loaded (T-Register contains 102077), set Loader switch to PROTECTED.

C. If the System Input/Output (SIO) subroutines are on a tape which is separate from FORTRAN Pass 1, load the tape using the Basic Binary Loader as in Step B.

D. Set Switch Register to starting address of FORTRAN Pass 1: 000100 (non-4K has optional starting address of 50 to enter control statement through teleprinter).

E. Press LOAD ADDRESS

F. Place source language tape in device serving as the Standard Input unit (e.g., Punched Tape Reader).

G. Press RUN.

H. If more than one source tape (halt with T-register = 102057), repeat Steps F and G for each tape (102002 in 4K).

I. Perform either of the following depending on memory size:

4K Memory

1. At end of Pass 1 (T-Register contains 102077) load Pass 2 using the Basic Binary Loader as in Step B.

2. Remove binary output from Standard Punch device and place in device serving as the Standard Input unit. (If only one output device, both binary and list output are on the same tape.)

3. Set Switch Register to: 000100

4. Press LOAD ADDRESS

5. Press RUN

6. At end of Pass 2 (T-Register contains 102077), load Pass 3 using the Basic Binary Loader as in Step B.

7. Remove binary output from Standard Punch device and place in device serving as Standard Input unit.

8. Set Switch Register to: 000100

9. Press LOAD ADDRESS.

10. Press RUN.

11. At end of Pass 3 (T-Register contains 102077), load Pass 4 using the Basic Binary Loader as in Step B.

12. Remove binary output from Standard Punch device and place in device serving as Standard Input unit.

13. Set Switch Register to: 000100.

14. Press LOAD ADDRESS.

15. Press RUN.

16. At end of Pass 4, the relocatable binary object tape is on the Standard Punch unit. Either of the following conditions may exist:

    a. If the T-Register contains 102077, the compilation is complete. If an assembly listing was requested, it is on the List Output Device.

    b. If the T-Register contains 102001, an assembly listing pass is to be performed:

        (1) Place binary output from Pass 3 in device serving as Standard Input unit. (Turn off Teleprinter punch unit.)

        (2) Press RUN.

        (3) At end of listing pass, T-Register contains 102077.

8K Memory

1. At end of Pass 1 (T-Register contains 102077), load Pass 2 using the Basic Binary Loader as in Step B.

2. Remove binary output from Standard Punch device and place in device serving as the Standard Input Unit. (If only one output device, both binary and list output are on the same tape.)

3. Set Switch Register to: 000100

4. Press LOAD ADDRESS.

5. Press RUN.

6. At end of Pass 2, the relocatable binary object tape is on the Standard Punch unit. Either of the following conditions may exist:

   a. If the T-Register contains 102077, the compilation is complete. If an assembly listing was requested, it is on the List Output device.

   b. If the T-Register contains 102001, an assembly listing pass is to be performed:

      (1) Place binary output from Pass 1 in device serving as Standard Input unit. (Turn off Teleprinter punch unit.)

      (2) Press RUN.

      (3) At end of listing pass, T-Register contains 102077.

J. The Basic Control System Loader is used to load the object programs generated by FORTRAN and any referenced library routines. Listed below is a summary of procedures for normal loading of relocatable object programs and library routines (and for the printing of a Memory Allocation Listing): †

† See Section 9.8 for details and options.

1. Load the Basic Control System tape using the Basic Binary Loader as in Step B.

2. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.

3. Place FORTRAN or Assembler generated relocatable object tape in device serving as Standard Input unit.

4. Press RUN. The loader types "LOAD" when the tape is loaded.

5. If more than one relocatable object tape is to be loaded, repeat Steps J3 and J4 for each. Otherwise, set Switch Register to 000004 to load library routines.

6. Place FORTRAN library tape in device serving as Program Library unit.

7. Press RUN. When the loading operation is complete, the Loader types "*LST". Press RUN to print Loader Symbol Table. When the Loader types "*RUN", the program is ready for execution.

8. Press RUN to initiate execution.

During the operation of the Compiler, the following halts may occur:

T-Register	Explanation	Action
102000	Memory overflow: the program is too large; has too many symbols	Irrecoverable error; program must be revised.
102001	End of binary object tape output, start of assembly listing.	If only one output device, place intermediate binary output from previous pass in Standard Input unit and press RUN.
102002	End of source tape. (4K compiler)	Place next input tape in reader.

T-Register	Explanation	Action
102007	For all passes except first, unrecognizable record on intermediate binary tape: 1) Punch error on previous pass. 2) Wrong tape supplied as input for pass.	If punch error, restart with Pass 1. If wrong tape, restart current pass: a) Load FORTRAN pass. b) Set Switch Register to 000100. c) Press LOAD ADDRESS. d) Place previous intermediate binary tape in input device. e) Press RUN.
102010	External symbol table overflow: the number of symbols exceeds 255.	Irrecoverable error; program must be revised.
102011	Checksum error on intermediate tape; indicates probable punch error.	Attempt to re-read record (binary records are separated by 4 feed frames). Otherwise, restart with Pass 1.
	If a magnetic tape is used for intermediate, indicates MT parity error or write not enabled	(irrecoverable).
102027	A Magnetic Tape Read error has occurred during Pass 2.	Restart.
102057	End of source tape.	Place next input tape in reader.

T-Register	Explanation	Action
102066	Tape supply low on 2753A Tape Punch.	Load new tape and press RUN.
102077	Normal end of pass or compilation.	Proceed as indicated in above steps.

For diagnostic messages that might occur during loading see the Basic Control System Programmer's Reference Manual. Diagnostics are also issued by the input/output system provided by FORTRAN and by the FORTRAN library routines (Section 9.9).

9.5.1 SWITCH REGISTER BITS

If bit 0 is set during pass 2 (or pass 4 in 4K), the compiler suppresses leader and trailer on punch out.

9.6 DIAGNOSTIC MESSAGES

Errors detected in the source program are indicated by a numeric code inserted before or after the statement in the List Output.

The format is as follows:

E-eeee:	ssss + nnnn
eeee	The error diagnostic code shown below.
ssss	The statement label of the statement in which the error was detected. If unlabeled, 0000 is typed.
nnnn	Ordinal number of the erroneous statement following the last labeled statement. (Comment statements are not included in this count.)

Error Code	Description
0001	Statement label error: a) The label is in positions other than 1-5. b) A character in the label is not numeric. c) The label is not in the range 1-9999. d) The label is doubly defined. e) The label indicated is used in a GO TO, DO, or IF statement or in an I/O operation to name a FORMAT statement, but it does not appear in the label field for any statement in the program (printed after END).
0002	Unrecognized Statement: a) The statement being processed is not recognized as a valid statement. b) A specifications statement follows an executable statement. c) The specification statements are not in the following order: DIMENSION COMMON EQUIVALENCE d) A statement function precedes a specification statement.

Error Code	Description
0003	Parenthesis error: There are an unequal number of left and right parentheses in a statement.
0004	Illegal character or format: a) A statement contains a character other than A through Z, 0 through 9, or space =+-/(),.$'. b) A statement does not have the proper format. c) A control statement is missing, misspelled, or does not have the proper format. d) An indexing parameter of a DO-loop is not an unsigned integer constant or simple integer variable or is specified as zero.
0005	Adjacent operators: An arithmetic expression contains adjacent arithmetic operators.
0006	Illegal subscript: A variable name is used both as a simple variable and a subscripted variable.
0007	Doubly defined variable: a) A variable name appears more than once in a COMMON statement. b) A variable name appears more than once in a DIMENSION statement. c) A variable name appears more than once as a dummy argument in a statement function. d) A program, subroutine, or function name appears as a dummy parameter; in a specifications statement of the subroutine or function; or as a simple variable in a program or subroutine.
0008	Invalid parameter list: a) The dummy parameter list for a subroutine or function exceeds 63. b) Duplicate parameters appear in a statement function.
0009	Invalid arithmetic expression: a) Missing operator b) Illegal replacement
0010	Mixed mode expression: integer constants or variables appear in an arithmetic expression with real constants or variables.

Error Code	Description
0011	Invalid subscript: a) Subscript is not an integer constant, integer variable, or legal subscript expression. b) There are more than two subscripts (i.e., more than two dimensions. c) Two subscripts appear for a variable which has been defined with one dimension only.
0012	Invalid constant: a) An integer constant is not in the range of -2^{15} to $2^{15}-1$. b) A real constant is not in the approximate range of 10^{38} to 10^{-38}. c) A constant contains an illegal character.
0013	Invalid EQUIVALENCE statement: a) Two or more of the variables appearing in an EQUIVALENCE statement are also defined in the COMMON block. b) The variables contained in an EQUIVALENCE cause the origin of COMMON to be altered. c) Contradictory equivalence; or equivalence between two or more arrays conflicts with a previously established equivalence.
0014	Table overflow: Too many variables and statement labels appear in the program.
0015	Invalid DO loop: a) The terminal statement of a DO loop does not appear in the program or appears prior to the DO statement. b) The terminal statement of a nested DO loop is not within the range of the outer DO loop. c) DO loops are nested more than 10 deep. d) Last statement in a loop is a GO TO, arithmetic IF, RETURN, STOP, PAUSE, or DO.

Error Code	Description
0016	Statement function name is doubly defined.

9.7 OBJECT PROGRAM LOADING

If absolute binary output was specified, the Basic Binary Loader is used to load the object program tape.

If relocatable binary output was specified, the BCS Relocating Loader is used to load the object program tape. If the program refers to other Assembler or FORTRAN generated object programs, these tapes are loaded by the Relocating Loader at the same time. In general, the FORTRAN Library tape must be submitted for loading also.

Listed below are summaries of procedures for normal loading of object programs:

BASIC BINARY LOADER
OPERATING PROCEDURES SUMMARY

A. Place binary object tape in Standard Input unit.

B. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).

C. Press LOAD ADDRESS.

D. Set LOADER switch to ENABLED.

E. Press PRESET.

F. Press RUN.

G. When the computer halts with T-Register containing 102077, set LOADER switch to PROTECTED.

H. Set Switch Register to starting address of object program.

I. Press LOAD ADDRESS.

J. Check that all I/O devices are ready and loaded for operation of the program.

K. Press RUN.

BASIC CONTROL SYSTEM LOADER OPERATING PROCEDURES SUMMARY

A. Load the Basic Control System tape using the Basic Binary Loader.

B. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.

C. Place Assembler (or FORTRAN) generated relocatable object tape in Standard Input unit.

D. Press RUN. The loader types "LOAD" if it expects another relocatable or library program.

E. If more than one relocatable object tape is to be loaded, repeat Steps C and D for each. Otherwise, set Switch Register to 000004 to load library routines.

F. Place FORTRAN Library tape in device serving as Program Library unit.

G. Press RUN. When the loading operation is complete, the Loader types "*LST". Press RUN. The Loader types "*RUN" indicating the program is ready for execution. (See the Basic Control System Programmer's Reference manual for error message.)

H. Press RUN to initiate execution.

9.8 OBJECT PROGRAM DIAGNOSTIC MESSAGES

During execution of the object program, diagnostic messages may be printed on the Teleprinter 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:

Teleprinter Message	A-Register	Explanation	Action
*EOR	Unit Number	Equipment Error: End of input tape on 2752A Teleprinter or 2737A Punched Tape Reader; tape supply low on 2753A Tape Punch. B-Register contains status word of Equipment Table entry.	Place next tape in input device, or for Tape Punch, load new reel of tape. Press RUN.
*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.

Teleprinter Message	A-Register	Explanation	Action
*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 E's, two decimal points, two signs, etc.)	Verify data.

During the execution of an object program referring to the FORTRAN library routines, the following errors codes may be printed on the Teleprinter Output unit when error conditions are encountered by the specified subroutine:†

Error Code	Subroutine	Condition
02 UN	ALOG	$a \leq 0$
03 UN	SQRT	$a < 0$
04 UN	.RTOR	$x = 0, y \leq 0$
		$x < 0, y \neq 0$
05 OR	SIN, COS	$\|a\| > 2^{14}$
06 UN	.RTOI	$x = 0, i \leq 0$
07 OF	EXP	$\|a\| * \log_2 e \geq 124$
08 OF	.ITOI	i^j out of range
08 UN	.ITOI	$i = 0, j \leq 0$
09 OR	TAN	$\|a\| > 2^{14}$

UN = Floating point underflow

OF = Integer or floating point overflow

OR = Out of range

† For complete details, see FORTRAN Library Routines manual.

ASC II CHARACTER FORMAT

b_7				0	0	0	0	1	1	1	1
b_6				0	0	1	1	0	0	1	1
b_5				0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NULL	DC_0	b	0	@	P		
0	0	0	1	SOM	DC_1	!	1	A	Q		
0	0	1	0	EOA	DC_2	"	2	B	R		U
0	0	1	1	EOM	DC_3	#	3	C	S		N
0	1	0	0	EOT	DC_4 (STOP)	$	4	D	T		A
0	1	0	1	WRU	ERR	%	5	E	U	UNASSIGNED	S
0	1	1	0	RU	SYNC	&	6	F	V		I
0	1	1	1	BELL	LEM	(APOS)	7	G	W		G
1	0	0	0	FE_0	S_0	(	8	H	X		N
1	0	0	1	HT / SK	S_1	)	9	I	Y		E
1	0	1	0	LF	S_2	*	:	J	Z		D
1	0	1	1	V_{TAB}	S_3	+	;	K	[		
1	1	0	0	FF	S_4	(COMMA)	<	L	\		ACK
1	1	0	1	CR	S_5	-	=	M	]		①
1	1	1	0	SO	S_6	.	>	N	↑		ESC
1	1	1	1	SI	S_7	/	?	O	←		DEL

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

EXAMPLE: The code for "R" is: b_7 b_6 b_5 b_4 b_3 b_2 b_1 = **1 0 1 0 0 1 0**

LEGEND

NULL	Null/Idle
SOM	Start of message
EOA	End of address
EOM	End of message
EOT	End of transmission
WRU	"Who are you?"
RU	"Are you...?"
BELL	Audible signal
FE_0	Format effector
HT	Horizontal tabulation
SK	Skip (punched card)
LF	Line feed
V_{TAB}	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DC_0	Device control reserved for data link escape
DC_1-DC_3	Device Control
DC_4 (Stop)	Device control (stop)
ERR	Error
SYNC	Synchronous idle
LEM	Logical end of media
S_0-S_7	Separator (information)
b	Word separator (space, normally non-printing)
<	Less than
>	Greater than
↑	Up arrow (Exponentiation)
←	Left arrow (Implies/Replaced by)
\	Reverse slant
ACK	Acknowledge
①	Unassigned control
ESC	Escape
DEL	Delete/Idle

FORTRAN STATEMENTS AND FUNCTIONS B

Functions

Function Name	Definition	Symbolic Name	No. of Arguments	Type of	
				Argument	Function
Absolute Value	$\|a\|$	ABS	1	Real	Real
		IABS	1	Integer	Integer
Float	Conversion from integer to real	FLOAT	1	Integer	Real
Fix	Conversion from real to integer	IFIX	1	Real	Integer
Transfer sign	Sign of a_2 times $\|a_1\|$	SIGN	2	Real	Real
		ISIGN	2	Integer	Integer
Exponential	e^a	EXP	1	Real	Real
Natural Logarithm	$\log_e(a)$	ALOG	1	Real	Real
Trigonometric Sine	sine (a)†	SIN	1	Real	Real
Trigonometric Cosine	cos (a)†	COS	1	Real	Real
Trigonometric Tangent	tan (a)†	TAN	1	Real	Real
Hyperbolic Tangent	tanh (a)	TANH	1	Real	Real
Square Root	$(a)^{1/2}$	SQRT	1	Real	Real
Arctangent	arctan (a)	ATAN	1	Real	Real
And (Boolean)	$a_1 \wedge a_2$	IAND	2	Integer	Integer
Or (Boolean)	$a_1 \vee a_2$	IOR	2	Integer	Integer
Not (Boolean)	$\neg a$	NOT	1	Integer	Integer
Sense Switch	Sense Switch Register Switch (n)	ISSW	1	Integer	Integer

† a is in radians

ASSEMBLY LANGUAGE SUBPROGRAMS C

A FORTRAN program can refer to a subprogram that has been prepared using Assembler source language. The subprogram may be treated as a subroutine or as a function. The object code programs generated by FORTRAN and by the Assembler are then linked together by the Basic Control System Relocating Loader when the programs are loaded.

FORTRAN REFERENCE

In the FORTRAN program, a subroutine is called using the following statement:

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

The symbolic name, s, identifies the subroutine and the a's are the actual arguments.

If the subprogram is a function, it is referenced by using the name and the actual arguments in an arithmetic expression:

f(a_1, a_2, . . . , a_n)

As a result of either the call or the reference, FORTRAN generates the following coding sequence:

JSB s/f	Transfers control to subroutine or function
DEF*+n+1	Defines return location
DEF a_1	Defines address of a_1
DEF a_2	Defines address of a_2
⋮	
DEF a_n	Defines address of a_n

The words defining the addresses of the arguments may be direct or indirect depending on the actual arguments. For example, an integer constant as an actual argument would yield a direct reference; an integer variable might yield an indirect reference.

If the subprogram being referenced is a subroutine, it may return none, one, or more than one value through its arguments or through common storage. If the subprogram is a function, it is assumed to return a single value in the accumulators: a function of type integer returns a value in the A-Register; a function of type real returns a value in the A- and B-Registers.

The subprogram may transfer values directly by accessing the words in the calling sequence or it may make use of the FORTRAN library subroutine .ENTR to aid in the transfer.

DIRECT TRANSFER OF VALUES

Any suitable technique may be used to obtain or deliver values for the arguments and to return control to the calling program. If address arithmetic is used in conjunction with an argument (e.g., to process elements of an array), the base location must be a direct reference; the location given in the calling sequence must be checked to determine if it is a direct or indirect reference. If it is an indirect reference the location to which it points must also be checked, and so forth.

Example:

PROGRAMMER | DATE | PROGRAM

STATEMENT

Label	Operation	Operand	Comments
	NAM	AMSUB	
	ENT	AMSUB	
AMSUB	NOP		AMSUB TO CONTAIN ADDR OF "*+N+I"
	LDA	AMSUB,I	A CONTAINS VALUE OF "*+N+I".
	STA	RETRN	RETRN CONTAINS VALUE OF "*+N+I".
NXTAG	ISZ	AMSUB	AMSUB CONTAINS ADDR OF LOCATION
	LDA	AMSUB	OF ARGUMENT. TEST IF ALL ARGU-
	CPA	RETRN	MENTS PROCESSED: COMPARE VALUE
	JMP	RETRN,I	OF "*+N+I" WITH ADDR OF CURRENT
			LOCATION OF ARGUMENT. IF EQUAL
PRSAG			RETURN TO CALLING PROGRAM, IF NOT,
	.		PROCESS ARGUMENT AS REQUIRED.
	.		
	.		
	LDA	AMSUB,I	A CONTAINS LOCATION OF ARGUMENT.
	LDA	Ø,I	LOAD ONE-WORD (FIXED POINT)
	.		VALUE INTO A.
	.		
	.		
	LDA	AMSUB,I	LOAD TWO-WORD (FLOATING POINT)

```
        DLD   Ø,I          VALUE INTO A AND B.
          .
          .
          .
        LDA   AMSUB,I      STORE ONE-WORD VALUE IN ARGUMENT
        STA   OUTAD        LOCATION.
        LDA   W1VAL
        STA   OUTAD,I
          .
          .
          .
        LDA   AMSUB,I      STORE TWO-WORD IN ARGUMENT
        STA   OUTAD        LOCATIONS.
        DLD   W2VAL
        DST   OUTAD,I
          .
          .
          .
        LDA   AMSUB,I      A CONTAINS ADDR OF LOCATION OF
        SSA                ARGUMENT. TO DETERMINE IF REF IS
        JMP   *+2          INDIRECT, TEST BIT 15. IF ONE,
        JMP   *+5          SET TO ZERO WITH AND, THEN LOAD
        AND   ANMSK        A WITH REFERENCED LOCATION.
        LDA   Ø,I          REPEAT TEST WITH NEXT REF. WHEN
        JMP   *-5          DIRECT REF ENCOUNTERED, PROCEED
ANMSK   OCT   Ø77777       WITH PROCESSING.
          .
          .
          .
        JMP   NXTAG        RETURN THROUGH HERE WHEN NEXT
RETRN   BSS   1            ARGUMENT IS REQUIRED.
OUTAD   BSS   1
W1VAL   BSS   1
W2VAL   BSS   2
        END
```

The preceding example assumes that each argument is processed or partially processed before the next is obtained or delivered. Control returns to the calling program when all arguments have been picked up or delivered.

TRANSFER VIA .ENTR

The transfer of values to or from the locations listed in the calling sequence may be facilitated through use of the FORTRAN library subroutine .ENTR. This subroutine moves the addresses of the arguments into an area reserved within the Assembly language subroutine. The addresses stored in the reserved area are all direct references; .ENTR performs all the necessary direct/indirect testing, etc. It also sets the correct return address in the entry point location.

The general form of the subroutine is:

```
        NAM s          The subroutine name is s.
        ENT s
        EXT .ENTR      .ENTR must be declared as external.
a       BSS n          Reserves n words of storage for the
s       NOP            addresses of the arguments; this pseudo
                       instruction must directly precede the
                       entry point location, s.

        JSB .ENTR
        DEF a          Defines first location of area used to
  (First instruction)  store argument addresses.
            .
            .
            .
        JMP s, I
        END
```

Example:

PROGRAMMER | DATE | PROGRAM

STATEMENT

Label	Operation	Operand	Comments
	NAM	AMSUB	
	ENT	AMSUB	
	EXT	.ENTR	
AGMTS	BSS	5	
AMSUB	NOP		
	JSB	.ENTR	
	DEF	AGMTS	
PRSAG			PROCESS ARGUMENTS AS REQUIRED
		.	
		.	
		.	
	LDA	AGMTS,I	PICK UP VALUE OF FIRST ARGUMENT
		.	
		.	
		.	
	DLD	AGMTS+1,I	PICK UP VALUE OF SECOND ARGUMENT
		.	
		.	
		.	
	LDA	W1VAL	STORE VALUE FOR THIRD ARGUMENT
	STA	AGMTS+2,I	
		.	
		.	
		.	
	DLD	W2VAL	STORE VALUE FOR FOURTH ARGUMENT
	DST	AGMTS+3,I	
		.	
		.	
		.	
	LDA	AGMTS+4	PICK UP ADDRESS OF FIFTH ARGUMENT
		.	
		.	
		.	
	JMP	AMSUB,I	RETURN TO CALLING PROGRAM
W1VAL	BSS	1	
W2VAL	BSS	2	
	END		

SAMPLE PROGRAM D

Using Simpson's rule, calculate the value of the integral:

$$\int_a^b \frac{\cos x}{x}\,dx$$

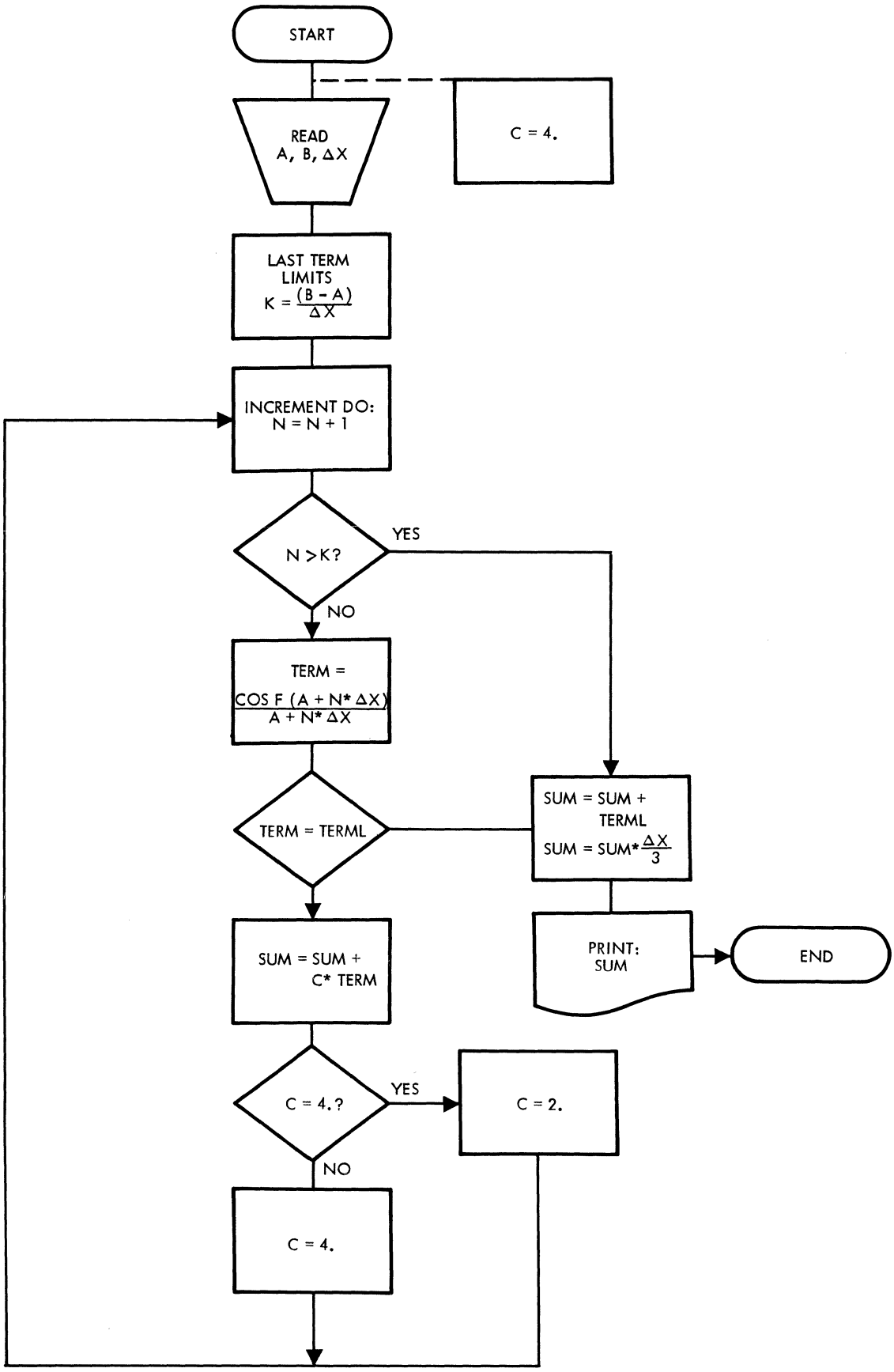
for the following possible values:

Variable	Range of Values
a	-6.99 to +6.99
b	-6.99 to +6.99
Δx	-.25 to +.25

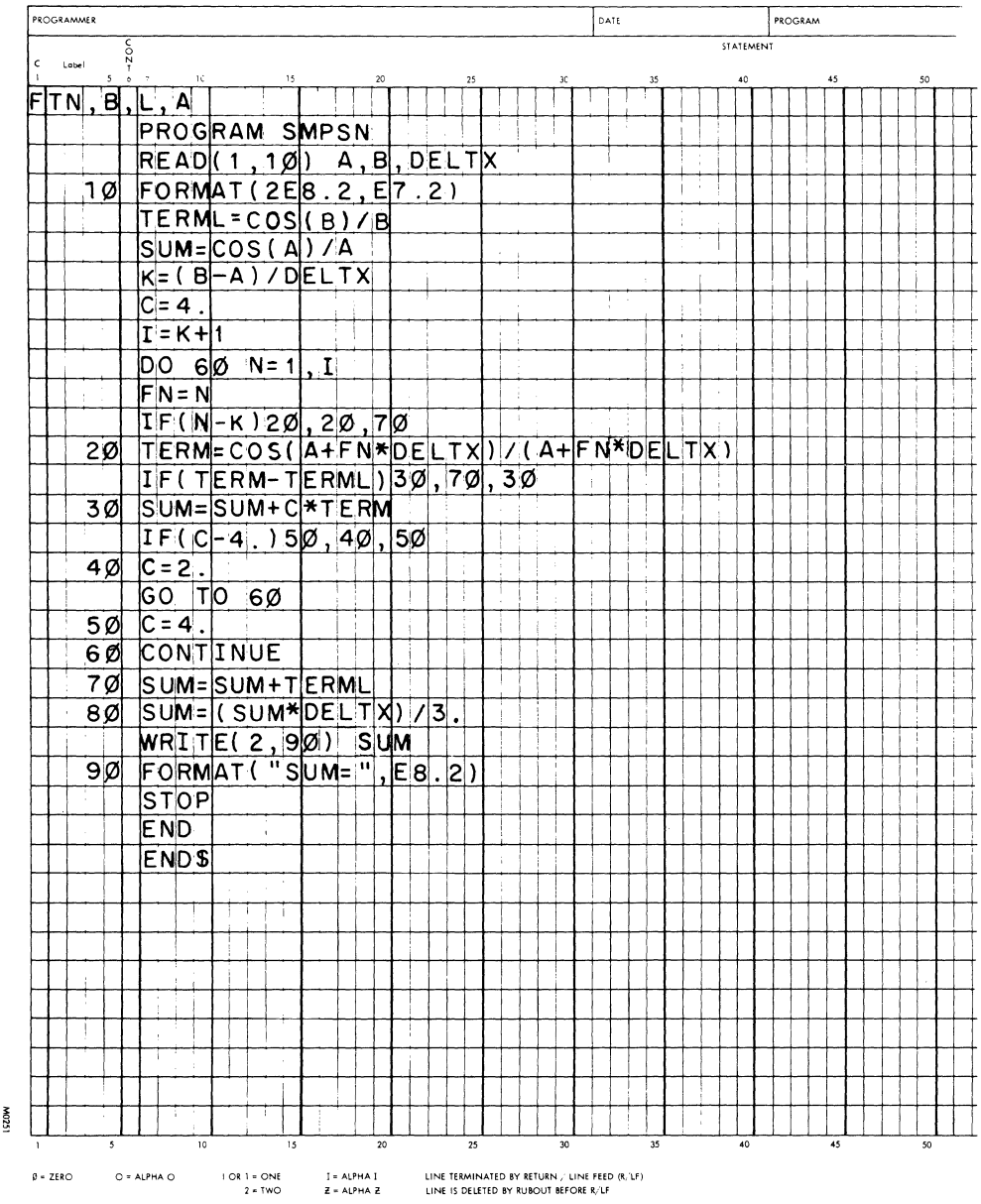
Simpson's rule for approximating a definite integral is:

$$\int_a^b f(x)dx = \frac{\Delta x}{3}\left(f(a)+4f(a+\Delta x)+2f(a+2\Delta x)+4f(a+3\Delta x)+\ldots+f(b)\right)$$

The last term is reached when $(a+k\Delta x)=b$, and when neither a 2 nor a 4 appears in front of the first or last term.



SAMPLE PROGRAM FLOWCHART


PROGRAMMER
DATE
PROGRAM
STATEMENT
C Label CONT
FTN,B,L,A
PROGRAM SMPSN
READ(1,10) A,B,DELTX
10 FORMAT(2E8.2,E7.2)
TERML=COS(B)/B
SUM=COS(A)/A
K=(B-A)/DELTX
C=4.
I=K+1
DO 60 N=1,I
FN=N
IF(N-K)20,20,70
20 TERM=COS(A+FN*DELTX)/(A+FN*DELTX)
IF(TERM-TERML)30,70,30
30 SUM=SUM+C*TERM
IF(C-4.)50,40,50
40 C=2.
GO TO 60
50 C=4.
60 CONTINUE
70 SUM=SUM+TERML
80 SUM=(SUM*DELTX)/3.
WRITE(2,90) SUM
90 FORMAT("SUM=",E8.2)
STOP
END
END$
MR251
Ø = ZERO
O = ALPHA O
1 OR 1 = ONE
2 = TWO
I = ALPHA I
Z = ALPHA Z
LINE TERMINATED BY RETURN / LINE FEED (R/LF)
LINE IS DELETED BY RUBOUT BEFORE R/LF

SOURCE PROGRAM LISTING

```
FTN,B,L,A
      PROGRAM SMPSN
      READ(1,10) A,B,DELTX
   10 FORMAT(2E8.2,E7.2)
      TERML=COS(B)/B
      SUM=COS(A)/A
      K=(B-A)/DELTX
      C=4.
      I=K+1
      DO 60 N=1,I
      FN=N
      IF(N-K)20,20,70
   20 TERM=COS(A+FN*DELTX)/(A+FN*DELTX)
      IF(TERM-TERML)30,70,30
   30 SUM=SUM+C*TERM
      IF(C-4.)50,40,50
   40 C=2.
      GO TO 60
   50 C=4.
   60 CONTINUE
   70 SUM=SUM+TERML
   80 SUM=(SUM*DELTX)/3.
      WRITE(2,90) SUM
   90 FORMAT("SUM=",E8.2)
      STOP
      END
      END$
```

OBJECT PROGRAM LISTING
Assembly Level Language

```
PAGE 0001  SMPSN


00000  000000  BSS 000000
00000  000000  OCT 000000
00001  062314R LDA 000314
00002  006404  OCT 006404
00003  016001X JSB .DIO.
00004  100277R DEF 000277,I
00005  100300R DEF 000300,I
00006  016002X JSB .IOR.
00007  016003X JSB .DST
00010  000246R DEF 000246
00011  016002X JSB .IOR.
00012  016003X JSB .DST
00013  000250R DEF 000250
00014  016002X JSB .IOR.
00015  016003X JSB .DST
00016  000252R DEF 000252
00017  126301R JMP 000301,I
00020  024040  OCT 024040
00021  031105  OCT 031105
00022  034056  OCT 034056
00023  031054  OCT 031054
00024  042467  OCT 042467
00025  027062  OCT 027062
00026  024400  OCT 024400
00027  016004X JSB .DLD
00030  000250R DEF 000250
00031  016005X JSB COS
00032  016006X JSB .FDV
00033  000250R DEF 000250
00034  016003X JSB .DST
00035  000254R DEF 000254
00036  016004X JSB .DLD
00037  000246R DEF 000246
00040  016005X JSB COS
00041  016006X JSB .FDV
00042  000246R DEF 000246
00043  016003X JSB .DST
00044  000256R DEF 000256
00045  016004X JSB .DLD
00046  000250R DEF 000250
00047  016007X JSB .FSB
00050  000246R DEF 000246
00051  016006X JSB .FDV
00052  000252R DEF 000252
```

```
00053  016010X JSB IFIX
00054  072260R STA 000260
00055  016004X JSB .DLD
00056  000315R DEF 000315
00057  016003X JSB .DST
00060  000261R DEF 000261
00061  062260R LDA 000260
00062  042314R ADA 000314
00063  072263R STA 000263
00064  062314R LDA 000314
00065  072264R STA 000264
00066  062264R LDA 000264
00067  016011X JSB FLOAT
00070  016003X JSB .DST
00071  000265R DEF 000265
00072  062264R LDA 000264
00073  003004  OCT 003004
00074  042260R ADA 000260
00075  003004  OCT 003004
00076  002020  OCT 002020
00077  126302R JMP 000302,I
00100  002002  OCT 002002
00101  126303R JMP 000303,I
00102  126302R JMP 000302,I
00103  016004X JSB .DLD
00104  000265R DEF 000265
00105  016012X JSB .FMP
00106  000252R DEF 000252
00107  016013X JSB .FAD
00110  000246R DEF 000246
00111  016003X JSB .DST
00112  000271R DEF 000271
00113  016005X JSB COS
00114  016003X JSB .DST
00115  000273R DEF 000273
00116  016004X JSB .DLD
00117  000265R DEF 000265
00120  016012X JSB .FMP
00121  000252R DEF 000252
00122  016013X JSB .FAD
00123  000246R DEF 000246
00124  016003X JSB .DST
00125  000275R DEF 000275
00126  016004X JSB .DLD
00127  000273R DEF 000273
00130  016006X JSB .FDV
00131  000275R DEF 000275
00132  016003X JSB .DST
00133  000267R DEF 000267
00134  016007X JSB .FSB
00135  000254R DEF 000254
00136  002020  OCT 002020
00137  126304R JMP 000304,I
00140  002002  OCT 002002
```

```
00141   126304R JMP 000304,I
00142   126303R JMP 000303,I
00143   016004X JSB .DLD
00144   000261R DEF 000261
00145   016012X JSB .FMP
00146   000267R DEF 000267
00147   016013X JSB .FAD
00150   000256R DEF 000256
00151   016003X JSB .DST
00152   000256R DEF 000256
00153   016004X JSB .DLD
00154   000261R DEF 000261
00155   016007X JSB .FSB
00156   000315R DEF 000315
00157   002020  OCT 002020
00160   126305R JMP 000305,I
00161   002002  OCT 002002
00162   126305R JMP 000305,I
00163   126306R JMP 000306,I
00164   016004X JSB .DLD
00165   000317R DEF 000317
00166   016003X JSB .DST
00167   000261R DEF 000261
00170   126307R JMP 000307,I
00171   016004X JSB .DLD
00172   000315R DEF 000315
00173   016003X JSB .DST
00174   000261R DEF 000261
00175   062264R LDA 000264
00176   002004  OCT 002004
00177   072264R STA 000264
00200   003004  OCT 003004
00201   042263R ADA 000263
00202   002021  OCT 002021
00203   026066R JMP 000066
00204   016004X JSB .DLD
00205   000256R DEF 000256
00206   016013X JSB .FAD
00207   000254R DEF 000254
00210   016003X JSB .DST
00211   000256R DEF 000256
00212   016004X JSB .DLD
00213   000256R DEF 000256
00214   016012X JSB .FMP
00215   000252R DEF 000252
00216   016006X JSB .FDV
00217   000321R DEF 000321
00220   016003X JSB .DST
00221   000256R DEF 000256
00222   062323R LDA 000323
00223   006400  OCT 006400
00224   016001X JSB .DIO.
00225   100311R DEF 000311,I
00226   100312R DEF 000312,I
```

```
00227  016004X JSB .DLD
00230  000256R DEF 000256
00231  016002X JSB .IOR.
00232  016014X JSB .DTA.
00233  126313R JMP 000313,I
00234  024040  OCT 024040
00235  021123  OCT 021123
00236  052515  OCT 052515
00237  036442  OCT 036442
00240  026105  OCT 026105
00241  034056  OCT 034056
00242  031051  OCT 031051
00243  002400  OCT 002400
00244  016015X JSB .STOP
00245  016015X JSB .STOP
00246  000000  BSS 000030
00276  000000  OCT 000000
00277  000020R DEF 000020
00300  000017R DEF 000017
00301  000027R DEF 000027
00302  000103R DEF 000103
00303  000204R DEF 000204
00304  000143R DEF 000143
00305  000171R DEF 000171
00306  000164R DEF 000164
00307  000175R DEF 000175
00310  000212R DEF 000212
00311  000234R DEF 000234
00312  000233R DEF 000233
00313  000243R DEF 000243
00314  000001  OCT 000001
00315  040000  OCT 040000
00316  000006  OCT 000006
00317  040000  OCT 040000
00320  000004  OCT 000004
00321  060000  OCT 060000
00322  000004  OCT 000004
00323  000002  OCT 000002
               TRA SMPSN
*** END
```

OBJECT PROGRAM LISTING
Symbol Table

```
 PAGE 0005  SMPSN


  SYMBOL  TABLE
SMPSN  R 000000
A      R 000246
B      R 000250
DELTX  R 000252
00010  R 000020
10000  R 000017
10001  R 000027
TERML  R 000254
SUM    R 000256
K      R 000260
C      R 000261
I      R 000263
N      R 000264
FN     R 000265
00020  R 000103
00070  R 000204
TERM   R 000267
00030  R 000143
00050  R 000171
00040  R 000164
00060  R 000175
00080  R 000212
00090  R 000234
10002  R 000233
10003  R 000243
```

BASIC CONTROL SYSTEM
Relocating Loader Memory Allocation

```
SMPSN

 02000 02323

LOAD

FRMTR

 02324 04234
 00240 00730

COS

 04235 04244

SIN

 04245 04346

CHEBY

 04347 04437

••FCM

 04440 04447

••DLC

 04450 04461
```

```
FADSB

 04462 04617

FDV

 04620 04723

FMP

 04724 05007

MPY

 05010 05120

.IENT

 05121 05155

FLOAT

 05156 05162

.PACK

 05163 05267

DIV

 05270 05362
```

```
DLDST

 05363 05420

IFIX

 05421 05455

.STOP

 05456 05476

.ERRR

 05477 05517

PWR2

 05520 05543

.FLUN

 05544 05556
```

```
*LST

.IOC.   15434
.SQT.   15405
.MEM.   15400
.BUFR   15602
SMPSN   02000
.DIO.   03635
.IOR.   03505
.DST    05373
.DLD    05363
COS     04235
.FDV    04620
.FSB    04465
IFIX    05421
FLOAT   05156
.FMP    04724
.FAD    04462
.DTA.   03733
.STOP   05456
.BIO.   03710
.IOI.   03532
.IAR.   03571
.RAR.   03545
.FLUN   05544
.PACK   05163
.MPY    05010
SIN     04245
..FCM   04440
.ERRR   05477
.CHEB   04347
.IENT   05121
.PWR2   05520
..DLC   04450
.DIV    05270


*LINKS
 01723 01777


*RUN
```

OBJECT PROGRAM
Input and Output Data

```
    1.23     4.72     .25
SUM=-.63E+00
STOP
    1.23     2.01     .10
SUM=-.12E-01
STOP
    0.34     1.01     .02
SUM= .88E+00
STOP
    0.00     1.00     .01
SUM= .57E+36
STOP
    1.00     1.25     .05
SUM= .92E-01
STOP
```

INDEX

CUT ALONG LINE

HEWLETT hp PACKARD

READER COMMENT SHEET

HP FORTRAN

HP 2116-9015 April, 1970

Hewlett-Packard welcomes your evaluation of this text. Any errors, suggested additions, deletions, or general comments may be made below. Use extra pages if you like.

FROM PAGE__OF__

NAME: _____

ADDRESS: _____

_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AS SHOWN ON OTHER SIDE AND TAPE

FOLD FOLD

FIRST CLASS
PERMIT NO. 141
CUPERTINO
CALIFORNIA

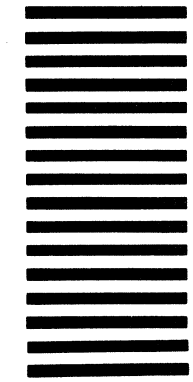
BUSINESS REPLY MAIL

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

SUPERVISOR, SOFTWARE PUBLICATIONS

HEWLETT - PACKARD

CUPERTINO DIVISION
11000 Wolfe Road
Cupertino, California
95014



FOLD FOLD