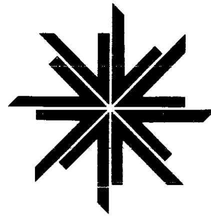


Control Data®
7600 Computer System

FORTRAN
Reference Manual

REVISION RECORD

REVISION	NOTES
6-70	Original Printing (Preliminary)
A	
12-70	This printing obsoletes the Preliminary edition.

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Pub. No. 60280400

© 1970 Control Data Corporation
Printed in the United States of America

Address comments concerning this manual to:

Control Data Corporation
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

CONTENTS

INTRODUCTION			vii
CHAPTER 1	CODING PROCEDURES		1-1
	1.1	Coding Line	1-1
		1.1.1 Statement	1-1
		1.1.2 Continuation	1-2
		1.1.3 Statement Label	1-2
		1.1.4 Identification Field	1-2
		1.1.5 Comments	1-2
	1.2	Punched Cards	1-2
CHAPTER 2	ELEMENTS OF FORTRAN		2-1
	2.1	FORTRAN Character Set	2-1
	2.2	Symbolic Names	2-1
	2.3	Data Types	2-2
	2.4	Constants	2-2
		2.4.1 Integer Constants	2-2
		2.4.2 Real Constants	2-2
		2.4.3 Double-Precision Constants	2-3
		2.4.4 Complex Constants	2-3
		2.4.5 Logical Constants	2-4
		2.4.6 Hollerith Constants	2-5
		2.4.7 Octal (Masking) Constants	2-6
	2.5	Variables	2-6
		2.5.1 Integer Variables	2-7
		2.5.2 Real Variables	2-7
		2.5.3 Double-Precision Variables	2-8
		2.5.4 Complex Variables	2-8
		2.5.5 Logical Variables	2-8
	2.6	Subscripted Variable	2-8
	2.7	Arrays	2-9
CHAPTER 3	EXPRESSIONS		3-1
	3.1	Arithmetic Expressions	3-1
		3.1.1 Forming Arithmetic Expressions	3-2
		3.1.2 Arithmetic Evaluation	3-2
		3.1.3 Mixed-Mode Arithmetic Expressions	3-4
	3.2	Relational Expressions	3-7
	3.3	Logical Expressions	3-8
	3.4	Masking Expressions	3-10

CHAPTER 4	ASSIGNMENT STATEMENTS		4-1
	4.1	Arithmetic Assignment	4-1
	4.2	Mixed-Mode Assignment	4-1
	4.3	Logical Assignment	4-4
	4.4	Masking Assignment	4-4
	4.5	Multiple Assignment	4-4
CHAPTER 5	TYPE DECLARATIONS AND STORAGE ALLOCATION		5-1
	5.1	Type Declaration	5-1
	5.2	Dimension Declaration	5-2
		5.2.1 Variable Dimensions	5-3
	5.3	Common Declaration	5-3
	5.4	Equivalence Declaration	5-6
	5.5	Data Declaration	5-8
		5.5.1 Block Data Subprogram	5-12
CHAPTER 6	CONTROL STATEMENTS		6-1
	6.1	GO TO Statement	6-1
		6.1.1 Unconditional GO TO	6-1
		6.1.2 Assigned GO TO	6-1
		6.1.3 ASSIGN Statement	6-2
		6.1.4 Computed GO TO	6-2
	6.2	IF Statements	6-3
		6.2.1 Three-Branch Arithmetic IF	6-3
		6.2.2 One-Branch Logical IF	6-4
		6.2.3 Two-Branch Logical IF	6-4
	6.3	DO Statement	6-5
		6.3.1 DO Loop Execution	6-6
		6.3.2 DO Nests	6-6
		6.3.3 DO Loop Transfer	6-8
	6.4	CONTINUE Statement	6-9
	6.5	PAUSE Statement	6-9
	6.6	STOP Statement	6-10
	6.7	RETURN Statement	6-10
	6.8	END Statement	6-10
CHAPTER 7	PROGRAM, PROCEDURES AND SUBPROGRAMS		7-1
	7.1	Source Program	7-1
	7.2	Main Program	7-1
	7.3	Program Communication	7-2
	7.4	Subprogram Communication	7-2
	7.5	Procedures and Subprograms	7-2
		7.5.1 Procedure Identifiers	7-3
		7.5.2 Formal Arguments	7-3
		7.5.3 Actual Arguments	7-5
	7.6	Statement Function	7-6

	7.7	Supplied Function	7-6
		7.7.1 Intrinsic Functions	7-7
		7.7.2 Basic External Functions	7-7
	7.8	Subprograms	7-7
		7.8.1 Function Subprogram	7-8
		7.8.2 Subroutine Subprogram	7-11
		7.8.3 Library Subroutines	7-12
	7.9	CALL Statement	7-14
	7.10	EXTERNAL Statement	7-15
	7.11	ENTRY Statement	7-17
	7.12	Variable Dimensions in Subprograms	7-18
	7.13	Program Arrangement	7-20
CHAPTER 8		OVERLAYS	8-1
	8.1	Levels	8-1
	8.2	Identification	8-1
	8.3	Composition	8-2
	8.4	Call	8-2
	8.5	Overlay Format	8-3
	8.6	Loader Cards	8-3
	8.7	Overlay Cards	8-4
CHAPTER 9		INPUT/OUTPUT FORMATS	9-1
	9.1	Input/Output List	9-1
	9.2	Array Transmission	9-2
	9.3	Format Declaration	9-4
	9.4	Conversion Specifications	9-5
		9.4.1 Ew.d Output	9-5
		9.4.2 Ew.d Input	9-6
		9.4.3 Fw.d Output	9-8
		9.4.4 Fw.d Input	9-9
		9.4.5 Gw.d Output	9-9
		9.4.6 Gw.d Input	9-10
		9.4.7 Dw.d Output	9-10
		9.4.8 Dw.d Input	9-10
		9.4.9 Iw Output	9-11
		9.4.10 Iw Input	9-12
		9.4.11 Ow Output	9-12
		9.4.12 Ow Input	9-13
		9.4.13 Aw Output	9-13
		9.4.14 Aw Input	9-14
		9.4.15 Rw Output	9-14
		9.4.16 Rw Input	9-14
		9.4.17 Lw Output	9-15
		9.4.18 Lw Input	9-15
	9.5	nP Scale Factor	9-15
		9.5.1 Fw.d Scaling	9-16
		9.5.2 Ew.d or Dw.d Scaling	9-17
		9.5.3 Gw.d Scaling	9-17

	9.6	Editing Specifications	9-17
		9.6.1 wX	9-17
		9.6.2 wH Output	9-18
		9.6.3 wH Input	9-19
		9.6.4 New Record	9-19
		9.6.5 *...*	9-20
	9.7	Repeated Format Specifications	9-22
	9.8	Unlimited Groups	9-22
	9.9	Variable Format	9-23
	9.10	USASI Compatibility	9-24
		9.10.1 Unlimited Groups for USASI	9-24
		9.10.2 Scale Factor for USASI	9-24
CHAPTER 10		INPUT/OUTPUT STATEMENTS	10-1
	10.1	Output Statements	10-1
	10.2	READ Statements	10-4
	10.3	NAMELIST Statement	10-5
	10.4	File Handling Statements	10-8
	10.5	Buffer Statements	10-9
	10.6	ENCODE/DECODE Statements	10-11
APPENDIX A		7000 SERIES FORTRAN CHARACTER CODES	A-1
APPENDIX B		FORTRAN STATEMENT LIST	B-1
APPENDIX C		FORTRAN FUNCTIONS	C-1
APPENDIX D		COMPUTER WORD STRUCTURE OF CONSTANTS-7600	D-1
APPENDIX E		COMPILATION AND EXECUTION	E-1
APPENDIX F		DIAGNOSTICS	F-1
APPENDIX G		PROGRAM-SUBPROGRAM FORMAT	G-1
APPENDIX H		SYSTEM ROUTINE	H-1
APPENDIX I		EXECUTION DIAGNOSTICS	I-1
APPENDIX J		FORTRAN CROSS-REFERENCE MAP	J-1
APPENDIX K		ADDITIONAL STATEMENTS	K-1

INTRODUCTION

FORTRAN for the CONTROL DATA® 7600 Computer System is a procedural language designed for solving problems of a mathematical or scientific nature. It incorporates a majority of the features of standardized FORTRAN as specified by the X3.9-1966 committee of the United States of America Standards Institute. The few variations are a matter of 7600 hardware requirements. These variations, as well as the extensions, are flagged throughout this manual.

Selected design extensions include FORTRAN IV features, FORTRAN 2.3 features as implemented on the CONTROL DATA® 6000 Series Computers, and new source statements relating to the dual memory of the 7600 Computer. 7600 FORTRAN is also compatible with FORTRAN II. (Since new programs should be written in the later versions of the language as specified above, FORTRAN II formats are not described.)

LANGUAGE FEATURES

- Constants and variables of the following types:

  Integer

  Single-precision floating point (real)

  Double-precision floating point

  Complex

  Logical

  Octal (constant only)

  Hollerith (constant only)

- Mixed mode arithmetic expressions
- Masking (Boolean), logical, and relational operators
- Shorthand notation for logical operators and constants
- Library functions (intrinsic functions and external functions)
- Independently compilable subprograms
- Multiple entry points to subroutines and functions
- Multiple subroutine exits
- Expressions as subscripts

- Variable dimensions
- Variable FORMAT capability
- Intermixed COMPASS subprograms
- Block transfers between large core and small core memory
- Allocation of large core memory
- Conversion formats for all data forms
- Array reference with fewer subscripts than dimensioned
- Large array size - approximately one-half million words
- Hollerith constants in expressions and DATA statements
- More than one statement per line
- Left- or right-justified Hollerith constants
- Two-branch logical IF statements
- NAMELIST capability
- BUFFER IN/BUFFER OUT and ENCODE/DECODE statements
- Overlay capability
- Multiple assignment statement form
- DATA statement usable in main program
- Abbreviated forms of DATA statement
- DO loop indexing parameters changeable within loop

COMPILER FEATURES

7600 FORTRAN is a one-pass compiler. It uses both Large Core Memory and Small Core Memory and takes advantage of the 48-parcel instruction stack and segmented functional units. Optimization of DO loops is accomplished by:

- Performing multiple-dimension index calculation before entering the loop
- Evaluating common subexpressions only once
- Evaluating invariant subexpressions before entering the loop

The compiler and execution time routines execute under 7000 SCOPE. Subprograms are compiled independently, and a file consisting of relocatable binary subprograms is produced. Upon option, the compiler also produces a source listing, an object code listing, a cross reference listing, and a relocatable binary deck.

The compiler can execute as a load-and-go program and can produce 7600 machine language output. It executes as an independent program under control of the operating system and uses only the storage required for compilation of a particular program. Two compilations may proceed simultaneously using the upper and lower job features of SCOPE. Overlays can be loaded at execution time without relocation.

FORTRAN accepts main programs and subprograms written in either FORTRAN source language or 7600 COMPASS assembly language. This feature permits a flexible program arrangement for each particular job.

CODING PROCEDURES 1

1.1 CODING LINE

A FORTRAN coding line contains 80 columns in which FORTRAN characters are written one per column. The four types of coding lines are listed below:

Statement	1-5	statement label
	6	blank or zero
	7-72	FORTRAN statement
	73-80	identification field
Continuation	1-5	blank
	6	FORTRAN character other than blank or zero
	7-72	continued FORTRAN statement
	73-80	identification field
Comment	1	C or $ or *
	2-80	comments
Data	1-80	data

USASI FORTRAN, X3.9-1966, does not specify the identification field or the use of $ and * in comment lines.

1.1.1 STATEMENT

Statement information is written in column 7 through 72. Statements longer than 66 columns may be continued on the next line. Blanks are ignored by the FORTRAN compiler except in H fields. The character $ may be used to separate statements when more than one is written on a coding line. However, it cannot be used with a program name, a subroutine name, a function statement, or any statement which requires a statement number. A blank card may be used to separate statements.

USASI FORTRAN, X3.9-1966, does not specify $.

1.1.2 CONTINUATION

The first line of every statement must have a blank or zero in column 6. If statements occupy more than one line, all subsequent lines must have a FORTRAN character other than blank or zero in column 6. Continuation cards may be separated by cards whose first 72 columns are blank. A statement may have up to 19 continuation lines.

1.1.3 STATEMENT LABEL

A statement label is a string of 1 to 5 digits occupying any column position 1 through 5. It serves as a reference to that particular statement. Only statements referred to elsewhere in the program require statement labels. These references can only be executable statements and FORMAT statements. The same statement label cannot be given to more than one statement in a program unit. A zero number is ignored.

1.1.4 IDENTIFICATION FIELD

Columns 73 through 80 are always ignored in the compilation process. They may be used for identification when the program is punched on cards. Usually, these columns contain sequencing information provided by the programmer.

USASI FORTRAN, X3.9-1966, does not specify the identification field.

1.1.5 COMMENTS

Each line of comment information is designated by a C, *, or $ in column 1. Comment information appears in the source program and the source program listing, but it is not translated into object code. The continuation character in column 6 is not applicable to comment cards.

USASI FORTRAN, X3.0-1966, does not specify * or $.

1.2 PUNCHED CARDS

Each line of the coding form corresponds to one 80-column card; the term "line" and "card" are often used interchangeably. Source programs and data can be read into the computer from cards; a relocatable binary deck or data can be punched onto cards.

ELEMENTS OF FORTRAN 2

2.1 FORTRAN CHARACTER SET

Forty-seven characters are used in forming the elements of a FORTRAN program. These are divided into alphanumeric characters and special characters. The alphanumeric characters are the alphabetic capital letters from A to Z and the decimal numerics from 0 to 9. The special characters are the following:

	blank	(	left parenthesis
=	equals	)	right parenthesis
+	plus	,	comma
-	minus	.	decimal point
*	asterisk	$	dollar sign
/	slash		

Appendix A includes a list of additional characters which may appear in Hollerith literals and, with the exception of the semi-colon, in DATA statements.

All characters appear internally in display code (Appendix A). A blank is ignored by the compiler except in Hollerith fields. Otherwise, it may be used freely to improve program readability.

2.2 SYMBOLIC NAMES

Symbolic names are used to identify data, programs, subprograms, input/output units, and labeled common blocks. With one exception, a symbolic name can be any combination of one to seven alphanumeric characters beginning with a letter.† The exception is a form of the octal constant which is the letter O followed by six octal digits. Embedded blanks in a symbolic name are ignored.

Examples:

Legal Symbolic Names	Illegal Symbolic Names	
IOTA	3BETA	begins with numeric character
A123456	REMAINDER	more than seven characters
O12345	+ 234	begins with special character
O12K345	O123456	illegal as a symbolic name but legal as an octal constant

† USASI FORTRAN, X3.9-1966, limits all symbolic names to six characters.

2.3 DATA TYPES

Seven different data types are specified for 7600 FORTRAN: integer, real, double-precision, complex, logical, Hollerith, and octal. Implicit declaration of type is applicable to integer and real only. In the case of a constant, the absence of a decimal point indicates type integer, and the presence of a decimal point indicates type real. In the case of a variable, an I, J, K, L, M, or N as an initial letter indicates type integer. Any other alphabetic character used as an initial letter indicates type real. Double-precision, complex, and logical data must be declared in a type statement. Hollerith and octal constants are treated as type integer when they appear in arithmetic expressions or assignment statements.

2.4 CONSTANTS

A constant can be any of the seven data types listed above. Complex and double-precision constants are formed from real constants. The type of a constant is determined by its form. The computer word structure for each type is given in appendix D.

2.4.1 INTEGER CONSTANTS

An integer constant, N, is a string of up to 18 decimal digits in the range $-(2^{59}-1) \leq N \leq (2^{59}-1)$. The maximum value of the result of integer addition or subtraction must not exceed $2^{59}-1$. Subscripts and DO-indexes are limited to $2^{17}-2$.

Examples:

63	3647631	314159265	574396517802457165
247	464646464		

During execution, the maximum allowable value is $2^{48}-1$ when an integer constant is converted to real. If the result is greater than $2^{48}-1$, bits 48-58 will be ignored and errors may result. The maximum value of the operands and the result of integer multiplication or division must be less than $2^{48}-1$. High order bits will be lost if the value is larger, but no diagnostic is provided. Values greater than $2^{48}-1$ are not printed and the field contains the character "R" right justified.

2.4.2 REAL CONSTANTS

A real constant is a signed or unsigned string of up to 14 decimal digits that includes a decimal point and/or an exponent. A real constant has one of the following forms:

n.n	n.nE±s	nE±s
n.	n.E±s	
.n	.nE±s	

Where n is the decimal base, s is the exponent to this base 10, and E is the optional symbol used to indicate exponentiation. The plus sign may be omitted for positive s. The range of a non-zero constant is approximately 10^{-294} to 10^{+322}. If the range is exceeded, a compiler diagnostic is provided. If the magnitude is less than 10^{-294}, the value will be zero.

All real numbers are carried in normalized form.

Examples:

3.E1	(means 3.0 x 10^1; i.e., 30.)
3.1415768	31.41592E-01
314.0749162	.31415E03
-3.141592E+279	.31415E+01

2.4.3 DOUBLE-PRECISION CONSTANTS

A double-precision constant is a signed or unsigned string of up to 29 decimal digits that includes a decimal point. It is optionally followed by an exponent. It is represented internally by two words. The forms are similar to real constants:

.nD±s n.nD±s n.D±s nD±s

Where n is the decimal base, s is the exponent to the base 10, and D is the symbol indicating double precision. D must always appear. The plus sign may be omitted for positive s. The range of non-zero constant is, approximately, from 10^{-294} to 10^{+322}; if the range is exceeded, a compiler diagnostic is provided. If s is omitted, it is assumed to be zero.

Examples:

3.1415927D	3141.593D3
3.1416D0	31416.D-04
3141.593D-03	

2.4.4 COMPLEX CONSTANTS

A complex constant appears as an ordered pair of optionally signed real constants. Its form is

(r_1, r_2)

Where the real part of the complex number is represented by r_1 and the imaginary part by r_2.

If the range of the real numbers comprising the constant is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the pair contains integer constants, including (0, 0).

Examples:

FORTRAN Representation	Complex Number
(1., 6.55)	1. + 6.55i
(15., 16.7)	15. + 16.7i
(-14.09, 1.654E-04)	-14.09 + .0001654i
(0., -1.)	0 - 1.0i

2.4.5 LOGICAL CONSTANTS

A logical constant is a truth value. It may assume only the value of true or the value of false. A true constant is stored internally as the one's complement of binary zero. A false constant is stored internally as binary zero. The two permissible forms of a logical constant are:

.TRUE.

.FALSE.

or the briefer alternate forms

.T.

.F.

The latter forms are not specified in USASI X3.9 FORTRAN.

Example:

```
LOGICAL X1, X2

X1 = .TRUE.

X2 = .FALSE.
```

2.4.6 HOLLERITH CONSTANTS

A Hollerith constant is a string of FORTRAN characters which is represented in memory by display code and is treated as an integer. The general form is:

$nHh_1h_2 \ldots h_n$

where n is an unsigned decimal integer indicating the number of characters following H which are part of the constant. H is the symbol indicating Hollerith type. The h_i are the FORTRAN characters that make up the constant. Blanks are significant.

The maximum number of characters allowed in a Hollerith constant of H form depends on its usage. When used in an expression, it is limited to 10 characters. In a DATA statement, or when passed as an actual argument to a subprogram, it is limited only by the necessity that the statement containing it be limited to 19 continuation lines. The long Hollerith constant must be dimensioned in a subprogram when used as an argument.

Alternate forms of the Hollerith constant are:

nLh (left justified)

nRh (right justified)

Both left and right justification are with binary zero fill. If more than ten characters are used in a DATA statement involving such a constant, only the last word has the zero fill. These forms may be used in an arithmetic statement.

USASI FORTRAN, X3.9-1966, does not specify the alternate forms nLh and nRh.

Examples:

```
6HCOGITO              12HCONTROL DATA
4HERGO                5LSUMbb=SUMbb00000
3HSUM                 1H)
5RSUMbb=00000SUMbb    3LbTT=bTT0000000
```

A semicolon (display code 77) cannot appear in Hollerith constants since this bit configuration is recognized as a Hollerith field terminator.

2.4.7 OCTAL (MASKING) CONSTANTS

An octal constant is an optionally signed string of octal digits. It may have a minus sign prefix. It is considered type integer. The two forms of the octal constant are:

$On_1 \ldots n_i$	$6 \leq i \leq 20$
$n_1 \ldots n_i B$	$1 \leq i \leq 20$

The first form consists of 6 to 20 octal digits preceded by the letter O. The second form consists of 1 to 20 octal digits followed by the letter B.

Octal constants are right justified with zero fill. If the constant exceeds 20 digits, or if a non-octal digit appears, a compiler diagnostic is provided.

USASI FORTRAN, X3.9-1966, does not specify octal constants.

Examples:

```
O7777777700077777        777776B

O2323232323232323        777000777000777B
```

2.5 VARIABLES

FORTRAN recognizes simple and subscripted variables. A simple variable represents a single quantity and references a storage location. The value specified by the name is always the current value stored in the location. Variables are identified by a symbolic name of 1-7 alphanumeric characters, the first of which must be alphabetic.

The compiler does not check to see if a variable has been assigned a value. The user must make certain that all variables are defined. Otherwise, unexpected values may result.

The type of variable is defined in one of two ways:

Explicit — Variables may be declared a particular type with the FORTRAN type declarations.

Implicit — A variable not defined in a FORTRAN type declaration is assumed to be integer if the first character of the symbolic name is I, J, K, L, M, or N.

Examples:

I15, JK26, KKK, NP362L, M

All other variables not declared in a FORTRAN type declaration are assumed to be real.

Examples:

TEMP, ROBIN, A55, R3P281

2.5.1 INTEGER VARIABLES

Integer variables are defined explicitly or implicitly. They may assume values in the range

$-(2^{59}-1) \leq I \leq (2^{59}-1)$.

The maximum absolute value a particular integer variable may assume depends on usage. The result of conversion from integer to real, of the integer multiplication, integer division, or input/output under the I-format specification is limited to $2^{48}-1$. The result of integer addition or subtraction can be as great as $2^{59}-1$. Subscripts and DO indexes are limited to $2^{17}-2$. Each integer variable occupies one word of storage.

Examples:

IOTA	LLLLLL
J	M58A
K2S04	NEGATE

2.5.2 REAL VARIABLES

Real variables are defined explicitly or implicitly. They may assume values in the range

$10^{-294} < X < 10^{+322}$

with approximately 14 significant digits. More specifically, X may assume the following values:

$-10^{+322} < X < -10^{-294}$

$X = 0$

$10^{-294} < X < 10^{+322}$

Each real variable is stored in floating-point format and occupies one word.

Examples:

ALPHA	XXXX
BETA	Z62597
GAMMA	REAL22

2.5.3 DOUBLE - PRECISION VARIABLES

Double-precision variables must be defined explicitly by a type declaration. Each double precision variable occupies two words of storage and can assume values in the range $10^{-294} \leq d \leq 10^{+322}$ with approximately 29 significant digits. Essentially, the double-precision variable is a real variable with storage extended in order to achieve greater precision.

Examples:

```
DOUBLE PRECISION  OMEGA, X, IOTA
```

2.5.4 COMPLEX VARIABLES

Complex variables must be explicitly defined by a type declaration. A complex variable occupies two words in storage. Each word contains a number in real variable format. This ordered pair of real variables (C_1, C_2) represents the complex number C_1+iC_2)

Example:

```
COMPLEX   ZETA, MU, LAMDA
```

2.5.5 LOGICAL VARIABLES

Logical variables must be defined explicitly by a type declaration. Each logical variable occupies one word of storage. It can assume the value of true or false. A logical variable with a positive zero value is false. Any other value is true. When a logical variable appears in an expression whose dominant mode is real, double, or complex, it is not packed and normalized prior to its use in the evaluation of an expression (as is the case with an integer variable).

Example:

```
LOGICAL VALUE, L33, PRAVDA
```

2.6 SUBSCRIPTED VARIABLE

A subscripted variable may have one, two, or three subscripts enclosed in parentheses. More than three produce a compiler diagnostic. Subscripts can be expressions in which the operands are simple integer variables and integer constants, and the operators are addition, subtraction, multiplication, and division only. Such expressions must result in positive integers. Use of other values such as zero, real, negative integer, complex, or logical may invalidate results.

When a subscripted variable represents the entire array, the subscripts are the dimensions of the array. When a subscripted variable references a single element in an array, the subscripts describe the relative location of the element in the array.

2.7 ARRAYS

An array is a block of successive storage locations. The entire array may be referenced by the array name without subscripts (I/O lists and Implied DO-loop notation). Arrays may have one, two, or three dimensions. The array name and dimensions must be declared in a DIMENSION, COMMON, or TYPE declaration prior to the first program reference to that array.

Each element in an array may be referenced by the array name plus a subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array. The maximum number of elements in an array is the product of the dimensions.

Array elements are stored by columns in ascending locations. In the array declared as A(3,3,3):

A_{111} A_{121} A_{131}

A_{211} A_{221} A_{231}

A_{311} A_{321} A_{331}

A_{112} A_{122} A_{132}

A_{212} A_{222} A_{232}

A_{312} A_{322} A_{332}

A_{113} A_{123} A_{133}

A_{213} A_{223} A_{233}

A_{313} A_{323} A_{333}

The planes are stored in order, starting with the first, as follows:

$A_{111} \rightarrow L$ $A_{121} \rightarrow L+3$ . . . . $A_{133} \rightarrow L+24$

$A_{211} \rightarrow L+1$ $A_{221} \rightarrow L+4$ . . . . $A_{233} \rightarrow L+25$

$A_{311} \rightarrow L+2$ $A_{321} \rightarrow L+5$ . . . . $A_{333} \rightarrow L+26$

Array allocation is discussed under DIMENSION declaration. The location of an array element with respect to the first element is a function of the maximum array dimensions and the type of the array.

Given DIMENSION A(L,M,N), the location of A(i,j,k), with respect to the first element A of the array, is given by A+(i-1+L*(j-1+M*(k-1)))*E.

The quantity enclosed by the outer parentheses is the subscript expression. E is the element length--the number of storage words required for each element of the array. For real, logical, and integer arrays, E = 1. For complex and double precision arrays, E = 2.

Example:

In an array defined by DIMENSION A(3,3,3), the location of A(2,2,3) with respect to A(1,1,1) is:

Locn A(2,2,3) = (Locn A(1,1,1) + (2-1+3(1+3(2))))*1

= (L + 22)*1 = L + 22

An array reference is never checked to see if it is within the limits of the array as defined.

7600 FORTRAN permits the following relaxation of the representation of subscripted variables:

Given A(D_1, D_2, D_3), where the D_i are integer constants,

then A(I,J,K) implies A(I,J,K)

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

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

A implies A(1,1,1)

similarly, for A(D_1, D_2)

A(I,J) implies A(I,J)

A(I) implies A(I,1)

A implies A(1,1)

and for A(D_1)

A(I) implies A(I)

A implies A(1)

The elements of a single-dimension array A(D_1) may not be referred to as A(I,J,K) or A(I,J). Diagnostics occur if this is attempted.

USASI FORTRAN, X3.9-1966, does not specify the above relaxations.

EXPRESSIONS 3

An expression is a set of operands combined by operators and parentheses to produce, at time of execution, a single-valued result. The set may be a single character or it can be a complex string of operands and operators nested within parentheses. There are four kinds of expressions in 7600 FORTRAN: arithmetic, masking (Boolean), logical, and relational. Arithmetic and masking expressions produce numerical results. Logical and relational expressions produce truth values. Each type of expression is associated with particular sets of operators and operands.

3.1 ARITHMETIC EXPRESSIONS

An arithmetic expression is a combination of arithmetic operators and operands which, when evaluated, produces a single numerical value.

The arithmetic operators are:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

The arithmetic operands are:

Constants

Variables (simple or subscripted)

Functions

Examples:

```
A
3.14159
3 + 16.427
(XBAR + (B(I,J+1,K)/3))
- (C + DELTA * AERO)
(B - SQRT (B**2 - (4*A*C)))/(2.0*A)
```

3.1.1 FORMING ARITHMETIC EXPRESSIONS

Two arithmetic operators may not appear next to one another in an arithmetic expression. If minus is used to indicate a negative operand, the sign and the element must be enclosed in parentheses if preceeded by an operator.

```
B*A/(-C)
                    but      -A*B-C
A*(-C)
```

Parentheses may be used to indicate grouping as in ordinary mathematical notation but they may not be used to indicate multiplication.

When writing an integer expression, it is important to remember that dividing an integer quantity by an integer quantity always yields a truncated result. The expression I*J/K may not yield the same result as J/K*I as the following numerical examples show:

4*3/2 = 6   but 3/2*4 = 4

3.1.2 ARITHMETIC EVALUATION

Parenthetical and function expressions are evaluated first in a right-to-left scan. In parenthetical expressions within parenthetical expressions, evaluation begins with the innermost expression and proceeds outward. Separate parenthetical expressions are evaluated as they are encountered in the right-to-left scan.

In an expression with no parentheses or within a pair of parentheses in which unlike operators appear, evaluation proceeds according to the following hierarchy of operators:

**	exponentiation	performed first
/ *	division multiplication	performed next
+ -	addition subtraction	performed last

In an expression with like operators, evaluation proceeds from left to right.

Examples:

In the following examples, R indicates an intermediate result in evaluation:

A**B/C+D*E*F-G is evaluated:

$A**B \rightarrow R_1$

$R_1/C \rightarrow R_2$

$D*E \rightarrow R_3$

$R_3*F \rightarrow R_4$

$R_4+R_2 \rightarrow R_5$

$R_5-G \rightarrow R_6$

A**B/(C+D)*(E*F-G) is evaluated:

$E*F-G \rightarrow R_1$

$C+D \rightarrow R_2$

$A**B \rightarrow R_3$

$R_3/R_2 \rightarrow R_4$

$R_4*R_1 \rightarrow R_5$

H(I3)+C(I,J+2)*(COS(Z))**2 is evaluated:

$COS(Z) \rightarrow R_1$

$R_1**2 \rightarrow R_2$

$R_2*C(I,J+2) \rightarrow R_3$

$R_3+H(I3) \rightarrow R_4$

The following is an example of an expression with embedded parentheses.

A*(B+((C/D)-E)) is evaluated:

$C/D \rightarrow R_1$

$R_1-E \rightarrow R_2$

$R_2+B \rightarrow R_3$

$R_3*A \rightarrow R_4$

(A*(SIN(X)+1.)-Z)/(C*(D-(E+F))) is evaluated:

E+F → R_1

D-R_1 → R_2

C*R_2 → R_3

SIN(X) → R_4

R_4+1. → R_5

A*R_5 → R_6

R_6-Z → R_7

R_7/R_3 → R_8

3.1.3 MIXED-MODE ARITHMETIC EXPRESSIONS

Mixed-mode arithmetic is permissible for all combinations of types (integer, real, double-precision, complex, and logical operands) using any of the mathematical operations except exponentiation. The type of an evaluated mixed-mode arithmetic expression is the mode of the dominant operand type. The order of dominance of operand types within an expression is given by the following list which proceeds from highest to lowest:

Complex

Double-precision

Real

Integer

Logical

In expressions of the form A**B, the following rules apply:

If B is preceded by a minus operator, the form is A**(-B).

A and B are treated as integers if type logical.

For the various operand types, the type relationships of A**B are:

Type A \ Type B	I	R	D	C	L
I	I	no	no	no	I
R	R	R	D	no	R
D	D	D	D	no	D
C	C	no	no	no	C
L	I	no	no	no	I

no indicates an invalid operation

Examples:

1. Given real A, B; integer I, J. The type of expression A*B-I+J is real because the dominant operand type is real.

   The expression is evaluated:

   Convert I to real

   Convert J to real

   A*B → R_1 real

   R_1-I → R_2 real

   R_2+J → R_3 real

2. The use of parentheses can change the evaluation. A, B, I, J are defined as above. A*B-(I-J) is evaluated:

   I-J → R_1 integer

   A*B → R_2 real

   Convert R_1 to real

   R_2-R_1 → R_3 real

3. Given complex C, D, real A, B. The type of the expression A* (C/D)+B is complex because the dominant operand type is complex. The expression is evaluated:

   C/D → R_1 complex

   Convert A to complex

   A*R_1 → R_2 complex

   Convert B to complex

   R_2+B → R_3 complex

4. Consider the expression C/D+(A-B) where the operands are defined in 3 above. The expression is evaluated:

   A-B → R_1 real

   C/D → R_2 complex

   Convert R_1 to complex

   R_1+R_2 → R_3 complex

5. Mixed-mode arithmetic with all types is illustrated by this example:

Given: the expression C*D+R/I-L

C	Complex
D	Double
R	Real
I	Integer
L	Logical

The dominant operand type in this expression is complex; therefore, the evaluated expression is complex.

Evaluation:

Round D to real and affix zero imaginary part.

Convert D to complex

C*D → R_1 complex

Convert R to complex

Convert I to complex

R/I → R_2 complex

R_2+R_1 → R_3 complex

R_3-L → R_4 complex

If the same expression is rewritten with parentheses as C*D+(R/I-L) the evaluation proceeds:

Convert I to real

R/I → R_1 real

R_1-L → R_2 real

Convert D to complex

C*D → R_3 complex

Convert R_2 to complex

R_2+R_3→ R_4 complex

3.2 RELATIONAL EXPRESSIONS

A relational expression is a combination of two arithmetic expressions with a relational operator. The relational expression will have the value true or false depending on whether the stated relation is valid or not. A true relational expression is assigned the value minus zero (all one bits). A false relational expression is assigned the value plus zero (all zero bits). The general form of a relational expression is:

a_1 op a_2

where the a's are arithmetic expressions and op is one of the relational operators.

NOTE

A relational expression can have only two operands combined by one operator. The form a_1 op a_2 op a_3 is not valid.

The relational operators are

Symbol	Meaning
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

Relational expressions of the following forms where I is integer, R is real, D is double precision and C is complex, are allowed:

I.LT.R	
I.LT.D	(D converted to real)
I.LT.C	(Real part of C is used)

A relation of the form a_1 op a_2 is evaluated from left to right.

The relations a_1 op a_2, a_1 op (a_2), (a_1) op a_2, (a_1) op (a_2) are equivalent.

Examples:

```
A.GT.16.                R(I).GE.R(I-1)
R-Q(I)*Z.LE.3.141592    K.LT.16
B-C.NE.D+E              (I).EQ.(J(K))
```

Mixed-mode is permissible in relational expressions for all combinations of types integer, real, double-precision, and complex. The order of dominance of the operand types is the same as that stated for mixed-mode arithmetic expressions (section 3.1.3). When complex expressions are tested for zero or minus zero, only the real part is used in the comparison. For double precision numbers, the value is converted to real.

USASI FORTRAN, X3.9-1966, specifies that the length of the real shall be converted to double precision length for use in evaluating the relational expression.

Relational expressions are converted to equivalent arithmetical expressions at compile time. At execution time, these equivalent arithmetic expressions are evaluated with program-supplied values and compared with zero to determine the truth value of the corresponding relational expression. For example, the relation p.EQ.q is equivalent to p-q=0. At time of execution, the difference is computed and tested for zero. If the difference is zero (or minus zero), the relation is true; otherwise, it is false. Likewise, the relation p.GE.q is equivalent to p-q $\geq$ 0. At time of execution, the difference p-q is computed and compared with zero. If the difference is greater than or equal to zero, the relation is true. If the relation is less than zero, then it is false.

The relation I.GE.0 is treated as true if I assumes the value minus zero or plus zero.

3.3 LOGICAL EXPRESSIONS

A logical expression is a combination of logical operands and/or relational expressions with logical operators which, when evaluated, will have a value of true or false. The general form of a logical expression is

L_1 op L_2 op L_3 . . .

where the L's are logical operands or relational expressions and the op's are logical operators.

The logical operands are

Logical constant	Either the value .TRUE. or the value .FALSE.
Logical variable	A variable that has been declared in a LOGICAL type statement. It can only assume the values .TRUE. or .FALSE.

The logical operators are

.NOT.	Logical negation	Reverses the truth value of the logical expression that follows it
.AND.	Logical conjuntion	Combines two logical expressions to produce a value of .TRUE. whenever both expressions are true; otherwise, it gives a value of .FALSE.
.OR.	Logical disjunction	Combines two logical expressions to produce a value of .TRUE. whenever either or both expressions are true: otherwise, it gives a value of .FALSE.

Alternate forms of the logical operators are

.N.

.A.

.O.

USASI FORTRAN, X3.9-1966, does not specify the alternate forms of the logical operators.

The logical operator .NOT. indicating negation appears in the form:

.NOT. L_1

The value of the expression is examined. If the value is equal to plus zero, the logical expression has the value false. All other values are considered true.

The hierarchy of logical operations is:

First	.NOT.	or	.N.
then	.AND.	or	.A.
then	.OR.	or	.O.

A logical variable, logical constant, or a relational expression is, in itself, a logical expression. If L_1, L_2 are logical expressions, then the following are logical expressions:

.NOT. L_1

L_1.AND. L_2

L_1.OR. L_2

If L is a logical expression, then (L) is a logical expression.

If L_1, L_2 are logical expressions and op is .AND. or .OR. then L_1 op op L_2 is never legitimate.

.NOT. may appear in combination with .AND. or .OR. only as follows:

L_1.AND. .NOT. L_2

L_1.OR. .NOT. L_2

L_1.AND. (.NOT.•••)

L_1.OR. (.NOT.•••)

.NOT. may appear with itself only in the form .NOT. (.NOT. (.NOT. L))
Other combinations cause compilation diagnostics.

If L_1, L_2 are logical expressions, the logical operators are defined as follows:

.NOT. L_1	is false only if L_1 is true
L_1.AND. L_2	is true only if L_1, L_2 are both true
L_1.OR. L_2	is false only if L_1, L_2 are both false

Examples:

$B - C \leq A \leq B + C$ is written

B-C. LE. A. AND. A. LE. B+C

FICA greater than 176.0 and PAYNMB equal to 5889.0 is written
FICA. GT. 176. 0. AND. PAYNMB. EQ. 5889. 0

3.4 MASKING EXPRESSIONS

The masking expression is a generalized form of the logical expression in which the variables may be types other than logical.

In a FORTRAN masking expression, 60-bit logical arithmetic is performed bit-by-bit on the operands within the expression. The operands may be any type variable, constant, or expression. No mode conversion is performed during evaluation. If the operand is complex, operations are performed on the real part. Although the masking operators are identical in appearance to the logical operators, their meanings are different. They are listed according to hierarchy. The following definitions apply:

.NOT. or .N.	complement the operand
.AND. or .A.	form the bit-by-bit logical product of two operands
.OR. or .O.	form the bit-by-bit logical sum of two operands

The operations are described below:

p	v	p .AND. v	p. OR. v	.NOT. p
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Let B_i be masking expressions, variables or constants of any type except logical. The following are masking expressions:

.NOT.B_1 B_1.AND.B_2 B_1.OR.B_2

If B is a masking expression, then (B) is a masking expression.

.NOT. may appear with .AND. or .OR. only as follows:

.AND..NOT.

.OR..NOT.

.AND. (.NOT.···

.OR. (.NOT.···

Masking expressions of the following forms are evaluated from left to right.

A .AND. B .AND. C...

A .OR. B .OR. C...

Arithmetic expressions appearing in masking statements must be enclosed in parentheses, e.g., E=(E*100B).OR.F.

Examples:

A	77770000000000000000	octal constant
D	00000000777777777777	octal constant
B	00000000000000001763	octal form of integer constant
C	20045000000000000000	octal form of real constant

.NOT.A	is	00007777777777777777
A .AND. C	is	20040000000000000000
A .AND..NOT.C	is	57730000000000000000
B .OR..NOT.D	is	77777777000000001763

The last expression could also be written as B .O. .N. D

ASSIGNMENT STATEMENTS 4

4.1 ARITHMETIC ASSIGNMENT

The general form of the arithmetic assignment statement is A = E, where E is an arithmetic expression and A is any variable name, simple or subscripted. The operator = means that A is replaced by the value of the evaluated expression, E, with conversion for mode if necessary.

Examples:

```
A =-A
B(J,4) = CALC(I+1)*BETA+2.3478
XTHETA=7.4*DELTA+ A(I,J,K**BETA)
RESPSNE=SIN(ABAR(INV+2,JBAR)/ALPHA(J,KAPL(I) ) )
JMAX=19
AREA = SIDE1 *SIDE2
PERIM = 2.*(SIDE1 + SIDE2)
```

4.2 MIXED-MODE ASSIGNMENT

The type of an evaluated expression is determined by the type of the dominant operand. This, however, does not restrict the types that identifier A may assume. A complex expression may replace A, even if A is real. The following chart shows the A = E relationship for all the standard modes. The mode of A determines the mode of the statement.

When all the operands in the expression E are logical, the expression is evaluated as if all the logical operands were integers.

For example, if L_1, L_2, L_3, L_4 are logical variables, R is a real variable, and I is an integer variable, then $I = L_1 * L_2 + L_3 - L_4$ is evaluated as if the L_i were all integers and the resulting value is stored as an integer in I.

$R = L_1 * L_2 + L_3 - L_4$ is evaluated as stated above, but the result is converted to a real (a floating point quantity) before it is stored in R.

Type of A	Type of Expression E			
	Complex	Double Precision	Real	Integer
Complex	A = E	Set A = most significant half of E A_{real} = E A_{imag} = 0	A_{real} = E A_{imag} = 0	Convert E to real A_{real} = E A_{imag} = 0
Double Precision	A = E_{real} less significant is set to zero	A = E	A = E less significant is set to zero	Convert E to Real A = E less significant is set to zero
Real	A = E_{real}	Set A = most significant half of E A = E	A = E	Convert E to Real A = E
Integer	Truncate E_{real} to Integer A = E	Truncate E to 48 bit integer A = E	Truncate E to Integer A = E	A = E
Logical	If $E_{real} \neq$ 0, A=E_{real} If E_{real} = 0, A = 0	If E ≠ 0, A=most significant half of E If E = 0, A = 0	If E ≠ 0, A = E If E = 0, A = 0	If E ≠ 0, A = E If E = 0, A = 0

Examples:

Given	C_i, A_1	Complex
	D_i, A_2	Double
	R_i, A_3	Real
	I_i, A_4	Integer
	L_i, A_5	Logical

$A_1 = C_1 * C_2 - C_3 / C_4$ (6.905, 15.393) = (4.4, 2.1)* (3.0, 2.0) - (3.3, 6.8) / (1.1, 3.4)

The expression is complex; the result of the expression is a two-word, floating point quantity. A_1 is complex, and the result replaces A_1.

$A_3 = C_1$ 4.4000+000 = (4.4,2.1)

The expression is complex. A_3 is real; therefore, the real part of C_1 replaces A_3.

$A_3 = C_1*(0.,-1.)$ 2.1000+000 = (4.4,2.1)* (0.,-1.)

The expression is complex. A_3 is real; the real part of the result of the complex multiplication replaces A_3.

$A_4 = R_1/R_2*(R_3-R_4)+I_1-(I_2*R_5)$ 13=8.4/4.2*(3.1-2.1)+14-(1*2.3)

The expression is real. A_4 is integer; the result of the expression evaluation, a real, is converted to an integer replacing A_4.

$A_2 = D_1**2*(D_2+D_3*D_4))$ 4.96800000000000+001 = 2.0D**2*(3.2D+(4.1D*1.0D))
$+(D_2*D_1*D_2)$ +(3.2D*2.0D*3.2D)

The expression is double precision. A_2 is double precision; the result of the expression evaluation, a double precision floating quantity, replaces A_2.

$A_5 = C_1*R_1-R_2+I_1$ 1=(4.4,2.1)*8.4-4.2+14

The expression is complex. Since A_5 is logical, the real part of the evaluated expression replaces A_5. If the real part is zero, zero replaces A_5.

4.3 LOGICAL ASSIGNMENT

The general form of the logical assignment statement is L = E, where L is a logical variable and E may be a logical, relational, or arithmetic expression.

Examples:

```
LOGICAL A, B, C, D, E, LGA, LGB, LGC
REAL F, G, H
A = B .AND. C .AND. D
A = F .GT. G .OR. F .GT. H
A = .N. (A.A. .N. B) .AND. (C.O.D)
LGA = .NOT. LGB
LGC = E .OR. LGC .OR. LGB .OR. LGA .OR. (A .AND. B)
```

4.4 MASKING ASSIGNMENT

The general form of the masking assignment statement is M = E. E is a masking expression, and M is a variable of any type except logical. No mode conversion is made during the replacement.

Examples:

```
INTEGER I,J,K,L,M,N(16)
REAL B,C,D,E,F(15)

N(2) = I .AND. J
B = C .AND. L
F(J) = I .OR. .NOT. L .AND. F(J)
N(1) = I.O.J.O.K.O.L.O.M
I = .N.I
D = (B.LE.C) .AND. (C.LE. E) .AND. .NOT.I
```

4.5 MULTIPLE ASSIGNMENT

Expressions of the form

```
A=B=C=D=3.0*X
```

are permissible and are executed right to left. The above expression would result in code which is equivalent to the expressions:

```
D=3.0*X
C=D
B=C
A=B
```

CONTROL STATEMENTS 6

Program execution normally proceeds from statement to statement as they appear in a 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. A transfer to a nonexecutable statement will result in a program error, which is always recognized during compilation.

6.1 GO TO STATEMENT

GO TO statements transfer control to a labeled statement whose reference is fixed or which is selected during execution of the program. The statement labels used in the GO TO statements must be associated with executable statements in the same program unit as the GO TO statement.

6.1.1 UNCONDITIONAL GO TO

The form of the unconditional GO TO statement is:

```
GO TO k
```

k is a statement label.

Execution of this statement discontinues the current sequence of execution and resumes execution at the statement labeled k.

Example:

```
      GO TO 10
   5  DIF = DIF - SUM
  10  SUM = SUM +1
```

Statement 5 is skipped during execution of this sequence.

6.1.2 ASSIGNED GO TO

The form of the assigned GO TO statement is:

GO TO m, $(n_1, n_2, \ldots, n_m)$

GO TO m

This statement acts as a many-branched GO TO. m is a simple integer variable assigned an integer value n in a preceding ASSIGN statement (section 6.1.3). The n_i are statement labels. As shown, the parenthetical statement label list need not be present.

Once having been defined by an ASSIGN statement, the variable m may not be referenced by any statement other than GO TO m until it is redefined.

The comma after m is optional. However, when the list is omitted, the comma must be omitted. m cannot be defined as the result of a computation. No compiler diagnostic is given if m is computed, but the object code is incorrect. If an assignment has not been made for an assigned GO TO statement and m is equal to zero, a diagnostic is provided at object time. If m is non-zero and within the range, a valid assignment is assumed. FORTRAN does not preset all locations to zero.

6.1.3 ASSIGN STATEMENT

The form of the GO TO assignment statement is:

```
ASSIGN k TO m
```

k is one of the statement labels appearing in the GO TO list; m is the simple integer variable in the assigned GO TO statement. At the time of execution of an assigned GO TO statement, the current value of m must have been assigned by an ASSIGN statement.

Example:

```
ASSIGN 10 TO NN
•
•
•
GO TO NN, (5, 10, 15, 20)
```

Statement number 10 will be executed next.

6.1.4 COMPUTED GO TO

The form of the computed GO TO statement is:

GO TO $(n_1, n_2, \ldots, n_m)$, i

This statement acts as a many-branch GO TO; i is present or computed prior to its use in the GO TO.

The n_i are statement labels and i is a simple integer variable. If $i < 1$ or if $i > m$, the transfer is undefined and an object time diagnostic will be issued indicating the point at which the error was detected. If $1 \leq i \leq m$, the transfer is to n_i.

The comma separating the statement number list and the index is optional.

Example:

```
N=3
.
.
.
GO TO (100,101,102,103) N
```

Statement number 102 will be the selected control transfer.

For proper operations, i must not be specified by an ASSIGN statement. No compilation diagnostic is provided for this error, but the object code is incorrect.

Example:

```
    ISWICH = 1
    GO TO (10,20,30),ISWICH
           .
           .
           .
10  JSWICH = ISWICH +1
    GO TO (11,21,31),JSWICH
```

Control transfers to statement 21.

6.2 IF STATEMENTS

The IF statement is used to transfer control conditionally. At time of execution, an expression in the IF statement is evaluated and the result determines the statement to which the jump will be made.

6.2.1 THREE-BRANCH ARITHMETIC IF

The form of the three-branch arithmetic IF is:

IF (c)n_1, n_2, n_3

c is any expression, and the n_i are statement labels. This statement tests the evaluated expression c and jumps accordingly as follows:

$c < 0$ jump to statement n_1

$c = 0$ jump to statement n_2

$c > 0$ jump to statement n_3

In the test for zero, +0=-0. When the mode of the evaluated expression is complex, only the real part is tested.

Example:

```
IF (IOTA-6)3, 6, 9
```

If the evaluation of the expression IOTA-6 produces a negative result, control transfers to the statement labeled 3; if zero, to 6; if positive, to 9.

6.2.2 ONE-BRANCH LOGICAL IF

The form of the one-branch logical IF is:

IF (ℓ)s

ℓ is a logical or relational expression and s is any executable statement except another logical IF, a DO statement or an END. If ℓ is true (not plus zero), the statement s is executed. If ℓ is false (plus zero) the statement immediately following the IF statement is executed.

Examples:

```
IF (A.LE.2.5) A = 2.0
```

When this statement is executed, the value of A will be compared with 2.5. If it is less than or equal to 2.5, A will be set to the value 2.0. If the comparison shows A to be greater than 2.5, control will proceed to the statement following.

6.2.3 TWO-BRANCH LOGICAL IF

The form of the two-branch logical IF is:

IF (ℓ) n_1, n_2

ℓ is a logical or relational expression and the n_i are statement labels.

The evaluated expression is tested for true (not plus zero) or false (plus zero) condition. If ℓ is true, the jump is to statement n_1. If ℓ is false, the jump is to statement n_2.

USASI FORTRAN, X3.9-1966, does not specify the two-branch logical IF.

Example:

IF (ℓ) 5,6

At time of execution, ℓ is tested for true or false condition. If true, control transfers to statement 5. If false, control transfers to statement 6.

6.3 DO STATEMENT

A DO statement makes it possible to repeat a group of statements a designated number of times using an integer variable whose value is progressively altered with each repetition. The initial value, final value, and rate of increase of this integer variable is defined by a set of indexing parameters included in the DO statement. The range of the repetitions extends from the DO statement to the terminal statement and is called the DO loop. The form of a DO statement is:

DO n i = m_1, m_2

DO n i = m_1, m_2, m_3

n Label of the terminal statement of the loop.

i Simple integer variable called the index variable. With each repetition, its value is altered progressively by the increment parameter m_3. Upon exiting from the range of a DO, the control variable remains defined as the last value acquired in execution of the DO if the exit results from execution of a GO TO or IF only. If the exit results from the DO loop being satisfied, the index variable is no longer well defined.

m_1 Initial parameter, the value of i at the beginning of the first loop.

m_2 Terminal parameter. When the value of i surpasses the value of m_2, DO execution is terminated and control goes to the statement immediately following the terminal statement.

m_3 Increment parameter, the amount i is increased with each repetition. If it has the value 1, it may be omitted (first form above).

The DO statement, the statement labeled n, and any intermediate statements constitute a DO loop; n may not be an arithmetic IF or GO TO statement, a two branch logical IF, a RETURN, another DO statement or a nonexecutable statement.

The indexing parameters m_1, m_2, m_3 are either unsigned integer constants or simple integer variables. Subscripted variables and negative or zero integer constants cause a diagnostic.

The indexing parameters m_1, m_2, and m_3, if variable, may assume positive or negative values or zero.†

The values of m_1, m_2, and m_3 may be changed during the execution of the DO loop.

† USASI FORTRAN, X3.9-1966, states that at time of execution of the DO, m_1, m_2, and m_3 must be greater than zero.

Examples:

1.
```
      DO 25 I=1,100
   25 A(I)=A(I)+B(I)
```

The index variable I is incremented by one for each cyle until the DO loop is executed 100 times. The control is then transferred to the statement immediately following statement 25.

2.
```
      DO 12 I=1,10,2
      J=1+K
      X(J)=Y(J)
   12 CONTINUE
```

I is set to the initial value of one and incremented by two on each of the following cycles. When the execution of the fifth cycle (I=9) is completed, control passes out of the DO loop.

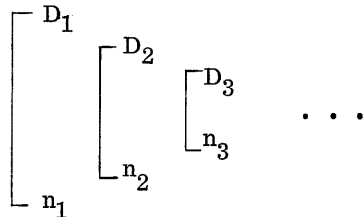
6.3.1 DO LOOP EXECUTION

The initial value of i, m_1, is increased by m_3 and compared with m_2 after executing the DO loop once, and if i does not exceed m_2, the loop is executed a second time. Then, i is again increased by m_3 and again compared with m_2; this process continues until i exceeds m_2. Control then passes to the statement immediately following n, and the DO loop is satisfied.

Should m_1 exceed m_2 on the initial entry to the loop, the loop is executed once and control is passed to the statement following n. When the DO loop is satisfied, the index variable i is no longer well defined. If a transfer out of the DO loop occurs before the DO is satisfied, the value of i is preserved and may be used in subsequent statements.

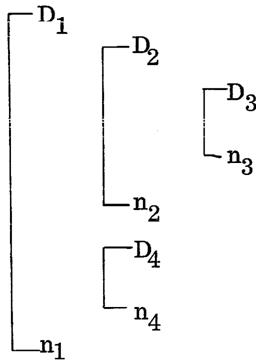
6.3.2 DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may be to any level. The last statement of a nested DO loop must either be the same as the last statement of the outer DO loop or occur before it. If $D_1, D_2, \ldots, D_m$ represent DO statements where the subscripts indicate that D_1 appears before D_2, D_2 appears before D_3 and $n_1, n_2, \ldots, n_m$ represent the corresponding limits of the D_i, then n_m must appear at or before n_{m-1}.

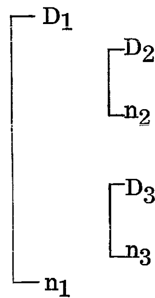


Examples:

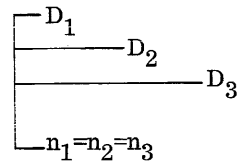
DO loops may be nested in common with other DO loops:




```
DO 1 I=1,10,2
     .
     .
     .
DO 2 J=1,5
     .
     .
     .
DO 3 K=2,8
     .
     .
     .
3 CONTINUE
     .
     .
     .
2 CONTINUE
     .
     .
     .
DO 4 L=1,3
     .
     .
     .
4 CONTINUE
     .
     .
     .
1 CONTINUE
```



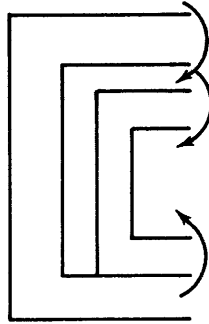

```
DO 100 L=2,LIMIT
     .
     .
     .
DO 10 I=1,10
     .
     .
     .
10 CONTINUE
     .
     .
     .
DO 20 K=K1,K2
     .
     .
     .
20 CONTINUE
     .
     .
     .
100 CONTINUE
```



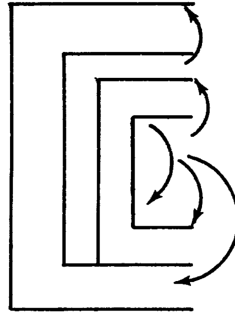

```
     DO 5 I=1,5
     DO 5 J=I,10
     DO 5 K=J,15
        .
        .
        .
5    A = B*C
```

6.3.3 DO LOOP TRANSFER

In a DO nest, a transfer may be made from one DO loop into a DO loop that contains it, but should not be made from the outer DO loop to the inner DO loop without first executing the DO statement of the inner DO loop.



Not Allowed



Allowed

One exception is allowed: once the DO statement has been executed and before the loop is satisfied, control may be transferred out of the DO range to perform some calculation and then transferred into the range of the DO. The return must not be made to the terminal statement.

When a statement is the terminal statement of more than one DO loop, the label of that terminal statement may not be used in any GO TO or arithmetic IF statement in the nest, except in the range of the innermost DO.

The following example is not acceptable since the statement GO TO 3 does not occur from the innermost DO loop.

```
3 IF(A(I))2,5,5
  DO 2 I=1,M
  GO TO 3
5 DO 2 J=1,N
1 A(I)=A(I)+B(I,J)
2 CONTINUE
```

(statement number 2 causes index to increment for inner DO loop, but not for the outer DO)

6.4 CONTINUE STATEMENT

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 CONTINUE is used elsewhere in the source program it acts as a do-nothing instruction and control passes to the next sequential program statement.

6.5 PAUSE STATEMENT

PAUSE

PAUSE n

n$\leq$5 octal digits without an O prefix or B suffix. PAUSE n rolls out the program and requests operator action at the station submitting the job. The words PAUSE n are displayed as a dayfile message. An operator entry from the console can continue or terminate the program. Program continuation proceeds with the statement immediately following PAUSE. If n is omitted, it is understood to be blank.

6.6 STOP STATEMENT

STOP

STOP n

n$\leq$5 octal digits without an O prefix or B suffix. STOP terminates the program execution and returns control to the monitor. If n is omitted, it is understood to be blank.

6.7 RETURN STATEMENT

A procedure subprogram must contain one or more RETURN statements to indicate the end of logic flow within the subprogram and return control to the calling program. If omitted, the successful execution of that subprogram will terminate the entire program.

In function subprograms, control returns to the statement containing the function reference. In a subroutine subprogram, control returns to the next executable statement following the CALL. A RETURN statement in the mainprogram causes an exit to the monitor.

6.8 END STATEMENT

END must be the final statement in a program or subprogram. It is executable in the sense that it effects termination of the program. The END statement may not be labeled.

The END statement may include the name of the program or subprogram which it terminates; however, any information appended to the END statement is ignored by the compiler.

USASI FORTRAN, X3. 9-1966, does not allow END as the last executable statement.

PROGRAM, PROCEDURES AND SUBPROGRAMS 7

7.1 SOURCE PROGRAM

A source program consists of a main program and optionally one or more auxiliary procedures and subprograms. The subprograms can be compiled separately and combined with the main program for execution.

7.2 MAIN PROGRAM

The first statement of a main program should be one of the following forms where name is an alphanumeric identifier of 1-7 characters. The parameter list is optional on all forms. If the first card of a program is not one of the following forms, a PROGRAM with a blank name and files of INPUT and OUTPUT are assumed. If more files than INPUT and OUTPUT are necessary, a PROGRAM card is required.

The form of the PROGRAM statement is:

PROGRAM name ($f_1,\ldots,f_n$)

The f_i represent the names of all input/output files required by the main program and its subprograms. n must not exceed 24. These parameters may be changed at execution time. At compile time, they must satisfy the following conditions:

1. The file name INPUT (references standard input unit) must appear if any READ statement is included in the program or its subprograms.

2. The file name OUTPUT (references standard output unit) must appear if any PRINT statement is included in the program or its subprograms. OUTPUT is required for obtaining a listing of execution diagnositcs.

3. The file name PUNCH must appear if any PUNCH statement is included in the program or its subprograms.

4. The file name TAPE i, must appear if a READ (i,n), WRITE (i,n), READ (i), or WRITE (i) statement is included in the program or its subprogram. (i is defined in section 10).

5. If I is an integer variable name for a READ (I,n) WRITE (I,n), READ (I), or WRITE (I) statement which appears in the program or its subprograms, the file names TAPE $i_1,\ldots,$TAPE i_k must appear. The integers $i_1,\ldots,i_k$ must include all values which are assumed by the variable I. The file name TAPE I may not appear in the list of arguments to the main program.

File names may be made equivalent at compile time. A PROGRAM statement having specified buffer lengths will be accepted, but the 7600 compiler will ignore them. In the list of parameters, equivalenced file names must follow those to which they are made equivalent. Their corresponding parameter positions may not be changed at execution even though the names of the files to which they are made equivalent may be changed at that time.

Examples:

```
PROGRAM ORB (INPUT, OUTPUT, TAPE 1 = INPUT, TAPE 2 = OUTPUT)
```

All input normally provided by TAPE 1 would be extracted from INPUT and all listable output normally recorded on TAPE 2 would be transmitted to the OUTPUT file.

7.3 PROGRAM COMMUNICATION

The main program and subprograms communicate with each other via arguments and COMMON variables. Subprograms may call or be called by any other subprogram as long as the calls are non-recursive. That is, if program A calls B, B may not call A. A calling program is a main program or subprogram that refers to another subprogram. A subroutine referenced by a program may not have the same name as the program.

7.4 SUBPROGRAM COMMUNICATION

Subprograms, functions, and subroutines use arguments as one means of communication. The arguments appearing in a subroutine call or a function reference are actual arguments. The corresponding arguments appearing with the program, subprogram, statement function, or library function name in the definition are formal arguments. One or more of the formal arguments or common variables can be used to return output to the calling program.

7.5 PROCEDURES AND SUBPROGRAMS

A FORTRAN program consists of a main program with or without auxiliary procedures and subprograms. Auxiliary sets of statements are used to evaluate frequently-used mathematical functions, to perform repetitious calculations, and to supply data specifications and initial values to the main program. 7600 FORTRAN provides six such procedures and subprograms:

Statement function

Intrinsic function

Basic external function

External function

External subroutine

Block data subprogram

The intrinsic function and the basic external function are furnished with the system. They are used to evaluate standard mathematical functions. The others are user-defined. The statement function and intrinsic function are compiled within the main program, the basic external function is furnished with the system, and the others are compiled separately. The first five are referred to as procedures, since each is an executable unit that performs its set of calculations when referenced. The first four are called functions. They return a single result to the point of reference. The last three, subprograms, are user-defined and are compiled independently. The block data subprogram supplies specifications and initial values to the main program. Table 7-1 outlines these categorical divisions.

The use of procedures and subprograms is determined by their particular capabilities and the needs of the program being written. If the program requires the evaluation of a standard mathematical function, an intrinsic function or a basic external function is used (Appendix C). If a single non-standard computation is needed repeatedly, a statement function is inserted in the program. If a number of calculations are required to obtain a single result, a function subprogram is written. If a number of calculations are required to obtain an array of values, a subroutine is written. When the program requires initial values, a BLOCK DATA subprogram is used.

7.5.1 PROCEDURE IDENTIFIERS

A procedure identifier is a symbolic name of up to seven alphanumeric characters, the first of which must be alphabetic.

USASI FORTRAN, X3.9-1966, limits all symbolic names to six characters.

There is no type associated with a symbolic name that identifies a SUBROUTINE. For a function subprogram, type is specified either implicitly by its name, explicitly in the FUNCTION statement or in a type statement. For a statement function, type is specified either implicitly by its name or explicitly in a type statement.

7.5.2 FORMAL ARGUMENTS

Formal arguments appear within the FUNCTION or SUBROUTINE statement or in the statement function definition and serve only to allocate data values in these auxiliary routines. For this reason, they are often referred to as dummy arguments.

Formal arguments may be the names of arrays, simple variables, library functions (basic external functions), and subprograms (FUNCTION and SUBROUTINE). Since formal arguments are local to the subprogram containing them, they may be the same as names appearing outside the procedure.

No element of a formal argument list may appear in an EQUIVALENCE, COMMON, or DATA statement within a subprogram. If it does, a compiler diagnostic results.

When a formal argument represents an array, it must be dimensioned within the subprogram. If it is not declared, the array name must appear without subscripts and only the first element of the array is available to the subprogram.

TABLE 7-1. SUBDIVISION OF PROCEDURES AND SUBPROGRAMS

<table>
<tr><th>Statement Function</th><th>Intrinsic Function</th><th>Basic External Function</th><th>External Function</th><th>External Subroutine</th><th>Block Data Subprogram</th></tr>
<tr><td>User-defined</td><td colspan="2">Compiler-defined</td><td colspan="3">User-defined</td></tr>
<tr><td colspan="2">Compiled within the referencing program</td><td>Not Compiled
- LIBRARY -</td><td colspan="3">Compiled externally to the referencing program</td></tr>
<tr><td colspan="5">PROCEDURE: Any defined calculation that can be referenced and which will exchange values between reference and definition through a list of arguments.</td><td></td></tr>
<tr><td colspan="2"></td><td colspan="3">EXTERNAL PROCEDURE: a procedure that is defined externally to the program unit that references it.</td><td></td></tr>
<tr><td colspan="4">FUNCTION: a procedure that supplies a single result to be used at the point of reference. It can also modify the arguments.</td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="2">EXTERNAL FUNCTION: a function defined externally to the program unit that references it.</td><td colspan="2"></td></tr>
<tr><td colspan="3"></td><td colspan="3">SUBPROGRAM: a user-defined set of statements compiled independently from the program unit which references it or to which it supplies specifications and initial values.</td></tr>
<tr><td colspan="3"></td><td colspan="2">PROCEDURE SUBPROGRAM: an external procedure that is defined by FORTRAN statements.</td><td>SPECIFICATION SUBPROGRAM: a subprogram without reference that supplies specifications and initial values to a main program.</td></tr>
</table>

7.5.3 ACTUAL ARGUMENTS

Actual arguments appear within a CALL statement referencing a SUBROUTINE or in any of the function references. They are associated with the corresponding formal arguments in the auxiliary procedure being referenced and serve to transmit values on a one-to-one basis. Accordingly, formal and actual arguments must agree in order and type. The compiler does not check for matching of type. The permissible forms of actual arguments are the following:

Arithmetic expression

Logical expression

Relational expression

Constant

Simple or subscripted variable

Array name

FUNCTION subprogram name

SUBROUTINE subprogram name

Basic external function name

Intrinsic function name

A calling program statement label identified by suffixing the label with the character S. This form should be used only when calling DUMP or PDUMP.

NOTE

The value of a constant used as an actual argument may be changed by the called routine.

Input/output buffer names may not be used as actual parameters but the following is allowed:

```
PROGRAM X(OUTPUT, TAPE 6 = OUTPUT)
CALL SUB (6,X)
    .
    .
    .
END
SUBROUTINE SUB(I, B)
    .
    .
    .
WRITE (I,n), B
    .
    .
    .
END
```

7.6 STATEMENT FUNCTION

A statement function is defined by a single expression and applies only to the program or subprogram containing the definition. The name of the statement function is an alphanumeric identifier. A single value is always associated with the name. A statement function has the form:

name $(p_1, \ldots, p_n) = E$

The p_i are formal arguments and must be simple variables. The maximum value of n is 60. E can be any arithmetic or logical expression. It may contain a reference to a library function, statement function, or function subprogram.

During the compilation, the statement function definition is compiled once at the beginning of the program and a transfer is made to this portion of the program whenever a reference is made to the statement function. A statement function reference has the form:

name $(p_1, \ldots, p_n)$

name is the alphanumeric identifier of the statement function. The actual arguments p_i may be any arithmetic expressions.

The statement function name must not appear in a DIMENSION, EQUIVALENCE, COMMON, or EXTERNAL statement. The name can appear in a type declaration but cannot be dimensioned. Statement function names must not appear as actual or formal arguments.

Actual and formal arguments must agree in number, order and mode. The mode of the evaluated statement function is determined by the name of the arithmetic statement function.

A statement function must precede the first statement in which it is used, but it must follow all declarative statements (DIMENSION, type, etc.) which contain symbolic names referenced in the statement function. All statement functions should precede the first executable statement. Otherwise, an informative diagnostic is provided.

A statement function may not reference itself and if such an attempt is made, a fatal diagnostic is provided.

7.7 SUPPLIED FUNCTION

To evaluate frequently-used mathematical functions, 7600 FORTRAN supplies predefined calculations as well as references to library routines contained in the system. The predefined calculations are called intrinsic or in-line functions and the references to the library routines are called basic external functions.

The intrinsic or in-line function inserts a simple set of calculations into the object program at compile time. The basic external function deals with more complex evaluations by inserting a reference to a library routine in the object program. The names of the supplied functions, their data types, and permissible arguments are predefined (Appendix C). References using these functions must adhere to the format defined in the tables. The type of a supplied function cannot be changed by a type statement.

7.7.1 INTRINSIC FUNCTIONS

An intrinsic function is a compiler-defined set of calculations that is inserted in the referencing program at compile time. The form of the intrinsic function and its reference are identical to the statement function outlined above. The table in Appendix C lists the intrinsic functions available. The name of an intrinsic function listed in this table must satisfy all of the following requirements:

The name must not appear in an EXTERNAL statement or be the name of a statement function

The name must not appear in a type statement declaring it to be other than the type specified in the table

Every appearance of the name must be followed by a list of parameters enclosed in parentheses, unless the name is in a type statement.

7.7.2 BASIC EXTERNAL FUNCTIONS

A basic external function is a call on one of the predefined library routines included with the system. These library routines are used to evaluate standard mathematical functions such as sine, cosine, square root, etc. A basic external function is referenced by the appearance of the function name with appropriate arguments in an arithmetic or logical statement. A list of basic external functions is given in Appendix C.

7.8 SUBPROGRAMS

Subprograms are used to implement programming capability beyond the limitations of supplied functions and the statement function. Although written as a subset of another program, the subprogram is compiled separately. It has its own independent variables, and its use is not limited to communication with the program for which it was written. Procedure subprograms handle routine calculations unique to the user. Specification subprograms are used to enter values into COMMON and supply program specifications.

Procedure subprograms are of two kinds: FUNCTION and SUBROUTINE. The FUNCTION subprogram is referenced by the appearance of its name in the calling program. The SUBROUTINE subprogram is referenced by a CALL statement in the calling program. A procedure subprogram returns control to a calling program through one or more RETURN statements. Because they are independent programs, procedure subprograms must terminate with an END statement to signal to the compiler that the physical end of the source program has been reached. An END statement is generated as a STOP. If a procedure subprogram does not contain at least one RETURN statement, the successful execution of that subprogram will terminate the entire program.

The fundamental difference between FUNCTION and SUBROUTINE subprograms is given in table 7-2.

There is one type of specification subprogram, the BLOCK DATA subprogram.

7.8.1 FUNCTION SUBPROGRAM

A FUNCTION subprogram is a collection of FORTRAN statements headed by a FUNCTION statement and written as a separate program to perform a set of calculations when its name appears in the referencing program. The mode of the function is determined by a type indicator or the name of the function. The first statement of a FUNCTION subprogram must be one of the following forms where name is an alphanumeric identifier and the p_is are formal arguments with n assuming any integer value up to 60. A FUNCTION statement must have at least one argument.

FUNCTION name $(p_1,\ldots,p_n)$

type FUNCTION name $(p_1,\ldots,p_n)$

Type is REAL, INTEGER, DOUBLE PRECISION, DOUBLE, COMPLEX, or LOGICAL. When the type indicator is omitted, the mode is determined by the first character of the function name.

The FUNCTION name must be assigned a value by appearing at least once in the subprogram as any one of the following:

The left-hand identifier of a replacement statement

An element of an input list

An actual argument of a subroutine reference

If not, the value returned is undefined. The name of a FUNCTION must not appear in an array declaration.

The FUNCTION subprogram accepts arguments from the referencing program through the argument list and returns a value through the FUNCTION name. The FUNCTION subprogram may define and redefine one or more arguments and return these values as is done in a SUBROUTINE (section 7.8.2).

When a function reference is encountered in an expression, control transfers to the FUNCTION subprogram indicated. When RETURN or END is encountered in the FUNCTION subprogram, control returns to the statement containing the function reference. An assigned GO TO statement transfers control to the statement indicated.

Example:

Referencing program	Function subprogram

```
PROGRAM IMPED                    FUNCTION VECTOR (X,Y)
      .                          Z=SQRT(X*X+Y*Y)
      .                          IF (Z)2,2,3
      .                        2 VECTOR=0.
RESULT=VECTOR (A,B)              GO TO 5
      .                        3 VECTOR=Z
      .                        5 RETURN
      .                          END
END
```

TABLE 7-2. DIFFERENCES BETWEEN A FUNCTION AND SUBROUTINE SUBPROGRAM

Function	Subroutine
Referenced by the name appearing in an arithmetic or logical statement	Referenced by a CALL statement
Must have one or more arguments	Need not have any arguments
Name is typed by first letter or by the type designation appearing before the word FUNCTION	No type associated with name

The function subprogram is referenced by the appearance of the name and list in the statement

```
RESULT=VECTOR (A,B)
```

The values represented by the actual arguments A and B are communicated to the subprogram through the dummy arguments X and Y.

The first calculation in the subprogram involves the appearance of a secondary reference: SQRT. This reference passes the calculated value in the parentheses to the basic external function for obtaining a square root. The result is returned to the subprogram and placed in storage location Z. Z is then tested to see if it is positive. If not, the function name VECTOR is equated to zero and that value is returned to the reference; if it is positive, the function name VECTOR is equated to that positive value and returned to the reference.

The following example shows how a FUNCTION subprogram can establish a value for the FUNCTION name by using an input statement rather than an arithmetic statement.

Referencing program

```
  PROGRAM INPUT
  INTEGER FUNCT
  J = FUNCT (1)
  WRITE (3,1) J
1 FORMAT (I5)
  STOP
  END
```

FUNCTION subprogram

```
  INTEGER FUNCTION FUNCT (I)
  READ (1,1) FUNCT
1 FORMAT (I2)
  RETURN
  END
```

Since the subprogram is intended to deal with integer values and its name is implicitely real, the name is typed integer in the referencing program and in the FUNCTION statement of the subprogram. The subprogram is referenced by the statement

J = FUNCT (1)

which arbitrarily passes the constant 1 as an actual argument. It enters the subprogram through the dummy argument I in the FUNCTION statement but is never used. This step is performed solely to satisfy the requirements of a FUNCTION subprogram. The subprogram reads in the value from a card and stores it in the location designated by the name of the FUNCTION subprogram, where it is available to the referencing program which stores it in J and then prints it out.

7.8.2 SUBROUTINE SUBPROGRAM

A SUBROUTINE subprogram is a collection of FORTRAN statements headed by a SUBROUTINE statement and written as a separate program to perform a set of calculations when called by a referencing program. It may return none, one, or more values. A value or type is not associated with the subroutine name itself.

The first statement of a subroutine subprogram must have one of the following forms:

SUBROUTINE name

or

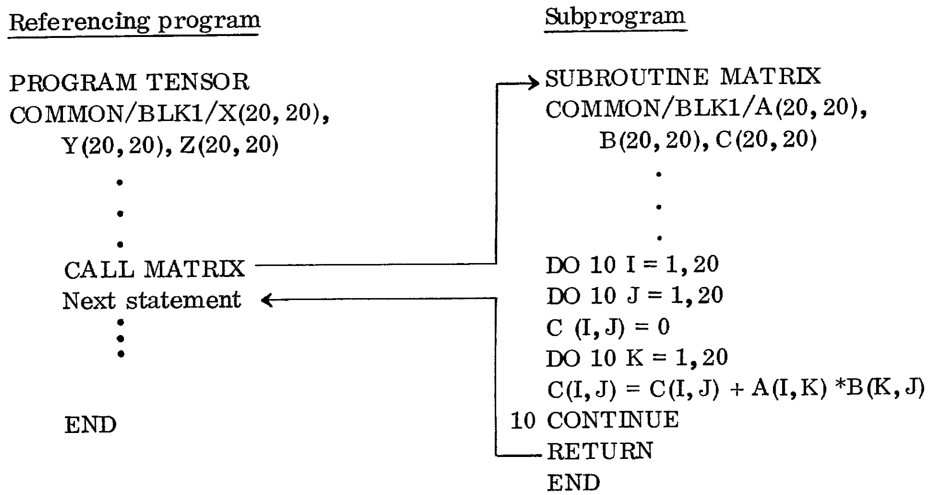
SUBROUTINE name $(p_1, \ldots, p_n)$

name is an alphanumeric identifier and p_i are formal arguments; n may be 1 to 60.

A CALL statement (section 7.9) transfers control from the calling program to the subroutine. A RETURN or END statement (section 6.7 and 6.8) returns control to the next executable statement following the CALL statement in the referencing program.

The SUBROUTINE subprogram may accept arguments from the calling program and can either return none, one, or more results through its arguments or in COMMON.

Example:



The referencing program reserves storage for three successive arrays in labeled COMMON. It is assumed that two of these arrays, X and Y, have values stored in them before the CALL statement is reached. The CALL statement transfers control to the subroutine without passing any arguments. The subroutine performs the matrix multiplication of the first two arrays and stores the results in the third. Control is returned to the next statement after the CALL in the referencing program. The subroutine obtains the values for its calculations from the labeled common block and returns the results it derives to the same labeled common block.

7.8.3 LIBRARY SUBROUTINES

FORTRAN contains several built-in subroutine subprograms which may be referenced by a program with a CALL statement. i must be an integer variable or constant; j is an integer variable.

CALL SLITE (i)

Turn on sense light i. If i = 0, turn all sense lights off. i is 0 to 6; if i > 6, the results are undefined and no diagnostic is provided.

CALL SLITET (i, j)

If sense light i is on, set j = 1, if sense light i is off, set j = 2; then turn sense light i off. i is 1 to 6. If i is out of range, the results are undefined.

CALL SSWTCH (i, j)

If sense switch i is down, set j = 1. If sense switch i is up, set j = 2. i is 1 to 6. If i is out of the range, the results are undefined. The switches are set by SCOPE control cards.

CALL OVERFL (j)†

If a floating point overflow condition exists, set j = 1. If no overflow exists, set j = 2; and set the machine to a no overflow condition.

CALL DVCHK (j)†

If division by zero occurred, set j = 1 and clear the indicator; if division by zero did not occur, set j = 2.

CALL SECOND (t)

Returns CP time from start of job in seconds in floating point format to three decimal places. t is a real variable.

CALL EXIT

Terminate program execution and return control to the monitor.

CALL DUMP ($a_1, b_1, f_1, \ldots, a_n, b_n, f_n$)

CALL PDUMP ($a_1, b_1, f_1, \ldots, a_n, b_n, f_n$) ($n \leq 20$)

Dump storage on OUTPUT file in indicated format. For PDUMP, control returns to the calling program; for DUMP, execution terminates and control returns to the operating system. If no arguments are provided, an octal dump of all storage occurs.

The a_i and b_i are SCM core addresses, variables, or statement numbers. They indicate the first word and the last word of the storage area to be dumped.

The statement numbers must be 1 to 5 digits trailed by an S; CALL DUMP (10S, 20S, 0). If b_i is the last statement of a DO loop, then b_iS is not allowed to be used as the last word of the storage area to be dumped.

The dump format indicators are as follows:

f = 0 or 3 octal dump

f = 1 real dump

f = 2 integer dump; if bit 48 is set (normalize bit)

† Currently J is always set to 2 (see LEGVAR in Appendix C).

7.9 CALL STATEMENT

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

```
CALL name
    or
CALL name (p1,...,pn)
```

name is the name of the subroutine being called, and p is an actual argument; n is 1 to 60. The name should not appear in any declarative statement in the calling program, with the exception of the EXTERNAL statement when name is also an actual argument.

The CALL statement transfers control to the subroutine. When a RETURN statement is encountered in the subroutine, control is returned to the next executable statement following the CALL statement in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the DO loop is satisfied. The CALL statement is executed each time the terminal statement is reached.

Examples:

1.
```
      SUBROUTINE BLDX(A,B,W)
      W=2.*B/A
      RETURN
      END
```

<u>Calls</u>

```
      CALL BLDX(X(I),Y(I),W)
      CALL BLDX(X(I)+H/2.,Y(I) + C(J),PROX)
      CALL BLDX(SIN(Q5),EVEC(I+J),OVEC(L))
```

2.
```
      SUBROUTINE MATMULT
      COMMON/ITRARE/X(20,20),Y(20,20),Z(20,20)
      DO 10 I = 1,20
      DO 10 J =1,20
      Z(I,J)=0.
      DO 10 K=1,20
   10 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
      RETURN
      END
```

Operations in MATMULT are performed on variables contained in the common block ITRARE. This block must be defined in all calling programs.

```
COMMON/ITRARE/AB(20,20), CD(20,20), EF(20,20)
CALL MATMULT
```

3.
```
      SUBROUTINE AGMT (SUB, ARG)
      COMMON/ABL/XP(100)
      ARG = 0.
      DO 5 I =1,100
    5 ARG = ARG + XP(I)
      CALL SUB
      RETURN
      END
```

Here the dummy argument SUB is used to transmit another subprogram name. The call to SUBROUTINE AGMT might be CALL AGMT (MULT, FACTOR), where MULT is specified in an EXTERNAL statement. (section 7.10)

7.10 EXTERNAL STATEMENT

When the actual argument list which calls a function or subroutine program contains a function or subroutine name, that name must be declared in an EXTERNAL statement.

EXTERNAL $name_1$, $name_2$, . . .

The EXTERNAL statement must precede the first statement of any program which calls a function or subroutine subprogram using the EXTERNAL name. When it is used, EXTERNAL always appears in the calling program; it may not be used with statement functions. If it is, a compiler diagnostic is provided.

Examples:

1. A function name used as an actual argument requires an EXTERNAL statement.

Calling Program Reference

```
            .
            .
            .
EXTERNAL SIN
CALL PULL(SIN,R,Q)
            .
            .
            .
```

Called Subprogram

```
SUBROUTINE PULL(X,Y,Z)
          .
          .
          .
Z=X(Y)
          .
          .
          .
```

But a function reference used as an actual argument does not need an EXTERNAL statement.

Calling Program Reference

```
          .
          .
          .
CALL PULL(SIN(R),Q)
          .
          .
          .
```

Called Subprogram

```
SUBROUTINE PULL(X,Z)
          .
          .
          .
Z=X
          .
          .
          .
END
```

2. A subroutine used as an actual argument must have its name declared in an EXTERNAL statement in the calling program.

```
COMMON/ABL/ALS(100)
EXTERNAL RTENTA, RTENTB
CALL AGMT(RTENTA,V1)
CALL AGMT(RTENTB,V1)
```

When a subprogram name appears as an actual argument, any arguments to be associated with a call of this subprogram can be passed via actual arguments or COMMON.

Example:

Calling Program

```
EXTERNAL ADDER
        .
        .
        .
CALL SUB(ADDER,A,B)
        .
        .
        .
```

Called Subprogram

```
SUBROUTINE SUB(X,Y,Z)
        .
        .
        .
CALL X(Y,Z)
        .
        .
        .
END
```

CALL SUB(ADDER(A,B)) would imply that ADDER is a function reference, not a subroutine name.

7.11 ENTRY STATEMENT

The statement provides alternate entry points to a FUNCTION or SUBROUTINE subprogram.

ENTRY name

Name is an alphanumeric identifier, and may appear within the subprogram only in the ENTRY statement. Each entry identifier must appear in a separate ENTRY statement. The dummy arguments, if any, appearing with the FUNCTION or SUBROUTINE statement do not appear with the ENTRY statement. ENTRY may appear anywhere within the subprogram except it should not appear within a DO; ENTRY statement cannot be labeled. The first executable statement following ENTRY becomes an alternate entry point to the subprogram.

In the calling program, the reference to the entry name is made just as if reference were being made to the FUNCTION or SUBROUTINE in which the ENTRY is imbedded. The name may appear in an EXTERNAL statement, and if a function entry name, in a type statement.

The ENTRY name may not be given type explicitly in the defining program; it assumes the same type as the name in the FUNCTION statement.

Examples:

```
      FUNCTION JOE(X,Y)
   10 JOE=X+Y
      RETURN
      ENTRY JAM
      IF (X.GT.Y) 10,20

   20 JOE=X-Y
      RETURN
      END
```

This could be called from the main program as follows:

```
      .
      .
      .
      Z=A+B-JOE(3.*P,Q-1)
      .
      .
      .
      R=S+JAM(Q,2.*P)
```

USASI FORTRAN, X3.9-1966, does not specify the ENTRY statement.

7.12 VARIABLE DIMENSIONS IN SUBPROGRAMS

In many subprograms, especially those performing matrix manipulation, the programmer may wish to vary array dimensions each time the subprogram is called.

This is accomplished by specifying the array name and its dimensions as dummy arguments in the FUNCTION or SUBROUTINE statement. The corresponding actual arguments specified in the calling program are used by the called subprogram. The maximum dimensions that any given array may assume are determined by dimensions in a DIMENSION, COMMON or type statement in the program that initially declared the array.

The dummy arguments representing the array dimensions must be simple integer variables. The array name must also be a dummy argument. The actual argument representing the array dimensions must have integer values.

The total number of elements of the corresponding array in the subprogram may not exceed the total number of elements of a given array in the program that initially declared the array.

Example:

Consider a simple matrix add routine written as a subroutine:

```
    SUBROUTINE MATADD (X,Y,Z,M,N)
    DIMENSION X (M,N),Y(M,N),Z(M,N)
    DO 10 I = 1,M
    DO 10 J = 1,N
 10 Z(I,J)=X(I,J)+Y(I,J)
    END
```

The arrays X, Y, Z and the variable dimensions M, N all appear as dummy arguments in the SUBROUTINE statement and also in the DIMENSION statement as shown. If the original calling program contains the array allocation declaration

```
DIMENSION A(10,10),B(10,10),C(10,10),E(5,5),G(5,5),H(10,10)
```

The program may call the subroutine MATADD from several places within the main program as follows:

```
CALL MATADD(A,B,C,10,10)
CALL MATADD(E,F,G,5,5)
CALL MATADD(B,C,A,10,10)
CALL MATADD(B,C,H,10,10)
```

The compiler does not check to see if the limits of the array established by the DIMENSION statement in the main program are exceeded.

The variable dimensions need not be passed as arguments; they can be in COMMON or computed internally.

7.13 PROGRAM ARRANGEMENT

FORTRAN assumes that all statements and comments appearing between a PROGRAM, SUBROUTINE, or FUNCTION statement and an END statement belong to one program. A typical arrangement of a set of main program and subprograms follows (Also see appendix E.)

```
PROGRAM WHAT
     .
     .
     .
END

SUBROUTINE S1(A,B)
     .
     .
     .
END

SUBROUTINE S2
     .
     .
     .
END

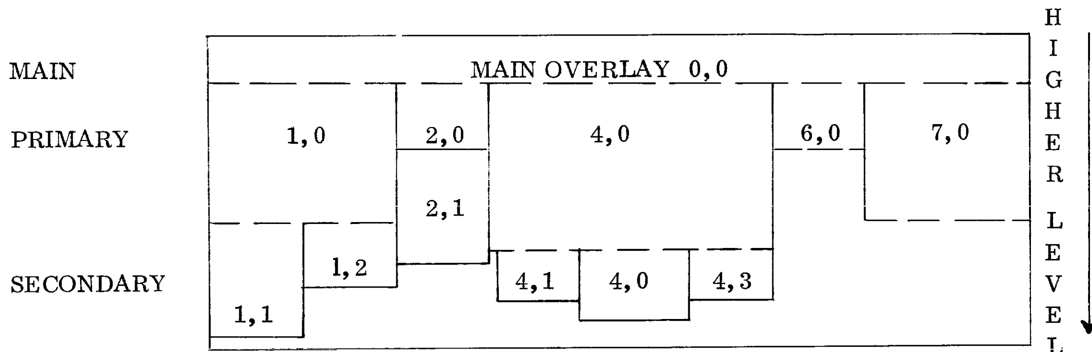
REAL FUNCTION F1(P1)
     .
     .
     .
END
```

OVERLAYS 8

An overlay is a portion of a program written on a file in absolute form and loaded at execution time without delay for relocation. The user defines an overlay with the OVERLAY card. He calls it with the CALL OVERLAY statement.

8.1 LEVELS

Levels are used to describe the sequence of loading overlays and to specify which sections of code are to overlay others. In 7000 SCOPE .V1. 0, there are three levels of overlaying, MAIN, PRIMARY, and SECONDARY. Up to three overlays may be in core simultaneously. They are usually loaded contiguously. The primary or secondary levels may be replaced by other overlays. The following diagram demonstrates the relationship of the levels when they are loaded into core. This example shows a number of different core loads which might exist for a single job:



8.2 IDENTIFICATION

Overlays may be loaded from specified files. A single overlay may be loaded only from a single file, although many files may be used for loading by a single job. An overlay is identified by its level number. The level number is a pair of two-digit octal numbers (0-77). The first number is the primary level, the second is the secondary level. An overlay with a non-zero primary level and a zero secondary level (1,0) is a primary overlay. Any overlay with the same primary level and a non-zero secondary level (1,1) is associated with and subordinate to the corresponding primary and is called a secondary overlay. This difference is significant when overlays are loaded. Level 0,0 is reserved for the initial, or main overlay which is neither primary nor secondary; it is a special case which remains in memory during overlay execution. Overlay numbers (0,1) to (0,77) are illegal.

The main overlay (0, 0) is loaded first. All primary overlays are loaded at the same point immediately following the main overlay. Secondary overlays are loaded immediately following their associated primary overlays. Loading the next primary overlay destroys the first loaded primary overlay and any associated secondary overlays. Likewise, the loading of a secondary overlay destroys a previously loaded secondary overlay.

8.3 COMPOSITION

Each overlay must have at least one program having the characteristics of a FORTRAN main program.

An overlay may consist of one or more FORTRAN or COMPASS programs. The program name becomes the primary entry point for the overlay through which control passes when the overlay is called. An overlay cannot reference entry points in higher level overlays. The only method of reference for a MAIN overlay to primary and secondary overlays is through the CALL OVERLAY statement. However, the primary overlay may reference any entry point in the MAIN overlay, while the secondary overlay may reference any entry point in the primary or MAIN overlay.

Blank common and labeled common may be defined in any level overlay and referenced by that overlay and higher level overlays. (The same rules apply as for entry points).

An OVERLAY is established by an OVERLAY card which precedes the program cards. The overlay consists of all programs appearing between the OVERLAY card and the next OVERLAY card or an end-of-file or an end-of-record.

8.4 CALL

Overlays are called by the following statement:

CALL OVERLAY (fn, ℓ_1, ℓ_2, p)

OVERLAY FORTRAN subroutine which translates the FORTRAN call into a call to the loader

fn	variable name of the location containing the name of the file (left justified display code) which includes the overlay
ℓ_1	primary level of the overlay
ℓ_2	secondary level of the overlay
p	recall parameter. If p equals 6HRECALL, the overlay is not reloaded if it is in memory

The first three parameters must be specified; the absence of any one could result in a MODE error at execution time. The levels appearing on the OVERLAY loader card are always octal. The normal mode for parameters in FORTRAN calls is decimal. This fact should be considered when coding the ℓ_1, ℓ_2 parameters. The programmer can keep his level numbers straight by using octal notation on both control and call cards.

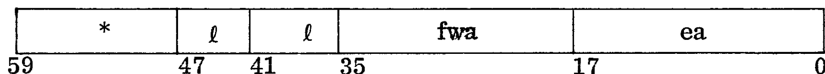
If uniqueness is ensured at execution time, more than one overlay may be created with the same level numbers. Uniqueness is determined by the level numbers, the file name from which the OVERLAY is to be loaded, and the position of the overlay on the file. Since the loader selects the first overlay encountered on the specified file with level numbers which match those in the call, it is possible to position a number of overlays on a file with the same identifier and by properly sequencing the calls thereto, have available a number of different overlays.

Loading from a file requires an end-around search of the file for the specified overlay; this can be time consuming in large files. When speed is essential, each overlay should be written to a separate file.

8.5 OVERLAY FORMAT

Each overlay consists of a logical record in the following format:

Word 1



*	50_8 (specified on overlay header)
ℓ_1	Primary overlay level
ℓ_2	Secondary overlay level
ea	Entry point to the overlay
fwa	First word address of overlay (overlay is loaded at fwa)

Word 2 through end of record: 60-bit data words.

8.6 LOADER CARDS

Loader cards are processed directly by the loader. They provide the loader with information necessary for generating overlays. All loader cards must precede the subprogram text to be loaded. Formats are the same as for SCOPE control cards. However, if they are in the FORTRAN decks, the loader cards must be punched in columns 7 through 72.

8.7 OVERLAY CARDS

OVERLAY (fn, ℓ_1, ℓ_2, Cnnnnnn)

fn — File name onto which the generated overlay is to be written

ℓ_1 — Primary level number

ℓ_2 — Secondary level number

(ℓ_1, ℓ_2) must be (0, 0) for first overlay card and must be in octal[†]

Cnnnnnn optional; nnnnnn can be up to 6 octal digits. If absent, overlay is loaded normally. If present, overlay is loaded nnnnnn words from the start of blank common. This provides a method for changing the size of blank common at execution time.

The first overlay card must have an fn. Subsequent cards may omit fn, and the overlay is written on the same fn.

Each OVERLAY card must be followed by a program card. The program card for the main overlay must specify all needed file names, such as INPUT, OUTPUT, TAPE 1, etc, for all overlay levels. File names should not appear in program cards for other than the (0, 0) OVERLAY.

The groups of relocatable decks processed by the loader in forming overlays must be presented to the loader in proper order. This requires that the 0, 0 overlay group be first. After this may be any primary group followed by all of its associated secondary groups, then any other primary group followed by its associated secondary groups, etc.

[†] Level numbers given in the CALL OVERLAY, however, are decimal; e.g., the overlay card for overlay 1, 9 would be OVERLAY(fn, 1, 11) and its call would be CALL OVERLAY(fn, 1, 9)

Example:

```
EXAM, S70.
RUN(S)
LGO.
7/8/9
OVERLAY(XFILE,0,0)
PROGRAM ONE(INPUT,OUTPUT, PUNCH)
              .
              .
              .
CALL OVERLAY(5HXFILE,1,0)
              .
              .
              .
STOP
END
OVERLAY(XFILE,1,0)
PROGRAM ONE ZERO
CALL OVERLAY(5HXFILE,1,1)
              .
              .
              .
RETURN
END
OVERLAY(XFILE,1,1)
PROGRAM ONE ONE
              .
              .
              .
RETURN
END
6/7/8/9
```

INPUT/OUTPUT FORMATS 9

Data is transferred between storage and the files for external units in one of two modes: binary coded decimal (BCD) and binary. BCD transmissions are dependent on the structure of the data they contain and as such must have their format specified. This is accomplished by means of a FORMAT statement. Binary data is transferred as a single string and does not require a format specification. Both forms require an input/output statement that identifies the unit involved and specifies the list of data to be moved.

9.1 INPUT/OUTPUT LIST

The list portion of an input/output statement indicates the data items and the order, from left to right, of transmission. The input/output list can contain any number of elements. List items may be array names, simple or subscripted variables, or an implied DO loop. Items are separated by commas, and their order must correspond to any FORMAT specification associated with the list. A double precision or complex element assumes a word-pair to be transmitted. There is no check whether the type of the list element matches the type of the conversion specified. External records are always read or written until the list is satisfied.

Subscripts in an input/output list may be any of the following forms in which c and k are unsigned integer constants and v is a simple integer variable:

Form	Example
(c)	(4)
(v)	(I)
(v±k)	(J+3)
(c*v)	(5*K)
(c*v±k)	(2*L-8)

Examples:

```
READ 100,A,B,C,D
READ 200,A,B,C(I),D(3,4),E(I,J,7),H
READ 101,J,A(J),I,B(I,J)
READ 102,DELTA(5*J+2,5*I-3,5*K),C,D(I+7)
```

The integer variable in a list must be previously defined, or it must be defined within an implied DO loop in the list.

Examples:

```
READ 300,A,B,C,(D(I),I=1,10),E(5,7),F(J),(G(I),H(I),I=2,6,2)
READ 400,I,J,K,(((A(II,JJ,KK),II=1,I),JJ=1,J),KK=1,K)
READ 500,((A(I,J),I=1,10,2),B(J,1),J=1,5),E,F,G(L+5,M-7)
```

9.2 ARRAY TRANSMISSION

An entire array or any part of an array can be transferred as a single specification in an input/output list by using an implied DO loop. An implied DO loop is of the form:

$(((A(I, J, K), L_1=m_1, m_2, m_3), L_2=n_1, n_2, n_3), L_3=p_1, p_2, p_3)$

m_i, n_i, p_i	Unsigned integer constants or simple integer variables. If m_3, n_3, or p_3 is omitted, it is assumed equal to 1.
I, J, K	Subscripts of A
L_1, L_2, L_3	Index variables I, J, K in same order

This is equivalent to the nest of DO loops:

```
      DO N L3=P1, P2, P3
      DO N L2=N1, N2, N3
      DO N L1=M1, M2, M3
N     READ 100, A (I, J, K)
```

where N is the statement number of the terminal statement and 100 is the statement number of the relative FORMAT statement.

An array name which appears without subscripts in an I/O list causes transmission of the entire array by columns.

Example:

```
DIMENSION B(10,15)
```

the statement

```
READ 13, B
```

is equivalent to

```
READ 13, ((B(I, J), I=1, 10), J=1, 15)
```

An implied DO loop can be used to transmit a simple variable more than one time. For example, the list item (A(K), B, K=1, 5) causes the transmission of the variable B five times. A list of the form K, (A(I), I=1, K) is permitted and the input value of K is used in the implied DO loop. The index variable in an implied DO list in a DATA statement should be an implicit integer.

Examples:

1. Simple implied DO loop list items.

```
    READ 400, (A(I), I=1, 10)
400 FORMAT (E20.10)
```

This statement is equivalent to the following DO loop.

```
      DO 5 I=1,10
5     READ 400, A(I)

      READ 100, ((A(JV,JX),JV=2,20,2),JX=1,30)
      READ 200, (BETA(3*JON+7),JON = JONA,JONB,JONC)
      READ 300,(((ITMSLST(I,J+1,K-2),I=1,25),J=2,N),K=IVAR,IVMAX,4)

      READ 600, (A(I),B(I),I=1,10)
600   FORMAT (F10.2,E6.1)
```

The previous statement is equivalent to the DO loop:

```
      DO 17 I = 1,10
17    READ 600, A(I),B(I)
```

2. Nested implied DO list items.

```
READ 100,(((((A(I,J,K),B(I,L),C(J,N),I=1,10),J=1,5),K=1,8),1L=1,15),N=2,7)
```

Data is transmitted in the following sequence:

```
A(1,1,1),B(1,1),C(1,2),A(2,1,1),B(2,1),C(1,2)...
...A(10,1,1),B(10,1),C(1,2),A(1,2,1),B(1,1),C(2,2)...
...A(10,2,1),B(10,1),C(2,2)...A(10,5,1),B(10,1),C(5,2)...
...A(10,5,8),B(10,1),C(5,2)...A(10,5,8),B(10,15),C(5,2)...
...A(10,5,8),B(10,15),C(5,7)
```

The following list item will transmit the array E(3,3) by columns:

```
READ 100,((E(I,J),I=1,3),J=1,3)
```

The following list item will transmit the array E(3,3) by rows:

```
READ 100, ((E(I,J),J=1,3),I=1,3)
```

3.

```
      DIMENSION MATRIX (3,4,7)
      READ 100, MATRIX
100   FORMAT (I6)
```

The above items are equivalent to the following statements:

```
      DIMENSION MATRIX (3,4,7)
      READ 100,(((MATRIX(I,J,K),I=1,3),J=1,4),K=1,7
```

The list is equivalent to the nest of DO loops:

```
      DO 5 K=1,7

      DO 5 J=1,4

      DO 5 I=1,3

5     READ 100, MATRIX(I,J,K)
```

9.3 FORMAT DECLARATION

BCD input/output statements require a FORMAT declaration which contains conversion and editing information relating to internal/external structure of the corresponding I/O list items. A FORMAT declaration has the following form:

FORMAT ($spec_1$,...,k($spec_m$,...), $spec_n$,...)

$Spec_i$	format specification
k	optional repetition factor, must be unsigned integer constant.

The FORMAT declaration is non-executable and may appear anywhere in the program. FORMAT declarations must have a statement label in columns 1-5. The compiler does not check the syntax of FORMAT statements at compile time.
The data items in an I/O list are converted from one representation to another (external/internal) according to FORMAT conversion specifications. FORMAT specifications may also contain editing codes.

Conversion specifications:

Ew. d	Single precision floating point with exponent
Fw. d	Single precision floating point without exponent
Dw. d	Double precision floating point with exponent
Gw. d	Single precision floating with or without exponent
Iw	Decimal integer conversion
Ow	Octal integer conversion†
Aw	Alphanumeric conversion
Rw	Alphanumeric conversion†
Lw	Logical conversion
nP	Scaling factor

Complex data items are converted on input/output according to a pair of consecutive Ew. d or Fw. d specifications.

Example:

```
      COMPLEX A,B
      PRINT 10,A
   10 FORMAT(F7.2,F9.2)
      READ 11,B
   11 FORMAT (E10.3,E10.3)
```

Editing specifications:

wX	Intraline spacing
wH	Heading and labeling
/	Begin new record
...	Heading and labeling

Both w and d are unsigned integer constants; w specifies the field width in number of character positions in the external record, and d specifies the number of digits to the right of the decimal within the field.

† Not specified in USASI FORTRAN, X3.9-1966.

9.4 CONVERSION SPECIFICATIONS

9.4.1 Ew.d OUTPUT

Real numbers in storage are converted to the BCD character form for output with the E conversion. The field occupies w positions in the output record with the real number right justified in the form:

ba.a...a±eee $\qquad 100 \le eee \le 322$

or

ba.a...aE±ee $\qquad 0 \le ee \le 99$

b indicates blank character. a's are the most significant digits of the integer and fractional part and eee are the digits in the exponent. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, signs, decimal point, E, and the exponent. Generally, $w \geq d+7$. Positive numbers need not reserve a space for the sign of the number.

If the field is not wide enough to contain the output value, an asterisk is inserted in the high order position of the field. If the field is longer than the output value, the quantity is right justified with blank fill to the left. Double precision numbers cannot be output under Ew.d.

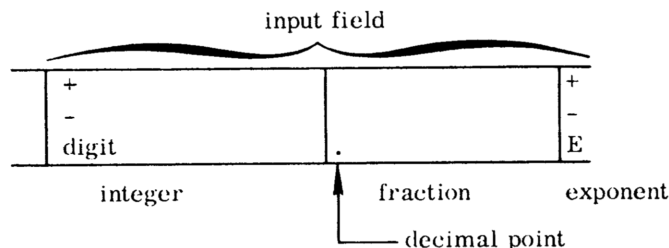
Examples:

```
      PRINT 10,A                A contains -67.32
10    FORMAT(E10.3)                   or + 67.32
            Result: -6.732E+01 or b6.732E+01
      PRINT 10,A
10    FORMAT (E12.4)
            Result: b-6.7320E+01  bb6.7320E+01
      PRINT 10,A                A contains -67.32
10    FORMAT(E9.3)              provision not made
                                for sign
            Result:  *.732E+01
      PRINT 10,A
10    FORMAT(E10.4)
            Result *.7320E+01
```

9.4.2 Ew.d INPUT

The E specification converts the number in the input field to a real number and stores it in the proper location.

Subfield structure of the input field:



The total number of characters in the input field is specified by w; this field is scanned from left to right; blanks are interpreted as zeros.

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

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

The exponent subfield may begin with D, E, + or -. When it begins with D or E, the + is optional between D or E and the string of digits of the subfield. The value of the string of digits in the exponent subfield must be less than 323.

Permissible subfield combinations:

+1.6327E-04	integer fraction exponent
-32.7216	integer fraction
+328+5	integer exponent
.629E-1	fraction exponent
+136	integer only
136	integer only
.07628431	fraction only
E-06 (interpreted as zero)	exponent only

In the Ew.d specification, d acts as a negative power-of-ten scaling factor when an external decimal point is not present. The internal representation of the input quantity is:

$$(\text{integer subfield}) \times 10^{-d} \times 10^{(\text{exponent subfield})}$$

For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as $3267 \times 10^{-8} \times 10^{5} = 3.267$.

A decimal point in the input field overrides d. The input quantity 3.67294+5 read by an E9.d specification is always stored as 3.6729×10^{5}. When d does not appear, it is assumed to be zero.

The field length specified by w in Ew.d should always be the same as the length of the field containing the input number. When it is not, incorrect numbers may be read, converted, and stored as shown below. The field w includes the significant digits, signs, decimal point, E or D, and exponent.

Example:

```
      READ 20,A,B,C
20    FORMAT (E9.3,E7.2,E10.3)
```

Input quantities on the card are in three contiguous fields columns 1 through 24:

```
    9       5        10
|+6.47E-01-2.36+5.321E+02bb
```

The second specification (E7.2) exceeds the width of the second field by two characters.

Reading proceeds as follows:

```
     9         7         10
[+6.47E-01]-2.36+5 .321E+02bb
+6.47E-01[-2.36+5].321E+02bb
+6.47E-01-2.36+5[.321E+02bb]
```

First, +6.47.01 is read, converted, and placed in location A. Next, -2.36+5 is read, converted, and placed in location B. The number actually desired was -2.36, but the specification error (E7.2 instead of E5.2) caused the two extra characters to be read. The number read (-2.36+5) is a legitimate input representation under the definitions and restrictions.

Finally, .321E+0200 is read, converted, and placed in location C. Here again, the input number is legitimate and is converted and stored, even though it is not the number desired.

The above example illustrates a situation where numbers are incorrectly read, converted, and stored, and yet there is no immediate indication that an error has occurred.

Examples:

Input Field	Specification	Converted Value	Remarks
+143.26E-03	E11.2	.14326	All subfields present
-12.437629E+1	E13.6	-124.37629	All subfields present
8936E+004	E9.10	.008936	No fraction subfield; input number converted as $8936.\times 10^{-10+4}$
327.625	E7.3	327.625	No exponent subfield
4.376	E5	4.376	No d in specification
-.0003627+5	E11.7	-36.27	Integer subfield contains - only
-.0003627E5	E11.7	-36.27	Integer subfield contains - only
blanks	Ew.d	-0	All subfields empty
1E1	E3.0	10.	No fraction subfield; input number converted as $1.\times 10^{1}$
E+06	E10.6	0.	No integer or fraction subfield; zero stored regardless of exponent field contents
1.bEb1	E6.3	10.	Blanks are interpreted as zeros

9.4.3 Fw.d OUTPUT

The field occupies w positions in the output record; the corresponding list item must be a floating point quantity, which appears as a decimal number, right justified:

b a. . . a. a. . . a

b indicates a blank. The a's represent the most significant digits of the number. The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, the decimal point and digits to the right do not appear. If the number is positive, the + sign is suppressed. If the field is too short to accommodate the number, one asterisk appears in the high-order position of the output field. If the field is longer than required to accommodate the number, the number is right justified with blank fill to the left.

Contents of A	Format Statement	Print Statement	Printed Result
+32.694	10 FORMAT (F7.3)	PRINT 10, A	b32.694
+32.694	11 FORMAT (F10.3)	PRINT 11, A	bbbb32.694
-32.694	12 FORMAT (F6.3)	PRINT 12, A	*2.694 (no provision for - sign and most significant digit)
.32694	13 FORMAT (F4.3, F6.3)	PRINT 12, A, A	.327b0.327

9.4.4 Fw.d INPUT

This specification is a modification of Ew.d. The input field consits of an integer and a fraction subfield. An omitted subfield is assumed to be zero. The restrictions described under Ew.d input apply.

Examples:

Input Field	Specification	Converted Value	Remarks
367.2593	F8.4	367.2593	Integer and fraction field
37925	F5.7	.0037925	No fraction subfield; input number converted as 37925 x 10^{-7}
-4.7366	F7	-4.7366	No d in specification
.62543	F6.5	.62543	No integer subfield
.62543	F6.2	.62543	Decimal point overrides d of specification
+144.15E-03	F11.2	.14415	Exponents are legitimate in F input and may have P-scaling
5bbbb	F5.2	500.00	No fraction subfield; input number converted as 50000x10^{-2}

9.4.5 Gw.d OUTPUT

The field occupies w positions of the output record, with d significant digits. The real data will be represented by F conversion unless the magnitude of the data exceeds the range that permits effective use of F conversion. In this case, the E conversion will represent the external output. Therefore, the effect of the scale factor is not implemented unless the magnitude of the data requires E conversion.

When F conversion is used under Gw.d output specification, 4 blanks are inserted within the field, right justified. Therefore, for effective use of F conversion, d must be $\leq$ w-6.

The method of representation in the output record is a function of the magnitude N of the real data being converted. The following table gives a correspondence between N and the method of conversion:

$0.1 \leq N < 1$	F (w-4). d, 4X
$1 \leq N < 10$	F (w-4). (d-1), 4X
⋮	⋮
$10^{d-2} \leq N < 10^{d-1}$	F (w-4). 1, 4X
$10^{d-1} \leq N < 10^{d}$	F (w-4). 0, 4X

Examples:

PRINT 101, XYZ　　　　XYZ contains 77.132

101 FORMAT (G10. 3)

Result: bb77. 1bbbb

PRINT 101,XYZ　　　　XYZ contains 1214635.1

101 FORMAT (G10. 3)

Result: b1. 215E+06

9.4.6 Gw.d INPUT

Gw. d specification is similar to the Fw. d input specification.

9.4.7 Dw.d OUTPUT

The field occupies w positions of the output record, the list item is a double precision quantity which appears as a decimal number, right justified:

ba. a. . . a+eee　　　　$100 \leq eee \leq 512$

or

ba. a. . . aD+ee　　　　$0 \leq ee \leq 99$

b indicates blank. D conversion corresponds to Ew. d Output.

Single precision numbers cannot be output under Dw. d.

9.4.8 Dw.d INPUT

D conversion corresponds to E conversion except that the list variables must be double precision names. D is acceptable in place of E as the beginning of an exponent subfield.

Example:

```
      DOUBLE Z,Y,X
      READ1,Z,Y,X
1     FORMAT (D18.11,D15,D17.4)
```

Input Card:

```
-6.31675298443E-03 +2.718926453147 6293477528869D-09
        18               15                17
```

9.4.9 Iw OUTPUT

I specification is used to output decimal integer values. The output quantity occupies w output record positions, right justified:

b̲a. . . a

b is a blank. The a's are the most significant decimal digits (maximum 15) of the integer. If the integer is positive, the + sign is suppressed. The range of numbers permitted is roughly $-2^{48}+1 \leq n \leq 2^{48}-1$.

If the field w is larger than required, the output quantity is right justified with blank fill to the left. If the field is too short, characters arestored from the right, an asterisk occupies the leftmost position.

Example:

```
      PRINT 10,I,J,K             I contains -3762
10    FORMAT (I8,I10,I5)         J contains +4762937
                                 K contains +13

                   Result:   bbb-3762bbb4762937bbb13
                                 8       10      5
```

9.4.10 Iw INPUT

The field is w characters in length, and the list item is a decimal integer constant. The input field w consists of an integer subfield, containing +, -, 0 through 9, or blank. When a sign appears, it must precede the first digit in the field. Blanks are interpreted as zeros. The value is stored right justified in the specified variable.

Example:

```
      READ 10,I,J,K,L,M,N
   10 FORMAT (I3,I7,I2,I3,I2,I4)
```

Input Card:

```
139bb-15bb18bb7b3b1b4
 3    7    2  3  2  4
```

In storage:

I contains	139
J	-1500
K	18
L	7
M	3
N	104

9.4.11 Ow OUTPUT

O specification is used to output octal integer values. The output quantity occupies w output record positions right justified:

aa. . . a

The a's are octal digits. If w is 20 or less, the rightmost w digits appear. If w is greater than 20, the number is right justified in the field with blanks to the left of the output quantity. A negative number is output in its one's complement internal form.

Octal output is not specified in USASI FORTRAN, X3. 9-1966.

9.4.12 Ow INPUT

Octal integer values are converted under O specification. The field is w characters in length, and the list item must be an integer variable.

The input field w consists of an integer subfield only (maximum of 20 octal digits) containing +, -, 0 through 7, or blank.

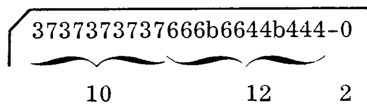
Only one sign may precede the first digit in the field. Blanks are interpreted as zeros.

Octal input is not specified in USASI FORTRAN, X3.9-1966.

Example:

```
      TYPE INTEGER P,Q,R
      READ 10,P,Q,R
 10   FORMAT (O10,O12,O2)
```

Input Card:



In storage:

```
P   00000000003737373737
Q   00000000666066440444
R   77777777777777777777
```

A negative number is represented in one's complement form.

A negative octal number is represented internally in seven's complement form (20 digits) obtained by subtracting each digit of the octal number from seven. For example, if -703 is an input quantity, its internal representation is 77777777777777777074.

```
That is, 77777777777777777777
        -00000000000000000703
         77777777777777777074
```

9.4.13 Aw OUTPUT

A conversion is used to output alphanumeric characters. If w is 10 or more, the quantity appears right justified in the output field, blank fill to left. If w is less than 10, the output quantity is represented by leftmost w characters.

9.4.14 Aw INPUT

This specification accepts FORTRAN characters including blanks. The internal representation is 7000 Series display code; the field width is w characters.

If w exceeds 10, the input quantity is the rightmost 10 characters in the field. If w is 10 or less, the input quantity is stored as a left justified BCD word; the remaining spaces are blank filled.

Example:

```
      READ 10,Q,P,O
10    FORMAT (A8,A8,A4)
```

Input Card:

```
LUX MENTIS LUX ORBIS
    8       8    4
```

In storage:

```
Q   LUXbMENTbb
P   ISbLUXbObb
O   RBISbbbbbb
```

9.4.15 Rw OUTPUT

This specification is similar to the Aw Output with the following exception: if w is less than 10, the output quantity represents the rightmost characters.

Rw output is not specified in USASI FORTRAN, X3.9-1966.

9.4.16 Rw INPUT

This specification is the same as the Aw Input with the following exception: if w is less than 10, the input quantity is stored as a right justified binary zero filled word.

Rw input is not specified in USASI FORTRAN, X3.9-1966.

Example:

```
      READ 10,Q,P,O
10    FORMAT (R8,R8,R4)
```

Input Card:

```
LUX MENTIS LUX ORBIS
    8       8    4
```

In storage:

```
Q   00LUXbMENT
P   00ISbLUXbO
O   000000RBIS
```

9.4.17 Lw OUTPUT

L specification is used to output logical values. The output field is w characters long, and the list item must be a logical element.

A value of TRUE or FALSE in storage causes w-1 blanks followed by a T or F to be output.

Example:

```
      LOGICAL I,J,K,L          I contains -0        J contains 0
      PRINT 5,I,J,K,L          K contains -0        L contains -0
    5 FORMAT (4L3)
                     Result: bbTbbFbbTbbT
```

9.4.18 Lw INPUT

This specification accepts logical quantities as list items. The field is considered true if the first non-blank character in the field is T or false if it is F. An all-blank field is considered false.

9.5 nP SCALE FACTOR

The D, E, F, and G conversion may be preceded by a scale factor which is:
External number = Internal number x $10^{\text{scale factor}}$. The scale factor applies to Fw.d and Gw.d on both input and output and to Ew.d and Dw.d on output only. A scaled specification is written as shown below; n is a signed integer constant.

nPDw.d   nPEw.d   nPFw.d   nPGw.d   nP

The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it holds for all D, E, F, and G specifications. To nullify this effect in subsequent D, E, F, and G specifications, a zero scale factor, 0P, must precede a D, E, F, or G specification. Scale factors for D, E, F, and G output specifications must be in the range $-8 \leq n \leq 8$.

Scale factors on D or E input specifications are ignored. For USASI compatible scale factor see section 9.10.2.

The scaling specification nP may appear independently of a D, E, F, or G specification; it holds for all subsequent D, E, F, and G specifications within the same FORMAT statement unless changed by another nP.

Example:

```
FORMAT(3PE12.6,F10.3,0PD18.7,-1P,F5.2)
```

The E12.6 and F10.3 specifications are scaled by 10^3, the D18.7 specification is not scaled, and the F5.2 specification is scaled by 10^{-1}.

The specification (3P,3I9,F10.2) is the same as the specification (3I9,3PF10.2).

9.5.1 Fw.d SCALING

Input

The number in the input field is divided by 10^n and stored. For example, if the input quantity 314.1592 is read under the specification 2PF8.4, the internal number is $314.1592 \times 10^{-2} = 3.141592$.

Output

The number in the output field is the internal number multiplied by 10^n. In the output representation, the decimal point is fixed; the number moves to the left or right, depending on whether the scale factor is plus or minus. For example, the internal number 3.145926538 may be represented on output under scaled F specifications as follows:

Specification	Output Representation
F13.6	3.141593
1PF13.6	31.415927
3PF13.6	3141.592654
-1PF13.6	.314159

9.5.2 Ew.d OR Dw.d SCALING

Output

The scale factor has the effect of shifting the output number left n places while reducing the exponent by n. Using 3.1415926538, some output representations corresponding to scaled E specifications are:

Specification	Output Representation
E20.2	3.14 E+00
1PE20.2	31.42 E-01
2PE20.2	314.16 E-02
3PE20.2	3141.59 E-03
4PE20.2	31415.93 E-04
5PE20.2	314159.27 E-05
-1PE20.2	0.31 E+01

9.5.3 Gw.d SCALING

Input

Gw.d scaling on input is the same as Fw.d scaling on input.

Output

The effect of the scale factor is suspended unless the magnitude of the data to be converted is outside the range that permits the effective use of F conversion.

9.6 EDITING SPECIFICATIONS

9.6.1 wX

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

Examples:

```
       INTEGER A                        A contains 7
       PRINT 10, A,B,C                  B contains 13.6
                                        C contains 1462.37
10     FORMAT (I2,6X, F6.2,6X, E12.5)

                  Result: b7bbbbbbb13.60bbbbbbb1.46237E+03

       READ 11,R,S,T
11     FORMAT (F5.2,3X, F5.2,6X, F5.2)
          or
11     FORMAT (F5.2,3XF5.2,6XF5.2)
```

Input Card:

```
14.62bb$13.78bCOSTb15.97
```

In storage:

```
R   14.62
S   13.78
T   15.97
```

9.6.2 wH OUTPUT

With this specification, 6-bit characters (including blanks) may be output in the form of comments, titles, and headings. w, an unsigned integer, specifies the number of characters to the right of H that are transmitted to the output record; w may specify a maximum of 136 characters. H denotes a Hollerith field. The comma following the H field is optional.

Examples:

Source program:

```
    PRINT 20
20  FORMAT (28HbBLANKSbCOUNTbINbANbHbFIELD.)
```

produces output record:

bBLANKSbCOUNTbINbANbHbFIELD.

Source program:

```
    PRINT 30,A                  A contains 1.5
30  FORMAT (6HbLMAX=, F5.2)
```

produces output record:

bLMAX = b1.50

9.6.3 wH INPUT

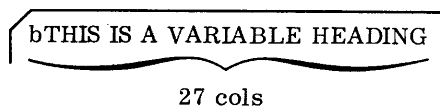
The H specification may be used to read Hollerith characters into an existing H field within the FORMAT specification.

Example:

Source program:

```
      READ 10
10    FORMAT (27Hbbbbbbbbbbbbbbbbbbbbbbbbbbb)
```

Input Card:



After READ, the FORMAT statement labeled 10 contains the alphanumeric information read from the input card; a subsequent reference to statement 10 in an output statement acts as follows:

PRINT 10

produces the print line:

bTHIS IS A VARIABLE HEADING

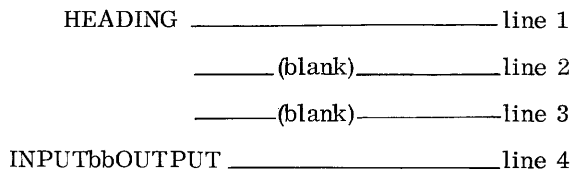
9.6.4 NEW RECORD

The slash(/) signals the end of a record anywhere in the specifications list. Consecutive slashes may appear in a list and they need not be separated from the other list elements by commas. During output, the slash is used to skip lines, cards, or tape records. During input, it specifies that control passes to the next record or card. K(/) or K/ results in K-1 lines being skipped.

Examples:

1.
```
      PRINT 10
10    FORMAT (6X,7HHEADING///3X,5HINPUT,2X,6HOUTPUT)
```

Printout:



Each line corresponds to a BCD record. The second and third records are null and produce the line spacing illustrated.

2. PRINT 11,A,B,C,D

11 FORMAT (2E10.2/2F7.3)

In storage:

```
A   -11.6
B   .325
C   46.327
D   -14.261
```

Printout:

```
b-1.16E+01bb3.25E-01

b46.327-14.261
```

3. PRINT 11,A,B,C,D

11 FORMAT (2E10.2/ /2F7.3)

Printout:

```
b-1.16E+01bb3.25E-01 ____________line 1
                     __ blank ____line 2
b46.327-14.261 _________________line 3
```

4. PRINT 15, (A(I),I=1,9)

15 FORMAT (8HbRESULTS2(/) (3F8.2))

Printout:

```
RESULTS __________________________________ line 1
                            __ (blank) ____ line 2
    3.62       -4.03       -9.78 __________ line 3
   -6.33        7.12        3.49 __________ line 4
    6.21       -6.74       -1.18 __________ line 5
```

9.6.5 * . . . *

The specification *...* can be used as an alternate form of wH to output headings, titles, and comments. Any 6-bit character (except asterisk) between the asterisks will be output. The asterisks delineate the Hollerith field. This specification need not be separated from other specifications by commas.

Output Examples:

1. Source program:

```
         PRINT 10
      10 FORMAT (*bSUBTOTALS*)
```

   produces the output record: bSUBTOTALS

2. Improper source program to output ABC*BE:

```
    PRINT 1
  1 FORMAT(*ABC*BE*)
```

   The * in the output causes the specification to be interpreted as *ABC* and BE*. BE* is an improper specification; therefore, the wH specification must be used to output ABC*BE.

For input, this specification may be used in place of wH to read a new heading into an existing Hollerith field. Characters are stored in the heading until an asterisk is encountered in the input field or until all the spaces in the format specification are filled. If the format specification contains n spaces and the mth character ($m \leq n$) in the input field is an asterisk, all characters to the left of the asterisk will be stored in the heading and the remaining character positions in the heading will be filled with blanks.

Input Examples:

1. Source program:

```
         READ 10
      10 FORMAT (*bbbbbbbbbbbbbbbbbbb*)
```

   Input card: FORTRAN FOR THE 7600

   A subsequent reference to statement 10 in an output control statement:

   PRINT 10 produces: FORTRAN FOR THE 7600

2. Source program:

```
         READ 10
      10 FORMAT (*bbbbbbb*)
```

   HEAD*LINE

   PRINT 10 produces: HEAD bbb

NOTE

Column 1 is printed and not used for carriage control.

9.7 REPEATED FORMAT SPECIFICATIONS

Format specifications may be repeated by using an unsigned integer constant repetition factor, k, as follows: k(spec), spec is any conversion specification except nP.† For example, to print two quantities K, L:

```
      PRINT 10,K,L
10    FORMAT (I2,I2)
```

Specifications for K, L are identical; the FORMAT statement may also be:

```
10    FORMAT (2I2)
```

When a group of FORMAT specifications repeats itself as in: FORMAT (E15.3,F6.1,I4,I4,E15.3, F6.1, I4,I4), the use of k produces: FORMAT (2(E15.3, F6.1,2I4))

Nesting of parenthetical groups preceded by repeat constants beyond two levels is not permitted in FORMAT specifications.

9.8 UNLIMITED GROUPS

FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an unlimited group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached, as in the following example:

Format specifications for output data:

```
(E16.3, F20.7,2(I4), (I3, F7.1), F8.2)
```

The first two fields are printed according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4 format. The remaining print fields follow (I3, F7.1), which does not have a repetition factor, until the list elements are exhausted. F8.2 is never reached.

USASI compatibility for unlimited groups is achieved by enabling a compile-time switch. (Refer to sections 9.10 and 9.10.1.

† USASI FORTRAN X3.9-1966 does not exclude nP from repeated format specifications.

9.9 VARIABLE FORMAT

FORMAT specifications may be specified at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, is read under A conversion or in a DATA statement and stored in an array or a simple variable. The name of the array containing the specifications may be used in place of the FORMAT statement labels in the associated input/output operation. The array name that appears with or without subscript specifies the location of the first word of the FORMAT information.

Examples:

1. Assume the following FORMAT specifications:

   ```
   (E12.2, F8.2, I7, 2E20.3, F9.3, I4)
   ```

   This information can be punched in an input card and read by the statements of the program such as:

   ```
       DIMENSION IVAR(3)
       READ 1 (IVAR(I), I=1, 3)
   1   FORMAT (3A10)
   ```

   The elements of the input card are placed in storage as follows:

   ```
   IVAR(1):        (E12.2, F8.
   IVAR(2):        2, I7, 2E20.
   IVAR(3):        3, F9.3, I4)
   ```

   A subsequent output statement in the same program can refer to these FORMAT specifications as:

   ```
   PRINT IVAR, A, B, I, C, D, E, J
   ```

   This produces exactly the same result as the program:

   ```
       PRINT 10, A, B, I, C, D, E, J
   10  FORMAT (E12.2, F8.2, I7, 2E20.3, F9.3, I4)
   ```

2. ```
   DIMENSION LAIS1(3), LAIS2(2), A(6), LSN(3),TEMP(3)
   DATA LAIS1/21H(2F6.3, I7, 2E12.2, 3I1)/, LAIS2/20H
   (I6, 6X, 3F4.1, 2E12.2)/
   ```

   Output statement:

   ```
   PRINT LAIS1, (A(I), I=1, 2), K, B, C, (LSN(J), J=1, 3)
   ```

   which is the same as:

   ```
       PRINT 1, (A(I), I=1, 2), D, B, C, (LSN(J), J=1, 3)
   1   FORMAT (2F6.3, I7, 2E12.2, 3I1)
   ```
   ```

Output statement:

```
PRINT LAIS2, LA, (A(M), M=3, 4), A(6), (TEMP(I), I=2, 3)
```

which is the same as:

```
    PRINT 2, LA, (A(M), M=3, 4), A(6), (TEMP(L), L=2, 3)
2   FORMAT (I6, 6X, 3F4.1, 2E12.2)
```

3. DIMENSION LAIS (3), VALUE(6)
   DATA LAIS/26H(I3, 13HMEANbVALUEbIS, F6.3)/

   Output statement:

```
    WRITE (10, LAIS)NUM, VALUE(6)
```

   which is the same as:

```
    WRITE (10, 10)NUM, VALUE(6)
10  FORMAT (I3, 13HMEANbVALUEbIS, F6.3)
```

9.10 USASI COMPATIBILITY

During compilation, a compiler parameter is available to select either the USASI FORTRAN X3.9-1966 compatibility features of the execution time routines or retain the 6000 compatible method of FORMAT/list interaction and output format. The switch is enabled by a parameter on the RUN control card and has effect only when a program is being compiled.

9.10.1 UNLIMITED GROUPS FOR USASI

Unlimited group repeat is implemented according to the USASI X3.9 specification. An innermost parenthetical group that has no repeat count specified in a FORMAT statement assumes a group repeat count of one. If the last outer right parenthesis of the format specification is encountered and the I/O list is not exhausted, control reverts to that group repeat specification terminated by the last preceding right parenthesis, or if none exists, then to the first left parenthesis of the format statement.

9.10.2 SCALE FACTOR FOR USASI

A scale factor is allowed for F, E, D, and G on input. On input, the scale factor has no effect if there is an exponent in the external field. G output makes use of the scale factor only if E conversion is necessary to convert the data. For E and D output, the basic real constant part of the output quantity is multiplied by 10^n and the exponent is reduced by n.

INPUT/OUTPUT STATEMENTS 10

The following definitions apply to all I/O statements:

i logical I/O unit number:

an integer constant of one or two digits (the first must not be zero)

integer variable name of no more than 6 characters, with a value of 1 to 99

n FORTRAN declaration identifier:

statement number

variable identifier which references the starting storage location of FORMAT information

L input/output list

The BCD record and the binary records for each I/O device as used with the 7000 SCOPE V 1.1 operating system are defined as follows:

Printer Card Reader Card Punch	A BCD record is a one-card image (80 characters)for the card I/O devices and 136 characters (1 print line) for the printer.
	A binary record, for the card I/O devices, is the number of cards between the EOR cards (7,8,9 punch in column 1); for the printer, it is the number of print-lines between the EOR marks.
Disk and Tape	A BCD record is a character string terminated by a 12-bit zero byte. A record will always begin at a word boundary, left justified. BCD records may be separated in groups terminated by an EOR control word.
	A binary record is a word string terminated by an EOR control word.
	All data, BCD or binary, is written in 512 word blocks consisting of one boundary word and 511 data and control words. A record may cross a block boundary.

10.1 OUTPUT STATEMENTS

PRINT n, L

Information in the list (L) is transferred from the storage locations to the standard output unit as line printer images, 136 characters or less per line in accordance with the FORMAT declaration, n.

The maximum record length is 136 characters, but the first character of every record is not printed as it is used for carriage control when printing on-line. Characters in excess of the print line appear on the succeeding line. Each new record starts a new print line.

Character	Action
Blank	Single-space before printing
0	Double-space before printing
1	Eject page before printing
+	Suppress spacing before printing; print two successive records on the same line

Consult the operating system manual for additional characters.

For off-line printing, the printer control is determined by the installation's printer routine.

PUNCH n, L

Information is transferred from the storage locations given by the list (L) identifiers to the standard punch unit. Information is transferred as Hollerith images, 80 characters or less per card in accordance with the FORMAT declaration, n. Records greater than 80 characters will be truncated.

WRITE (i, n) L

This statement transfers information from storage locations given by the list (L) to a specified output unit (i) according to the FORMAT declaration (n).

With a half inch tape unit, a record containing up to 136 characters is recorded in even parity (BCD mode). The number of words in the list and the FORMAT declaration determine the number of records that are written on a unit. If the record is less than 136 characters, the remainder of the record is filled with blanks.

The information is recorded in 7000 series display code with no special control characters added, and it represents a continuous stream of output records. Trailing blanks on each record are removed and two consecutive characters with a value of zero separate records on the tape.

If the file is to be printed, the first character of a record is not printed as it is a printer carriage control. If the programmer fails to allow for a printer control character, the first character of the output data is lost on the printed listing.

WRITE (i)L

This statement transfers information from storage locations given by the list (L) to a specified output unit (i). If L is omitted, the WRITE (i) statement acts as a do-nothing statement. The list written by this statement constitutes one binary record. See READ (i)L.

Examples:

```
      PRINT  50,A,B,C
 50   FORMAT(X8HMINIMUM=F17.7,2X8HMAXIMUM=F17.7,2X8HVALUEbISbF8.2)

      PUNCH  52,ACCT,LSTNME,FSTNME
 52   FORMAT (F8.2,3X2A10)

      WRITE (2,53)A,B,C,D
 53   FORMAT(4E21.9)

      DIMENSION A(10),B(50,5)
      DO 5  I=1,10
  5   WRITE(6)A(I),(B(I,J),J=1,5)
```

10.2 READ STATEMENTS

A check should be made for the end of the file either by counting records or by an IF EOF statement after each read (section 10.4). When the EOF is read, the data used for processing will be a blank record. If a read is issued after the EOF is read, the job will be terminated unless the EOF flag has been cleared by an IF EOF statement. An EOR control word encountered when reading a BCD file is ignored except for file INPUT where it is treated as an EOF.

READ n, L

One or more card images are read from the standard input unit. Information is converted from left to right in accordance with FORMAT specification (n), and it is stored in the locations named by the list (L). Input may be on 80-column Hollerith cards or magnetic tapes prepared off-line, containing 80-character records in BCD mode.

Example:

```
      READ 10,A,B,C
   10 FORMAT (3F10.4)
```

READ (i, n)L

This statement transfers one record of information from logical unit (i) to storage locations named by the list (L), according to FORMAT specification (n). The number of words in the list and the FORMAT specifications must conform to the record structure on the logical unit.

READ (i)L

This statement transfers one record of information from a specified unit (i) to storage locations named by the list (L).

Records to be read by READ (i) should be written in binary mode. The number of words in the list of READ(i)L must not exceed the number of words in the corresponding WRITE statement.

If L is omitted, READ (i) spaces over one record. See WRITE (i)L.

Examples:

```
1.    DIMENSION C(264)
      READ (10)C
      DIMENSION BMAX (10),M2(10,5)
      DO 7  I=1,10
7     READ(6)BMAX(I),(M2(I,J),J=1,5)
      READ (5)  (skip one logical record on unit 5)
      READ (6)  ((A(I,J),I=1,100),J=1,50)
      READ (6)  ((A(I,J),I=1,100),J=1,50)
```

```
2.      READ (10,50)X,Y,Z
    50  FORMAT (3F10.6)
        DOUBLE PRECISION DB(4)
        READ (10,51) DB
    51  FORMAT (4D20.12)
        READ 51,DB
        READ (2,52) (Z(J),J=1,8)
    52  FORMAT (F10.4)
```

10.3 NAMELIST STATEMENT

The NAMELIST statement permits the input and output of character strings consisting of names and values without a format specification.

NAMELIST $/y_1/a_1/y_2/a_2/.../y_n/a_n$

Each y is a NAMELIST name consisting of 1-7 characters which must be unique within the program unit in which it is used. Each a is a list of the form $b_1, b_2, ... b_n$;each being a variable or array name.

In any given NAMELIST statement, the list a of variable names or array names between the NAMELIST identifier y and the next NAMELIST identifier (or the end of the statement if no NAMELIST identifier follows) is associated with the identifier y; that is, the list a_i is associated with NAMELIST identifier y_i.

Examples:

```
PROGRAM MAIN
NAMELIST/NAME1/N1,N2,R1,R2/NAME2/N3,R3,N4,N1

SUBROUTINE XTRACT (A,B,C)
NAMELIST/CALL1/L1,L2,L3/CALL2/L3,P4,L5,B
```

A variable name or array name may be an element of more than one such list. In a subprogram, b may be a dummy parameter identifying a variable or an array, but the array may not have variable dimensions.

A NAMELIST name may be defined only once in a program unit preceding any reference to it. Once defined, any reference to a NAMELIST name may be made only in a READ or WRITE statement. The form of the input/output statements used with NAMELIST is as follows:

```
READ (u,x)
WRITE (u,x)
```

u is an integer variable or integer constant denoting a logical unit, and x is a NAMELIST name.

Example:

```
Assume A, I, and L are array names
     .
     .
     .
NAMELIST /NAM1/A,B,I,J/NAM2/C,K,L
     .
     .
     .
READ (5,NAM1)
     .
     .
     .
WRITE (8,NAM2)
```

These statements result in the BCD (coded) input/outputs on the device specified as the logical unit of the variables and arrays associated with the identifiers, NAM1 and NAM2.

INPUT DATA

The current file on unit u is scanned up to an end-of-file or a record with a $ in column 2 followed immediately by the name (NAM1) with no embedded blanks. Succeeding data items are read until a $ is encountered.

The data item, separated by commas, may be in any of three forms:

v=c

a=$d_1,\ldots,d_j$

a(n)=$d_1,\ldots,d_m$

v is a variable name, c a constant, a an array name, and n is an integer constant subscript. d_i are simple constants or repeated constants of the form k*c, where k is the repetition factor.

Example:

```
DIMENSION Y(3,5)
LOGICAL L
COMPLEX Z
NAMELIST /HURRY/I1,I2,I3,K,M,Y,Z,L
READ (5,HURRY)
```

and the input record:

```
$HURRYbI1+1,L=.TRUE.,I2=2,I3=3.5,Y(3,5)=26,Y(1,1)=11,12.0E1,13,4*14,
Z=(1.,2.),K=16,M=17$
```

produce the following values:

I1=1	Y(1,2)=14.0
I2=2	Y(2,2)=14.0
I3=3	Y(3,2)=14.0
Y(3,5)=26.0	Y(1,3)=14.0
Y(1,1)=11.0	K=16
Y(2,1)=120.0	M=17
Y(3,1)=13.0	Z=(1.,2.)
	L=.TRUE.

The number of constants, including repetitions, given for an unsubscripted array name must equal the number of elements in that array. For a subscripted array name, the number of constants need not equal, but may not exceed, the number of array elements needed to fill the array.

v=c	variable v is set to c
$a=d_1,\ldots,d_j$	the values $d_1,\ldots,d_j$ are stored in consecutive elements of array a in the order in which the array is stored internally.
$a(n)=d_1,\ldots,d_m$	elements are filled consecutively starting at a (n)

The specified constant of the NAMELIST statement may be integer, real, double precision, complex of the form (c_1, c_2), or logical of the form T, or .TRUE., F, or .FALSE.. A logical or complex variable may be set only to a logical and complex constant, respectively. Any other variable may be set to an integer, real or double precision constant. Such a constant is converted to the type of its associated variable.

Constants and repeated constant fields may not include embedded blanks. Blanks, however, may appear elsewhere in data records.

A maximum of 150 characters per input record is permitted. More than one record may be used for input data. All except the last record must end with a constant followed by a comma, and no serial numbers may appear; the first column of each record is ignored.

The set of data items may consist of any subset of the variable names associated with x. These names need not be in the order in which they appear in the defining NAMELIST statement.

OUTPUT DATA

Output to unit u of BCD information is as follows:

One record consisting of a $ in column 2 immediately followed by the identifier x. As many records as are needed to output the current values of all variables in the list associated with x. Simple variables are output as v=c.

Elements of dimensioned variables are output in the order in which they are stored internally.

The data fields are made large enough to include all significant digits. Logical constants appear as T and F. No data appears in column 1 of any record.

One record consisting of a $ in column 2 immediately followed by the letters END.

The records output by such a WRITE statement may be read by a READ (u,x) statement where x is the same NAMELIST identifier.

If unit u is the standard punch unit and a record is longer than 80 characters, the remaining characters are punched on the next card.

The maximum length of a record written by a WRITE (u,x) statement is 130 characters.

10.4 FILE HANDLING STATEMENTS

REWIND i

File tape i is rewound to load point. If the file is already rewound, the statement acts as a do-nothing statement. The REWIND statement may not reference the system INPUT and OUTPUT files.

BACKSPACE i

File tape i is backspaced one record in a binary file or a BUFFER IN/OUT file or one BCD record in a normal BCD file. If tape i is at load point (rewound) this statement acts as a do-nothing statement. The BACKSPACE statement may not reference the system INPUT and OUTPUT files.

END FILE i

An end-of-file is written on file tape i. The END FILE statement may not reference the system INPUT and OUTPUT files.

IF (ENDFILE i)n_1, n_2

IF (EOF, i)n_1, n_2

These statements check the previous read operation to determine if an end-of-file has been encountered on file tape i. If so, control is transferred to statement n_1; if not, control is transferred to statement n_2.

IF(UNIT, i)n_1, n_2, n_3, n_4

n_1 not ready

n_2 ready and no previous error

n_3 EOF sensed on last input operation

n_4 ineffective (parity error)

With the present system, for IF (UNIT, i)n_1, n_2, n_3, n_4, the parity error checking on a unit being tested is not accessible to the user. Therefore, n_4 may be omitted.

10.5 BUFFER STATEMENTS

The primary differences between buffer I/O and read/write I/O statements are given below:

1. The mode of transmission is assumed to be binary in BUFFER statements.

2. The read/write control statements are associated with a list and, in BCD transmission, with a FORMAT statement. The buffer control statements are not associated with a list; data transmission is to or from one area in storage.

3. Before buffered data is used, the status of the buffer operation must be checked by IF(UNIT, i). (see section 10. 4).

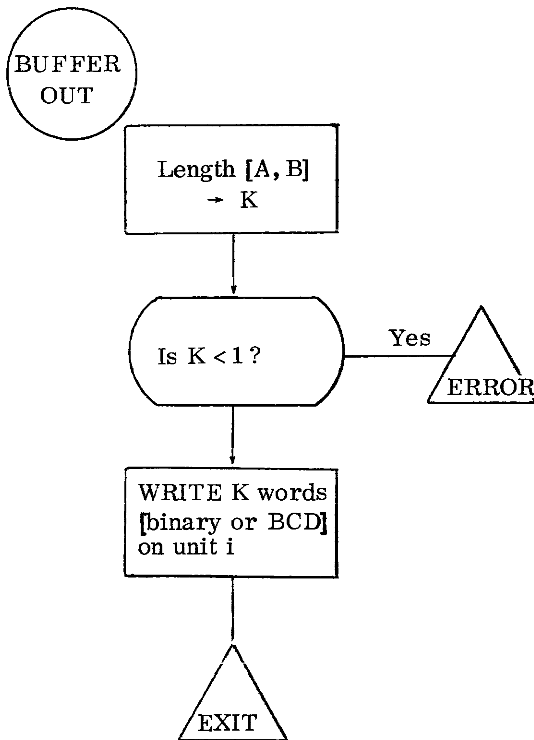
NOTE

The compiler does not check for subsequent inclusion of IF (UNIT).

In the descriptions that follow, these definitions apply.

i logical unit number

p parity key. Ignored.



i logical unit number

p recording mode
  0 even-BCD
  1 odd-binary

A variable identifier: first word of data block to be transmitted.

B variable identifier: last word of data block to be transmitted.

In the BUFFER statements the address of B must be greater than that of A. A unit referenced in a BUFFER statement may not be referenced in other I/O statements.

BUFFER IN (i,p) (A,B)

Information is transmitted from unit i to storage locations A through B. The transmission will terminate when either:

1. All the data from A to B has been read--in which case, the file will be left positioned at the end of the binary record

   or

2. A binary EOR is encountered.

In either case, the number of words actually transmitted can be obtained with

```
L = LENGTH(i)
```

BUFFER OUT (i,p) (A,B)

Information is transmitted from storage locations A through B as one logical record. It is written on unit i containing all the words from A to B inclusive.

Examples:

1.
```
      COMMON/BUFF/DATA(10), CAL(50)
      PAR=0
      BUFFER IN(9,PAR) (DATA(1), CAL(50))
    5 IF(UNIT,9)5,6,7
```

   Information is input from unit 9 to the labeled common area BUFF beginning at DATA(1), the first word of the block, and extending through CAL(50), the last word of the block.

2.
```
      DIMENSION A(100)
      N=6
      BUFFER OUT(N,1) (A(1),A(100))
    4 IF(UNIT,N)4,6,7
```

   Information is transmitted to unit N from the block area defined by A(1) and A(100), that is, all of array A is transmitted.

10.6 ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements are comparable to the BCD WRITE/READ statements; however, no peripheral equipment is involved. Information is transferred under FORMAT specifications from one area of storage to another. The parameters in these statements are defined as follows:

ENCODE (c, n, v)L where

- c unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the record. c may be an arbitrary number of BCD characters.
- n statement number or variable identifier representing the FORMAT statement
- v variable identifier or an array identifier which supplies the starting location of the BCD record
- L input/output list

When encoding or decoding is performed, the first record begins with the leftmost character position specified by v and continues c BCD characters (10 BCD characters per computer word). For ENCODE, if c is not a multiple of 10, the record ends in the middle of a word and the remainder of the word is blank filled. For DECODE, if the record ends with a partial word the balance of the word is ignored.

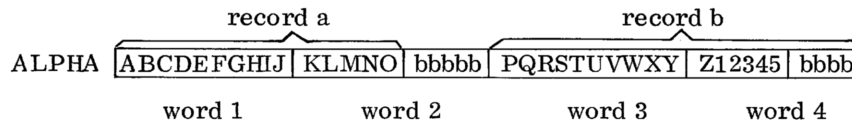
Since each succeeding record begins with a new computer word, an integral number of computer words is allocated for each record with $\frac{c+9}{10}$ words. The number of characters allocated for any single record in the encoded area must not exceed 150.

Example:

A(1) = ABCDEFGHIJ
A(2) = KLMNObbbbb
B(1) = PQRSTUVWXY
B(2) = Z12345bbbb

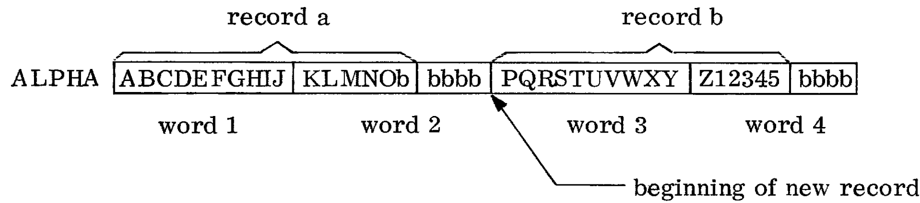
1. c = multiple of 10

```
     ENCODE (20,1,ALPHA)A,B
1    FORMAT (A10,A5/A10,A6)
```



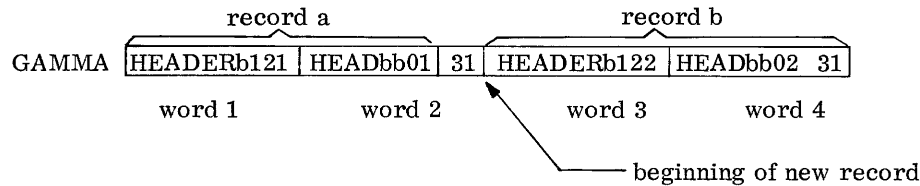
2. c ≠ multiple of 10

```
     ENCODE (16,1,ALPHA)A,B
1    FORMAT (A10,A6)
```



3. c ≠ multiple of 10

```
     DECODE (18, 1,GAMMA)A6,B6
1    FORMAT (A10,A8)
```



A6(1) = HEADERb121
A6(2) = HEADbb01bb
B6(1) = HEADERb122
B6(2) = HEADbb02bb

ENCODE (c, n, v) L

The information of the list variables, L, is transmitted according to the FORMAT (n) and stored in locations starting at v, c BCD characters per record. If the I/O list (L) and specification list (n) translate more than c characters per record, an execution diagnostic occurs. If the number of characters converted is less than c, the remainder of the record is filled with blanks.

DECODE (c, n, v) L

The information in c consecutive BCD characters (starting at address v) is transmitted according to the FORMAT n and stored in the list variables. If the number of characters specified by the I/O list and the specification list (n) is greater than c (record length) per record, an execution diagnostic occurs. If DECODE attempts to process an illegal BCD code or a character illegal under a given conversion specification, an execution diagnostic occurs.

Examples:

1. The following illustrates one method of packing the partial contents of two words into one word. Information is stored in core as:

```
LOC(1) SSSSSxxxxx
          .
          .
          .
LOC(6) xxxxxddddd

          10 BCD ch/wd
```

   To form SSSSSddddd in storage location NAME:

```
     DECODE (10,1,LOC(6))TEMP
1    FORMAT (5X,A5)
     ENCODE (10,2,NAME)LOC(1),TEMP
2    FORMAT(2A5)
```

   The DECODE statement places the last 5 BCD characters of LOC(6) into the first 5 characters of TEMP. The ENCODE statement packs the first 5 characters of LOC(1) and TEMP into NAME.

   With the R specification; the program may be shortened to:

```
     ENCODE (10,1,NAME)LOC(1),LOC(6)
1    FORMAT (A5,R5)
```

2. DECODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement FORMAT (2A8, Im) the programmer wishes to specify m at some point in the program, subject to the restriction $2 \leq m \leq 9$. The following program permits m to vary.

```
          IF(M.LT.10.AND.M.GT.1)1,2
    1     ENCODE (8,100,SPECMAT) M
  100     FORMAT (6H(2A8,I,I1,1H) )
          .
          .
          .
          PRINT SPECMAT,A,B,J
```

M is tested to insure it is within limits. If not, control goes to statement 2 which could be an error routine. If M is within limits, ENCODE packs the integer value of M with the characters: (A8, Im). This packed FORMAT is stored in SPECMAT. SPECMAT contains (2A8, Im).

A and B will be printed under specification A8, and the quantity J under specification I2, or I3, or ... or I9 according to the value of m.

3. ENCODE can be used to rearrange and change the information in a record. The following example also illustrates that it is possible to encode an area into itself and that encoding will destroy information previously contained in an area.

```
          I = 10HV = bbFT/SEC
          IA = 16
          ENCODE (10,1,I)I,IA,I
    1     FORMAT (A2,I2,R6)
```

Before executing the above code

I = 26545555062450230503

After execution

I = 26543441062450230503

TYPE DECLARATIONS AND STORAGE ALLOCATION 5

5.1 TYPE DECLARATION

The type declaration statement provides the compiler with information on the structure of variable and function identifiers.

Statement	Characteristics	
COMPLEX list	2 words/element	Floating Point
DOUBLE PRECISION list or DOUBLE list	2 words/element	Floating Point
REAL list	1 word/element	Floating Point
INTEGER list	1 word/element	Integer
LOGICAL list	1 word/element	Logical

TYPE may precede any of the above statements.

DOUBLE may replace DOUBLE PRECISION in any FORTRAN statement in which the latter is allowed.

USASI FORTRAN, X3.9-1966, does not specify DOUBLE as a replacement for DOUBLE PRECISION.

List is a string of identifiers separated by commas; integer constant subscripts are permitted. For example:

A, B1, CAT, D36F, GAR (1,2,3)

The type declaration is non-executable and must precede the first reference to the variable or function in a given program. If an identifier is declared in two or more type declarations, the first declaration holds until the second is read, the second holds until the third, etc. However, the second and ensuing declarations will result in informative diagnostics.

An identifier not declared in a type declaration is type integer if the first letter of the name is I, J, K, L, M, N; for any other letter, it is type real.

When subscripts appear in the list, the associated identifier is the name of an array, and the product of the subscripts determines the amount of storage to be reserved for that array. By this means, dimension and type information are given in the same statement. In this case no DIMENSION statement is needed. If a second declaration of storage appears, an informative diagnostic is issued and the original declaration is used.

Examples:

```
COMPLEX A412,DATA,DRIVE,IMPORT
DOUBLE PRECISION PLATE,ALPHA(20,20),B2MAX,F60,JUNE
REAL I,J(20,50,2),LOGIC,MPH
INTEGER GAR(60),BETA,ZTANK,AGE,YEAR,DATE
LOGICAL DISJ,IMPL,STROKE,EQUIV,MODAL
DOUBLE RL,MASS(10,10)
```

5.2 DIMENSION DECLARATION

A subscripted variable represents an element of an array of variables. Storage is reserved for arrays by the non-executable statements DIMENSION, COMMON, or a type statement.

The standard form of the DIMENSION declaration is:

DIMENSION $v_1, v_2, \ldots, v_n$

The variable names v_i may have 1, 2, or 3 integer constant subscripts separated by commas, as in SPACE (5, 5, 5). Under certain conditions within subprograms only, the subscripts may be constants or variables.

Example:

```
DIMENSION A(10),B(20,3)
```

The DIMENSION declaration is non-executable and it must precede the first reference to the array in a given program. The DIMENSION statement should precede the first executable statement and will result in an informative diagnostic otherwise.

The number of computer words reserved for an array is determined by the product of the subscripts in the subscript string and the type of the variable. A maximum of $2^{17}-1$ elements may be reserved in any one array. If the maximum is exceeded, a diagnostic is provided.

```
COMPLEX ATOM
DIMENSION ATOM (10,20)
```

In the above declarations, the number of elements in the array ATOM is 200. Two words are used to contain a complex element; therefore, the number of computer words reserved is 400. This is also true for double precision arrays. For real, logical, and integer arrays, the number of words in an array equals the number of elements in the array.

If an array is dimensioned in more than one delcaration statement, the first declaration holds and an informative diagnostic is provided.

Examples:

```
DIMENSION A(20,2,5)
DIMENSION MATRIX(10,10,10), VECTOR(100), ARRAY(16,27)
```

5.2.1 VARIABLE DIMENSIONS

It is possible to vary the dimension of an array associated with a subprogram each time that subprogram is called. This is done by specifying the array identifier and its dimensions as dummy arguments in the subprogram statement or by referencing them as variables in a COMMON declaration in the subprogram. The corresponding actual arguments or common values in the calling program are used by the called subprogram. The maximum dimension that any array may assume is determined by the dimensioning statement in the original calling program, i.e., the subsequent declarations shall not exceed the declaration that originally defined the array.

Example:

```
SUBROUTINE X(A, L, M)
DIMENSION A(L, 10, M)
```

5.3 COMMON DECLARATION

The COMMON declaration provides up to 61 blocks of storage that can be referenced by more than one subprogram. The declaration reserves blank, numbered, and labeled blocks. Starting addresses for these blocks are indicated on the core map.

Areas of common information may be specified by the declaration:

COMMON/i_1/$list_1$/i_2/$list_2$. . .

The common block identifier, i, may be 1-7 characters. If the first character is alphabetic, the identifier denotes a labeled common block; remaining characters may be alphabetic or numeric. If the first character is numeric, remaining characters must be numeric and the identifier denotes a numbered common block. Leading zeros in numeric identifiers are ignored. Zero by itself is an acceptable numbered common identifier. Labeled and numbered COMMON are treated identically by the compiler.

Example:

```
COMMON/200/A, B, C
```

The following are common identifiers:

Labeled	Numbered
AZ13	1
MAXIM	146
Z	6600
XRAY	0

A common statement without a label, or with blanks between the separating slashes is treated as a blank common block, for example:

COMMON / / A, B, C   or   COMMON X, Y, Z(5)

$List_i$ is a string of identifiers representing simple and subscripted variables; formal parameters are not allowed. If a non-subscripted array name appears in the list, the dimensions must be defined by a type or DIMENSION declaration in that program. If an array is dimensioned in more than one declaration, a compiler diagnostic is issued. The order of simple variables or array storage within a common block is determined by the sequence in which the variables appear in the COMMON statements.

The total of labeled and numbered common blocks is limited to 61. Labeled and numbered common blocks may be preset; data stored in them by DATA declarations is made available to any subprogram using the appropriate block. Data may not be entered into blank common blocks by the DATA declaration.

Examples:

1. COMMON/BLK/A(3)

   DATA A/1.,2.,3./

2. COMMON/100/I(4)

   DATA I/4,5,6,7/

COMMON is non-executable and can appear anywhere in the program. Any number of blank COMMON declarations may appear in a program. If DIMENSION, COMMON or type declarations appear together, the order is immaterial.

Since labeled and numbered common block Identifiers are used only within the compiler, they may be used elsewhere in the program as other kinds of identifiers. A list identifier in one common block may not appear in another common block. (If it does, the name is doubly defined.)

At the beginning of program execution, the contents of all common areas are unpredictable except labeled common areas specified in a DATA declaration.

Examples:

```
COMMON A,B,C       }
                   }  Blank Common
COMMON/ /E,F,G,H   }
COMMON/BLOCKA/A1(15),B1,C1/BLOCKD/DEL(5,2),ECHO
COMMON/VECTOR/VECTOR(5),HECTOR,NECTOR
COMMON/9999/AX,BX,CX
```

The length of a common block in computer words is determined from the number and type of the list variables. In the following statements, the length of common block A is 12 computer words. The origin of the common block is Q (1).

```
COMMON/A/Q(4),R(4),S(2)
REAL Q,R
COMPLEX S
```

	Block A	
origin	Q(1)	
	Q(2)	
	Q(3)	
	Q(4)	
	R(1)	
	R(2)	
	R(3)	
	R(4)	
	S(1)	real part
	S(1)	imaginary part
	S(2)	real part
	S(2)	imaginary part

If a subprogram does not use all of the locations reserved in a common block, unused variables may be necessary in the COMMON declaration to insure proper correspondence of common areas.

```
COMMON/SUM/A,B,C,D       (main program)
COMMON/SUM/E(3),D        (subprogram)
```

In the above example, only the variable D is used in the subprogram. The unused variable E is necessary to space over the area reserved by A, B, and C.

Each subprogram using a common block assigns the allocation of words in the block. The identifiers used within the block may differ as to name, type and number of elements; but the block identifier must remain the same.

Example:

```
PROGRAM MAIN
COMPLEX C
COMMON/TEST/C(20)/36/A,B,Z
.
.
.
```

The length of the block named TEST is 40 computer words. The length of the block numbered 36 is 3 computer words.

The subprogram may rearrange the allocation of words as in:

```
SUBROUTINE ONE
COMMON/TEST/A(10),G(10),K(10)
COMPLEX A
.
.
.
```

The length of TEST is 40 words. The first 10 elements (20 words) of the block represented by A are complex elements. Array G is the next 10 words, and array K is the last 10 words. Within the subprogram, elements of G are treated as floating point quantities; elements of K are treated as integer quantities.

The length of a common block other than blank common must not be increased by subprograms using the block unless that subprogram is loaded first by the SCOPE loader. The symbolic names used within the block may differ, however, as shown above.

5.4 EQUIVALENCE DECLARATION

The EQUIVALENCE declaration permits variables to share locations in storage. The general form is:

```
EQUIVALENCE (A,B,...),(A1,B1,...),...
```

(A,B,...) is an equivalence group of two or more simple or subscripted variable names; formal arguments are not allowed. Subscripts may only be integer constants. A multiple subscripted variable can be represented by a singly subscripted variable. The correspondence is:

```
A(i,j,k) is the same as A ((the value of (i+(j-1)*I+(k-1)*I*J))*E)
```

where E is 1 or 2 depending on A's word length, i, j, k are integer constants; I and J are the integer constants appearing in DIMENSION A(I,J,K). For example, in DIMENSION A(2,3,4), the element A(1,1,2) can be represented by A(7).

EQUIVALENCE is most commonly used when two or more arrays can share the same storage locations. The lengths need not be equal.

Example:

```
      DIMENSION A(10,10),I(100)
      EQUIVALENCE (A,I)
  5   READ 10, A
               .
               .
               .
  6   READ 20, I
```

The EQUIVALENCE declaration assigns the first element of array A and array I to the same storage location. The READ statement 5 stores the A array in consecutive locations. Before statement 6 is executed, all operations using A should be completed since the values of array I are read into the storage locations previously occupied by A.

Variables requiring two memory positions which appear in EQUIVALENCE statements must be declared to be COMPLEX or DOUBLE prior to their appearance in such statements.

USASI FORTRAN, X3.9-1966, does not require type declaration prior to equivalence.

Example:

```
      COMPLEX DAT,BAT
      DIMENSION DAT(10,10),BAT(10,10),CAT(10,10)
      DOUBLE PRECISION CAT
      COMMON/IFAT/FAT(2,2)
      EQUIVALENCE (DAT(6,3),FAT(2,2) ),(CAT,BAT)
      .
      .
      .
```

EQUIVALENCE is non-executable and can appear anywhere in the program or subprogram. However, if it appears after the first executable statement, an informative diagnostic is provided.

Any variable may be made equivalent to any other variable, provided that no two variables in any one group are in COMMON. The variables may be with or without subscript.

5.5 DATA DECLARATION

Values may be assigned to program variables or labeled common variables with the DATA declaration:

DATA $d_1,\ldots,d_n/a_1,k*a_2,\ldots,a_n/,d_1,\ldots,d_n/a_1,\ldots,a_n/,\ldots$

d_i identifiers representing simple variables, array names, or variables with integer constant subscripts or integer variable subscripts (implied DO-loop notation).

a_i literals and signed or unsigned constants.

k integer constant repetition factor that causes the literal following the asterisk to be repeated k times. If k is non-integer, a compiler diagnostic occurs.

USASI FORTRAN, X3.9-1966, specifies the form DATA $k_1/d_1/,k_2/d_2/,\ldots,k_n/d_n/$

A semicolon cannot be used in the character string of data entered under L, R or H control.

DATA is non-executable and can appear anywhere in the program or subprogram. When DATA appears with DIMENSION, COMMON, EQUIVALENCE, or a type declaration, the statement that dimensions any arrays used in the DATA statement must appear prior to the DATA statement. Variables in blank common or formal arguments may not be preset by a DATA declaration.

Only single-subscript, DO-loop-implying notation is permissible. This notation may be used for storing constant values in arrays.

USASI FORTRAN, X3.9-1966, does not specify the use of DO-loop-implying notation for storing constants in arrays.

Examples:

1. DIMENSION GIB(10)

   DATA (GIB(I), I=1, 10)/1.,2.,3.,7*4.32/

   Array GIB: 1.
   2.
   3.
   4.32
   4.32
   4.32
   4.32
   4.32
   4.32
   4.32

2. DIMENSION TWO(2,2)

```
DATA TWO(1,1),TWO(1,2),TWO(2,2),TWO(2,1)/1.,2.,3.,4./
```

Array TWO:	TWO(1,1)	1.
	TWO(2,1)	4.
	TWO(1,2)	2.
	TWO(2,2)	3.

3. DIMENSION SINGLE(3,2)

```
DATA (SINGLE(I),I=1,6)/1.,2.,3.,1.,2.,3./
```

Array SINGLE:	SINGLE(1,1)	1.
	SINGLE(2,1)	2.
	SINGLE(3,1)	3.
	SINGLE(1,2)	1.
	SINGLE(2,2)	2.
	SINGLE(3,2)	3.

In the DATA declaration, the type of the constant stored is determined by the structure of the constant rather than by the variable type in the statement. in DATA A/2/, an integer 2 replaces A, not a real 2 as might be expected from the form of the symbolic name A. Data types requiring two words per element must be properly specified to maintain correct correspondence in memory.

There should be a one-to-one correspondence between the variable names and the list. This is particularly important in arrays in labeled common. For instance:

```
COMMON/BLK/A(3),B
DATA A/1.,2.,3.,4./
```

The constants 1.,2.,3., are stored in array locations A,A+1,A+2; the constant 4. is discarded; B is unmodified and an error is issued. If this occurs unintentionally, errors may occur when B is referred to elsewhere in the program.

```
COMMON/TUP/C(3)
DATA C/1.,2./
```

The constants 1.,2. are stored in array locations C and C+1; the content of C(3), that is, location C+2, is not defined.

When the number of list elements exceeds the range of the implied DO, the excess list elements are not stored, and a diagnostic is issued.

```
DATA (A(I),I=1,5,1)/1.,2.,...,10./
```

The excess values 6. through 10. are discarded.

Examples:

1. DATA LEDA,CASTOR,POLLUX/15,16.0,84.0/

LEDA	15
	.
	.
	.
CASTOR	16.0
	.
	.
	.
POLLUX	84.0

2. DATA A(1,3)/16.239/

   ARRAY A

A(1,3)	16.239

3. DIMENSION B(10)

   DATA B/O000077, O000064,3*O000005, 5*O000200/

ARRAY B	O77
	O64
	O5
	O5
	O5
	O200
	O200
	O200
	O200
	O200

4. COMMON/HERA/C(4)

   DATA C/3.6,3*10.5/

ARRAY C	3.6
	10.5
	10.5
	10.5

5. COMPLEX PROTER (4)

DATA PROTER/4*(1.0,2.0)/

ARRAY PROTER	1.0
	2.0
	1.0
	2.0
	1.0
	2.0
	1.0
	2.0

6. DIMENSION MESSAGE (3)

DATA MESSAGE/9HSTATEMENT,2HIS,10HINCOMPLETE/

ARRAY MESSAGE	STATEMENT IS INCOMPLETE

Data declaration statements of the following forms may also be used to assign constant values to program or common variables at load time.

DATA(i_1=value list), (i_2=value list),...

The variable identifier, i, may be:

non-subscripted variable

array variable with constant subscripts

array name

The value list is either a single constant or set of constants whose number is equal to the number of elements in the named array.

List contains constants only and has the form:

$a_1, a_2, \ldots, k(b_1, b_2, \ldots), c_1, c_2, \ldots$

k is an integer constant repetition factor that causes the parenthetical list following it to be repeated k times. If k is non-integer, a compiler diagnostic is provided.

Examples:

```
COMMON/DATA/GIB
DATA ( (GIB(I),I=1,10)=1.,2.,3.,7(4.32) )
COMMON/DATA/ROBIN(5,5,5)
DATA (ROBIN(4,3,2)=16.)
```

5.5.1 BLOCK DATA SUBPROGRAM

A block data subprogram may be used to enter data into labeled or numbered common prior to program execution in place of a DATA declaration and it may appear more than once in a FORTRAN program.

The form of a BLOCK DATA subprogram is:

```
BLOCK DATA
     .
     .
     .
FORTRAN declaration statements only
     .
     .
     .
END
```

Examples:

```
BLOCK DATA

COMMON/ABC/A(5), B, C/DEF/D, E, F

COMPLEX D, E

DOUBLE PRECISION F

DATA (A(L), L=1, 5)/2.3,3.4,3*7.1/, B/2034.756/, D, E, F/2*(1.0,2.5),17.86972415872D30/

END

BLOCK DATA

COMMON/DEF/G,H,I
     .
     .
     .
END
```

7000 SERIES FORTRAN CHARACTER CODES A

Source Language Character	Console Display Code	External BCD Code	Punch Position in a Hollerith Card Column
A	01	61	12-1
B	02	62	12-2
C	03	63	12-3
D	04	64	12-4
E	05	65	12-5
F	06	66	12-6
G	07	67	12-7
H	10	70	12-8
I	11	71	12-9
J	12	41	11-1
K	13	42	11-2
L	14	43	11-3
M	15	44	11-4
N	16	45	11-5
O	17	46	11-6
P	20	47	11-7
Q	21	50	11-8
R	22	51	11-9
S	23	22	0-2
T	24	23	0-3
U	25	24	0-4
V	26	25	0-5
W	27	26	0-6
X	30	27	0-7
Y	31	30	0-8
Z	32	31	0-9
0	33	12	0
1	34	01	1
2	35	02	2
3	36	03	3
4	37	04	4
5	40	05	5
6	41	06	6
7	42	07	7
8	43	10	8
9	44	11	9
+	45	60	12
-	46	40	11
*	47	54	11-8-4

Source Language Character	Console Display Code	External BCD Code	Punch Position in a Hollerith Card Column
/	50	21	0-1
(	51	34	0-8-4
)	52	74	12-8-4
$	53	53	11-8-3
=	54	13	8-3
blank(space)	55	20	space
,	56	33	0-8-3
.	57	73	12-8-3

ADDITIONAL CHARACTERS

Character	Console Display Code	External BCD Code	Hollerith Card Punch
≡	60	36	0-8-6
[	61	17	8-7
]	62	32	0-8-2
%	63	16	8-6
≠	64	14	8-4
→	65	35	0-8-5
∨	66	52	11-0†
∧	67	37	0-8-7
↑	70	55	11-8-5
↓	71	56	11-8-6
<	72	72	12-0††
>	73	57	11-8-7
≤	74	15	8-5
≥	75	75	12-8-5
¬	76	76	12-8-6
;	77	77	12-8-7
end-of-line	0000	1632	

† 11-0 and 11-8-2 are equivalent
†† 12-0 and 12-8-2 are equivalent

FORTRAN STATEMENT LIST

B

SUBPROGRAM STATEMENTS

			Page
Entry Points	PROGRAM name $(f_1,\ldots,f_n)$	N	7-1
	SUBROUTINE name $(p_1,\ldots,p_n)$	N	7-12
	FUNCTION name $(p_1,\ldots,p_n)$	N	7-10
	type FUNCTION name $(p_1,\ldots,p_n)$	N	7-10
	ENTRY name	N	7-19
Intersubroutine	EXTERNAL $name_1, name_2\ldots$	N	7-16
Transfer Statements	CALL name	E	7-15
	CALL name $(p_1,\ldots,p_n)$	E	7-15
	RETURN	E	6-11

DATA DECLARATION AND STORAGE ALLOCATION

Type Declaration	COMPLEX List	N	5-1
	DOUBLE PRECISION List	N	5-1
	DOUBLE List	N	5-1
	REAL List	N	5-1
	INTEGER List	N	5-1
	LOGICAL List	N	5-1
	TYPE DOUBLE List	N	5-1
	TYPE COMPLEX List	N	5-1
	TYPE REAL List	N	5-1
	TYPE INTEGER List	N	5-1
	TYPE LOGICAL List	N	5-1

N = Non-executable   E = Executable

			Page
Storage Allocations	DIMENSION $V_1, V_2, \ldots, V_n$	N	5-2
	COMMON/I_1/$List_1$/I_2/$List_2$.../I_n/$List_n$	N	5-3
	EQUIVALENCE (A,B,...), (A1,B1,...)...	N	5-6
	DATA I_1/List/,I_2/List/,...	N	5-8
	DATA (I_1=List), (I_2=List),...	N	5-8
	BLOCK DATA	N	5-12
ARITHMETIC STATEMENT FUNCTION			
	Name $(p_1, p_2, \ldots, p_n)$ = Expressions	E	7-6
SYMBOL MANIPULATION, CONTROL AND I/O			
Replacement	A=E Arithmetic	E	4-1
	L=E Logical/Relational	E	4-4
	M=E Masking	E	4-4
Intraprogram Transfers	GO TO n	E	6-1
	GO TO m	E	6-1
	GO TO m, $(n_1, \ldots, n_m)$	E	6-1
	GO TO $(n_1, \ldots, n_m)$, i	E	6-2
	IF (c)n_1, n_2, n_3	E	6-3
	IF (ℓ)n_1, n_2	E	6-4
	IF (ℓ)s	E	6-4
	IF (ENDFILE i)n_1, n_2	E	10-8
	IF (EOF, i)n_1, n_2	E	10-8
	IF (UNIT, i)n_1, n_2, n_3, n_4	E	10-8
LOOP CONTROL	DO n i = m_1, m_2, m_3	E	6-5
	DO n i = m_1, m_2	E	6-5
MISCELLANEOUS PROGRAM CONTROLS			
	ASSIGN s to m	E	6-2
	CONTINUE	E	6-10
	PAUSE	E	6-10
	PAUSE n	E	6-10
	STOP	E	6-11
	STOP n	E	6-11

			Page
I/O FORMAT			
	FORMAT ($spec_1$, $spec_2$, ...)	N	9-4
	NAMELIST/name/list...	N	10-5
I/O CONTROL STATEMENTS			
	READ n, L	E	10-4
	PRINT n, L	E	10-1
	PUNCH n, L	E	10-2
	READ (i, n)L	E	10-4
	WRITE (i, n)L	E	10-2
	READ (i)L	E	10-4
	WRITE (i)L	E	10-3
	ENCODE (c, n, v)L	E	10-11
	DECODE (c, n, v)L	E	10-12
	BUFFER IN (u, p)(A, B)	E	10-11
	BUFFER OUT (u, p)(A, B)	E	10-11
I/O Tape Handling	END FILE i	E	10-8
	REWIND i	E	10-8
	BACKSPACE i	E	10-8
PROGRAM AND SUBPROGRAM TERMINATION			
	END	E	6-11
	END name	E	6-11
ADDITIONAL STATEMENTS			
	LARGE $t_1 . v_1(i_1), t_2 . v_2(i_2) ... t_n . v_n(i_n)$	N	L-1
	SMALLIN (s, l, w)	E	L-2
	SMALLOUT (s, l, w)	E	L-2

FORTRAN FUNCTIONS C

INTRINSIC FUNCTIONS (IN-LINE)

Form	Definition	Actual Parameter Type	Mode of Result
ABS(X)	Absolute value	Real	Real
AIMAG(C)	Obtain the imaginary part of a complex argument	Complex	Real
AINT(X)	Truncation integer. Sign of X times largest integer $\leq \lvert X \rvert$.	Real	Real
AMAX0($I_1, I_2, \ldots$)	Determine maximum argument	Integer	Real
AMAX1($X_1, X_2, \ldots$)	Determine maximum argument	Real	Real
AMIN0($I_1, I_2, \ldots$)	Determine minimum argument	Integer	Real
AMIN1($X_1, X_2, \ldots$)	Determine minimum argument	Real	Real
AMOD(X_1, X_2)†		Real	Real
AND($X_1, \ldots, X_n$)	Boolean AND of $X_1, \ldots, X_n$	-	Logical
CMPLX(X_1, X_2)	Convert real to complex (X_1+iX_2)	Real	Complex
COMPL(X)	Complement of X	-	Logical
CONJG(C)	Conjugate of C	Complex	Complex
DIM(X_1, X_2)	If $X_1 > X_2 : X_1 - X_2$ If $X_1 \leq X_2 : 0$	Real	Real
DMAX1($D_1, D_2, \ldots$)	Determine maximum argument	Double	Double
DMIN1(D_1, D_2,	Determine minimum argument	Double	Double
FLOAT(I)	Integer to real conversion	Integer	Real
IABS(I)	Absolute value	Integer	Integer
IDIM(I_1, I_2)	If $I_1 > I_2 : I_1 - I_2$ If $I_1 \leq I_2 : 0$	Integer	Integer
IFIX(X)	Real to integer conversion	Real	Integer
INT(X)	Truncation, integer. Sign of X times largest integer $\leq \lvert X \rvert$	Real	Integer
ISIGN(I_1, I_2)	Sign of I_2 times absolute value of I_1.	Integer	Integer
MAX0($I_1, I_2, \ldots$)	Determine maximum argument	Integer	Integer
MAX1($X_1, X_2, \ldots$)	Determine maximum argument	Real	Integer
MIN0($I_1, I_2, \ldots$)	Determine minimum argument	Integer	Integer
MIN1($X_1, X_2, \ldots$)	Determine minimum argument	Real	Integer
MOD(I_1, I_2) †		Integer	Integer
OR($X_1, \ldots, X_n$)	Boolean OR of $X_1, \ldots, X_n$	-	Logical
REAL(C)	Obtain the real part of a complex argument	Complex	Real
SIGN(X_1, X_2)	Sign of X_2 times absolute value of X_1.	Real	Real

† AMOD(X_1, X_2) is defined as $X_1 - [X_1/X_2]X_2$, where [x] is an integer with magnitude of not more than x and sign the same as x.

BASIC EXTERNAL FUNCTIONS (LIBRARY)

ACOS(X)	Arccosine in radians	Real	Real
ALOG(X)	Natural log of X	Real	Real
ALOG10(X)	Log to the base 10 of X	Real	Real
ASIN(X)	Arcsine in radians	Real	Real
ATAN(X)	Arctangent in radians	Real	Real
ATAN2(X_1, X_2)	Arctangent (X_1/X_2) in radians	Real	Real
CABS(C)	Absolute value	Complex	Real
CCOS(C)	Complex cosine, argument in radians	Complex	Complex
CEXP(C)	Complex exponent	Complex	Complex
CLOG(C)	Complex log	Complex	Complex
COS(X)	Cosine X radians	Real	Real
CSIN(C)	Complex sine, argument in radians	Complex	Complex
CSQRT(C)	Complex square root	Complex	Complex
DABS(D)[††]	Absolute value	Double	Double
DATAN(D)	Double arctangent in radians	Double	Double
DATAN2(D_1, D_2)	Double arctangent: D_1/D_2 in radians	Double	Double
DBLE(X)[††]	Real to double	Real	Double
DCOS(D)	Double cosine, argument in radians	Double	Double
DEXP(D)	Double exponent	Double	Double
DLOG(D)	Natural log of D	Double	Double
DLOG10(D)	Log to the base 10 of D	Double	Double
DMOD(D_1, D_2)[†]		Double	Double
DSIGN(D_1, D_2)[††]	Sign of: D_2 times absolute value of D_1	Double	Double
DSIN(D)	Sine of double precision argument in radians	Double	Double
DSQRT(D)	Double square root	Double	Double
EXP(X)	e to Xth power	Real	Real
IDINT(D)[††]	Double to integer. Sign of D times largest integer ≤ |D|	Double	Integer
LEGVAR(A)	Returns -1 if variable is indefinite, +1 if out of range, and 0 if normal. For LEGVAR to be effective, the program must be run in mode 0 to disable floating point interrupts.	Real	Integer
LENGTH(I)	Returns number of words transferred to CM from unit I after BUFFER IN	Integer	Integer
RANF(X)	Random number generator; typical call follows: Y=RANF(X) where X is type real. Three conditions exist on X. 1. If X is zero, the next random number is generated and returned. 2. If X is negative, a random number is not generated but the last previously generated random number (or the seed if not random number has been generated) is returned.	Real	Real

[†] DMOD(D_1, D_2) is defined as $D_1 - [D_1/D_2]D_2$, where [x] is an integer with magnitude of not more than x and sign the same as x.

[††] USASI FORTRAN, X3.9-1966, specifies these functions as intrinsic

RANF(X) (contd)	3. If X is positive, the exponent part of X is set to 1717_8 and the low order bit is set to one. This result is returned as the seed of a new sequence, and any additional calls to RANF will be based on a sequence using this seed.		
SNGL(D)[††]	Double to real (unrounded)	Double	Real
SIN(X)	Sine X radians	Real	Real
SQRT(X)	Square root of X	Real	Real
TAN(X)	Tangent X radians	Real	Real
TANH(X)	Hyperbolic tangent X radians	Real	Real

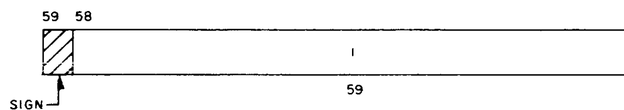
Following functions accept A as a variable address name for an actual parameter:

Form	Definition	Actual Parameter Type	Mode of Result
LOCF(A)	Returns address of argument A	-	Integer

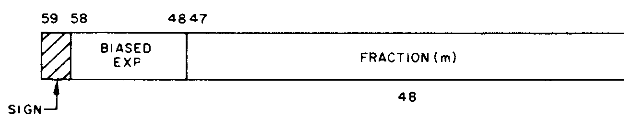
[††] USASI FORTRAN, X3.9-1966, specifies these functions as intrinsic

COMPUTER WORD STRUCTURE OF CONSTANTS — 7600 D

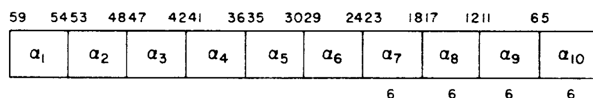
INTEGER



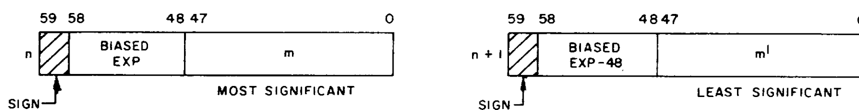
REAL



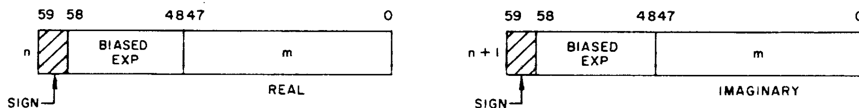
HOLLERITH BCD AND DISPLAY CODE



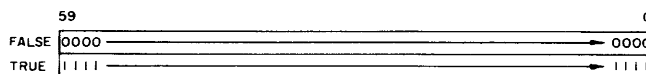
DOUBLE-PRECISION



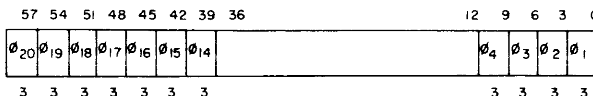
COMPLEX



LOGICAL



OCTAL



COMPILATION AND EXECUTION E

FORTRAN Control Card

The 7600 FORTRAN compiler is called by the control card:

RUN(cm, , , if, of, bf, lc, as, cs, t)

cm Compiler mode option; (if omitted, assume G)

- G compile and execute, no list unless explicit LIST cards appear in the deck
- S compile with source list, no execute
- P† compile with source list and punch deck on file PUNCHB, no execute
- L compile with source and object list which contains mnemonics, no execute
- M† compile with source and object list which contains mnemonics, produce a punch deck on file PUNCHB, no execute

if file name for compiler input; if omitted assumed to be INPUT

of file name for compiler output; if omitted assumed to be OUTPUT

bf file name on which the binary information is always written; if omitted, assumed to be LGO.

lc line-limit (octal) on the OUTPUT file of an object program. If omitted, assumed to be 10000_8.

as if non-zero or non-blank, the USASI switch causes the USASI I/O list/FORMAT interaction at execution time. This allows the unlimited group repeat and scale factor as defined in USASI. It has no effect on the compilation method.

cs cross-reference switch. If non-zero a cross reference listing is produced.

t error traceback. When this parameter is present, calls to library functions will be made with maximum error checking. Full error traceback will be done if an error is detected. When t is omitted, minimum error checking will be done and no traceback will be provided if errors are encountered. Thus, a significant saving in memory space and execution time is realized. This mode of compilation (t omitted) is not intended for use with programs in the debug stages.

The second and third positions are allowed so that 6000 compatibility is maintained. If used they are ignored by the 7600 compiler.

---

† Because COMPASS allows only one binary output file to be written, a RUN (P or M) and LGO will result in only the FORTRAN code of a FORTRAN–COMPASS job being placed on LGO.

Compiler output, except in the G mode, includes a reproduction of the source program, a variable map, and indications of fatal and non-fatal errors detected during compilation. If the G mode is selected, all output is suppressed unless fatal errors are detected in which case the output contains the statement in error and the error diagnostic messages. If the L or M mode is selected, the output includes an object list which contains mnemonics.

On L or M mode listings, the following lines will appear:

PS	(preamble start)
PT	(preamble terminate)
CS	(conclusion start)
CT	(conclusion terminate)

These identify statement sequences where some common computation has been extracted and performed before entering the sequence.

A copy of the compiled programs is always left in disk storage as a binary record on a file named either LGO or the name specified as the bf parameter in the call to the compiler. The compiled program may be called and executed repeatedly, until the end of job occurs, by using the name of the load-and-go file. In the output file at the end of compilation of each subprogram, the compiler indicates the amount of unused compilation space.

Two control cards LIST and NOLIST are available to allow the programmer more flexibility in requesting a list of his programs. These cards are free field beyond column 6 and appear between subprograms. When the LIST card is detected, the source cards of the following programs are listed. If the compiler mode was L, the object code is also listed. When the NOLIST card is detected no more listing takes place until a LIST card is detected.

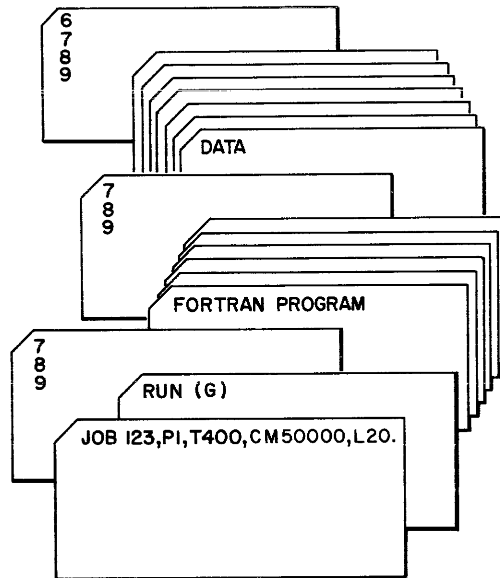
To aid in the preparation of overlay files, the FORTRAN compiler, upon detecting an OVERLAY card between subprograms, transfers them to the load-and-go file, and to the PUNCHB file if the P or M option is selected. They also are transferred to the output file.

The following control card is transferred to the load-and-go and PUNCHB file if mode is P or M:

```
OVERLAY (...
```

This statement must begin after column 6.

COMPILE AND EXECUTE



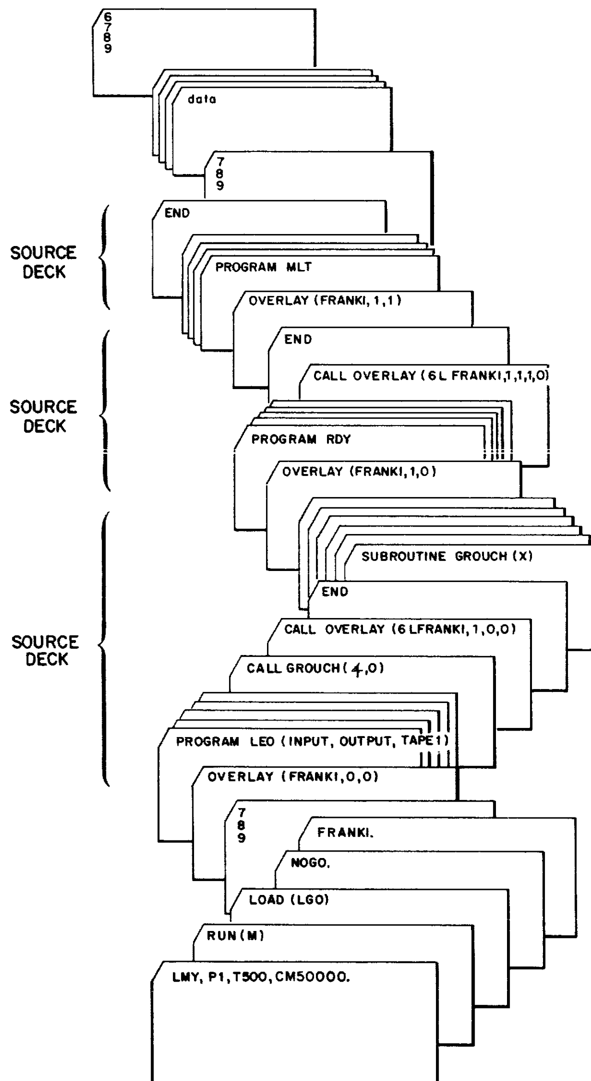
The above control card sequence will compile and run in a field of 50000_8 words.

DECK STRUCTURE FOR A NORMAL COMPILE AND EXECUTE

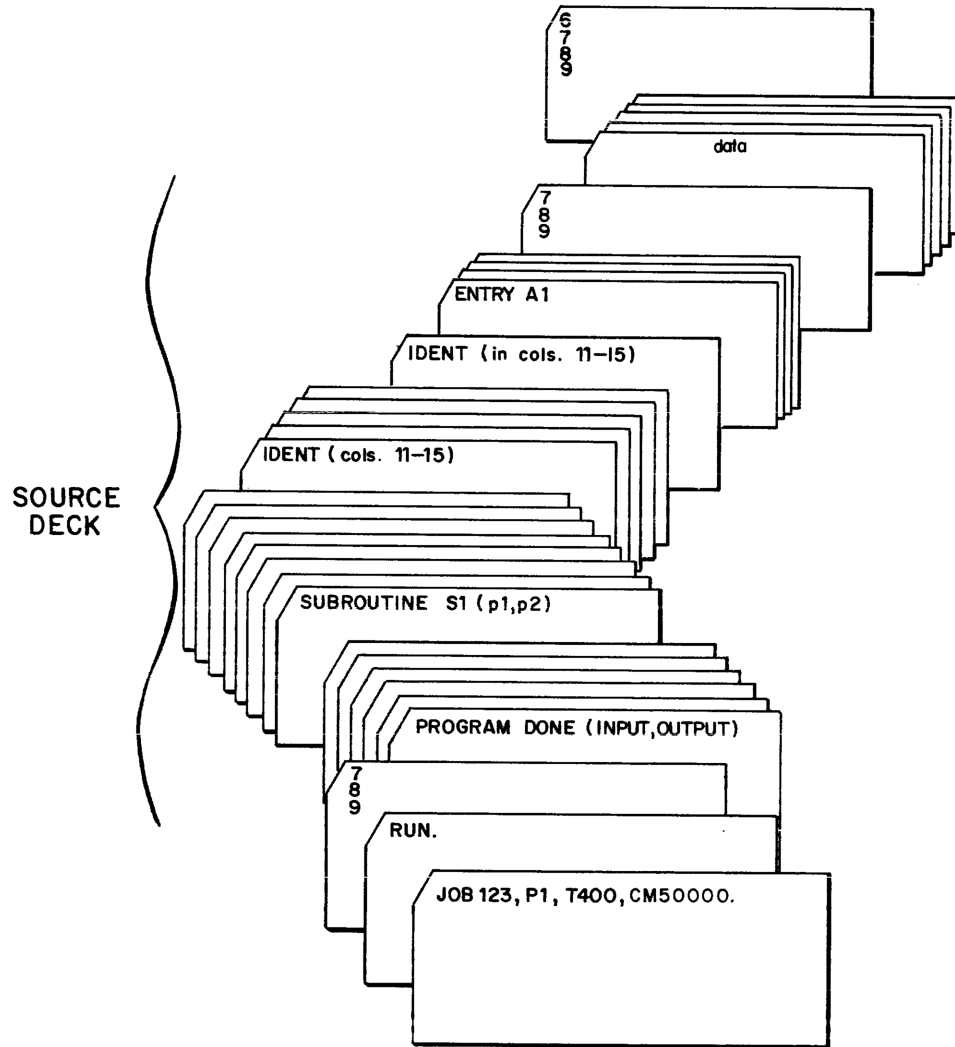
Job name	JOB123
Priority	1
Time limit	approx. 4 minutes
SCM Field Length	50000_8 words
LCM Field Length	20000_8 words

Compile and execute with no list and no binary deck.

Overlay Preparation of 0, 0;1, 0;1, 1



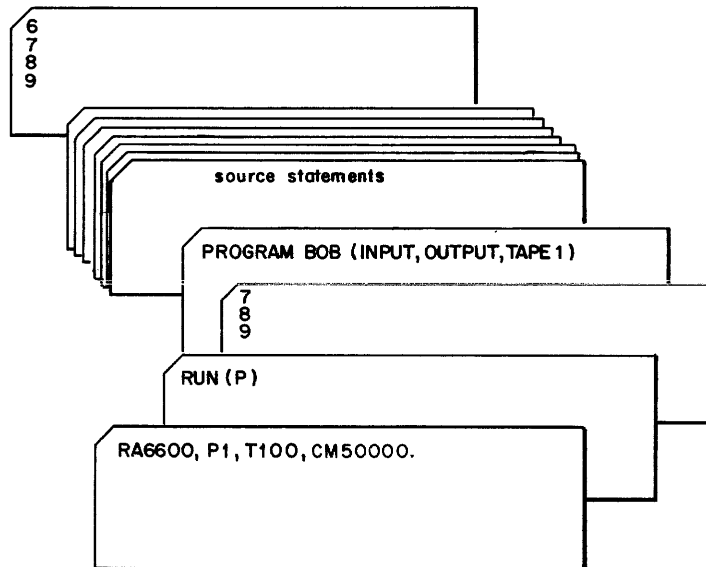
FORTRAN Compile and Execute with Mixed Deck



FORTRAN Compile and Produce Binary Card; no execution.

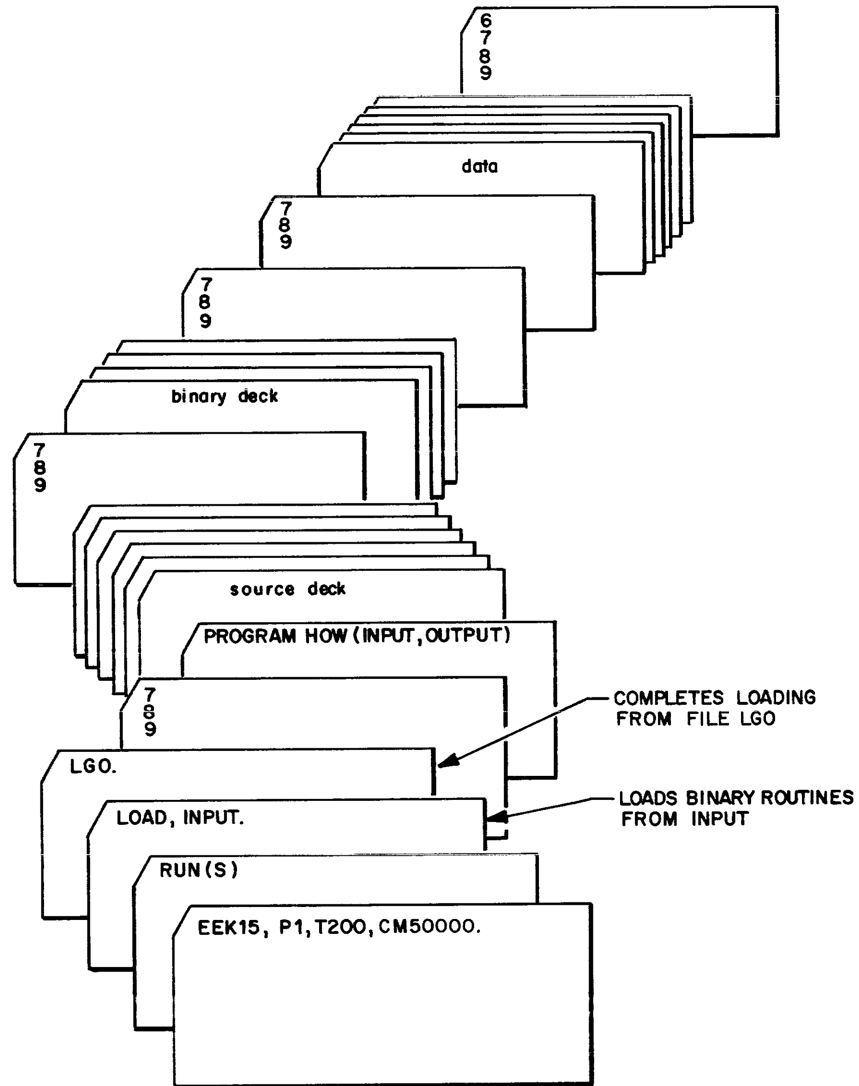
Three files of I/O - INPUT, OUTPUT AND TAPE1

Job name	RA6600
Priority	1
Time limit	approximately 1 minute
Field length	50000_8 words



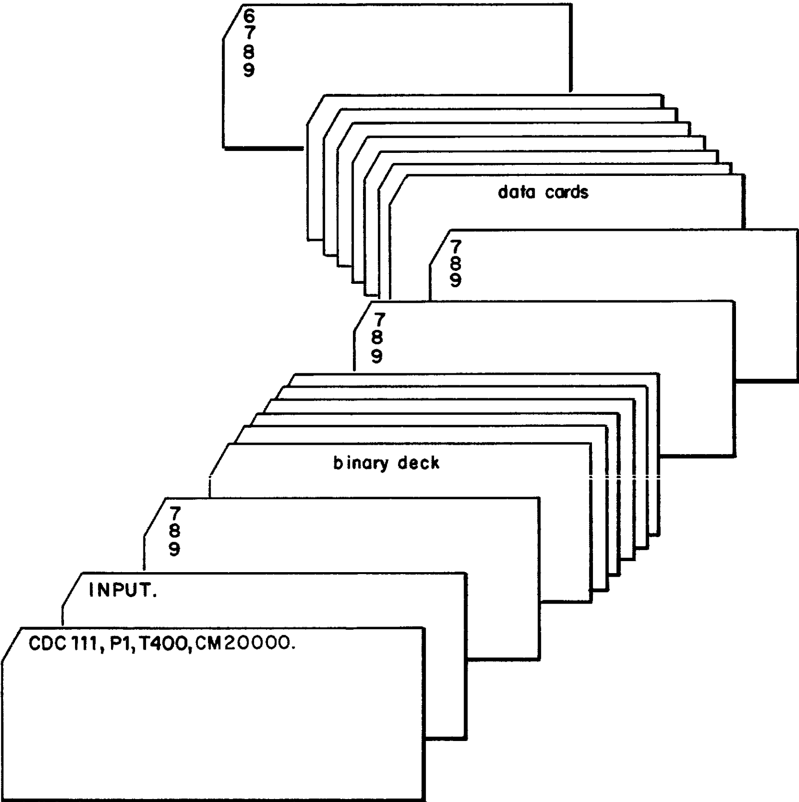
FORTRAN Compile and Execute (plus a prepunched binary subroutine deck)

A binary deck to be loaded with a compiled routine must be preceded by 7, 8, 9 card.

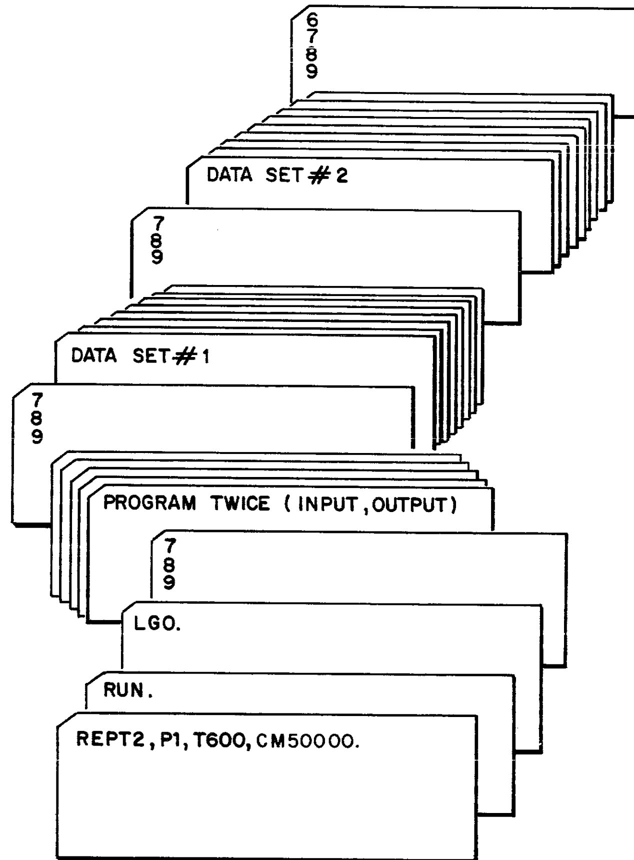


Load and Execute a Prepunched Binary Program

The binary cards in the input file following the record separator are loaded into central memory when the program call card INPUT is encountered.



Compile Once and Execute Twice† with Different Data Decks



† Program TWICE must read the end of record card.

DIAGNOSTICS F

During a FORTRAN compilation, 2- or 3-character error printouts follow statements which are incorrect; other printouts may follow the end statement indicating types of errors in the program. The short printouts produce a more descriptive full line diagnostic which is printed after the subprogram has been compiled.

The full line diagnostic contains a number to identify the statement in which the error was detected. In the short diagnostic an F as the third letter indicates a fatal error.

Fatal errors force diagnostic and statement in error to be listed and inhibit the production of a relocatable record of the subprogram.

Non-fatal errors do not force a listing and will only appear if some type of listing has been specified or if fatal errors have been detected.

The two-character error indicators are defined below:

*****AC** INCORRECT ARGUMENT COUNT

Indicates that the number of arguments in this reference to a subroutine differs from the number which occurred in a prior reference.

*****AE** RECURSIVE CALL

The arithmetic statement function being compiled references itself.

*****AF** MISPLACED STATEMENT FUNCTION

The arithmetic statement function has a statement number or appears after the first executable statement.

*****AL** SYNTAX ERROR IN ARGUMENT LIST

Indicates a format error in a list of arguments.

*****AS** SYNTAX ERROR IN ASSIGNMENT STATEMENT

Indicates a format error in an ASSIGN statement.

*****BC** SYNTAX ERROR IN BOOLEAN CONSTANT

*****BD** EXECUTABLE STATEMENTS IN A BLOCK DATA SUBPROGRAM

*****BX** BOOLEAN EXPRESSION ERROR

Indicates a format error in the designation of a FORTRAN boolean statement in a B-type expression

*****CB** LABELED COMMON BLOCKS EXCEED MAX OF 61

Attempt made to use more than 61 labeled common blocks.

*****CD** VARIABLE DUPLICATED IN A COMMON REGION

Indicates that a variable currently being assigned to the COMMON region has been previously assigned to this region.

*****CE** VARIABLES ASSIGNED TO COMMON ARE IMPROPERLY EQUIVALENCED

Indicates that two variables assigned to COMMON are improperly equivalenced.

*****CL** SYNTAX ERROR IN CALL STATEMENT

Indicates a format error in a call statement.

*****CM** SYNTAX ERROR IN COMMON STATEMENT

Indicates a format error in a COMMON statement.

*****CN** CONTINUATION CARD SEQUENCE ERROR

Indicates that more than 19 continuation cards appear in succession or that one such card appears in an illogical sequence.

*****CO** COMMON STORAGE EXCEEDED

Indicates that the amount of COMMON storage required by the main program or specified to the compiler is less than required by the current program or subroutine.

*****DA** DUPLICATE ARGUMENTS IN A FUNCTION DEFINITION STATEMENT

*****DB** ARRAY SIZE OUT OF RANGE

Indicates the requested array size exceeds 131K. A constant used as an array subscript cannot be contained in 17 bits.

*****DC** SYNTAX ERROR IN A DECIMAL CONSTANT

Indicates a format error in the expression of a FORTRAN decimal constant.

*****DD** VARIABLE BEING DIMENSIONED HAS BEEN PREVIOUSLY DIMENSIONED

Indicates a variable has appeared in more than one DIMENSION statement.

*****DF** DUPLICATE FUNCTION NAME

Indicates that the function name in the current function-definition statement has occurred as the name of a previously defined function.

*****DI** DO TERMINATOR PREVIOUSLY DEFINED

The terminator of this DO loop has already been defined.

*****DJ** INDEX OF OUTER DO REDEFINED BY INNER DO

*****DL** DECLARATIVE APPEARS AFTER FIRST EXECUTABLE STATEMENT

The declarative statement appears after the first executable statement.

*****DM** SYNTAX ERROR IN ARRAY DIMENSION

Indicates an error in the format of a DIMENSION statement.

*****DN** ILLEGAL DO TERMINATOR

This statement cannot be used as a DO terminator. Indicates the attempt to use a FORMAT, GO TO, arithmetic IF, or another DO statement as the termination statement of a DO.

*****DO** SYNTAX ERROR IN A DO STATEMENT

Indicates an error in the format of a DO statement.

*****DP** MULTIPLY DEFINED STATEMENT NUMBER

Indicates the current statement has previously appeared in the statement number field.

*****DQ** UNDIMENSIONED ARRAY IN DATA STATEMENT

Syntax error or undimensioned variable in DATA statement

*****DR** DATA RANGE ERROR

Attempt to store data out of range of array.

*****DS** UNDEFINED STATEMENT NUMBER IN A DO LOOP

Indicates that references have been made in DO statements to statement numbers which did not appear anywhere in the statement number field.

*****DT** SYNTAX ERROR IN DATA STATEMENT

Indicates an error in the format of a DATA statement.

*****DU** ATTEMPT HAS BEEN MADE TO PRESTORE BLANK COMMON.

*****EC** CONTRADICTION IN EQUIVALENCE STATEMENT

Indicates that a variable currently appearing in an EQUIVALENCE statement cannot be equivalenced because of an inherent contradiction in the statement.

*****EF** END OF FILE CARD ENCOUNTERED, END CARD ASSUMED

Indicates that an end of file card is detected before the last END card is encountered.

*****EM** SYNTAX ERROR IN INDICATED EXPONENTIATION

Indicates the mode of the base or the exponent of an indicated exponentiation process is improper.

*****EQ** SYNTAX ERROR IN EQUIVALENCE STATEMENT

Indicates an error in the format of an EQUIVALENCE statement.

*****EX** SYNTAX ERROR IN EXPONENT

Indicates an error in the exponent portion of an indicated exponentitation process.

*****FA** FUNCTION HAS NO ARGUMENT

The FUNCTION has void parameter list; at least one argument is required.

*****FL** SYNTAX ERROR IN EXTERNAL OR F-TYPE STATEMENT

Indicates an error in an EXTERNAL statement of F-TYPE statement.

*****FM** UNRECOGNIZABLE STATEMENT

Indicates a statement whose type cannot be determined.

*****FN** NO STATEMENT NUMBER ON FORMAT STATEMENT

Indicates that a FORMAT statement is missing a statement number.

*****FP** ILLEGAL USE OF FORMAL PARAMETER

Formal parameter used in a COMMON statement.

*****FS** ERROR INSPECIFICATION PORTION OF FORMAT STATEMENT

Indicates a format error in the specification portion of a FORMAT statement.

*****FT** SYNTAX ERROR IN FUNCTION TYPE STATEMENT

*****GF** CONFLICT IN USAGE OF FORMAT OR STATEMENT NUMBER

A FORMAT number is referenced in a control statement or an executable statement number is referenced as a FORMAT number.

*****GO** SYNTAX ERROR IN A GO TO STATEMENT

Indicates an error in the format of a GO TO statement.

*****HC** HOLLERITH CONSTANT LONGER THAN ONE WORD

*****IC** CHARACTER NOT IN FORTRAN CHARACTER SET

Attempt was made to use a character other than those listed in appendix A.

*****ID** IMPROPERLY NESTED DO LOOPS

The sequence of DO loops is improper.

*****IF** SYNTAX ERROR IN AN IF STATEMENT

Indicates an error in the format of an IF statement.

*****IL** SYNTAX ERROR IN AN INDEXED LIST OF I/O STATEMENT

Indicates a format error in an indexed list of the current input/output statement.

*****IN* ILLEGAL FUNCTION NAME

The name of a function reference starts with a number.

*****IO** ILLEGAL I/O DESIGNATOR

I/O designator has a variable name of more than six characters or a numeric value of more than two digits or is alphanumeric and begins with a number.

*****IS**	ILLEGAL USE OF PROGRAM, SUBROUTINE, OR FUNCTION NAME
*****IT**	ILLEGAL TRANSFER TO DO TERMINATOR
	A transfer to a DO terminator is not allowed if the terminator has already been defined and no transfer to it appeared before it was defined.
LF	SYNTAX ERROR IN LARGE STATEMENT
	Format error in LARGE statement
*****LN**	NAMELIST ERROR
LO	LCM OVERFLOW
*****LS**	SYNTAX ERROR INPUT/OUTPUT LIST
	Indicates an error in the format of an input/output list.
*****MA**	MISUSED SUBROUTINE ARGUMENT IN EQUIVALENCE STATEMENT
	Indicates that an argument of the subroutine or function being compiled has been misused in an EQUIVALENCE statement.
*****MO**	MEMORY OVERFLOW, FIELD LENGTH TOO SHORT
	Indicates that the compiler field length, as specified on the JOB card, is too short.
*****MS*	UNDEFINED STATEMENT NUMBER
	Indicates that references have been made to statement labels which did not appear somewhere in the statement-label field of a line.
*****NC**	SUBROUTINE OR FUNCTION NAME CONFLICTS WITH A PRIOR USAGE
*****NL**	NAMELIST NAME NOT UNIQUE
*****NM**	IMPROPER HEADER CARD
	Indicates an error in the formatting of the name (header) card.
*****NO**	NO OBJECT CODE GENERATED
	Source program has generated no object code. This error will occur if a void file is input to the compiler.

*****NP** NO PATH TO THIS STATEMENT

The flagged statement cannot be executed at object time; program continues.

*****NV** VARIABLE DIMENSIONED ARRAY IN NAMELIST

*****OD** REFERENCE TO AN ARRAY BEFORE IT IS DIMENSIONED

*****PM** FUNCTION PARAMETER MODE INCONSISTENCY

Indicates the parameters in a function reference do not agree in mode with the formal parameters of the statement function.

*****PN** UNBALANCED PARENTHESIS

Indicates an unequal number of open and closed parentheses in a statement.

*****PT** SYNTAX ERROR IN AN ENTRY STATEMENT

The entry statement being processed is labeled or has more than one name or is in a DO loop or name started with a number.

*****RN** SYNTAX ERROR IN A RETURN STATEMENT

Indicates an error in the format of a RETURN statement.

SA SMALL IN/OUT ARGUMENT ERROR

Error in SMALL IN/OUT arguments.

*****SB** ERROR IN AN ARRAY SUBSCRIPT

Indicates a format error in a subscript of an array reference currently being processed.

*****SE** SYNTAX ERROR IN SENSE STATEMENT

Indicates an error in the format of a sense statement.

*****SF** FIELD LENGTH OF ROUTINE BEING COMPILED EXCEEDS THE SPECIFIED FIELD LENGTH

*****SM** SYNTAX ERROR IN STATEMENT NUMBER

Indicates an error in the format of the statement label field.

*****SN** ILLEGAL CHARACTER IN STATEMENT NUMBER USAGE

Indicates an error in the format of the position where the statement label should appear.

*****SY** SYSTEM ERROR IN FORTRAN COMPILER

*****TM** SUBROUTINE HAS MORE THAN 60 ARGUMENTS

Indicates that a subroutine reference has more than 60 arguments or that the routine being compiled has more than 60 parameters.

*****TN** PROGRAM HAS MORE THAN 24 ARGUMENTS

*****TT** VARIABLE GIVEN CONFLICTING TYPES

A variable has appeared in more than one type statement.

****TY** SYNTAX ERROR IN A TYPE STATEMENT

Indicates an error in the format of a TYPE statement.

*****UA** REFERENCE MADE TO AN AS YET UNDIMENSIONED ARRAY

Indicates reference was made to an array which has not previsouly appeared in a DIMENSION statement.

*****UE** LOGICAL UNIT NUMBER IS NOT AN INTEGER

*****VC** VARIABLE NAME CONFLICTS WITH A PRIOR USAGE

Indicates that a variable name appears which conflicts with some prior use.

*****VD** ARRAY WHOSE DIMENSIONS ARE ARGUMENTS TO THE SUBROUTINE OR FUNCTION HAS BEEN MISUSED

Indicates improper use of an array with variable dimensions.

*****XF** SYNTAX ERROR IN THE EXPRESSIONS BEING PROCESSED

Indicates an error in the format of the expression currently being processed.

*****ZY** SYSTEM ERROR-UNKNOWN TWO LETTER CODE

PROGRAM-SUBPROGRAM FORMAT G

The starting address of all programs is RA+100_8 with the first 77_8 locations containing file and loader information. Only 24 files may be declared for any one program and the file names along with their associated buffer addresses begin at RA+2. An object time routine, Q8NTRY, transfers the file information to RA+2+n, where n is the number of declared files, at execution time. The I/O buffers are reserved as a portion of the main program and Q8NTRY also initializes the buffer parameters during execution.

The first word of a main program contains the name of the program in left justified display code and a parameter count greater than 77_8 in the right most position. Since no more than 74_8 parameters may be passed to a subprogram, a count of greater than 77_8 terminates trace back information. The second word of a main program is the entry point. It contains instructions to preset the parameters for Q8NTRY which performs initiation only once per execution. Therefore, entry into an overlay is through this word destroying its contents. Since Q8NTRY does not perform any function after the first entry, the destruction of the preset parameters for an overlay entry does not matter.

The addresses of the first six parameters to a subprogram are passed by B registers 1-6. One word is reserved for each parameter. The address of parameters after the 6th are actually passed through this reserved word. Immediately following these reserved words is a location containing the name of the subprogram in left justified display code and a parameter count in the lower six bits. Next is the entry/exit line for the subprogram. Therefore, a subprogram will have as few as two reserved words if the parameter count is zero. Otherwise, there will be a reserved word for each parameter plus the name and the entry words.

Subroutines written in the COMPASS assembly language that will operate in conjunction with FORTRAN coded routines should be formatted as in the following examples to take full advantage of the error tracing facility of FORTRAN Version 2.0. The called subroutines do not have to be concerned with register preservation.

Examples:

PROGRAM PETE (INPUT, OUTPUT, TAPE 1)

DATA 0		L00002	Name and argument count plus 100_8
SB1	L00002	L00001	Entry/Exit line
SB2	C00001		
RJ	Q8NTRY	L00003	

```
SUBROUTINE PHD      (A,B,C)

    DATA    0             L00005
    DATA    0             L00004
    DATA    0             L00003
    DATA    0             L00002             Name & argument count
    DATA    0             L00001             Entry/Exit line

SUBROUTINE PEN      (A,B,C,D,E,F,G,H,I,J)

    DATA    0             L00003             Reserved for A
    DATA    0             L00004             Reserved for B
    DATA    0             L00005             Reserved for C
    DATA    0             L00006             Reserved for D
    DATA    0             L00007             Reserved for E
    DATA    0             L00010             Reserved for F
    DATA    0             L00011             Reserved for G
    DATA    0             L00012             Reserved for H
    DATA    0             L00013             Reserved for I
    DATA    0             L00014             Reserved for J
    DATA    0             L00002             Name & argument count
    DATA    0             L00001             Entry/Exit line
```

Calling Sequence to PEN

```
CALLPEN             (M,N,O,P,Q,R,S,T,U,V)

    SB1     M
    SB2     N
    SB3     O
    SB4     P
    SB5     Q
    SB6     R
    SX6     Entry line of PEN
    SA1     X6-1     Name & argument count
    SB7     X1-6     Number of arguments less 6
    SX6     S
    SA6     A1-B7    Reserved word for S
    SX7     T
    SA7     A6+1     Reserved word for T
    SX6     U
    SA6     A7+1     Reserved word for U
    SX7     V
    SA7     A6+1     Reserved word for V
    RJ      PEN
    0712L00002       Where 12 (octal) is argument count and L00002 is word containing the name of calling
                     routine.
```

Where 12_8 is argument count.

SYSTEM ROUTINE H

The SYSTEM routine handles the following extensions for the mathematical routines of FORTRAN Version 2.1: error tracing, diagnostic printing, termination of output buffers and transfer to specified non-standard error procedures. The END processor also uses SYSTEM to dump the output buffers and print an error summary. Since SYSTEM, along with the initialization routine, Q8NTRY, and the end processors, END, STOP, EXIT, must always be available, these routines are combined into one with multiple entry points. Any of the parameters used by SYSTEM relating to a specific error may be changed by a user routine during execution by calling SYSTEMC.

CALLING SYSTEM

The calling sequence to SYSTEM from an assembly language routine passes the error number X1 and an error message address in X2. Therefore, one error number may have several different messages associated with it. The error summary at the end of the program lists the total number of times each error number was encountered.

FORTRAN routines call SYSTEM via a RJ to SYSTEMP, a special entry point. Because the addresses of the subprogram arguments must be passed to a non-standard recovery routine if one is specified, SYSTEMP must be called with eight parameters. The first six parameters are the first six formal parameters of the subprogram. If the subprogram does not have six parameters, dummy parameters must be supplied. The seventh parameter to SYSTEMP is the error number specified as an integer constant or integer variable. The array or simple variable containing the diagnostic message is the eighth parameter. After adjusting the parameters, SYSTEMP transfers to SYSTEM for error processing.

ERROR PROCESSING

If an error number of zero is accepted, this is a special call to end the output buffers and return. If no OUTPUT file is defined before SYSTEM is called, there is no error printing and an appropriate message appears in the Dayfile. Each line printed is subjected to the line limit of the OUTPUT buffer. When limit is exceeded, the job is terminated. The error table is ordered serially; the first error corresponds to the error number 1, and is expandable at assembly time. The last entry in the table is a catch-all for any error number that exceeds the table length. Following is an entry in the error table.

Error Table

PRINT FREQUENCY	PRINT FREQUENCY INCREMENT	PRINT LIMIT	ERROR DETECTION TOTAL	F/NF	A/NA	NON-STANDARD RECOVERY ADDRESS
8	8	12	12	1	1	18

Use of PRINT FREQUENCY

PRINT FREQUENCY = PF

PRINT FREQUENCY INCREMENT = PFI

1. If PF = 0 and PFI = 0, diagnostic and trace back information are never listed.
2. If PF = 0 and PFI = 1, diagnostic and trace back information are always listed until the print limit is reached.
3. If PF = 0 and PFI = n, diagnostic and trace back information are listed only the first n times unless the print limit is reached first.
4. If PF = n, diagnostic and trace back information are listed every n^{th} time until the print limit is reached.

Use of FATAL (F)/ non-FATAL (NF)

1. If the error is non-fatal and no non-standard recovery address is specified, the error messages are printed according to PRINT FREQUENCY and control is returned to the calling routine.
2. If the error is fatal and no non-standard recovery address is specified, the error messages are printed according to PRINT FREQUENCY, an error summary is listed, all the output buffers are terminated, and the job is terminated.
3. Non-standard recovery address is explained below:

TRACEBACK EXAMPLE

```
     DATA   0   L00002        Name and number of parameters
     DATA   0   L00001        Entry/exit line
       .
       .
       .
+    RJ         SYSTEM
-    07         L00002        07 is number of parameters passed to SYSTEM and L00002 is
                              address of word containing name of calling routine
```

The name of the routine always precedes the entry/exit line.

Use of NON-STANDARD RECOVERY

SYSTEM will supply the non-standard recovery routine with the following information:

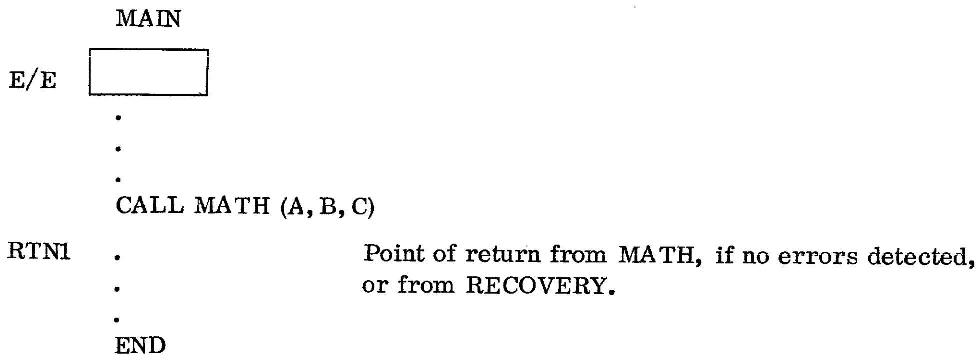
B1-B6	address of the first six parameters passed to the routine that detected the error
X1	error number passed to SYSTEM
X2	address of the diagnostic message available to SYSTEM
X3	address within an auxiliary table if A/NA bit is set
X4	instruction word consisting of the return jump to SYSTEM in the upper 30 bits and trace back information in the lower 30 bits for the routine which detected the error
A0	address of error number entry within SYSTEM's error table.

1. Non-fatal error

   The entry/exit line of the routine which called SYSTEM is set into the entry/exit line of the recovery routine. Control is then passed to the word immediately following the entry/exit line of the recovery routine. The traceback information available to SYSTEM from the routine which detected the error is passed to the recovery routine in X4.

   Any faulty parameters may be corrected, and the recovery routine is allowed to call the routine which detected the error with corrected parameters. Upon exit from the recovery routine, control is turned not to SYSTEM nor to the routine which detected the error, but rather back another level (see example). By not correcting the faulty parameters in the recovery routine, a three routine loop could develop between the routine which detects the error, SYSTEM, and the recovery routine. No checking is done for this case.

   Example:



```
        MATH
E/E     [jump to RTN1]          May be reentered from RECOVERY with corrected
                                parameters
        .
        .
        .
        RJ SYSTEM
        07XXAAAAAA.........     traceback information

RTN2    .
        .
        .
        END

        SYSTEM
        [jump to RTN2]          transfers E/E line of MATH to E/E
        .                       line of RECOVERY and gives control
        .                       to RECOVERY
        JUMP TO RECOVERY
        .
        .
        .
        END

        RECOVERY
E/E     [jump to RTN1]          corrects faulty parameters and may
                                recall MATH
        .
        .
        .
        RJ MATH
        .
        .
        .
        jump to E/E             returns to MAIN following reference
        END                     to MATH
```

2. Fatal error:

Into the entry/exit line of the recovery routine is set a return address back to SYSTEM. Control is then passed to the word immediately following the entry/exit line of the recovery routine. The traceback information available to SYSTEM from the routine which detects the error is set in X4. If control is returned to SYSTEM from the recovery routine, then an error summary is listed, all output buffers are terminated and the job is aborted.

```
        SYSTEM
E/E     [          ]
        .
        .
        .
TAG     jump to RECOVERY
        07XXAAAAAA              traceback information
RTN3    .
        .
        .
        END
```

```
        RECOVERY

E/E     jump to RTN3
        .
        .
        .
        jump to E/E
        END
```

Use of the A/NA Bit

The A/NA bit is for use only when a non-standard recovery address is specified. If this bit is set, the address within an auxiliary table is passed in X3 to the recovery routine. This bit allows more information than is normally supplied by SYSTEM to be passed to the recovery routine. Only during assembly of SYSTEM may this bit be set, because an entry must also be made into the auxiliary table. Each word in the auxiliary table must have the error number in its upper 10 bits so that the address of the first error number match is passed to the recovery routine. An entry in the auxiliary table for an error is not limited to any specific number of words.

The traceback information is terminated as soon as one of the following variables is detected:

Calling routine is a program (the number of arguments 77B)

Maximum trace back limit is reached.

No trace back information is supplied; a 07 instruction does not follow the return jump as is the case with I/O operations.

To change an error table during execution, a FORTRAN type call is made to SYSTEMC with the addresses of the following parameters in B1 and B2:

B1 error number

B2 parameter list in consecutive locations containing:

word 1 fatal/non-fatal (fatal = 1, non-fatal = 0)
word 2 print frequency
word 3 print frequency increment (only significant if word 2 = 0)

special values:
word 2 = 0, word 3 = 0 never list error
word 2 = 0, word 3 = 1 always list error
word 2 = 0, word 3 = X list error only the first X times encountered

word 4 print limit
word 5 non-standard recovery address
word 6 maximum trace back limit

If any word within the parameter list is negative, the value already in the table entry will not be altered.

(Since the auxiliary table bit may be set only during assembly of SYSTEM, only then can an auxiliary table entry be made.)

ERROR LISTING

< message supplied by calling routine >

ERROR NUMBER xxxx DETECTED BY zzzzzzz at yyyyyy
CALLED FROM cccccc at wwwwww

•

•

•

zzzzzzz and ccccccc are routine
names, yyyyyy and wwwwww are
absolute addresses and error
number is decimal

ERROR SUMMARY

ERROR TIMES

xxxx yyyy
xxxx yyyy
xxxx yyyy

all numbers are decimal

NO OUTPUT FILE FOUND

OUTPUT FILE LINE LIMIT EXCEEDED

Functions of entry points:

Q8NTRY	initialize I/O buffer parameters
STOP	enter STOP in the Dayfile and begin END processing
EXIT	enter EXIT in the Dayfile and begin END processing
END	terminate all output buffers, print an error summary; transfer control to main overlay if within an overlay or in any other case exit to monitor
SYSTEM	handles error tracing, diagnostic printing, termination of output buffers, and either transfers to specified non-standard error recovery address, aborts the job, or returns to calling routine depending on type of error
SYSTEMP	adjusts arguments for use by SYSTEM and transfers control to SYSTEM
SYSTEMC	changes entry in SYSTEM's error table according to arguments passed.
ABNORML	gains control from an execution routine when an error had been assembled as fatal and during the processing of the job was changed to non-fatal with no non-standard error recovery. An abnormal termination is given.

FILE NAME HANDLING BY SYSTEM

SYSTEM(Q8NTRY) places in RA+2 and the locations immediately following, the file names from the FORTRAN PROGRAM card. The file name is left justified and the file's FET address is right justified in the word. (Thus the declared names replace any actual file names at execution time in the RA area.)

The logical file name (LFN) which appears in the first word of the FET is determined in one of the three following ways:

CASE 1: If no actual parameters are specified, the LFN will be the file name from the PROGRAM card.

Example:

```
.
.
.
RUN(S)
LGO.
.
.
.
PROGRAM TEST1(INPUT,OUTPUT,TAPE1,TAPE2)
```

Before RA+2

```
SYSTEM(Q8NTRY)
000___________000
000___________000
```

After RA+2

		LFN in FET
INPUT	FET address	INPUT
OUTPUT	FET address	OUTPUT
TAPE1	FET address	TAPE1
TAPE2	FET address	TAPE2

CASE 2: If actual parameters are specified, the LFN will be that specified by the corresponding actual parameter, or the file name from the PROGRAM card if no actual parameter was specified. A one-to-one correspondence exists between the actual parameters and the file names found on the PROGRAM card.

Example:

```
.
.
.
RUN(S)
LGO(,,DATA,ANSW)
.
.
.
PROGRAM TEST2(INPUT,OUTPUT,TAPE1,TAPE2,TAPE3=TAPE1)
```

Before RA+2

```
000___________000
000___________000
DATA__________000
ANSW__________000
```

After RA+2			LFN in FET
	INPUT	FET address	INPUT
	OUTPUT	FET address	OUTPUT
	TAPE1	FET address	DATA
	TAPE2	FET address	ANSW
	TAPE3	FET address of TAPE1	Uses TAPE1 FET

CASE 3: An equivalenced file name from the PROGRAM card will ignore an actual parameter. The LFN will be that of the file to the right of the equivalence and no new FET will be created.

Example:

```
.
.
.
RUN(S)
LGO(,,DATA,ANSW)
.
.
.
PROGRAM TEST3(INPUT,OUTPUT,TAPE1=OUTPUT,TAPE2,TAPE3)
```

Before RA+2		
	000	000
	000	000
	DATA	000
	ANSW	000

After RA+2			LFN in FET
	INPUT	FET address	INPUT
	OUTPUT	FET address	OUTPUT
	TAPE1	FET address of OUTPUT	uses OUTPUT FET
	TAPE2	FET address	ANSW
	TAPE3	FET address	TAPE3

EXECUTION DIAGNOSTICS I

For the format of the error listing see page H-6.

The symbol INF denotes infinite and IND denotes indefinite internal words.

Some error conditions are preceded by "also". The routine in question calls on a subordinate library routine, giving it the arguments indicated; therefore the subordinate routine may detect some errors of its own and report them under its own error number.

Routine	Condition	Standard Recovery	Error Number
AGGOER	Called only upon detection of a computed or assigned GO TO error	Fatal	1
ACOS(R)	R = INF or R = IND or abs (R) .GT. 1.0	+IND +IND	2
ALOG(R)	R = INF or R = IND or R .LT. 0 R = 0	+IND - INF	3
ALOG10(R)	R = INF or R = IND or R .LT. 0 R = 0	+ IND - INF	4
ASIN(R)	R = INF or R = IND or abs (R) .GT. 1.0	+IND	5
ATAN(R)	R = INF or R = IND	+IND	6
ATAN2(R1, R2)	(R1 or R2) =(INF or IND) R1 = R2 = 0	+IND +IND	7
CABS(Z)	(real (Z) or imag (Z))= (INF or IND)	+IND	8
CBAIEX:Z**I	(real (Z) or imag (Z)) = (INF or IND) Z = (0,0) and I .LE. 0	(+IND,+IND) (+IND,+IND)	9
CCOS(Z)	(real (Z) or imag (Z)) =(INF or IND) also: COS (real (Z)) and EXP (imag(Z)) and imag (Z) .LT. -675.82	(+IND,+IND)	10
CEXP (Z)	(real (Z) or imag (Z)) = (INF or IND) Abs(real (Z)).GT.741.67 abs(imag(Z)).GT.2.2E14	(+IND,+IND)	11
CLOG (Z)	(real (Z) or imag (Z)) = (INF or IND) Z = (0,0)	(+IND,+IND)	12
COS(R)	R = INF or R = IND or abs (R) .GT. 2.2E14	+IND	13

Routine	Condition	Standard Recovery	Error Number
CSIN (Z)	(real (Z) or imag (Z)) = (INF or IND) also: SIN(real(Z)) and EXP (imag (Z)) and imag (Z) .LT.-675.82	(+IND,+IND)	14
CSQRT (Z)	(real (Z) or imag(Z)) = (INF or IND)	(+IND,+IND)	15
DABS (D)	D = INF D = IND	+INF +IND	16
DATAN (D)	D = INF or D = IND	+IND	17
DATAN2 (D1, D2)	(D1 or D2) = (INF or IND) D1 = D2 = 0	+IND +IND	18
DBADEX: D1**D2	(D1 or D2) = (INF or IND) D1 = 0 and D2 .LE. 0 D1 .LT. 0	+IND +IND +IND	19
DBAIEX: D1**I2	D1 = INF or D1 = IND D1 = 0 and I2 .LE. 0	+IND +IND	20
DBAREX: D1**R2	(D1 or R2) = (INF or IND) D1 = 0 and R2 .LE. 0 D1 .LT. 0	+IND +IND +IND	21
DCOS (D)	D = INF or D = IND or abs (D) .GT. 2.2E14	+IND	22
DEXP (D)	D = INF or D = IND D .GT. 741.67	+IND +INF	23
DLOG (D)	D = INF or D = IND or D .LT. 0 D = 0	+IND -INF	24
DLOG10 (D)	D = INF or D = IND or D .LT. 0 D = 0	+IND -INF	25
DMOD (D1,D2)	(D1 or D2) = (INF or IND) D2 = 0 D1 / D2 .GE. 2 ** 96	+IND +IND +IND	26
DSIGN (D1,D2)	D1 = IND or D2 = (0 or INF or IND) D1 = INF	+IND INF with sign of D2	27
DSIN (D)	D = INF or D = IND or abs (D) .GT. 2.2E14	+IND	28
DSQRT (D)	D = INF or D = IND or D .LT. 0	+IND	29
EXP (R)	R = INF or R = IND R .GT. 741.67 R. LT. -675.82	+IND +INF	30
IBAIEX: I1**I2	I1 = 0 and I2 .LE. 0 I1 ** I2 .GE. 2** 48	0 0	31
IDINT (D)	D = +INF or D = IND or D .GE. 2** 59 D = -INF or D .LE. -2**59	2**59-1 1-2**59	32

Routine	Condition	Standard Recovery	Error Number
RBADEX: R1**D2	(R1 or R2) = (INF or IND) R1 = 0 and D2 .LE. 0 R1 .LT. 0	+IND +IND +IND	33
RBAIEX: R1**I2	R1 = INF or R1 = IND R1 = 0 and I2 .LE. 0 R1**I2 = INF	+IND +IND +INF	34
RBAREX: R1**R2	(R1 or R2) = (INF or IND) R1 = 0 and R2 .LE. 0 R1 .LT. 0	+IND +IND +IND	35
SIN (R)	R = INF or R = IND or abs (R) .GT. 2.2E14	+IND	36
SLITE (I)	I .GT. 6 or I .LT. 0	Proceed	37
SLITET (I1, I2)	I1 .GT. 6 or I1 .LE. 0	Proceed	38
SQRT (R)	R = INF or R = IND or R .LT. 0	+IND	39
SSWTCH (I1, I2)	I1 .GT. 6 or I1 .LE. 0	Proceed	40
TAN (R)	R = INF or R = IND or abs (R) .GT. 8.4E14	+IND	41
TANH (R)	R = INF or R = IND	+IND	42
INPUTN	Too few constants Loss of precision Attempt to read after write	Fatal	49
BUFFEI	Attempt to read past EOF on Buffer In. Last operation was a write, no data available to read Starting address greater than terminal address	Fatal Fatal Fatal	55 56 57
BUFFEO	Starting address greater than terminal address	Fatal	59
INPUTB	Attempt to read past EOF - binary input	Fatal	63

Routine	Condition[††]	Standard Recovery	Error Number
INPUTC	Attempt to read past EOF - coded input	Fatal	65
INPUTS	Attempt to transfer more than 150 characters/ record on DECODE processing	Fatal	66
INPUTN	Namelist name not found Wrong type or too many constants	Fatal	67
KODER (Coded output)	Illegal letter as format specification	Fatal	68
	Format specification has more than 2 levels of parentheses (3 levels under USASI)	Fatal	69
	Exceeded record size (format specified more than 136 characters per line).	Fatal	70
	Field width specified as zero.	Fatal	71
	Field width specified is less than or equal to the specified decimal width.	Fatal	72
	Attempt to output data under Hollerith format.	Fatal	73
KRAKER (Coded input)	Illegal letter used as format specification.	Fatal	74
	Format specification with more than 2 levels of parentheses.	Fatal	75
	Field width specified as zero.	Fatal	76
	Coded read past end of record.	Fatal	77
	Attempt to input data under Hollerith format.	Fatal	78
	Illegal data in the external field. [†††]	Fatal	79
	Data converted is out of range. [†††]	Fatal	80
ALL I/O ROUTINES	Unassigned medium[†]	Fatal	82
OUTPTC OUTPTN	Line limit as specified on RUN card exceeded	Fatal	84
OUTPTS	Attempt to transfer more than 150 characters/ record on ENCODE processing.	Fatal	86
KODER (Coded output)	Attempt to output a single array under "D" format specification.	Fatal	87

[†] Execution time diagnostic occurs when a variable file name is undefined. It is printed as Unassigned medium, file xxxxxxx (where xxxxxxx is the name of the undefined file).

[††] All input/output errors at execution time are fatal. Standard error recovery for all the above cases is to terminate the job after standard error tracing.

[†††] Card image will be printed.

Routines	Condition	Standard Recovery	Error Number
INPUTC	Last operation was a write, no data available to read.	Fatal	88
INPUTB	List exceeds data on file, attempt to read more data than exists in the logical record.	Fatal	89
INPUTB	Last operation was a write, no data available to read	Fatal	90

FORTRAN CROSS-REFERENCE MAP J

If the ninth field of the run control card is non-zero, FORTRAN supplies the programmer with a cross-reference map after each PROGRAM, SUBROUTINE, or FUNCTION, purely as an aid to program debugging. The following information is furnished:

Program length including I/O Buffers

Statement function references with the relative core locations, general compiler tag assigned, symbolic tag given in the program and the references to the statement function

Statement number references with the same information as above

Block names and lengths

Variable references - also with location, general tag, symbolic tag, and a list of references

Start of constants (relative address)

Start of temporaries (relative address)

Start of indirects (relative address)

Unused compiler space

The programmer should bear in mind that because of the operation of the compiler not all references will be listed. An actual physical reference is necessary before the reference is placed in the reference map. If the required variable address is already in a register, the compiler will use the address in the register and not make an actual variable reference by name. A reference to a statement number will not be listed if an actual jump is not necessary, such as when the code simply falls through to the next statement and the compilation of a jump instruction is therefore unnecessary.

ADDITIONAL STATEMENTS K

The LARGE statement and the SMALL statement as described in this appendix will be implemented in this version of 7600 FORTRAN. This implementation is subject to change in subsequent versions.

LARGE Statement

The LARGE statement allocates space in large core memory. The statement format is:

LARGE $t_1 . v_1 (i_1), t_2 . v_2 (i_2), \ldots, t_n . v_n (i_n)$

where t specifies the type of the following variable or array name. It may be any of the characters C, D, I or R specifying complex, double, integer, logical, or real respectively. If t is not present, the type of the variable is determined by the first character of its name. v is an array name. i is the dimension(s) of the array; it may be fixed or variable.

A LARGE statement may appear in a PROGRAM, or callable SUBROUTINE--but not in a FUNCTION subprogram. Arrays named in LARGE statements are, unless they correspond to arguments of a callable SUBROUTINE, assigned to large core memory starting at relative address 0000000. The assignments of array locations are made in the order in which the array names appear in LARGE statements. An array name appearing in a LARGE statement may not appear in any other nonexecutable statement and must appear in the LARGE statement before any reference is made to the array. The length of an array of names in a LARGE statement may be through 131071 (decimal) locations. 393215 (decimal) large core memory locations are available for an object program or subprogram.

The LARGE statement provides blocks of storage in the same manner as the COMMON declaration (section 5.3). The length of a LARGE block in computer words is determined by the number and type of the list variables. If a subprogram does not use all the locations reserved, the LARGE statement must contain dummy variables to insure proper correspondence.

If a large core variable appears as an argument in a reference to a FUNCTION subprogram or to a library function subroutine, instructions are compiled for transferring the large core variable to a small core memory location; the small core memory address is transmitted as the argument to the FUNCTION subprogram or library function subroutine, which implies that the original large core argument may not specify a result location to the FUNCTION subprogram or library function subroutine.

If a large core variable appears as an argument in a reference to a called SUBROUTINE, the corresponding argument within the SUBROUTINE must appear in a LARGE statement, i.e., the arguments must agree in memory type, and in this case, the corresponding array must reside in the large core relative region $0000000\text{-}0777776_8$ (inclusive). If an array in large core memory is not named by an argument in a callable SUBROUTINE, it may reside anywhere in the relative region $0000000\text{-}1377776_8$ (inclusive) of that memory. When a large core variable appears as a formal argument in a SUBROUTINE, a LARGE statement containing that argument does not cause storage to be allocated for that array.

Large core memory arrays cannot be used in an I/O list.

SMALL Statement

The SMALL statement transfers a block of words between large core and small core memories.

A SMALL statement may appear in a PROGRAM, or callable SUBROUTINE--but not in a FUNCTION subprogram. Such a statement has one of the forms

SMALLIN (s, ℓ, w)
SMALLOUT (s, ℓ, w)

where s is a small-core memory simple variable name, array name, or array element; ℓ is a large-core memory array name, or array element; w is a simple integer variable name or a constant. The SMALLIN statement causes the number of words specified by w to be block-transferred from LCM starting at location ℓ to small-core memory starting at location s. The SMALLOUT statement causes the number of words specified by w to be block-transferred from small-core memory starting locations s to large-core memory starting location ℓ. The word count for double precision and complex arrays must specify twice as many words as elements to be transferred.

Example:

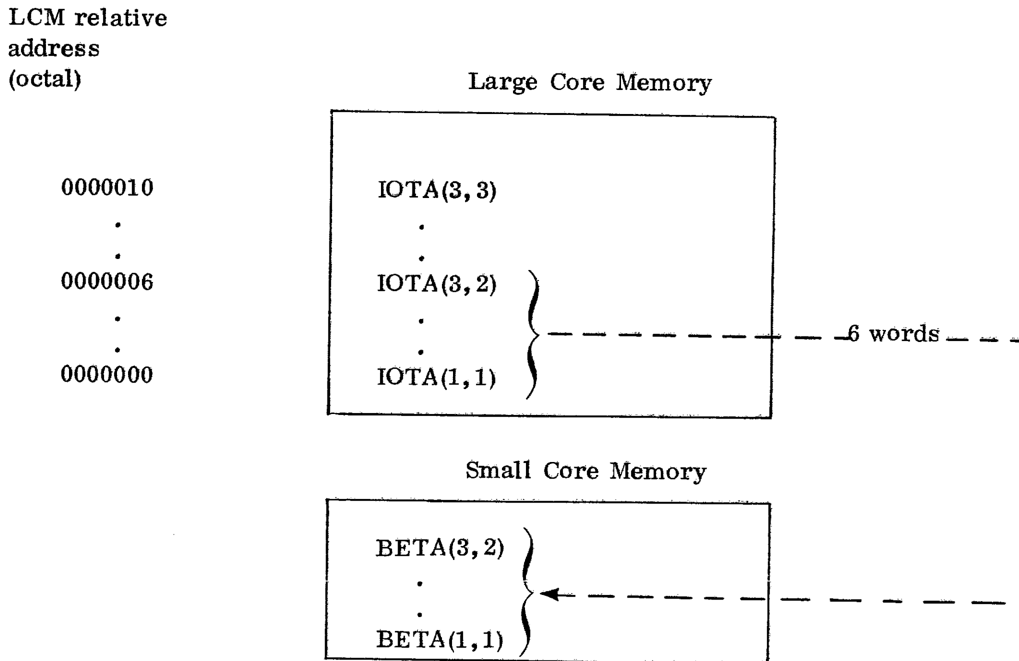
Given the following statements:

LARGE R. IOTA(3, 3)

REAL BETA(3, 2)

SMALLIN(BETA(1, 1), IOTA(1, 1), 6)

the elements will be stored and transferred according to the following illustration:



INDEX

Actual arguments 7-5
Arguments, actual 7-5
Arguments, formal 7-3
Arithmetic assignment 4-1
Arithmetic evaluation 3-2
Arithmetic expressions 3-1
Arithmetic mixed-mode expressions 3-4
Arrays 2-9
    Transmission of 9-2
ASSIGN statement 6-2
Assigned GO TO statement 6-1
Assignment statements 4-1
    Arithmetic 4-1
    Logical 4-4
    Masking 4-4
    Mixed-mode 4-1
    Multiple 4-4
Asterisks 9-20
Aw input specification 9-14
Aw output specification 9-13

BCD record 10-1
BACKSPACE i 10-8
Basic External functions 7-7
Binary record 10-1
BLOCK DATA subprogram 5-12
BUFFER IN (u,p) (A,B) 10-11
BUFFER OUT (u,p) (A,B) 10-11
BUFFER statements 10-9

CALL OVERLAY 8-2
CALL statement 7-14
Calling system H-1
Cards, loader 8-3
Character codes A-1
Coding 1-1
    Character code, FORTRAN A-1
    Comments 1-2
    Continuation cards 1-2
    Identification fields 1-2
    Line 1-1
    Statement label 1-2
    Statements 1-1

Comments, coding 1-2
COMMON declaration 5-3
Compilation and execution E-1
Compiler mode options E-1
Complex constants 2-3
Complex variables 2-8
Composition, overlay 8-2
Computed GO TO statement 6-2
Computer word structure of constants-7600 D-1
Constants 2-1, D-1
    Complex 2-3
    Computer word structure D-1
    Double precision 2-3
    Hollerith 2-5
    Integer 2-2
    Logical 2-4
    Octal 2-6
    Real 2-2
    Word Structure D-1
Continuation cards, coding 1-2
CONTINUE statement 6-9
Control statements 6-1
    ASSIGN 6-2
    CONTINUE 6-10
    DO 6-5
    END 6-11
    GO TO 6-1
    IF 6-3
    PAUSE 6-10
    RETURN 6-11
    STOP 6-11
Conversion specifications 9-5
    Aw input 9-14
    Aw output 9-13
    Dw.d input 9-10
    Dw.d output 9-10
    Ew.d input 9-6
    Ew.d output 9-5
    Fw.d input 9-9
    Fw.d output 9-8

Conversion specifications (cont'd)
  Gw.d input 9-10
  Gw.d output 9-9
  Iw input 9-12
  Iw output 9-11
  Lw input 9-15
  Lw output 9-15
  Ow input 9-13
  Ow output 9-12
  Rw input 9-14
  Rw output 9-14
Cross-reference map J-1

DATA declaration 5-8
Data types 2-2
DECK structures E-1
Declarations 5-1
  COMMON 5-3
  DATA 5-8
  DIMENSION 5-2
  EQUIVALENCE 5-6
  Type 5-1
DECODE (c,n,v)L 10-11
Diagnostics F-1
Diagnostics, execution I-1
DIMENSION declaration 5-2
Dimensions, variable 5-3,7-18
DO loop execution 6-6
DO loop transfer 6-8
DO nests 6-6
DO statement 6-5
Double precision constants 2-3
Double precision variables 2-8
DUMP 7-14
DVCHK (j) 7-14
Dw.d input specification 9-10
Dw.d output specification 9-10
Dw.d scaling 9-17

Editing specifications 9-17
  ... 9-20
  New record 9-19
  wH input 9-19
  wH output 9-18
  wX 9-17
Eject, page 10-2

Elements of FORTRAN 2-1
ENCODE (c,n,v)L 10-11
ENCODE/DECODE statement 10-11
END FILE i 10-8
END statement 6-10

ENTRY statement 7-17
EQUIVALENCE declaration 5-6
Error processing H-1
Evaluation, arithmetic 3-2
Ew.d input specification 9-6
Ew.d output specification 9-5
Ew.d scaling 9-17
Execution, compilation and E-1
Execution diagnostics I-1
EXIT 7-14
Expressions 3-1
  Arithmetic 3-1
  Logical 3-8
  Masking 3-10
  Relational 3-7
EXTERNAL statement 7-15

File handling statements 10-8
Formal arguments 7-3
FORMAT declaration 9-4
Format, overlay 8-3
Format, program-subprogram G-1
FORTRAN character set 2-1, A-1
FORTRAN control card E-1
FORTRAN cross-reference map J-1
FORTRAN functions C-1
FORTRAN library routine entry points J-1
FORTRAN statement list B-1
Function, library Appendix C
Function, statement 7-6
FUNCTION subprogram 7-8
Functions, FORTRAN C-1
Fw.d input specification 9-9
Fw.d output specification 9-8
Fw.d scaling 9-16

GO TO statements 6-1
  Assigned 6-1
  Computed 6-2
  Unconditional 6-1
Gw.d input specification 9-10

Gw.d output specification 9-9
Gw.d scaling 9-17

Hollerith constants 2-5

Identification field, coding 1-2
Identification, overlays 8-1
IF (ENDFILE, i) n_1, n_2 10-8
IF (EOF, i) n_1, n_2 10-8
IF (UNIT, i) n_1, n_2, n_3, n_4 10-8
IF statements 6-3
  One branch logical IF 6-4
  Three branch arithmetic IF 6-3
  Two branch logical IF 6-4
Integer constants 2-2
Integer variables 2-7
Intrinsic function 7-7
I/O formats 9-1
I/O list 9-1
I/O statements 10-1
  BUFFER statements 10-9
  NAMELIST statements 10-5
  Output statements 10-1
    PRINT 10-1
    PUNCH 10-2
    WRITE 10-2
    READ statements 10-5
  File handling statements 10-8
Iw input specification 9-12
Iw output specification 9-11

Label, statement 1-2
LARGE statement K-1
Levels, overlay 8-1
Library functions Appendix C
Library routine entry points, FORTRAN J-1
Library subroutines 7-12
Lines, coding 1-1
LIST E-2
Loader cards 8-3
Logical assignment 4-4
Logical constants 2-4
Logical expressions 3-8
Logical variables 2-8
Lw input specification 9-15
Lw output specification 9-15

Main program 7-1
Masking assignment 4-4
Masking expressions 3-10
Mixed-mode arithmetic expressions 3-4
Mixed-mode assignment 4-1
Multiple assignment 4-4

NAMELIST statement 10-5
New record specification 9-19
NOLIST E-2

Octal constants 2-6
One-branch logical IF 6-4
Output statements 10-1
OVERFL (i) 7-14
Overlays 8-1, E-2
  CALL OVERLAY 8-2
  Cards 8-4
  Composition 8-2
  Format 8-3
  Identification 8-1
  Levels 8-1
  Loader cards 8-3
Ow input specification 9-13
Ow output specification 9-12

Page eject 10-2
PAUSE statement 6-9
PDUMP 7-14
PRINT n, L 10-1
Procedure identifiers 7-3
Procedures 7-2
Program arrangement 7-20
PROGRAM card 7-1
Program communication 7-2
Program, main 7-1
Program-subprogram format G-1
PUNCH n, L 10-2
Punched cards 1-2

READ statements 10-4
  READ (i) L 10-4
  READ (i, n) L 10-4
  READ n, L 10-4
Real constants 2-2
Real variables 2-7
Relational expressions 3-7

Repeated format specifications 9-22
Unlimited groups 9-22
Unlimited groups for USASI 9-24
RETURN statement 6-10
REWIND i 10-8
RUN E-1
Rw input specification 9-14
Rw output specification 9-14

Scaling
Dw.d scaling 9-17
Ew.d scaling 9-17
Fw.d scaling 9-16
Gw.d scaling 9-17
nP scale factor 9-15
Scale factor for USASI 9-24
SECOND (t) 7-14
SLITE(i) statement 7-13
SLITET (i,j) statement 7-13
Source program 7-1
SMALL statement K-2
Specifications 9-17
Conversion 9-5
Editing 9-17
Repeated format 9-22
Variable format 9-23
SSWTCH (i,j) statement 7-14
Statement, ASSIGN 6-2
Statement, CALL 7-14
Statement, DO 6-5
Statement, ENTRY 7-17
Statement, EXTERNAL 7-15
Statement function 7-6
Statement, GO TO 6-1
Statement label 1-2
Statement list, FORTRAN B-1
Statement, NAMELIST 10-5
Statement, READ 10-4
Statements, buffer 10-9
Statements, coding 1-1
Statements, control 6-1
Statements, ENCODE/DECODE 10-11
Statements, IF 6-3
Statements, I/O 10-1
Statements, output 10-1
Statements, tape handling 10-8
STOP statement 6-10
Structure, array 2-9
Subprogram, block data 5-12
Subprogram communication 7-2
Subprogram format, program
Subprogram, function 7-8
Subprogram, subroutine 7-11
Subprograms 7-2, 7-7
Subprograms, variable dimensions 7-18
Subroutine 7-12
Subroutine subprogram 7-11
Subroutines, library 7-12
Subscripted variables 2-8
Supplied function 7-6
Symbolic names 2-1
System routine H-1

Tape handling statements 10-8
Three-branch arithmetic IF statement 6-3
Two-branch logical IF statement 6-4
Type declarations 5-1

Unconditional GO TO statement 6-1
Unlimited groups specification 9-22
Unlimited groups for USASI 9-24
USASI compatibility 9-24
scale factor 9-24
unlimited groups 9-24

Variable dimensions 5-3
Variable dimensions in subprograms 7-18
Variable format 9-23
Variables 2-6
Complex 2-8
Double precision 2-8
Integer 2-7
Logical 2-8
Real 2-7
Subscripted 2-8

wH input specification 9-19
wH output specification 9-18
Word structure of constants D-1
WRITE i, L 10-3
WRITE (i) L 10-3
WRITE (i,n) L 10-2
wX specification 9-17

COMMENT SHEET

MANUAL TITLE 7600 FORTRAN Reference Manual

PUBLICATION NO. 60280400 REVISION A

FROM: NAME: ______

BUSINESS ADDRESS: ______

COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

AA3419 REV. 11/69 PRINTED IN U.S.A

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

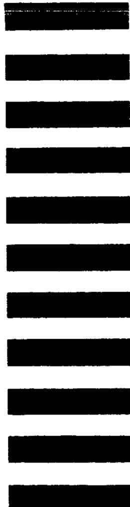
STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.



CUT ALONG LINE

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Technical Publications
4201 No. Lexington Ave.
St. Paul, Minnesota 55112

ARH220

FOLD

FOLD

Control Data 7600 FORTRAN Reference Manual Pub. No. 60280400

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORL