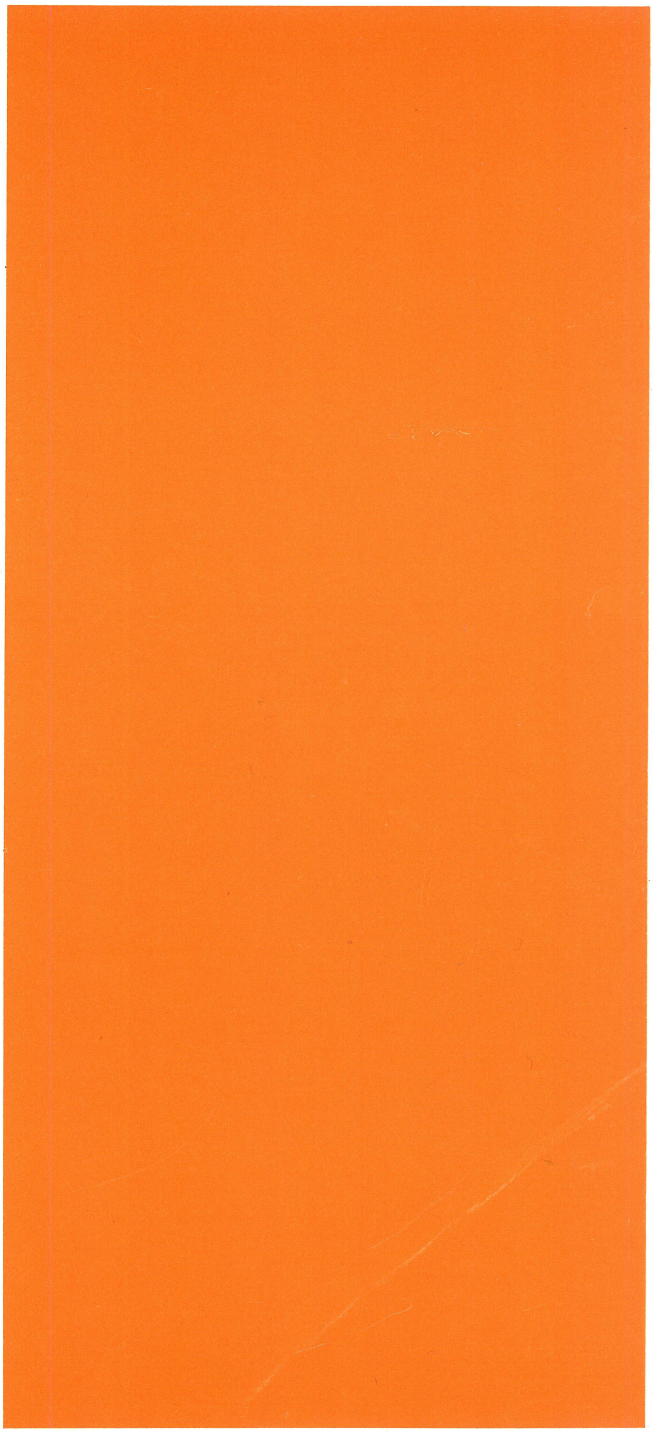
Honeywell Bull

FORTRAN

SERIES 600/6000

SOFTWARE



BJ67, Rev. 1

Ref. : 19.30.222 A

Honeywell Bull FORTRAN

SERIES 600/6000

SUBJECT:

General Description, Capabilities, Rules and Definitions, User Interfaces, Statements, Input/Output, and Subroutines of the Series 600/6000 FORTRAN.

SPECIAL INSTRUCTIONS:

This manual, Order Number BJ67, Revision 1, supersedes BJ67, Revision 0, dated June 1971, and Addendum 1, dated September 1971, Addendum B, dated December 1971, and Addendum C, dated July 1972.

Change bars in the margins indicate technical additions and changes; asterisks indicate deleted material.

SOFTWARE SUPPORTED:

Series 600 Software Release 7.0
Series 6000 Software Release E

DATE:

March 1973

ORDER NUMBER:

BJ67, Rev. 1

Printed in France

Ref: 19.30.222 A

PREFACE

This manual describes the Honeywell Series 6000 FORTRAN compiler, which is intended as a replacement for Series 600 FORTRAN IV (Batch) and Series 600 Time Sharing FORTRAN. This compiler is a compatible extension of the current Series 600 compilers and provides an extended software capability for the Series 6000 computers. This reference manual assumes that the reader is familiar with FORTRAN programming principles and basic concepts. All of the necessary Series 6000 FORTRAN rules and statements are included in this manual.

Fortran is a coded system designed to extend the power of Series 600/6000 in the area of program preparation and maintenance. It is supported by comprehensive documentation and training; periodic maintenance and, where feasible, improvements are furnished for the current version of the system, provided it is not modified by the user.

 File No.: 1623,1723

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 600 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
Series 600:			
Hardware reference:			
Series 600	System Manual	371	BM78
DATANET 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards Reference Manual	1688	BS19
System initialization:			
GCOS Startup	System Operating Techniques	DA10	DA10
Communications System	GRTS/355 Startup Procedures Reference Manual	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File System	File Management Supervisor	DB54	DB54
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processor	DA37	DA37
Multi-Access I-D-S	Multi-Access I-D-S Implementation Guide	DA80	DA80
File Input/Output	File and Record Control	1003	BN85
I-D-S Data Query System	I-D-S Data Query System Installation	DB57	DB57
I-D-S Data Query System	I-D-S Data Query System User's Guide	DB56	DB56
Program maintenance:			
Object Program	Source and Object Library Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
On-Line Peripheral testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Total On-Line testing	Total On-Line Test System (TOLTS)	DA49	DA49
Language processors:			
Macro Assembly Language	Macro Assembler Program	1004	BN86
COBOL Language	COBOL Compiler	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
FORTRAN IV Language	FORTRAN IV	1006	BN88
DATANET 355	DATANET 355 Macro-Assembler Program	1660	BB98
Generators:			
Sorting	Sort/Merge Program	1005	BN87
Merging	Sort/Merge Program	1005	BN87
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator Reference Manual	1663	BW23

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 600:		
Remote terminal system:			
DATANET 30	NPS/30 Programming Reference Manual	1558	BR68
DATANET 30/305/355	GRTS Programming Reference	DA79	DA79
Service and utility routines:			
Loader	General Loader	1008	BN90
Utility Programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Programs	1651	BS07
FORTRAN	FORTRAN Subroutine Libraries Reference Manual	1620	BR95
Controller Loader	Relocatable Loader	DA12	DA12
Service Routines	Service Routines	DA97	DA97
Software Debugging	Trace and Debug Routines	DB20	DB20
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch Interface Facility	1642	BR99
System Programming	GCOS Time-Sharing System Programmers' Reference Manual	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	Time-Sharing FORTRAN	1566	BR70
Text Editing	Time-Sharing Text Editor	1515	BR40
Transaction processing:			
User's Procedures	Transaction Processing System User's Guide	DA82	DA82
Handbooks:			
Console Messages	Console Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards & Abort Codes	1691	BJ69
Software maintenance (SMD):			
Table Definitions	GCOS Introduction & System Tables SMD	1488	BR17
Startup program	Startup (INIT) SMD	1489	BR18
Input System	System Input SMD	1490	BR19
Peripheral Allocation	Dispatcher and Peripheral Allocation SMD	1491	BR20
Core Allocation/Rollcall	Rollcall, Core Allocation and Operator Interface SMD	1492	BR21
Fault Processing	Fault Processing SMD	1493	BR22
Channel Modules	I/O Supervisor (IOS) SMD	1494	BR23
Error Processing	GCOS Exception Processing SMD	1495	BR24
Output System	Termination and System Output SMD	1496	BR25
File System Modules	File System Maintenance SMD	1497	BR26
Utility Programs	GCOS Utility Routines SMD	1498	BR27
Time-Sharing System	Time-Sharing Executive SMD	1501	BR29

Rev. 7303

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 6000 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 6000:		
Hardware reference:			
Series 6000	Series 6000 Summary Description	DA48	DA48
DATANET 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards Reference Manual	1688	BS19
System initialization:			
GCOS Startup	System Startup and Operation	DA06	DA06
Communications System	GRTS/355 Startup Procedures Reference Manual	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File System	File Management Supervisor	DB54	DB54
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processor	DA37	DA37
Multi-Access I-D-S	Multi-Access I-D-S Implementation Guide	DA80	DA80
File Input/Output	File and Record Control	1003	BN85
I-D-S Data Query System	I-D-S Data Query System Installation	DB57	DB57
I-D-S Data Query System	I-D-S Data Query System User's Guide	DB56	DB56
Program maintenance:			
Object Program	Source and Object Library Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
On-Line Peripheral Testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Total On-Line Testing	Total On-Line Test System (TOLTS)	DA49	DA49
Error Analysis and Logging	Honeywell Error Analysis and Logging System	DB50	DB50
Language processors:			
Macro Assembly Language	Macro Assembler Program	1004	BN86
COBOL Language	COBOL Compiler	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
DATANET 355	DATANET 355 Macro-Assembler Program	1660	BB98
Generators:			
Sorting	Sort/Merge Program	1005	BN87
Merging	Sort/Merge Program	1005	BN87

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 6000:		
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator Reference Manual	1663	BW23
Service and utility routines:			
Loader	General Loader	1008	BN90
Utility Programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Programs	1651	BS07
FORTRAN	FORTRAN Subroutine Libraries Reference Manual	1620	BR95
Controller Loader	Relocatable Loader	DA12	DA12
Service Routines	Service Routines	DA97	DA97
Software Debugging	Trace and Debug Routines	DB20	DB20
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch Interface Facility	1642	BR99
System Programming	GCOS Time-Sharing System Programmers' Reference Manual	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	FORTRAN	1686	BJ67
Text Editing	Time-Sharing Text Editor	1515	BR40
Remote terminal system:			
DATANET 30	NPS/30 Programming Reference	1558	BR68
DATANET 30/305/355	GRTS Programming Reference	DA79	DA79
Transaction processing:			
User's Procedures	Transaction Processing System User's Guide	DA82	DA82
Handbooks:			
Console Messages	Console Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards and Abort Codes	1691	BJ69

CONTENTS

ILLUSTRATIONS

TABLES

SECTION I

INTRODUCTION

GENERAL

FORTRAN is an automatic coding language. It closely resembles the ordinary language of mathematics and provides the facility for expressing any problem requiring numerical computation. In particular, problems involving large sets of equations and containing many variables may be handled easily. FORTRAN is especially suited for solving scientific and engineering problems, and it is also suitable for many business applications.

The FORTRAN Language consists of words and symbols arranged into statements. A set of FORTRAN statements, describing each step in the solution of a problem, is a FORTRAN program (a source language program).

The Series 6000 FORTRAN compiler is a processor which translates a FORTRAN program into machine language. Each computing system that has a FORTRAN compiler translates FORTRAN programs into its own machine language. This processor is provided as a part of the Series 600/6000 Software System to translate FORTRAN source language programs to machine language programs in the form acceptable for execution with the General Comprehensive Operating Supervisor (GCOS).

The FORTRAN language is augmented by a prewritten library of routines which accompany the system. These routines evaluate the standard arithmetical functions, provide all input/output for the program, and furnish the user with other services to aid in the problem solution. Special purpose routines may be written by the user for use as subprograms.

CAPABILITIES

The Series 6000 FORTRAN services both batch and time sharing, using the same compiler modules for both environments. Users have the capability of developing programs for eventual use in the batch environment with the convenience of the interactive time sharing environment, and after debug is complete, submitting them to batch without concern for time sharing/batch language incompatibilities.

This compiler allows users to enter FORTRAN programs in exactly the same form regardless of the input medium or location. The only difference in the input stream at the user interface is the mandatory presence of GCOS control cards for local and remote batch and the required use of command language in the time sharing environment. Remote accessed use of GCOS, including both time sharing and remote batch, will contribute significantly to the job load at the Central Computer Site.

A number of the extensions which were developed by other manufacturers add significant capabilities to the FORTRAN language. Many of these new features have been included in the Series 6000 FORTRAN; particularly extensions contained in the IBM 360 FORTRAN and the UNIVAC FORTRAN V. Some entirely new features, not available on any other FORTRAN compiler, have also been included.

COMPARISON OF FORTRAN COMPATIBILITIES

Table 1-1 contains the capabilities of the Series 6000 FORTRAN as compared with the Series 600 Time Sharing FORTRAN, the Series 600 FORTRAN IV (batch), and the ANSI standard FORTRAN. As indicated, the Series 6000 FORTRAN is the most comprehensive.

1. ANSI FORTRAN IV is a subset of Series 6000 FORTRAN.

2. Series 600 FORTRAN IV is a subset of Series 6000 FORTRAN except:

   a. FORMAT GENERATOR feature is not in Series 6000 FORTRAN. (See Appendix D for a conversion routine.)

   b. The DEBUG Statement is not in Series 6000 FORTRAN. (See Appendix D for a conversion routine.)

3. Series 600 Time Sharing FORTRAN is not a subset of Series 6000 FORTRAN; however, Series 6000 FORTRAN contains many of the time sharing FORTRAN extensions with slightly different syntax and form.

Series 6000 FORTRAN is fully compatible, at the object level, with Series 600 FORTRAN (Batch).

Table 1-1. Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
GENERAL PROPERTIES				
Statement Numbers	1 thru 99,999	1 thru 99,999	1 thru 99,999	1 thru 99,999
Embedded Blanks Allowed	Yes	Yes	Yes	Yes
Continuation Cards	19	19	No limit	19
Specification Statement Must Precede 1st Executable Statement	No	Yes	Yes	No
Arithmetic Statement Function Must Precede All Executable Statements	Yes	Yes	Yes	Yes
Arithmetic Statement Function must follow all specification statements	No	Yes	Yes	Yes
CHARACTER variables and constants	Yes	No	No	No
IMPLICIT	Yes	No	No	No
ENTRY	Yes	Yes	No	No
FORMAT GENERATOR	No [a]	Yes	No	No
DEBUG	No [a]	Yes	No	No
Adjustable Dimensions	Yes	Yes	Yes	Yes
Array Dimensions	7	7	63	3
Initialize Data in Type Statement	Yes	No	No	No
$ Acceptable within a Symbol	No	No	No	No

[a] Refer to Appendix D for a description of a conversion program for these statements.

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
Mixed Mode Expressions Allowed	Yes	No	Yes	No
General Arithmetic Expressions Within Subscripts	Yes	No	Yes	No
General Arithmetic Expressions in Output Lists	Yes	No	Yes	No
ALTERNATE RETURNS Statement number used as an argument is preceded by	$	$	$	No
Switch variable as alternate return	Yes	No	No	No
Multiple entry points to a subprogram	Yes	Yes	No	No
Size-in-Bytes Specification in TYPE statements	Yes	No	No	No
END=Clause in READ statements	Yes	No	Yes	No
ERR=Clause in I/O statements	Yes	No	No	No
CHAIN Overlays	No	No	Yes	No
LINK Overlays	Yes	Yes	No	No
Constant Zero Accepted as the Initial Value of a DO index	No	No	Yes	No
Value (in storage) of the DO index is always updated	No (OPTZ) Yes (NOPTZ)	No	Yes	— —

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
STATEMENT EXTENSIONS				
Namelist without file reference	Yes	No	Yes	No
NAMELIST	Yes	Yes	Yes	No
NAMELIST Array Available Dimensions	7	3	63	No
ENCODE/DECODE	Yes	No	Yes	No
FILENAME and ASCII Variables and Constants	No	No	Yes	No
PARAMETER $P_1 = V_1$, $P_2 = V_2$, ...	Yes	No	No	No
ABNORMAL	Yes	No	No	No
CONSTANTS, VARIABLES SUBSCRIPTS, EXPRESSIONS				
Complex Numbers	Yes	Yes	No	Yes
Double Precision Numbers	Yes	Yes	No	Yes
Logical Constants	Yes	Yes	Yes	Yes
Subscript Forms: V,C,V±C,C*V,C*V±C'	Yes	Yes	Yes	Yes
Any arithmetic expression as a subscript	Yes	No	Yes	No
Relational Expressions	Yes	Yes	Yes	Yes
Variable Length Names	1-8	1-6	Unlimited	1-6
Mixed Mode Expressions Allowed	Yes	Floating pt. forms only.	Yes	No
Data Initialization Statement	Yes	Yes	Yes	Yes
Quoted Literals	Yes	Yes	Yes	No

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
Integer Constants:				
Number Digits	1-11	1-11	1-11	---
Magnitude	$2^{35}-1$	$2^{35}-1$	$2^{35}-1$	---
Real Constants:				
Number Digits	1-9	1-9	1-9	---
Magnitude	10^{38}	10^{38}	10^{38}	---
Subscript Magnitude	2^{18}	2^{18}	2^{18}	---
A**B**C Prohibited	No	Yes	Yes	Yes
CONTROL STATEMENTS				
GO TO n	Yes	Yes	Yes	Yes
GO TO $(n_1,n_2,\ldots,n_m)i$	Yes	Yes	Yes	Yes
IF (a) n_1,n_2,n_3 (Arithmetic IF)	Yes	Yes	Yes	Yes
IF (a)s (Logical IF)	Yes	Yes	Yes	Yes
Assign i to n	Yes	Yes	Yes	Yes
GO TO N, $(n_1,n_2,\ldots,n_m)$	Yes	Yes	Yes	Yes
DO n i = m_1,m_2,m_3	Yes	Yes	Yes	Yes
CONTINUE	Yes	Yes	Yes	Yes
PAUSE, PAUSE n	Yes	Yes	No (Library Routine)	Yes
STOP	Yes	Yes	Yes	Yes
STOP n	Yes	No	No	Yes
END	Yes	Yes	Yes	Yes
INPUT/OUTPUT STATEMENTS				
FORMAT $(c_1,c_2,\ldots,cn/c'_1,c'_2,\ldots,c'_n)$	Yes	Yes	Yes	Yes

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
Formatted Sequential Character I/O Statements				
PRINT/READ t,list	Yes	Yes	Yes	No
PUNCH t,list	Yes	Yes	No	No
PRINT/READ, list	Yes	No	Yes	No
PUNCH,list	Yes	No	No	No
PRINT/READ X	Yes	Yes	Yes	No
PUNCH X	Yes	No	No	No
READ (f,t) list	Yes	Yes	Yes	Yes
WRITE (f,t) list	Yes	Yes	Yes	Yes
READ (f,x)	Yes	Yes	Yes	No
WRITE (f,x)	Yes	Yes	Yes	No
Unformatted Sequential Binary I/O Statements				
READ (f) list	Yes	Yes	Yes	Yes
WRITE (f) list	Yes	Yes	Yes	Yes
Unformatted Random Binary I/O Statements				
READ (f'n) list	Yes	No	Yes	No
WRITE (f'n) list	Yes	No	Yes	No
Format Property				
A-Conversion	Yes	Yes	Yes	Yes
R-Conversion	Yes	No	No	No
F-Conversion	Yes	Yes	Yes	Yes
E-Conversion	Yes	Yes	Yes	Yes
G-Conversion	Yes	Yes	Yes	Yes
H-Conversion	Yes	Yes	Yes	Yes

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
I-Conversion	Yes	Yes	Yes	Yes
X-Conversion	Yes	Yes	Yes	Yes
D-Conversion	Yes	Yes	No	Yes
V-Conversion	Yes	No	Yes	No
L-Conversion	Yes	Yes	Yes	Yes
T-Conversion	Yes	No	No	No
O-Conversion	Yes	Yes	Yes	No
For n Slashes at End, Input Records Skipped	n	n	n	n
For n Slashes at End, Blank Records Written	n-1	n-1	n-1	n
Quoted Literals	Yes	Yes	Yes	No
Scale Factors	Yes	Yes	Yes	Yes
Field Repetition	Yes	Yes	Yes	Yes
Parentheses Levels	2	2	10	2
Carriage Controls:				
blank	Yes	Yes	Yes	Yes
0	Yes	Yes	Yes	Yes
1	Yes	Yes	Yes	Yes
+	Yes	Yes	Yes	Yes
Format Statements Read In at Object Time	Yes	Yes	Yes	Yes
File Manipulation Statements:				
ENDFILE a	Yes	Yes	Yes	Yes
REWIND a	Yes	Yes	No	Yes
BACKSPACE a	Yes	Yes	Yes	Yes
OPENFILE a	No	No	Yes	No
BEGINFILE a	No	No	Yes	No

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
CLOSEFILE a	No	No	Yes	No
SPECIFICATION STATEMENTS				
INTEGER $a(i_1),b(i_2)...$	Yes	Yes	Yes	Yes
REAL $a(i_1),b(i_2)...$	Yes	Yes	Yes	Yes
DOUBLE PRECISION $a(i_1)$, $b(i_2)...$	Yes	Yes	No	Yes
LOGICAL $a(i_1)$, $b(i_2)$..	Yes	Yes	Yes	Yes
COMPLEX $a(i_1)$, $b(i_2)$..	Yes	Yes	No	Yes
CHARACTER $a(i_1),b(i_2)$.	Yes	No	No	No
ASCII $a(i_1),b(i_2)...$	No	No	Yes	No
FILENAME a,b...	No	No	Yes	No
DIMENSION $a_1(k_1)$, $a_2(k_2)$,...	Yes	Yes	Yes	Yes
COMMON a,b,c,...,d,e,f	Yes	Yes	Yes	Yes
COMMON a,b,c,.../r/ d,e,f,.../s/ g,h,...	Yes	Yes	(1)	Yes
EQUIVALENCE (a,b,c,...)(d,e,f,...)	Yes	Yes	No	Yes
NAMELIST	Yes	Yes	Yes	No
SUBPROGRAMS				
FUNCTION name $(a_1,a_2,...,a_n)$	Yes	Yes	Yes	Yes
REAL FUNCTION name $(a_1,a_2,...,a_n)$	Yes	Yes	(2)	Yes
INTEGER FUNCTION name $(a_1,a_2,...,a_n)$	Yes	Yes	(2)	Yes
DOUBLE PRECISION FUNCTION name $(a_1,a_2,...,a_n)$	Yes	Yes	(2)	Yes

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
COMPLEX FUNCTION name $(a_1,a_2,\ldots,a_n)$	Yes	Yes	(2)	Yes
LOGICAL FUNCTION name $(a_1,a_2,\ldots,a_n)$	Yes	Yes	(2)	Yes
CHARACTER FUNCTION name $(a_1,a_2,\ldots,a_n)$	Yes	No	No	No
SUBROUTINE name $(a_1,a_2,\ldots,a_n)$	Yes	Yes	Yes	Yes
CALL name $(a_1,a_2,\ldots,a_n)$	Yes	Yes	Yes	Yes
RETURN	Yes	Yes	Yes	Yes
RETURN i	Yes	Yes	Yes	No
EXTERNAL X,Y,Z,...	Yes	Yes	Yes	Yes
BLOCK DATA	Yes	Yes	No	Yes
ENTRY name $(a_1,a_2,\ldots,a_n)$	Yes	Yes	No	No
FORMATS				
Format	(3)	Fixed	Free-Field	Fixed
Sequence Numbers (73-80)	Yes	Yes	No	Yes
Line Numbers	(3)	No	(4)	No
Line Length	(5)	80	72	72
Statements per line	Multiple (uses ;)	1	Multiple (uses ;)	1
Comment Lines	C or * in Col. 1	C or * in Col. 1	*	C

Table 1-1 (cont). Comparison of FORTRAN Features

FORTRAN FEATURE	Series 6000 FORTRAN	Series 600 FORTRAN IV	Series 600 Time Sharing FORTRAN	ANSI FORTRAN
Continuation	(3)	(6)	& after Line No.	(6)
Character Set (source)	BCD or ASCII	BCD	ASCII	---
Source Medium -Decks	Yes	Yes	No	Yes
-Comdks	Yes	Yes	No	No
-TS Files	Yes	No	Yes	No
-Alter	Yes	Yes	No	No

Note (1): Blank Common is Allowed; Labeled Common is not.
Note (2): Done presently in type statement.
Note (3): Determined by the FORM option (described below)

FORM Specified

1. Cols. 1-5 are reserved for statement numbers.
2. Continuation indicated by a non-blank, non-zero Col. 6.
3. Comments indicated by * or C in Col 1.
4. Cols. 7-72 contain FORTRAN statements.
5. Cols. 73-80 may contain sequence number.

NFORM and LNO Specified

1. Cols. 1-n (n≤8) contain a sequence number.
2. Statements begin anywhere between Cols. 1-80 following sequence number.
3. Continuation indicated by an ampersand as the first non-blank character in statement.
4. Statement numbers must be separated from sequence number by at least one blank or by pound sign (#).
5. Comments indicated by an asterisk or C as first character in statement.

NFORM and NLNO Specified

1. Statements begin anywhere between Cols. 1-72.
2. Continuation indicated by ampersand as first non-blank character of statement.
3. Comments indicated by * or C as first character.
4. Cols. 73-80 may contain a sequence number.

Note (4): 8 numerics (not referenced in program).
Note (5): 80 characters if card, otherwise unlimited.
Note (6): Non-zero, non-blank character in Col. 6.

SECTION II

RULES AND DEFINITIONS

CHARACTER SET

FORTRAN utilizes two character sets - ASCII and Series 600/6000 BCD. The character set and byte size of the internal representation of generated object code is controlled by an option on the $ FORTY or $ FORTRAN card or the YFORTRAN or FORTRAN RUN command. The byte size will be 6 or 9 bits depending on the option selected (BCD or ASCII). Appendix A contains the ASCII and BCD character set with the octal and card representation for each character. The character set of the source program is self-determining and requires no options.

The FORTRAN character set is a subset of the full 128 ASCII characters and is used as follows:

1. FORTRAN statements and the verbs or prepositions do not differentiate between upper and lower case alphabetic characters.

2. No distinction is made between the cases in forming variable, function, common, etc. names.

3. Upper and lower case letters are recognized as different only in user character data and literals.

4. Character restrictions may be necessary for certain external routine procedures. For example, symbols in assembly language subroutines may be restricted to upper case.

5. Any character in the ASCII character set is valid as literal data.

A program unit is written using the following characters:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p,
q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and

CHARACTER	NAME OF CHARACTER
⌀	Space
=	Equals
+	Plus
-	Minus
∧ or ↑	Vertical Arrow or Caret
*	Asterisk
&	Ampersand
/	Slash
(	Left Parenthesis
)	Right Parenthesis
,	Comma
.	Radix Point
$	Currency Symbol
' or	Apostrophe or Acute Accent
;	Semicolon
"	Quotation Marks

The order in which the characters are listed does not imply a collating sequence. All are ASCII characters.

Special Characters

The following special characters are used for FORTRAN syntax punctuation:

Space " $ ( ) + - , / ; = . ' ↑ & *

The space character is not meaningful to the compiler except in character literals and may be used freely to enhance readability of programs.

Quotation marks (") and apostrophes (') are used as character literal delimiters. The apostrophe also precedes the record number in random file input/output statements.

The currency symbol ($) identifies statement numbers used as arguments. It also serves as a delimiter of input data for NAMELIST read.

Parentheses ( ) are used to enclose subexpressions, complex constants, equivalence groups, format specification, argument lists, subscripts, and to specify the ranges of implied DO loops.

Plus sign (+) indicates algebraic addition, Printer carriage control, or a unary operator.

Minus sign (-) indicates algebraic subtraction or a unary operator.

The comma (,) is used as a separator for data symbols and expressions for parameter lists, equivalence groups, complex constants and format specifications.

The slash (/) is used to indicate algebraic division, as a delimiter for data lists, labeled common statements, and as a record terminator in a format statement.

The semicolon (;) is used as a statement delimiter.

The equality sign (=) indicates the assignment operator in arithmetic and logical assignment statements, Parameter statements, DO statements, and implied DO statements in I/O and data lists.

The asterisk (*) designates a comment line or an alternate return argument in a subroutine statement. The asterisk is also used as the multiplication operator, and a double asterisk (**) is one of the exponentiation operators. The quantity to the left of the sign is raised to the power indicated on the right.

The period (.) is used as a radix point, and serves as a delimiter for symbolic logical, and relational operators and logical constants.

The up arrow and caret (↑ or ∧) serve as additional exponentiation operators. They are alternates to the double asterisk (**) and may be freely used interchangeably.

The ampersand (&) serves as one of the continuation line indicators.

SOURCE PROGRAM FORMAT

Source File Types

Source programs generally originate as either punched cards or typed lines on a teletypewriter. They may also be the product of (output from) the execution of some program, or one may be compressed in a compilation activity through use of the COMDK option. These source programs may be kept in the form of decks, paper tape, magnetic tape files, or permanent mass storage files. To be compiled, decks and paper tape media programs must be copied to magnetic tape, or mass storage first. The mass storage file need not be permanent; a normal deck setup will produce the compiler input file (S*) on a temporary file. The source program file must be recorded in Standard System Format (see the File and Record Control manual). The Series 6000 FORTRAN compiler will accept magnetic tape or mass storage files, in Standard System Format, with any of the following media codes:

0 - formatted BCD line images, without slew control for the printer
1 - compressed BCD card images
2 - (uncompressed) BCD card images
3 - formatted BCD line images, with trailing printer slew control information
5 - old time sharing ASCII format
6 - new time sharing ASCII standard system format

Card images are limited to eighty (80) characters, while line images are limited according to the device on which they were prepared. For simplification, wherever "card images" and "line images" could both be used, this document will simply use the term "line".

Relationship of Statements to Lines

A source program file is made up of statements and comments. A statement may be contained on from one to twenty lines. The first is called an initial line and the rest are called continuation lines. A comment is contained on one line, it is not considered as a statement, and merely provides information for documentary purposes. Comment lines may be placed freely in the program file, even between consecutive continuation lines.

Every program unit (subprogram, main program, etc.) must terminate with an end line. This line contains an END statement and serves to separate individual program units. Any subsequent units must begin on a new line.

When the first line of a program unit is a comment line, page titles and object deck labels are extracted from that line as follows:

Characters 2-7	identification label of the object deck
Characters 8-72	page title for listings

When the first line of a program unit is not a comment line, the deck label will be the first SYMDEF of the routine; no page title will be used.

Format Rules for Lines

A variety of source line formats are acceptable to Series 6000 FORTRAN, ranging from the ANSI standard 80 character fixed format to the standard line formats used with the time sharing system. Specification of which format has been used is via two options: FORM/NFORM and LNO/NLNO. These options may appear on the $ FORTY or $ FORTRAN control card or in the option list of the YFORTRAN or FORTRAN RUN command.

Source files in ANSI standard format should be run using the FORM option. Time sharing source files should normally use NFORM+LNO. These are the default options when jobs originate from batch and time sharing, respectively.

FORM FORMATTED LINES

Lines in FORM format have the following characteristics:

1. Comment lines are recognized by a C or * in character position 1.
2. Continuation lines are recognized by a non-blank, non-zero character in position 6.

Lines containing more than 72 characters (e.g., card images) in FORM format have the following additional characteristic:

3. Character positions 73-80 may be used for sequence identification information. This field is not considered part of the statement, it is provided for programmer convenience.

4. No more than 80 characters will be processed. If more are present, they are ignored.

The LNO/NLNO option is not applicable to files in FORM format. Only NFORM format files can have line numbers.

NFORM FORMATTED LINES - NLNO

Lines in NFORM format with no line numbers (NLNO) have the following characteristics:

1. Comment lines are recognized by a C or * in character position 1.

2. A continuation line is indicated by the ampersand character (&) as the first non-blank character of the line.

Card images in this format also have the characteristic:

3. Character positions 73-80 may be used for sequence identification information.

NFORM FORMATTED LINES - LNO

Lines in NFORM format with line numbers (LNO) have the following characteristics:

1. A line number field begins in character 1. The line number field may contain up to eight characters and may contain leading blanks. The magnitude of this line number is treated modulo 2^{18} (262144).

2. Line numbers which are less than eight characters long must be terminated by a non-numeric character.

3. If the character following the line number is a # it is ignored and the next character is considered to be following the line number.

4. Comment lines are recognized by a C or * as the next character following the line number.

5. A continuation line is indicated by the ampersand character (&) as the first non-blank character following the line number.

Card images in this format do not reserve characters 73-80 for sequence identification information. The statement text may extend into these positions.

Format Rules Common to FORM/NFORM

The above rules show that the format options are used to control the following functions:

1. Elimination of line numbers and sequence identification fields from the lines.

2. Separation of comment lines from statement lines.

3. Distinction between initial statement lines and continuation lines.

4. Determination of the position numbers of the first and last characters of the statement text.

Beyond this the line format is the same. Initial lines may begin with a statement number. The statement number may begin anywhere on the line but must be in the range $1 \leq n \leq 99999$. There may be up to 19 continuation lines and the statement text continues with the first character following the continuation character.

A statement may be terminated by the semi-colon (;) character on either an initial or continuation line. The information remaining on the line is processed as an initial line. The new statement may begin with a statement number and may be continued. Note that it is not possible to put comments on the same line as the statement line which ends with a semi-colon.

Figure 2-1 illustrates the appearance and general properties of a FORTRAN program written on a coding sheet. This example illustrates the FORM format.

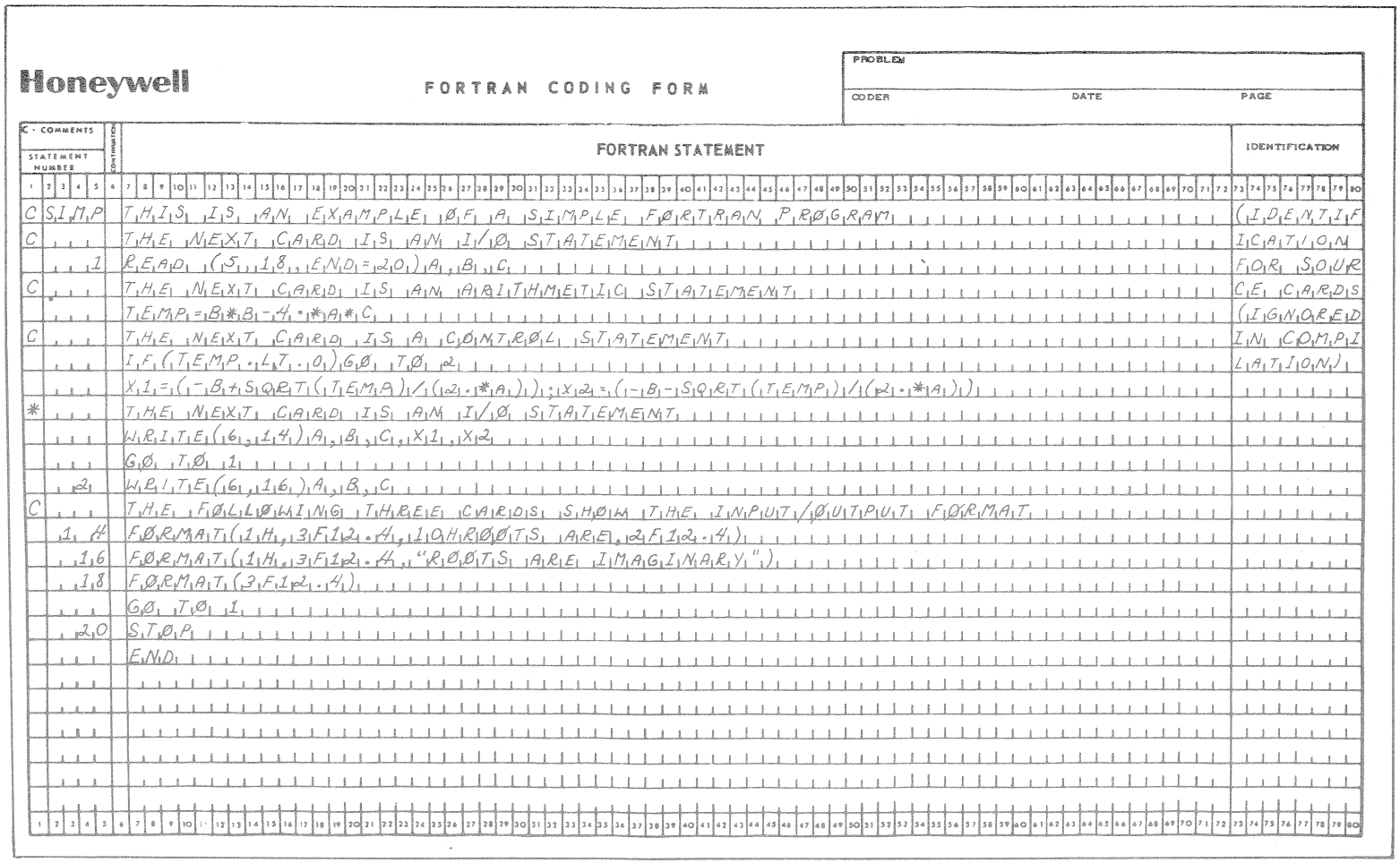


Figure 2-1. FORTRAN Coding Sheet

SYMBOL FORMATION

A symbolic name consists of one to eight alphanumeric characters, the first of which must be alphabetic. Data types may be associated with a symbolic name either implicitly or explicitly. The implicit associations are determined by the first character of the symbol; integer if the name begins with the letters I,J,K,L,M, or N; otherwise real. This default implicit associative rule may be changed by the use of the IMPLICIT statement. This allows implicit association for all data types - integer, real, double- precision, complex, logical, or character. An explicit declaration of type for some symbol always overrides its implicit type. Data type is explicitly associated with a symbol when it appears in one of the type statements: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, or CHARACTER, or when it appears in a FUNCTION statement with a type prefix (e.g., REAL FUNCTION MPYM(A,B)).

No case distinction is made in forming symbols. The symbol ABC is identical to the symbols abc and Abc.

A symbolic name representing a function, variable, or array has only one data type association for each program unit. Once associated with a particular data type, a specific name implies that type for any usage of that symbolic name that requires a data type association throughout the program unit in which it is defined.

DATA TYPES

The mathematical and representational properties for each of the data types are defined below. The value zero is considered neither positive nor negative.

1. An integer datum is always an exact representation of an integer value. It may assume positive, negative, or zero integral values. Each integer datum requires one 36 bit word of storage in fixed point format. The permissible range of values for integer type is -2^{35} to $2^{35}-1$.

2. A real datum is a processor approximation to the value of a real number. It may assume positive, negative, or zero values, possibly fractional. A real datum requires one 36 bit word of storage in floating point format. The permissible range of values for real type is approximately $\pm 10^{38}$ to $\pm 10^{-38}$ with a precision of eight digits.

3. A double precision datum is a processor approximation to the value of a real number. It may assume positive, negative, or zero values. A double precision datum requires two consecutive 36-bit words of storage in double precision floating point format. The permissible range of values for double precision type is approximately $\pm 10^{38}$ to $\pm 10^{-38}$ with a precision of 18 digits.

4. A complex datum is a processor approximation to the value of a complex number. The representation of the approximation is in the form of an ordered pair of real data. The first of the pair represents the real part and the second, the imaginary part. Each part has, accordingly, the same degree of approximation as for a real datum. A complex datum requires two consecutive words of storage, each in floating point format. Each part of a complex datum has the same range of values and precision as a real datum.

5. A logical datum is a representation of a logical value of true or false. The source representation of the logical value "true" may be either .TRUE. or .T., and in DATA statements, the single character "T" may also be used. For the value "false", .FALSE. and .F. may be generally used with "F" being allowable in DATA statements. A logical datum requires one 36-bit word of storage with the value zero representing "false", and non-zero representing "true". Where input/output is involved, the external representations of "true" and "false" are the single letters "T" and "F".

6. A character datum is a processor representation of a string of ASCII or BCD characters. This string may consist of any characters capable of being represented in the processor. The space character is a valid and significant character in a character datum. Character strings are delimited by quotes, apostrophes, or by preceding the string by nH. The character set (BCD or ASCII) is declared by an option on the $ FORTY or $ FORTRAN control card or the YFORTRAN or FORTRAN RUN command.

The term "reference" is used to indicate an identification of a datum implying that the current value of the datum will be made available during the execution of the statement containing the reference. If the datum is identified but not necessarily made available, the datum is said to be "named". One case of special interest in which the datum is named is that of assigning a value to a datum, thus defining or redefining the datum.

CONSTANTS

There are three general types of constants - character, single word, and double word. Each of these types is divided as follows:

1. Single Word Constants

   a. Integer
   b. Octal
   c. Real
   d. Logical

2. Double Word Constants

   a. Double Precision
   b. Complex

3. Character Constants

A constant is a value that is known prior to writing a FORTRAN statement and which does not change during program execution.

Integer Constants

An integer constant consists of 1 to 11 decimal digits with an accuracy of 10 digits. The decimal point of the integer is always omitted; however, it is always assumed to be immediately to the right of the last digit in the string. An integer constant may be as large as $2^{35}-1$ ($\cong 3.4 \times 10^{10}$), except when used for the value of a subscript or as an index of a DO or a DO parameter, in which case the value of the integer is computed modulo 2^{18}.

Example:

```
-7
 152
 843517
```

Octal Constants

An octal constant is written as a non-empty string of up to 12 octal digits preceded by the letter O and an optional sign. The sign affects only bit 0 of the resulting literal (complementation does not take place). Octal constants may be used in preset data lists only (e.g., DATA statement).

Example:

```
Ø  777000
Ø - 377777777
```

Real Constants

A real constant is in floating-point mode and is contained in one computer word (single precision). This constant consists of one of the following:

1. One to nine significant decimal digits written with a decimal point, but not followed by a decimal exponent.

2. One to nine significant decimal digits written with or without a decimal point, followed by a decimal exponent written as the letter E followed by a signed or unsigned one or two digit integer constant. When the decimal point is omitted, it is always assumed to be immediately to the right of the rightmost digit. The exponent value may be explicitly 0, and the field following the E may not be blank.

Examples:

```
  75.
1234.
  21.083
  -3.2105
   7.0E2      (means 7.0 x 10^2; 700)
   7E-3       (means 7.0 x 10^-3; .007)
```

A real constant has precision to eight digits. The magnitude must be between the approximate limits of 10^{-38} and 10^{38}, or must be zero.

Double Precision Constants

A double-precision constant is in floating-point mode and is contained in two computer words. This constant consists of one of the following:

1. Ten to eighteen significant decimal digits written with a decimal point, but not followed by a decimal exponent.

2. Up to 18 significant decimal digits written with or without a decimal point, followed by a decimal exponent written as the letter D followed by a signed or unsigned one or two digit integer constant. When the decimal point is omitted, it is always assumed to be immediately to the right of the rightmost digit. The exponent value may be explicitly 0, and the field following the D may not be blank.

Examples:

```
 12.34567891
-13.57D0
   .1234D0
  7.0D4       (means 7.0 x 10^4, 70000.)
  7D-3        (means 7.0 x 10^-3, .007)
```

Double-precision constants have precision to 18 digits. The magnitude of a double-precision constant must lie between the approximate limits of 10^{-38} and 10^{38}, or must be zero.

Complex Constants

A complex constant consists of an ordered pair of signed or unsigned real constants separated by a comma and enclosed in parentheses.

Examples:

```
(10.1, 7.03) is equal to 10.1 + 7.03i
(5.41, 0.0) is equal to 5.41 + 0.0i
(7.0E4, 20.76) is equal to 70000. + 20.76i
```

where i is the square root of -1.

The first real constant represents the real part of the complex number; the second real constant represents the imaginary part of the complex number. The parentheses are required regardless of the context in which the complex constant appears. Each part of the complex constant may be preceded by a plus sign or a minus sign, or it may be unsigned.

Logical Constants

A logical constant may take either of the two forms:

```
.TRUE. (or .T.)
.FALSE. (or .F.)
```

and is represented in the machine as

```
TRUE≠0
FALSE=0
```

Representation may be T and F respectively in DATA statements or externally when doing input/output.

Character Constants

Character constants are of two kinds characterized by their representation in either the ASCII or the BCD character set. The kind is determined by an option on the $ FORTY or $ FORTRAN card or the YFORTRAN or FORTRAN RUN command. Character constants are formed in one of the following ways:

1. Preceding the character string by nH.
2. Enclosing the string in quotation marks.
3. Enclosing the string in apostrophes.

Character constants can be used as arguments to external subprograms, as literals in the DATA statement, as part of a FORMAT statement, as the display object of the STOP and PAUSE statements, in a character assignment statement, or in a relational expression.

The maximum length of a character constant is 511 characters.

The interpretation of quoted strings of both types is such that the appearance of the string delimiter in two consecutive character positions within a string is considered as a single occurrence of the delimiter as a member of that string. For example, the representation: "abc""ef" is represented internally as the literal abc"ef. Alternatively, the other delimiter type can be used (e.g., 'abc"ef').

VARIABLES

Variable Type Definition

A variable is any quantity that is referred to by name rather than by value. A variable may take on many values and may be changed during the execution of the program.

The type of a variable is specified implicitly by its name, or explicitly by use of a type statement.

1. Default implicit type association enables the declaration of real and integer variables and function names according to the following rules:

   a. If the first character of the name is I,J,K,L,M, or N, (upper or lower case) it is an integer name.

   b. If the first character is any other alphabetic character, it is a real name.

2. The IMPLICIT type statement is used to redefine the implicit typing. See the IMPLICIT statement description in Section IV of this document.

3. The explicit type statements (described in this document) are used to assign a type to a variable or function subprogram.

4. Function subprogram names may be typed on the FUNCTION statement by use of the type prefix.

Scalar Variable

The six types of scalar variables are: character, integer, real, logical, double-precision, and complex. A scalar variable may take on any value its corresponding constant may assume. A scalar variable occupies the same number of storage locations as a constant of the same type.

External Variable

An external variable is the name of a subprogram that appears as an actual argument in the calling sequence to some subprogram. It must appear in an EXTERNAL statement before its first use in the source program.

Parameter Symbols

A parameter symbol is used when it is desired to compile a program several times when the only changes from one compilation to the next are to certain constants. The parameter symbol (described under the PARAMETER statement) is used under these circumstances.

Character Variable

Character variables may be implicitly typed via the IMPLICIT statement or explicitly using the CHARACTER statement. The limit is 511 characters.

Array

An array is an ordered set of data with from one to seven dimensions. The array is referenced by a symbolic name. Identification of the entire ordered set is achieved by the use of the array name.

Array Element

An array element is an item of data in an array. It is identified by immediately following the array name with a subscript that points to the particular element of the array. In some instances the array name may be used in unsubscripted notation to reference the first element of the array.

Subscripts

A variable may be made to represent any element of an array containing from one to seven dimensions by appending one to seven subscripts to the variable name. Subscript expressions are separated by commas. The number of subscript expressions must correspond with the declared dimensionality except in an EQUIVALENCE statement. Following evaluation of all of the subscript expressions, the array element successor function determines the identified element.

Form of Subscript

A subscript expression may take the form of any legal FORTRAN arithmetic expression. The result of any such expression is truncated to an integer before use.

Examples:

```
IMAS        8*IQUAN        9+J
J9          5*L+7          B**2
K2          H*M-3          6**A-(1-SQRT(3.14))/8
N+3         7+2*K          LIST (J)
```

The value of a subscript expression must be greater than zero and not greater than the corresponding array dimension. The value of a subscript expression containing real variables is truncated to an integer after evaluation.

Subscripted Variables

A subscripted variable consists of a variable name, followed by parentheses, enclosing one to seven subscripts separated by commas.

Examples:

```
A(I)
K(3)
BETA (8*J+2,K-2,L)
MAX (K,J,K,L,M,N)
```

1. During execution, the subscript is evaluated so that the subscripted variable refers to a specific member of the array.

2. Each variable that appears in subscripted form must have the size of the array specified. This must be done by DIMENSION, COMMON or type statements that contains the dimension information. The specification of dimensionality must precede the first reference to the array.

Array Element Successor Function

The general algorithm to linearize a subscript involving n terms (for an array of n dimensions) is:

$$S = \sum_{I=1}^{n} ((e_I - 1) \cdot \prod_{J=0}^{I-1} d_J) + 1$$

where each e_I is a subscript term and each d_J is an array dimension

The term d_0 is the "zero-th dimension" of the array. It reflects the number of words of core store required for one element. For example: integer, logical, and real quantities require one word per element ($d_0 = 1$); double-precision and complex quantities require a word pair ($d_0 = 2$); and character variables, which use the size in bytes notation to provide the number of characters per element may have a d_0 value of up to 64 (since they have a maximum of 511 characters). The formula for reducing size in characters to size in words is a function of the BCD/ASCII option. Let n be the number of characters specified, and m be the number of characters per word (6 for BCD, 4 for ASCII). Then d_0 is computed as:

$$d_0 = (n+m-1)/m$$

The following are examples using integer and complex quantities:

```
INTEGER   X(3,2,4)
X(2,2,2)= 1
```

Expanding the algorithm for three dimensions:

$$S = (e_1-1)*d_0 + (e_2-1)*d_0*d_1 + (e_3-1)*d_0*d_1*d_2 + 1$$

$$S = (2-1)*1 + (2-1)*1*3 + (2-1)*1*3*2 + 1$$

$$S = 11$$

Looking at the array in storage in ascending order, the elements are:

```
X(1,1,1), X(2,1,1), X(3,1,1), X(1,2,1), X(2,2,1),

X(3,2,1), X(1,1,2), X(2,1,2), X(3,1,2),

X(1,2,2), X(2,2,2), ..., X(3,2,4)
```

X(2,2,2) is the eleventh word of the array.

```
COMPLEX X (3,2,4)
X(2,2,2) = (1.0, 0.0)

S = (2-1)*2 + (2-1)*2*3 + (2-1)*2*3*2 + 1

S = 21
```

In this example, the first word of the word pair for this element is the twenty-first word of the array.

Array Declarator

An array declarator specifies an array used in a program unit. The array declarator indicates the symbolic name, the number of dimensions (one to seven) and the size of each dimension. The array declarator form may be in a type statement, dimension statement, or common statement. An array declarator has the form:

```
v(i) or v*n(i)
```

where v is the symbolic array name, n is the size-in-bytes of an element, and i is the declarator subscript. Declarator subscript (i) is composed of from one through seven elements each of which may be an integer constant, a parameter symbol or an integer variable. Each element is separated by a comma (if more than one).

The appearance of a declarator subscript in a declarator statement informs the processor that the declarator name is an array name. The number of subscripts indicates the dimensions of the array. The magnitude of the value for the subscript expressions indicates the maximum value that the subscript name may attain in any array element reference.

Adjustable Dimensions

The name of an array and the constants that are its dimensions may be passed as arguments to a subprogram. In this way a subprogram may perform calculations on arrays whose sizes are not determined until the subprogram is called. The following rules apply to the use of adjustable dimensions:

1. Variables may be used as dimensions of an array only in the array declarator of a FUNCTION or SUBROUTINE subprogram. For any such array, the array name and all the variables used as dimensions must appear as dummy arguments in at least one FUNCTION, SUBROUTINE, or ENTRY statement.

2. The adjustable dimensions may not be altered within the subprogram.

3. The true dimensions of an actual array must be specified in a DIMENSION, COMMON, or type statement of some calling program.

4. The calling program passes the specific dimensions to the subprogram. These specific dimensions are those that appear in the DIMENSION, COMMON, or type statement of the calling program. Variable dimension size may be passed through more than one level of subprogram. The specific dimensions passed to the subprogram as actual arguments cannot exceed the true dimensions of the indicated array.

5. Variables used as dimensions must be integers. If the variables are not implicitly typed by their initial letters, a type statement must precede the dimension statement in which they are used as adjustable dimensions.

6. If an adjustable array name or any of its adjustable dimensions appears in a dummy argument list of a FUNCTION, SUBROUTINE, or ENTRY statement, that array name and all its adjustable dimensions must appear in the same dummy argument list.

Example:

```
DIMENSION K(4,5),J(2,3)          SUBROUTINE SETFLG(K,J,I,L,M,N)
      .                                .
      .                                .
      .                          DIMENSION K(I,L),J(M,N)
CALL SETFLG (K,J,4,5,2,3)              .
      .                                .
      .                          DO 20 NO = 1,I
      .                          DO 20 MO = 1,L
                                 K(NO,MO) = 0
                               20 CONTINUE
                                       .
                                       .
                                       .
                                       .
```

EXPRESSIONS

Arithmetic

An arithmetic expression consists of certain legal sequences of constants, subscripted and nonsubscripted variables, and arithmetic function references separated by arithmetic operation symbols, commas, and parentheses.

The following arithmetic operation symbols denote addition, subtraction, multiplication, division, and exponentiation, respectively:

+ - * / ** ↑ or ∧

The rules for constructing arithmetic expressions are:

1. Figures 2-2 and 2-3 indicate which constants, variables, and functions may be combined by the arithmetic operators to form arithmetic expressions. The intersection of a row and column gives the type of the result of such an expression. Figure 2-2 gives the valid combinations with respect to the arithmetic operators +,-,*, and /. Figure 2-3 gives the valid combinations with respect to the arithmetic operators **, ↑ or ∧.

	I	R	D	C	T
I	I	R	D	C	T
R	R	R	D	C	N
D	D	D	D	C	N
C	C	C	C	C	N
T	T	N	N	N	T

Legend

C - Complex
D - Double-precision
I - Integer
N - Nonvalid
R - Real
T - Typeless

Figure 2-2. Arithmetic Expressions +, -, *, and /

POWER

BASE	I	R	D	C	T
I	I	R	D	N	N
R	R	R	D	N	N
D	D	D	D	N	N
C	C	C	C	C	N
T	N	N	N	N	N

Figure 2-3. Arithmetic Expressions - Exponent (**, ↑ or ∧)

2. Any expression may be enclosed in parentheses.

3. Expressions may be connected by the arithmetic operation symbols to form other expressions, provided that:

   a. No two operators appear in sequence except **, which is a single operator and denotes exponentiation.

   b. No operation symbol is assumed to be present. For example, (X) (Y) is not valid.

4. The expression A**B**C is evaluated as A**(B**C).

5. Preceding an expression by a plus or minus sign does not affect the type of the expression.

6. In the hierarchy of operations, parentheses may be used in arithmetic expressions to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows:

   a. Function Reference
   b. **, ↑ or ∧ Exponentiation
   c. * and / Multiplication and Division
   d. + and - Addition and Subtraction

   This hierarchy is applied first to the expression within the innermost set of parentheses in the statement; this procedure continues through the outer parentheses until the entire expression has been evaluated.

7. Operations on the same level (e.g., A*B/C) are evaluated left to right. Parentheses may be used to reorder this sequence if necessary.

The FORTRAN expression

```
A*6+Z/Y**(W+(A+B)/X**K)
```

represents the mathematical expression

$$6A+\frac{Z}{Y^{\left(W+\frac{(A+B)}{X^{K}}\right)}}$$

Logical

A logical expression consists of certain sequences of logical constants, logical variables, references to logical functions, and relational expressions separated by logical operation symbols. A logical expression always results in a true or false evaluation.

The logical operation symbols (where a and b are logical expressions) are:

Symbol	Definition
.NOT.a	This has the value .TRUE. only if a is .FALSE.; it has the value .FALSE. only if a is .TRUE.
a.AND.b	This has the value .TRUE. only if a and b are both .TRUE.; it has the value .FALSE. if a or b or both are .FALSE.
a.OR.b	(INCLUSIVE OR) This has the value .TRUE. if either a or b or both are .TRUE.; it has the value .FALSE. only if both a and b are .FALSE.

The logical operators NOT, AND, and OR must always be preceded and followed by a period.

Logical expression evaluation proceeds to determine the true/false state of the simpler subexpressions first, and stops (evaluation) as soon as the true/false state for the complete expression has been determined. Thus, it is a distinct possibility that the entire expression may not be evaluated. Since this may be of significance to some applications, the following example is given:

```
IF (RAND (X) .GT. 0 .OR. L) GOTO 100
```

Assuming that RAND is an external function and L is a logical variable, the expression is true when either RAND(X) is greater than zero or L is true. The second alternative is clearly simplier to determine than the first. Further, since there is no need to evaluate RAND(X) .GT. 0 when L is true, the statement will be optimized into an equivalent pair of statements:

```
IF (L) GO TO 100

IF (RAND(X) .GT. 0) GO TO 100
```

The significance of this is that the function RAND is called only when L is false. If evaluation of RAND(X) can have side effects, this may be of consequence. For those applications impacted by this implementation, the solution would be to make the evaluation of RAND(X) unconditional. For example:

```
T = RAND(X)

IF(T.GT. 0 .OR. L) GO TO 100
```

Relational

A relational expression consists of two arithmetic expressions connected by a relational operator. Relational expressions always result in a true or false evaluation. Relational expressions are logical operands and can be used in a logical replacement statement, a logical IF statement, as arguments to functions/subroutines, a Parameter statement, or an output list.

The six relational operation symbols are:

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

Example:

A.GT.B has the value .TRUE. if the quantity A is greater than the quantity B, and value .FALSE. otherwise.

The relational operators must always be preceded and followed by a period. The following are the rules for constructing logical and relational expressions:

1. Figure 2-4 indicates which constants, variables, functions, and arithmetic expressions may be combined by the relational operators to form a relational expression. In Figure 2-4, L indicates a valid combination and N indicates an invalid combination. The relational expression will have the value .TRUE. if the condition expressed by the relational operator is met; otherwise, the relational expression will have the value .FALSE.

.GT.,.GE.,.LT., .LE.,.EQ.,.NE.	I	R	D	C	L	S	T
I	L	L	L	*	N	L	L
R	L	L	L	N	N	N	N
D	L	L	L	N	N	N	N
C	*	N	N	*	N	N	N
L	N	N	N	N	N	N	N
S	L	N	N	N	N	L	N
T	L	N	N	N	N	N	L

Legend

C - Complex
D - Double-Precision
I - Integer
L - Logical
N - Nonvalid
R - Real
S - Character
T - Typeless
* - .EQ.,.NE. only

Figure 2-4. Use of Relational Operators

2. The numeric relationships that determine the true or false evaluation of relational expressions are:

   a. For numeric values having unlike signs, the positive value is considered larger than a negative value, regardless of the respective magnitude. E.g., +3 > -5 and +5 > -5.

   b. For numeric values having like signs, the magnitude of the values determines the relationship. E.g., +3 > +2 and -8 < -4.

3. A logical term may consist of a relational expression, a single logical constant, a logical variable, or a reference to a logical function. A logical expression is a series of logical terms or logical expressions connected by the logical operators .AND.,.OR., and .NOT.

4. The logical operator .NOT. must be followed by a logical or relational expression, and the logical operators .AND. and .OR. must be preceded and followed by logical or relational expressions.

5. Any logical expression may be enclosed in parentheses.

6. In the hierarchy of operations, parentheses may be used in logical, relational, and arithmetic expressions to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows (from innermost operation to outermost operations):

a.	Function Reference	
b.	**,↑or∧	Exponentiation
c.	+ and -	Unary addition and subtraction
d.	* and /	Multiplication and Division
e.	+ and -	Addition and Subtraction
f.	.LT.,.LE.,.EQ.,.NE.,.GT.,.GE.	
g.	.NOT.	
h.	.AND.	
i.	.OR.	

This hierarchy is applied first to the expression within the innermost set of parentheses in the statement; this procedure continues through the outer parentheses until the entire expression has been evaluated.

Typeless

The following functions are considered as typeless:

FLD
AND
OR
XOR
BOOL
COMPL

A typeless result is regarded as a 36-bit string (one computer word). Typeless entities may be combined with integer, logical or other typeless entities. With the arithmetic operators the result is typeless; with relational operators the result is logical; the logical operations may not be used on typeless entities.

Evaluation of Expressions

A part of an expression need be evaluated only if such action is necessary to establish the value of the expression. The rules for formation of expressions imply the binding strength of operators. It should be noted that the range of the subtraction operator is the term that immediately succeeds it.

When two elements are combined by an operator, the order of evaluation of the elements is undefined because of possible reordering during optimization. If mathematical use of operators is associative, commutative, or both, full use of these facts may be made to revise orders of combinations, provided only that integrity of parenthesized expressions is not violated. The value of an integer element is the nearest integer whose magnitude does not exceed the magnitude of the mathematical value represented by that element. The associative and commutative laws do not apply in the evaluation of integer terms containing division, hence the evaluation of such terms must effectively proceed from left to right.

Any use of an array element name requires the evaluation of its subscript. The evaluation of functions appearing in an expression may not validly alter the value of any other element within the expression, assignment statement, or call statement in which a function reference or subscript appears. No factor may be evaluated that requires a negative valued primary to be raised to a real or double precision exponent. No factor may be evaluated that requires raising a zero valued primary to a zero valued exponent. No element may be evaluated whose value is not mathematically defined.

The mode of evaluation of arithmetic expressions is determined by the following order of type dominance:

1. Complex
2. Double Precision
3. Real
4. Typeless
5. Integer

When two primaries are combined by any of the arithmetic operators except the exponentiation operator, their respective types are examined according to the stated order of type dominance. The type of the recessive primary is converted to that of the dominant primary (if necessary) and the operation is performed.

Unary Operators

The unary operators, negative, positive, and logical not, may immediately precede a constant or a variable in an expression; however, if the placement causes the unary negative or positive operator to be adjacent to another operator, it must be enclosed in parentheses with the constant or variable.

Examples:

```
A=+1.6
C=D/(-Z)*W
IF(-3.+T4)1,2,3
L1=R2.GT.(-2.)
L2=.NOT.L1
A=B**(-2)
```

FORTRAN STATEMENTS

Types of FORTRAN Statements

The basic unit of FORTRAN is the statement. Statements are classified according to the following uses:

1. Arithmetic statements specifying numerical, character, or logical value assignment.
2. Control statements governing the order of execution in the object program.

3. Input/Output statements and input/output formats which describe the form of the data.

4. Subprogram statements enabling the programmer to define and use subprograms.

5. Specification statements providing information about variables used in the program, information about storage allocation and data assigned.

6. Compiler control statements direct the compilation activity.

Arithmetic Statements

assignment statements
arithmetic statement functions

Control Statements

ASSIGN
CONTINUE
DO
GOTO
IF
PAUSE
STOP

Input/Output Statements

BACKSPACE
DECODE
ENCODE
END FILE
FORMAT
PRINT
PUNCH
READ
REWIND
WRITE

Subprogram Statements

BLOCK DATA
CALL
ENTRY
FUNCTION
RETURN
SUBROUTINE

Specification Statements

ABNORMAL
COMMON
DATA
DIMENSION
EQUIVALENCE
EXTERNAL
IMPLICIT
NAMELIST
type
    INTEGER
    REAL
    DOUBLE PRECISION
    COMPLEX
    LOGICAL
    CHARACTER
PARAMETER

Compiler Control Statement

END

Index of Statements

Table 2-1 contains an alphabetical listing of FORTRAN statements giving an example with the page number for the statement in Section IV.

Table 2-1. Alphabetical Listing of FORTRAN Statements

Statement	Example	Page
arithmetic statement function	F(X,Y)=(X+1)*Y(I)	4-2
assignment statement	A=4*B-SINE(C**2)	4-2
ABNORMAL	ABNORMAL SINE	4-7
BACKSPACE	BACKSPACE 5	4-8
BLOCK DATA	BLOCK DATA	4-9
CALL	CALL MATMPY (X,5,10,4,Z)	4-10
CHARACTER	CHARACTER ARRAY*14(10,10)	4-12
COMMON	COMMON X,Y,Z	4-13
COMPLEX	COMPLEX T,N1,D1	4-15
CONTINUE	CONTINUE	4-16
DATA	DATA A,B,C /3.5,2.9,6.0/	4-17
DECODE	DECODE (CHARS,95 01) A,I(3),X	4-20
DIMENSION	DIMENSION A(50)	4-21
DO	DO 35 K=10,20,2	4-22
DOUBLE PRECISION	DOUBLE PRECISION DENOM,PREF	4-26
ENCODE	ENCODE (CHARS(6),9001)A,I(3),X	4-27
END	END	4-28
ENDFILE	ENDFILE 5	4-29
ENTRY	ENTRY INVRT (B,C,D)	4-30
EQUIVALENCE	EQUIVALENCE (A,B,C)	4-31
EXTERNAL	EXTERNAL SIN,COS,RESULT	4-34

Table 2-1 (cont). Alphabetical Listing of FORTRAN Statements

Statement	Example	Page
FORMAT	10 FORMAT (E17.2,F20.0)	4-35
FUNCTION	FUNCTION CALC (B,C,D)	4-37
GO TO, assigned	GO TO S4, (3,4,7)	4-40
GO TO, computed	GO TO (3,4,7),K	4-41
GO TO, unconditional	GO TO 20	4-40
IF,arithmetic	IF (A(J,K)-B)10,4,30	4-42
IF,logical	IF (A.GT.B) GO TO 3	4-43
IMPLICIT	IMPLICIT INTEGER (A-F,X,Y)	4-44
INTEGER	INTEGER I,ABC	4-45
LOGICAL	LOGICAL A1,K	4-46
NAMELIST	NAMELIST/LIST/ R,S,T,U,V	4-47
PARAMETER	PARAMETER I=5/2,J=I*3	4-48
PAUSE	PAUSE 1234	4-49
PRINT, list directed	PRINT,A	4-51
PRINT,formatted	PRINT 20,A	4-51
PRINT,namelist	PRINT LIST	4-51
PUNCH,list directed	PUNCH,A	4-52
PUNCH,formatted	PUNCH 20,A	4-52
PUNCH,namelist	PUNCH LIST	4-52
READ,list directed	READ,A	4-53
READ,formatted	READ 20,A	4-53
READ,namelist	READ LIST	4-53
READ,formatted file	READ(5,20,END=90,ERR=95) A	4-53
READ,unformatted file	READ(5,END=90,ERR=95) A	4-54

Table 2-1 (cont). Alphabetical Listing of FORTRAN Statements

Statement	Example	Page
READ,random binary file	READ(8'I)A	4-54
READ,namelist file	READ(5,LIST)	4-54
REAL	REAL J	4-55
RETURN	RETURN	4-56
REWIND	REWIND 5	4-57
STOP	STOP 100	4-58
SUBROUTINE	SUBROUTINE ALPHA (B,C,D)	4-59
type	INTEGER A,B,C,D	4-61
WRITE,formatted file	WRITE(6,30,ERR=S4)A	4-63
WRITE,unformatted file	WRITE(6,ERR=S4)A	4-63
WRITE,namelist file	WRITE(6,LIST)	4-63
WRITE,random binary	WRITE(8'I)A	4-63

SECTION III

USER INTERFACES

Users create programs by entering FORTRAN statements into remote and local peripheral or terminal devices which are connected to a computer operating under GCOS.

The interface between the user and the FORTRAN system consists of the transmission to the user's I/O device of compilation error messages and run-time diagnostics. The messages transmitted are sufficient to locate for the user the line on which the error occurred, and the form of the message is such that the error is explicitly defined.

Three modes of operation are available to the user: local batch, remote batch, and time sharing. The only user differences among the three modes are the I/O device assignments for the system output and input files, the presence of necessary user-GCOS communication via control cards or command language, and the assumed compiler options for the compilation process.

BATCH MODE

In the local batch mode, the system I/O devices are the card reader card punch and line printer. The user communicates directly with GCOS for system services via the GCOS control cards and the usable slave mode instructions. The execution of user programs submitted via the local batch mode is carried out directly under GCOS and the user's program exists under GCOS as a separate batch job. Input processing is performed by System Input and allocation by the GCOS allocator.

The remote batch mode is exactly equivalent to the local batch mode in capability. The only difference is the assignment of the system I/O device to the remote terminal as remote files (not direct access) rather than to the local card reader and local printer/punch.

Batch Call Card

The system call card for Series 6000 FORTRAN in batch mode is:

1	8	16 (operand field)
$	FORTY	OPTIONS
	or	
$	FORTRAN	OPTIONS

Operand Field:

The Operand Field specifies the system options. The following options are available with FORTRAN (Standard Batch options are underlined):

LSTIN - A listing of source input will be prepared by the FORTRAN compiler.

NLSTIN - No listing of the source input will be prepared.

LSTOU - A listing of the compiled object program output will be prepared.

NLSTOU - No listing of the compiled object program output will be prepared.

DECK - A binary object program deck will be prepared as output.

NDECK - No binary object program deck will be prepared.

COMDK - A compressed source deck will be prepared as output.

NCOMDK - No compressed source deck will be prepared as output.

MAP - A storage map of the program labels, variables and constants will be prepared as output.

NOMAP - No storage map will be prepared.

XREF - A cross reference report will be prepared as output. This is the same as the SYMTAB option in FORTRAN IV. Both are acceptable to Series 6000 FORTRAN. A TO-FROM transfer table will also be generated.

NXREF - No cross reference report will be prepared.

DEBUG - A run time debug symbol table (.SYMT.) will be included in the object program. This is the same as the STAB option in FORTRAN IV. Both are acceptable to Series 6000 FORTRAN.

NDEBUG - No debug symbol table will be prepared.

BCD - The object program character set will be standard BCD (see Appendix A).

ASCII - The object program character set will be ASCII (see Appendix A).

FORM - The source program is in standard statement format (see Format Rules for Statements).

NFORM - The source program is "free form".

LNO - The source input records are line numbered beginning in Column 1 and terminating with the first non-numeric character. This option is only operable with the NFORM option.

NLNO - The source records are not line numbered.

OPTZ - Global optimization procedures will be performed so that the object program produced is highly efficient. It should be noted that this option will slow the compilation rate, though not significantly.

NOPTZ - Global optimization of the object program will not be performed.

DUMP - Slave core dump will be given if the compilation activity terminates abnormally.

NDUMP - Program registers upper SSA, and slave program prefix will be dumped if the compilation activity terminates abnormally.

NOTE: Independent of the DUMP/NDUMP option, Series 6000 FORTRAN has built in the capability of producing a symbolic dump of the internal tables in the event of a compiler abort. The presence of a $ SYSOUT *F control card will activate this process.

Sample Batch Deck Setup

The following are the required control cards for the compilation and execution of a batch FORTRAN activity. The $ control cards are fully described in the Control Cards Reference Manual.

```
1       8         16
$       SNUMB
$       IDENT
$       OPTION    FORTRAN
$       FORTY     Options     or $ FORTRAN Options
          .
          .                   FORTRAN Source Deck(s)
          .
$       EXECUTE   Options
$       File Cards
$       ENDJOB
***EOF
```

TIME SHARING SYSTEM OPERATION

From a user point of view there are two time sharing versions of the Series 6000 FORTRAN compiler. Each version is invoked by a different call. These versions and the language call for each are as follows:

Compiler Version	Language Call
Batch based time sharing compiler	YFORTRAN
Time sharing based compiler	FORTRAN

In this document, the batch based time sharing system is referred to as the YFORTRAN Time Sharing System and the time sharing based system is referred to as the FORTRAN Time Sharing System. The time sharing based Series 6000 FORTRAN compiler compiles under the time sharing system (rather than being spawned as in the case of the batch based time sharing compiler) and differs from the batch based time sharing compiler in the following areas.

1. Compiles under the GCOS Time Sharing System.

2. Eliminates the need for configuring batch core for YFORTRAN compiles through DRL TASK.

3. Retains essentially the current RUN syntax with modifications as noted in this section.

4. Interfaces with a new 4K Time Sharing loader module.

5. Significant overhead reduction in FORTRAN time sharing system.

6. Blank common allocation is common.

7. "CORE=" clause is not required for compiles.

8. Compilers are identical except for the executive phase (YEXC vs YTEX).

The only user differences, other than the ones noted above, are the I/O device assignments for the system output and input files, the presence of necessary user GCOS communication via control cards or command language, and the assumed compiler options for the compilation process.

Time Sharing System Command Language

The standard means of communication with the Series 600/6000 GCOS Time Sharing System (TSS) is by way of a teletypewriter used as a remote terminal. Other compatible devices may also be used, but use of a teletypewriter is assumed in this manual. The user may choose either the keyboard/printer or paper-tape teletypewriter unit for input/output, or combine both. In either case, the information transmitted to and from the system is displayed on the terminal-printer. Keyboard input will be used for purposes of description; instructions for the use of paper tape are given under "Paper Tape Input" in this section.

The user "controls" the time sharing system primarily by means of a command language, a language distinct from any of the specialized programming languages that are recognized by the individual time sharing compilers/processors (e.g., the Time Sharing FORTRAN language). The command language is, for the most part, the same for users of any component of the time sharing system; i.e., FORTRAN, BASIC, Text Editor, etc. A few of the commands pertain to only one or another of the component time sharing systems, but the majority of them are, in form and meaning, common to all component systems.

The commands relate to the generation, modification, and disposition of program and data files, and program compilation/execution requests. The complete time sharing command language is described in Time Sharing General Information Manual.

Once communication with the system has been established, any question or request from the system must be answered within ten minutes, except for the initial requests for user identification (user-ID) and sign-on password, which must be given within one minute. If these time limits are exceeded, the user's terminal will be disconnected.

Time Sharing Commands of the YFORTRAN and FORTRAN Time Sharing Systems

The valid time sharing system commands are listed in Table 3-1. These commands are fully described in the Time Sharing General Information Manual. The RUN command for the YFORTRAN and FORTRAN Time Sharing Systems is more fully described in this manual.

Table 3-1. YFORTRAN and FORTRAN Time Sharing Systems Commands

Command	Applicable At Build Mode
ABC	Yes
ACCESS	Yes
ASCASC	Yes
ASCBCD	Yes
AUTOMATIC [a]	Yes
BCDASC	Yes
BPRINT	Yes
BPUNCH	Yes
BYE	Yes
CATALOG	Yes
DELETE [a]	Yes
DONE	Yes
EDIT	Yes
ERASE [a]	Yes
FDUMP	Yes
GET	Yes
HELP	Yes
HOLD	Yes
JABT	Yes
JOUT	Yes
JSTS	Yes
LENGTH	Yes
LIB [a]	Yes
LIST	Yes
NEW [a]	Yes
NEWUSER	Yes
NO PARITY	Yes
OLD [a]	Yes
PARITY	Yes
PERM [a]	Yes
PRINT [a]	Yes
PURGE [a]	Yes
RECOVER	Yes
#RECOVER	No
RELEASE [a]	Yes
REMOVE	Yes
RESAVE [a]	Yes
RESEQUENCE [a]	Yes
ROLLBACK	Yes
#ROLLBACK	No
RUN [a]	Yes
SAVE [a]	Yes
SCAN	Yes
SEND	Yes
STATUS	Yes
SYSTEM [a]	Yes
TAPE [a]	Yes

[a] not applicable at subsystem-selection level

Log-On Procedure

The user, to initiate communication with the GCOS Time Sharing System, performs the following steps:

- Turns on the terminal unit
- Obtains a dial-tone
- Dials one of the numbers of his time sharing center

The user will then receive either a high-pitched tone indicating that his terminal has been connected to the computer or a busy signal. The busy signal indicates, of course, that no free line is presently available.

Once the user's terminal has been connected to the computer, the time sharing system begins the log-on procedure by transmitting the following message:

```
HIS SERIES 6000, SERIES 600 ON(date)AT(time)CHANNEL(nnnn)
```

where time is given in hours and thousandths of hours (hh.hhh), and nnnn is the user's line number.

Following this message, the system asks for the user's identification:

```
USER ID --
```

The user responds, on the same line, with the user-ID that has been assigned to him by the time sharing installation management. This user-ID uniquely identifies a particular user already known to the system, for the purposes of locating his programs and files and accounting for his usage of the time sharing resources allocated to him. An example request and response might be:

```
USER ID -- J.P.JONES
```

Note: A carriage return must be given following any complete response, command, or line of information typed by the user.

(The user's response is underlined here for illustration.) After the user responds with his user-ID, the system asks for the sign-on password that was assigned to him along with his user-ID, as follows:

```
PASSWORD
```

The user types his password directly on the "strikeover" mask provided below the request PASSWORD. The password is used by the system as a check on the legitimacy of the named user. The "strikeover" mask insures that the password, when typed, cannot be read by another person. (In the event that either the user-ID or password is twice given incorrectly, the user's terminal is immediately disconnected from the system.) At this point, if the accumulated charges for the user's past time sharing usage equals or slightly exceeds 100 per cent of his current resource allocation, he will receive a warning message. If his accumulated charges exceeds 110 per cent of his current resources, he receives the message:

RESOURCES EXHAUSTED - CANNOT ACCEPT YOU

and his terminal is immediately disconnected. (The user may also receive the following information message if his situation warrants it:

n BLOCKS FILE SPACE AVAILABLE

This condition does not affect the log-on procedure.)

Assuming that the user has responded with a legitimate user-ID and password and has not over extended his resources, the time sharing system then asks the user to select the processing system that he wants to work with; this is called the system-selection request. In this case, the user would respond with YFORTRAN or FORTRAN:

SYSTEM ? YFORTRAN or FORTRAN

The user is then asked whether he now wants to enter a new program (NEW) or if he wants to retrieve and work with a previously entered and saved program (OLD); the request message is:

OLD OR NEW -

If the user wishes to start a new program (i.e., build a new source file), he responds simply with:

NEW

If, on the other hand, he wants to recall an old source-program file, he responds with:

OLD filename

where filename is the name of the file on which the old program was saved during a previous session at the terminal (see the SAVE command).

Following either response, the system types the message READY, returns the carriage, and prints an asterisk in the first character position of the next line:

```
READY
*
```

An example of a complete log-on procedure, up to the point where the YFORTRAN or FORTRAN system is ready to accept program input or control commands, might be as follows:

```
HIS SERIES 6000, SERIES 600 ON 07/26/68 AT 14.768 CHANNEL 0012

USER ID - J.P.JONES
PASSWORD
XXXXXXXXXX      - (user's password is typed over the mask)
SYSTEM-YFORTRAN or FORTRAN
OLD OR NEW - NEW - (NEW is shown arbitrarily for illustration)
READY
*               - (the user begins entering input on this line)
```

Entering Program-Statement Input

After the message:

```
READY
*
```

the system is in build-mode (as indicated by the initial asterisk) and is ready to accept FORTRAN program-statement input or control commands. All lines of input other than control commands are accumulated on the user's current file. Normally the current file will be the file that contains the program he wants to compile and run at this session. If he is building a new file (NEW response to OLD OR NEW--), his current file will initially be empty. If he has recalled an old file (OLD filename) the content of the named old file will initially be on his current file, and any input typed by the user -- excepting control commands -- will be either added to, merged into, or will replace lines in the current file, depending upon the relative line numbering of the lines in the file and the new input. (This process is explained under the heading "Correcting or Modifying a Program," below.)

Following each line of noncommand-language input and the terminating carriage response, the system will supply another initial asterisk, indicating that it is ready to accept more input.

Format of Program-Statement Input

A line of FORTRAN input -- as distinct from a control command -- can contain one of the following:

1. One or more FORTRAN statements.
2. A partial statement.
3. A continuation of a statement left incomplete in the preceding line of input.
4. A comment.
5. A combination of (3) and (1) or (2), in that order.
6. A combination of (1) and (2).

A line of input must begin with a line-sequence number of from one to eight numeric characters. The line-sequence number facilitates correction and modification of the source program (described below); hereinafter, the line-sequence number will be referred to simply as the "line number". (Note that a line number is distinct from a statement number; a statement number is a part of the FORTRAN-language statement itself.)

The line number is always terminated with (i.e., immediately followed by) a single control character which may be a blank, an ampersand, a number sign, an asterisk, or the letter C. The control character merely serves to indicate what type information is to follow (new statement, continuation, or comment) and is not compiled as part of the program.

The semicolon may be used to indicate the end of one complete FORTRAN statement and the beginning of another on the same line of input. A carriage return must, of course, be used to terminate a complete line of input.

This line format is suitable for direct processing by the FORTRAN compiler with the options NFORM and LNO.

The general format of a line of FORTRAN input is then as follows:

```
nnnnnnnncstatement or continuation ;statement...;statement

                        or

    nnnnnnnnc comment
```

where: nnn...n is a numeric line number, the magnitude of which is less than 2^{18}(262144), and

c is a single-character control character which may be a blank, an ampersand, an asterisk, a number sign, or the letter C, and must immediately follow the last digit of the line number.

SIGNIFICANCE OF THE CONTROL CHARACTER

The control character identifies the type of information that follows it.

Ø (blank) -- if the character position immediately following the last digit of the line number contains a blank, and the next nonblank character is not an ampersand, then the next nonblank character is assumed to begin a new FORTRAN statement. In this case, the next nonblank character may begin a FORTRAN statement number (i.e., mm...m statement-text).

& (ampersand) -- if an ampersand is the first nonblank character following the line number, the next nonblank character is assumed to be a continuation of the previous statement in the previous line of input. The effect of "&" is to suppress the previous carriage return as an end-of-statement indicator.

* (asterisk) or C -- if the line number is terminated with an asterisk or the letter C, the information following is assumed to be a comment. The comment itself is terminated by a carriage return.

(pound sign) -- if the user wants a numeric in column 1 of the card image and line numbers exist in the source file, a pound sign (#) character immediately following the line number will cause the character following it to go into column 1.

A semicolon within a noncomment line indicates both the end of the preceding statement and that any significant information (nonblank, noncarriage return) following it begins a new statement. The new statement may include a FORTRAN statement number, mm...m.

The format of a statement, as typed in following a blank control character, is:

...nnb̲ b̲...b̲ mm...m FORTRAN-language text

(The statement-format portion is underlined.)

where: b̲...b̲ are optional blanks, and

mm...m is an optional numeric statement number which must be equal to or less than 99999

BLANKS (OR SPACING) WITHIN A LINE OF INPUT

Initial, imbedded, or trailing blanks in a line of input have no significance in its interpretation, excepting only that blanks are illegal within the line number and that the nonnumeric character (including b) immediately following the line number is interpreted as a control character. Thus, spacing can be used quite freely within a line of input in the interest of legibility. (Blanks within character constants and nH fields -- i.e., alphanumeric information -- are meaningful however, and are retained in the object program coding.)

Note that the line/statement format is, except for the relative position of the control character, completely free-form, or position independent.

To this point the discussion of line format has been oriented to the NFORM form described earlier in this discussion. This is generally the most convenient form to use in time sharing. It is not mandatory, however. The source file may be built using the text editor and be without line numbers. The NLNO option permits this. Or, the source may be in "fixed" format (without line numbers). The FORM option may be used to indicate this. The full spectrum of line formats and source file recording modes is available to the time sharing user.

Correcting or Modifying a Program

Keyboard input is sent to the computer and written onto the user's current file in units of complete lines. A line of terminal input is terminated by a carriage return and no part of the line is transmitted to the system until that carriage return is given. Therefore, corrections or modifications can be done at the terminal at two distinct levels:

1. Correction of a line-in-progress (i.e., a partial line not yet terminated).

2. Correction or Modification of the program (i.e., the contents of the user's current source file) by the replacement or deletion of lines contained therein, or the insertion of new lines.

The correction of a typing error that is detected by the user before the line is terminated can be done in one of two ways. He may delete one or more characters from the end of the partial line or he may cancel the incomplete line and start over. The rules are as follows:

1. Use of the commercial "at" character (@) deletes from the line the character preceding the @ character; use of n consecutive @ characters deletes the n preceding characters (including blanks).

   Examples:

   *ABCDF@E would result in ABCDE being transmitted to the program file.

   *ABCØDEF@@@@GHJ would result in ABCGHJ being transmitted. (The characters to be deleted are underlined for illustration.)

2. Use of the CTRL (control) and X keys, depressed simultaneously, causes all of the line to be deleted. The characters DEL are printed to indicate deletion and the carriage is automatically returned. For example:

   *ACDEFG CTRL/X DEL (all characters deleted)

   (ready for new input)

Correction or modification of the current source file is done on the basis of line numbers and proceeds according to the following rules:

1. Replacement. A numbered line will replace any identically numbered line that was previously typed or contained on the current file (i.e., the last entered line numbered nnn will be the only line numbered nnn in the file).

2. Deletion. A "line" consisting of only a line number (i.e., nnn) will cause the deletion of any identically numbered line that was previously typed or contained on the current file.

3. Insertion. A line with a line-number value that falls between the line-number values of two pre-existing lines will be inserted in the file between those two lines. If the line number is less than the first line number it is inserted at the beginning of the file; if greater than the largest line number, it is inserted at the end of the file.

At any point in the process of entering program-statement input, the LIST command may be given, which results in a "clean", up-to-date copy of the current file being printed. In this way, the results of any previous corrections or modifications can be verified visually. Following the response (or command) OLD filename, the LIST command can be used initially to inspect the contents of the current source file (i.e., the "old" program).

Input Error Recovery

The decimal input/output routine permits the time sharing user (BCD or ASCII) to correct a string of characters input from a teletypewriter when a character is illegal for the current format conversion. For example: a decimal point is illegal in an "I" field. The current input line is output with a pointer to the illegal character. The user can now input a correction to replace the corresponding characters previously input. The input/output routine resumes with the new string. If the user responds with a carriage return, the usual error message is output.

The YFORTRAN Time Sharing System Run Command

The RUN command has the form:

RUN H -nnn fs = fh; fc(opt) ulib # fe

where: RUN H is the command RUN or RUNH. The latter form is used to display a heading line on the terminal giving date, time, and SNUMB.

-nnn nnn is the maximum time in seconds of processor time, that the program is to be allowed for execution.

fs is the set of file descriptors for source files in the standard BCD card image format, in compressed card image format (COMDK), or in time sharing ASCII format and/or descriptors for binary card image object files. These files serve as inputs to the compiler and/or loader. Where a BCD or COMDK source file is supplied, fs may also include a descriptor for an alter file in BCD format. The alter file must begin with a $ UPDATE card and must be in alter number sequence. If there are many BCD or COMDK source files in the list, the alter file will update the first. Alternatively, the list fs may consist of a single file descriptor that points to a previously generated system loadable (H*) file.

A file descriptor consisting of the single character * indicates the current file (*SRC). The list is optional and, when missing, indicates that only the current file (*SRC) is to be compiled.

fh is a single file descriptor of a random file into which the system loadable file produced by General Loader will be saved if the compilation is successful. This file will be written if no fatal errors occur during compilation. If the named file does not exist, a permanent random file of 36 blocks will be created and added to the user's catalog. If the field is missing, the H* file is generated into a temporary file. The presence of this option is valid only when the program indicated by the list fs, the FORTRAN library, and the user library (if any) is bindable (no outstanding SYMREFS). If General Loader indicates that outstanding SYMREF's exist, an executable H* file will be created, but any reference to an unsatisfied SYMREF will cause the program execution to be abnormally terminated (General Loader inserts a MME GEBORT at references to unsatisfied SYMREF's. When a MME is encountered during the execution of a time sharing subsystem, GCOS and the Time Sharing Executive simulate an illegal operation fault.)

fc is a single file descriptor of a sequential file into which the compiler is to place the binary (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs. If the named file does not exist, a permanent linked file of 3 blocks will be created and added to the user's catalog. This file will expand as necessary to hold the object decks. In this case the field fs plus the libraries need not indicate a complete program (individual or collections of SUBROUTINES may be compiled and saved). When this optional field is missing, a C* file will not be generated. When present the DECK option is turned on for the compilation process.

opt is a list of options. Some of these options affect the compilation process, and some the loading. The following compiler options are available for time sharing; they are described under the $ FORTY card; underlined is default.

DEBUG - Generate run-time debug symbol table

NDEBUG - No run-time debug symbol table is generated

BCD - Object character set is BCD. If applicable, this option must be specified whenever General Loader is to be called. This is required for compile, compile and load, and load activities; it is not required for execute only runs (run H* file).

ASCII - Object character set is ASCII

FORM - Source is in "fixed" format

NFORM - Source is in "free" format

LNO - Source is line numbered

NLNO - Source is not line numbered

OPTZ - Optimize the object module

NOPTZ - Optimization of the object module will not be done

The remaining options have to do with the loading process. The underlined option is the default case.

GO - The program will be executed at the completion of compilation.

NOGO - The program will not be executed at the completion of the compilation. If specified, the object program will be saved. If no object (H*) save file is specified, only the compilation will be performed (General Loader will not be called).

ULIB - File descriptors exist following the end of the options field which locate user libraries which are to be searched for missing routines prior to searching for them in the system library.

NOLIB - No user libraries are to be used.

TIME=nnn The batch compilation and/or General Loader activity time limits will be set to nnn seconds; where nnn $\leq$ 180. If not specified, nnn is set to 60.

CORE=nn The batch compilation activity core requirement will be set to nnK+6K or 24K, whichever is larger. If not specified, nn is set to 16.

URGC=nn - The urgency for the batch compilation and/or General Loader activity will be set to nn, where nn $\leq$ 40. If not specified, nn is set to 40.

TEST - A test version of the compiler and/or General Loader is to be used for the batch activity. There must be an accessed file (in the AFT) of the name FORTRANY. If these two conditions are met, then file FORTRANY will be allocated as file code ** in the batch activity.

REMO - Temporary files created for the batch process will be removed from the AFT as they are no longer needed. This option keeps the number of files in the AFT down to a minimum but causes more time to be spent processing each RUN command.

NAME=name- Provides a name for the main link of the saved H* file. May be used both at time of creation of this file and subsequently as it is reused. This name is placed in the SAVE/field of the $ OPTION card.

ulib is a sequence of file descriptors pointing to random files containing user libraries to be searched before the system library.

fe is a set of file descriptors for files which will be required during execution. Each catalog/file description is separated by a semicolon (see Time Sharing Command Language and File Usage in the Time Sharing General Information Manual). The file description may be in any of the following formats.

1. filename specifying a filename in the form nn where 01 ≤ nn ≤ 44 and nn represents a logical file code referenced by the I/O statements in the program.

2. filedescr specifying a full description.

   a. filename

   b. filename$password

   c. userid/catalog$password...

The FORTRAN Time Sharing System RUN Command

This command has the following form:

```
RUNH - nnn fs = fh; fc (opt) ulib #fe
```

RUNH - is the command RUN or RUNH. RUNH is used to display a heading line on the terminal giving date and time.

-nnn - is the maximum time in seconds of processor time the compiled object program is allowed to execute.

fs - is the set of file descriptions for source files in the Time Sharing ASCII format, in the standard BCD format, in COMDK form, and/or descriptors for binary card image object files. These files serve as inputs to the compiler and/or time sharing loader. When a BCD or COMDK source file is supplied, fs may also include a descriptor for an alter file in BCD format. The $ ALTER file must begin with a $ UPDATE card and must be in alter number sequence. If there is more than one BCD or COMDK source file in the list, the alter file will update the first. The list fs may also consist of a single file descriptor that points to a previously generated system loadable (H*) file.

A file descriptor consisting of the single character * indicates the current file (*SRC). The fs list is optional and, when missing, indicates that only the current file (*SRC) is to be compiled.

fh - is a single file descriptor of a random file into which the system loadable file (H*) produced by the time sharing loader is saved providing the compilation is successful. If the named permanent file does not exits, a permanent (quick access) random file of 36 llinks is created and added to the users' catalog. If the field is missing, no temporary H* file is created, instead the time sharing loader creates a complete bound core-image of the object execution program, "releases" itself via DRL RELMEn, and enters the execution directly.

If the time sharing loader indicates outstanding SYMREF's exist, any reference to them during the object execution will cause abnormal termination via a DRL ABORT.

fc - is a single file descriptor of a sequential file into which the compiler is to place the binary object (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs.

If the named file does not exist, a quick access permanent file of 3 llinks is created. This file will expand as necessary up to a maximum of 20 llinks to hold the object deck(s). When C* is specified, a compiler temporary file (*1 scratch) file of 48 llinks is defined and it's name is placed into the AFT.

opt - is a list of compiler/loader options available in the time sharing based Series 6000 FORTRAN system. Those options available in the batched based Series 6000 FORTRAN system but not specified here are not currently used in the time sharing based Series 6000 FORTRAN system. They are ignored if specified. Default options are underlined.

The options are as follows:

BCD - The internal character set for the object execution is BCD. If applicable, this option must be specified whenever the time sharing loader is called. This is required for compile, compile and load, and load activities; it is not required (or interpreted) for execute only runs (from H* save file). The user should not load object deck files compiled under different options; i.e., one under BCD and another under ASCII, as execution time results are very unpredictable.

ASCII - Internal character set of the object execution is ASCII.

FORM - Source is in "fixed" format.

NFORM - Source is in "free" format.

LNO - Source is line numbered.

NLNO - Source is not line numbered.

OPTZ - Optimizer phase is called.

NOPTZ - Optimizer phase is not called.

The following remaining options have to do with the load process.

GO - The program will be executed at the successful completion of the compile-load process.

NOGO - The program will not be executed at the completion of the compilation. If specified, the object program will be saved. If no object (H*) save file is specified, only the compilation will be performed.

ULIB - File descriptors exist following the end of the options field which allocate user libraries which are to be searched for missing routines prior to searching the system library. Up to nine user library files may be specified separated by semi-colons.

NOLIB - No user libraries are searched. Specification of user libraries in this case will cause a RUN diagnostic.

CORE = nn where nn is additional core (mod 1024) to be added to the standard time sharing loader allocation of 22K. This should be done if the message "<F> PROGRAM EXCEEDS STORE SIZE" appears. The compiler will attempt to "second guess" the space requirements for the load process by accumulating the size of the generated code, .DATA. region, labeled common and blank common for each subprogram compiled; then adding a constant (11K for the standard library) to this to arrive at the size of a load space requirement. If the message "NOT ENOUGH CORE TO RUN JOB" appears, TSS allocation is too small to compile/load this program.

ulib - is a list of file descriptors (separated by semi-colon) pointing to file(s) containing subprograms that have SYMDEF symbols which satisfy the undefined SYMREF's in the load table. The user library or libraries are searched in the order in which they are encountered and before the system subroutine library. The user may create his own library files using UTILITY, RANLIB, and the Object File Editor.

The user library file must be a random permanent file when creating a user library file through the batch procedure.

fe - is a set of file descriptors for files which will be required during execution. Each catalog/file description is separated by a semicolon (see Time Sharing Command Language and File Usage in the Time Sharing General Information Manual). The file description may be in any of the following formats:

1. filename specifying a filename in the form nn where $01 \leq nn \leq 44$ and nn represents a logical file code referenced by the I/O statements in the program.

2. file descr specifying a full description

   a. filename
   b. filename $ password
   c. userid/catalog $ password

Example:

1. Create a random file to contain the user's library with the ACCESS subsystem. ACCESS CF,/ULIB1,B/50,50/,R,MODE/R/

2. Deck setup for creation and saving a user library file (through CARDIN or batch).

```
1       8           16
$       IDENT
$       USERID      UMC$PASSWD
$       UTILITY
$       LIMIT       ,9K
$       FILE        AA,F1S,10L
$       PRMFL       A1,R,S,UMC/OBJDECK1
$       PRMFL       A2,R,S,UMC/OBJDECK2
$       PRMFL       A3,R,S,UMC/OBJDECK3
$       FUTIL       A1,AA,MCOPY/1F,HOLD/AA/
$       FUTIL       A2,AA,MCOPY/1F/,HOLD/AA/
$       FUTIL       A3,AA,COPY/1F/,REW/AA/
$       FILEDIT     NOSOURCE,OBJECT,INITIALIZE
$       FILE        R*,J1C,10L
$       FILE        *C,F1R,10L
$       PROGRAM     RANLIB
$       PRMFL       A4,R/W,R,UMC/ULIB1
$       FILE        R*,J1R,10L
$       ENDJOB
```

Link Overlays Under Time Sharing

The subroutine FTLK may be invoked to create a time sharing loadable-executable link/overlay. The following example illustrates the batch activity to build the link/overlay H* perm-file.

```
1               8                   16
$               IDENT
$               USERID
$               LOWLOAD             36
$               USE                 .GTLIT,.TSGF.,.FTLK,.FTSU.
$               OPTION              FORTRAN,NOGO
$               FORTY               NFORM,NLNO
  CALL          LINK ("X1")
  CALL          LINK ("X2")
  CALL          LINK ("X3")
  PRINT,        "ALL  DONE          ";STOP; END
$               ENTRY               ......
$               LINK                X1
$               FORTY               NFORM,NLNO
  SUBROUTINE        A
  PRINT, "LINK      X1 EXECUTING"   ;RETURN;END
$               ENTRY               A
$               LINK                X2, X1
$               FORTY               NFORM,NLNO
  SUBROUTINE        B
  PRINT, "LINK      X2 EXECUTING"   ; RETURN;END
$               ENTRY               B
$               LINK                X3,X2
$               FORTY               NFORM,NLNO
  SUBROUTINE        C
  PRINT, "LINK      X3 EXECUTING"   ; RETURN;END
$               ENTRY               C
$               EXECUTE
$               PRMFL               H*,R/W,R,UMC/HSTAR
$               ENDJOB
```

The RUN subsystem must be used to load the main link from the saved H* perm-file. Use of the time sharing pre-execution initialization .SETU. routine is required to assure that the ASCII name of the accessed H* perm-file is retained for use by FTLK. FTLK will pick up this name for use as a parameter to DRL RESTOR.

COMMON INFORMATION FOR THE FORTRAN AND YFORTRAN TIME SHARING SYSTEMS

The user will, most commonly, apply the alternate name specified in the following format.

filedescr "altname" where altname = nn; attaching the logical file code nn to the specified file.

File codes 05, 06, 41, 42, and 43 are implicitly defined for teletypewriter directed I/O and need not be mentioned in the RUN command unless I/O is to be directed to a file. Other logical file codes may be teletypewriter directed by specifying a descriptor of the form "nn". For example:

```
RUN#"10"
```

If a given file descriptor consists of only an unquoted 2-digit logical file code, a temporary file will be created for the user unless a quick-access permanent file with the same name already exists. The PERM command can subsequently be used to make the temporary file permanent. Alternatively, such temporary files can be made permanent at the time the user logs off. For example:

```
RUN PROGRAM#10
```

If no file exists in the user's catalog with the name 10, a linked temporary file will be created with that name and I/O directed to the logical file code 10 will be routed to the temporary file.

The fe list of the RUN command serves two additional functions: creation of a file control block and association of the logical file code with some specific file or the teletypewriter. When this association involves a catalog file descriptor, that file is accessed (or created if so indicated) and added to the user's available file table (AFT). The file is then said to be allocated to the process. This is analogous to the allocation by the $ PRMFL and $ FILE control cards in a batch operation.

When a file is first referenced by an executing program, a general file "open" function is invoked. At this time, the file control block comes into play. There are three possibilities.

1. There is no file control block for the referenced file.
2. The file control block indicates that the teletypewriter is to be used.
3. The file control block indicates that a file is to be used.

If there is no file control block, one is automatically generated indicating that a file is to be used. When the file control block indicates that the teletypewriter is to be used, the device attachment is completed and I/O proceeds. When the file control block indicates that a file is to be used (cases 1 and 3), the AFT is searched. If a match is found (some allocated file has a 2-digit file code/name equivalent to the file description in the I/O statement), attachment is made to that file and I/O proceeds. If no match is found (there has been no file allocation for the current file designator), a comment is displayed on the teletypewriter identifying the undefined file designator as follows:

```
FILE XX NOT IN AFT. ACCESS CALLED -
```

XX is the 2-digit file designator being referenced by the running program. At this point the ACCESS subsystem is called (as indicated by the above message) and the following is displayed (by ACCESS):

```
FUNCTION?
```

Commands may now be given to ACCESS. When the dialogue is finished, ACCESS will return to the user's program. The "open" routine will then make a fresh search of the AFT. If a match is now found (indicating the user accessed some file), attachment is made to that file and I/O proceeds. If a match is not found, the file control block is changed to indicate attachment to the teletypewriter and I/O proceeds. For example, consider that PROGRAM contains I/O statements with a file designator of 10 and the following dialogue has transpired:

```
SYSTEM? YFORTRAN or FORTRAN
OLD OR NEW -- OLD PROGRAM
READY
*RUN

FILE 10 NOT IN AFT. ACCESS CALLED

FUNCTION?
```

If the user responds with a carriage return, the teletypewriter will be used for file 10. If the user responds:

```
AF,/MYFILE"10",R,W
```

The ACCESS subsystem will access the file MYFILE of the user's master catalog under the alternate name 10 with read and write permissions. ACCESS then repeats the query "FUNCTION?". If the user now responds with a carriage return, I/O for file 10 will be directed to MYFILE.

One additional option exists for the purpose of collecting the results of a compiler abort. If at the time the RUN command is issued there exists a file in the AFT of name ABRT, that file will be allocated to the compilation activity as file code *F. In the event of a compiler abort, a core dump and symbolic display of the internal tables will be written to this file in a form suitable for printing.

Specify RUN Command as First Line of Source File

A user can include the RUN command as the first line or lines of his source file subject to the following restrictions:

1. This feature is available on time sharing ASCII files only.
2. The line may be in the current file (*SRC) or a referenced perm-file; however, it must begin with the first line of the first source file.
3. The first two characters following the line number must be *#, intervening blanks are not permitted.
4. Multiple *# lines may appear in a source file, provided the total number of characters does not exceed 240 (3 80-character lines).
5. The lines must conform with the RUN syntax continuation as documented in this manual.
6. The line(s) are treated as comment line(s) by the Series 6000 FORTRAN compiler.
7. The user can override the first-line contained RUN command by indicating save files, options, or concatenation on his RUN type-in.

The following examples illustrate this capability.

```
*SYSTEM? YFORTRAN or FORTRAN
 OLD OR NEW?  NEW
*010*#RUN    *(20,30)=HSTAR(BCD,NOGO)
*020  PRINT, "HELLO DOLLY..."
*030  STOP; END
*RUN INVOKES FIRST LINE SYNTAX
```

Run Examples

1. RUN

   The current *SRC FORTRAN source file will be compiled and executed.

2. RUNH-20 FR001=HSTAR; CSTAR1 (ULIB) ABC; XYZ #

   INPUT "01" ; OUTPUT "02"

   FORTRAN program file FR001 is to be compiled and executed. The H* will be saved on file HSTAR and C* on file CSTAR1. For the execution, the random user libraries ABC and XYZ will be scanned for outstanding SYMREFs in FR001. Logical file codes 01 and 02 have been used as alternate names for the quick-access permanent files INPUT and OUTPUT. A heading line for date, time, and SNUMB will be displayed and the object program will be limited to 20 seconds of execution time.

3. RUN #"10"

   The current *SRC file will be compiled and executed and I/O through logical file code 10 will be directed to/from the teletypewriter.

4. RUN BCDIOM = ; CSTAR2 (BCD,NOGO)

   FORTRAN file BCDIOM will be compiled and the object deck will be saved on file CSTAR2. The user intends to execute the object file in the BCD mode.

5. RUN HSTAR #02

   Execute a previously bound and saved H* file. The quick-access file "02" is accessed by the RUN subsystem. If no such file exists a temporary is created.

6. RUN = HSTAR (TIME=60, CORE=18, URGC=10, ULIB,

   COMMON=50) SEARCH

   Compile and execute the CURRENT *SRC file, saving the bound H* on random file HSTAR. Limit the compile time to 60 seconds, increase the General Loader limits to 18,000 words, and enter the batch compile activity with an urgency of 10 (default urgency = 40). The random user library "SEARCH" is searched by the General Loader to satisfy outstanding SYMREFs prior to searching the standard system library. The RUN subsystem causes a $ LOWLOAD 86 (50 + 36) to be placed on the R* file from the COMMON = option.

7. RUN *; CSTAR1; CSTAR2

   Compile and execute the current *SRC file and bind it with two previously saved C* files: CSTAR1 and CSTAR2.

Additional examples are given in Section V under File Designation.

Batch Activity Spawned by the YFORTRAN Time Sharing System RUN Command

As an example of the simplest case, consider that some source file is current in *SRC, and a RUN command is typed with none of the optional fields. A job setup comparable to the following will be dispatched to the batch system.

```
$       SNUMB       nnnnT,40
$       USERID
$       IDENT
$       LOWLOAD
$       USE         .GRBG./36/36
$       OPTION      NOFCB
$       OPTION      NOGO,NOSREF,NOMAP,SAVE/OBJECT
$       USE         .GTLIT,.TSGF.,.FTSU.,.FXEMA
$       FORTY       NLSTIN,NFORM,ASCII
$       LIMITS      2,25K
$       FILE        S*,X1R          source file *SRC
$       FILE        P*,X2S          diagnostic report only
$       EXECUTE
$       FILE        P*,X2R
$       FILE        H*,X3R ,3R      bound program
$       ENDJOB
```

The results of compilation and loading are returned on files P* and H*. P* is read and scanned for compiler and/or loader diagnostics. These are displayed on the teletypewriter and if there have been no fatal errors, the fully bound program will be loaded from H* and execution will proceed.

Example of a Time Sharing Session

A comprehensive example of program creation, testing, correction and modification follows. Replies to the user from the system are underlined here; in actual use, no underlining is done. Explanations are enclosed in parentheses; they are not part of the printout.

```
USER ID - J.P.JONES
PASSWORD--
XKKHKKKHXXKKX
SYSTEM?YFORTRAN or FORTRAN
ØLD ØR NEW-NEW
READY
*AUTØX - (enter automatic-line-number mode)
*0010     READ,A,B,C
*0020     X1=A*B/C
*0030     X2=A**2;B**2
*0040     ANS=X2/X1
*0050     PRINT 10,X1,X2, ASN@@@ANS - (typing error correction)
*0060 10  FØRMAT(1X,"X1=",F6.S@2,"X2=",F7.2,"ANS=",
*0070&    F6.2)
*0080     STØP
*0090     END
*0100     (end automatic mode by carriage return)
*0030     X2=A**2+B**2-C - (replacement of line 30)
*SAVE FØRT01
DATA SAVED--FØRT01

*LIST     - (display corrected program)
0010      READ,A,B,C
0020      X1=A*B/C
0030      X2=A**2+B**2-C
0040      ANS=X2/X1
0050      PRINT 10,X1,X2, ANS
0060 10   FØRMAT(1X,"X1=",F6.2,"X2=",F7.2,"ANS=",
0070&     F6.2)
0080      STØP
0090      END

READY
*RUN     (run program)

= 3.2,10.5,2.2 - (type input data)
X1= 15.27X2= 118.29ANS= 7.75 - (output - correct,
                                but poor format)

*0060 10 FØRMAT(1X,"X1="   ,F6.2," X2=",F7.2,"  ANS=", -
                                (correct format statement)
*RUN

= 3.2,10.5,2.2
X1= 15.27  X2= 118.29  ANS= 7.75 - (improved output format)
*RESAVE FØRT01
DATA SAVED--FØRT01 - (corrected version of program saved)

*BYE - (finished)
**RESØURCES USED $  2.08, USED TØ DATE $  263.85= 27%
**TIME SHARING ØFF AT  15.421 ØN  10/10/68
```

Supplying Direct-Mode Program Input

During program execution, keyboard input may need to be supplied to satisfy one or more READ statements in the program. Each time input is required, the equal-sign character, "=", will be printed at the terminal. The user begins typing the input immediately following the equal sign.

It is also possible to input data from a paper tape. The actual characters transmitted to the terminal from a READ statement are: carriage return (CR), line feed (RO), equal sign (=), and sign-on (X-ON). The sign-on character activates the paper tape reader if the reader is in the ready state. A ready state is achieved by having the paper tape "loaded" and the reader switches set to "AUTO" or "TD-ON". Paper tapes which are to be used in this way should end each line with the characters: carriage return (CR), sign off (X-OFF), line feed (LF), rubout (RO). The sign-off character turns off the reader but leaves it in a ready state for any subsequent READ's.

Teletypewriter output from the PUNCH statement automatically appends this control information to the end of each line, facilitating preparation of tapes. In any event the user must manually begin such tapes with an appropriate leader of RO characters.

Emergency Termination of Execution

The use of the BREAK key will terminate program execution. The teletypewriter buffer will be flushed. Control will return to the build-mode after the use of the break key.

Paper Tape Input

In order to supply build-mode input from paper tape, the user gives the command TAPE. The system responds with READY. At this point, the user should position his tape in the reader and start the device. Input is terminated when either the end-of-tape occurs, the user turns off the reader, an X-OFF character is read by the paper tape reader, or a jammed tape causes a delay of over one second between the transmission of characters.

At present a maximum of 80 characters are permitted per line of paper tape input. Excessive lines will be truncated at 80 characters with the remaining data placed in the next line. A maximum of two disk links (7680 words) of paper tape input will be collected during a single input procedure. All data in excess of two disk links will be lost.

REMOTE BATCH INTERFACE

Refer to the GRTS Programming Reference Manual, for a description of the deck setup required for submitting a batch job from a remote terminal.

FILE SYSTEM INTERFACE

The File System provides multiprocessor access to a common data base. The file system allocates permanent file space and controls file access for users in local and remote batch and time sharing. The file system is fully described in the GCOS File System Reference Manual.

TERMINAL BATCH INTERFACE

The CARDIN time sharing subsystem allows the user to submit a batch job from a time sharing terminal. This capability is fully described in the GCOS Time Sharing Terminal Batch Interface Facility Reference Manual.

ASCII/BCD CONSIDERATIONS

The Series 6000 FORTRAN enables the programmer to choose the character set that best meets the needs of the application or that is most convenient for the normal mode of execution.

Specification of BCD or ASCII is possible in both batch and time sharing. In batch, the $ FORTY card provides BCD by default. In time sharing, the RUN command provides ASCII by default. The selection is made at compile time and need not normally be designated for execute-only runs. One exception exists, and that is when running in time sharing in the BCD mode, where the run consists of object decks only (no compilations required; not running a saved H*). In this case the BCD option must be given in the RUN command.

When BCD is elected, internal character data and FORMATS are carried in BCD; storage is allocated at a rate of six characters per word; and for I/O, ENCODE, PAUSE, etc., library calls are made to the entry names which work with BCD.

Similarily, when ASCII is elected, the object module will have all ASCII properties. Character data and FORMATs are carried in ASCII; storage is allocated at a rate of four characters per word; and, for I/O, ENCODE, PAUSE, etc., library calls are made to the entry names which work with ASCII.

Therefore, one generally cannot mix object modules of contradicting character sets. Conflicts arise over which routines are to be loaded from the library, how to index through character arrays, how to analyze FORMAT statements, etc.

BCD or ASCII internal programs execute in either batch or time sharing with certain automatic convenience functions for dealing with the variety of file and device types accessible to the program. In terms of specific problems, automatic file transliteration and/or reformatting on a logical record basis is provided for the following:

1. Execution of a BCD program under time sharing.
   a. Input and output may be directed to the teletypewriter.
   b. Input files may be ASCII.
2. Execution of an ASCII program in the batch mode.
   a. Input and output may be directed to the reader, printer, punch, or SYSOUT. Printer output will not have slew interpretation.
   b. Input files may be BCD (media 0, 2, or 3) or ASCII.
3. Execution of a BCD program in the batch mode. Input files may be ASCII.

4. Execution of an ASCII program under time sharing. Input files may be ASCII or BCD (media 0, 2, or 3).

Use of the word "may" in the lists above implies an optional capability. This capability capitalizes on the existence of a collection of alternate entry names in the File and Record Control which are called from FORTRAN library modules. Specification of this optional capability in batch is under programmer control. The proper linkage is accomplished when the following control card is presented to the General Loader:

```
$ USE .GTLIT
```

In YFORTRAN time sharing, the RUN command will place the $ USE .GTLIT control card on the R* file (see Batch Activity Spawned by RUN in Section III).

Files not requiring transliteration and/or reformatting are, of course, acceptable as input. Output files are always recorded in the media code relative to the internal character set of the executing program independent of the batch/time sharing environment. BCD programs will output files with media codes 0, 2, 3 only; ASCII programs will output files with media code 6 only.

FILE FORMATS

All output files generated by Series 6000 FORTRAN, whether formatted or unformatted, ASCII or BCD, sequential or random[1], generated in time sharing or batch, are in Standard System Format (as described in the File and Record Control Reference Manual).

Files generated in time sharing in the build-mode or by the Text Editor may be used directly as ASCII input data files for a FORTRAN object program. BCD file output may be listed (using the SCAN subsystem) at either the user's terminal teletypewriter or at a high speed line printer (BATCH verb of SCAN).

---

[1]Random files may optionally be treated as non-standard format. The file format consists of fixed length records without record control words and block control words. See Section V, Unformatted Random File Input/Output Statements.

GLOBAL OPTIMIZATION

Global Optimization gives the user some control over the balance between compilation and object program efficiency. This analysis has been collected into a single optional compiler phase that is elected by the OPTZ option on the $ FORTY or $ FORTRAN control card or the FORTRAN or YFORTRAN Time Sharing Systems RUN command. The analyses performed include:

1. Common Subexpression Analysis - This analysis provides a determination of multiple occurrences of the same subexpression within a program block. The goal is to perform a given computation only one time.

2. Expression Compute Point Analysis - This analysis provides a determination of the optimal place and time for the computation of some expression in relation to the loop structure of the program and the redefinition points of the expression's constituent elements.

3. Induction Variable Expression Analysis - This analysis determines the optimal computation sequence. Its intent is to reduce expressions to simple operations upon an index register at the loop boundaries.

4. Loop Collapsing Analysis - This analysis attempts to reduce two or more nested loops into a single loop.

5. Register Management Analysis - This analysis determines how registers and temporary storage are to be allocated. Priorities are assigned according to the number of references to an expression and the loop level of these references. Candidates for global assignment over one or more program loops are selected.

6. Induction Variable Materialization Analysis - This analysis determines the necessity for materializing in core the current value of a DO index.

The use of Global Optimization does not always result in a faster running program; furthermore, there are situations where the object code generated by Global Optimization will not be an exact functional equivalent of No-Global-Optimization generated code using the same source. For example:

1. If a program contains multiple references to invariant expressions, code for the evaluation of that expression will follow the program prologue. This placement could result in the unnecessary evaluation of the expressions, if references were from statements which are conditionally executed; i.e., the conditions may be such that the expressions are not to be referenced. For example:

```
      COMMON A,B,C, L1,L2,L3
      ........
      ........
      IF(L1) 1,2,1
1     Z=A+B
      Y=A+B
2     IF(L2) 3,4,3
3     Z2=(B+C)
      ........
      Z3=(B+C)
      ........
4     IF(L3) 5,6,5
5     Y1=(A+C) + (A+C)**2
      ........
      Y2=(A+C)
6     CONTINUE
```

Expressions (A+B), (B+C) and (A+C) have multiple references under conditional code.

They will be pre-calculated following the prologue. However, if L1, L2, and L3 were all zero, this evaluation will have been done unnecessarily.

Another example demonstrates how results can actually be different (OPTZ vs NOPTZ). Consider the following example where the programmer is attempting to avoid a divide check fault.

```
      FUNCTION FX(A,B)
      .......
      .......
 10   IF(B) 1,2,1
  1   FX=A/B+(A/B)**2+(A/B)**3
      GO TO 3
  2   FX=A+A**2+A**3
  3   CONTINUE
      .......
      .......
      END
```

The OPTZ generation will sometimes produce a divide check when (A/B) is evaluated following the prologue; i.e., whenever B = 0.

This situation can be avoided in either of two ways.

a. The above example could be rewritten as:

```
      FUNCTION FX(A,B)

 10   IF(B.NE.0.)FX=A/B+(A/B)**2+(A/B)**3
      IF(B.EQ.0)FX=A+A**2+A**3
      CONTINUE
      .......
      END
```

The optimization phase is "sensitive" to logical if statements. Expressions which are only referenced within the truth clause of a logical if statement will not be removed from such a conditional setting.

b. The following modification to the original example will eliminate the side effect.

```
      FUNCTION FX(A,B)
      .......
 10   IF(B) 1,2,1
  1   B=B
      FX=A/B+(A/B)**2+(A/B)**3
      GO TO 3
  2   FX=A+A**2+A**3
  3   CONTINUE
      .......
      END
```

In this case, the assignment statement B=B will, through redefinition analysis, inhibit removal of the expression (A/B) from the conditional area. Also, there will be no code generation for the statement B=B since a local optimization in code generation recognizes the superfluous nature of the assignment.

2. Another situation results from using certain outdated library "flag" routines. For example, if a program uses FLGEOF, FLGERR to set an end-of-file or error flag, expressions involving these flag variables may appear to the optimizer as invariant over some range of statements when there actually may be a redefinition due to input/output. For example:

```
      CALL FLGEOF (UNT,IF)
           |
           V
      DO 100 I=1,N
           |
           V
      READ(UNT)V ,V
           |
           V
      IF(IF.EQ.0)READ(UNT)V ,V
           |
           V
      IF(IF.EQ.0READ(UNT)V ,V
           |
           V
  100 CONTINUE
```

Since the optimizer does not consider each of the READ statements as a potential redefinition point for the variable IF, the expression (IF.EQ.0) will be removed from the DO 100 I=1,N loop. Thus, in this case the EOF will never be sensed; however, the use of the END= clause will avoid this problem. For example,

```
      DO 100 I=1,N
      READ(UNT,END=10)V1,V2
      .......
      READ(UNT,END=10)V3,V4
      .......
  100 READ(UNT,END=10)V5,V6
      .......
   10 PRINT,"END OF FILE ON",UNT
```

In summary, Global Optimization does not guarantee the generation of faster running programs, and in some instances undesirable faults can be introduced. However, analysis of this optimization technique has shown that in general, significant improvement of object code usually results.

COMPILATION LISTINGS AND REPORTS

The compilation listings and reports produced by the system are controlled by options on the $ FORTY or $ FORTRAN control card.

The following listings and reports are produced when the indicated options are specified (default options are underlined).

Option	Listing or Report Produced
LSTIN	Source Program Listing.
LSTOU	Source and Object Program Listing with a Program Preface Summary.
XREF	Cross Reference Report and TO-FROM Transfer Table.
MAP	Storage Map and Program Preface Summary.

Any diagnostics pertinent to the program will be included in the LSTIN report if it is not suppressed. When the NLSTIN option is present, the diagnostics will appear as a free-standing report.

The Compilation Statistics Report will be produced if any other report is produced or DECK or COMDK is called for.

Figure 3-1 contains an example program with all reports. The following descriptions explain each report in more detail using Figure 3-1 as a base for the description.

Source Program Listing

Each line of this report is divided into three fields. The left-most field contains the line or alter number for each source line. If the source program is line-numbered (NFORM and LNO options specified), the actual line number will be displayed in this field. If the source program is not line-numbered (FORM or NFORM and NLNO options specified), this field contains the alter number (relative sequence number of the line).

The second field contains the text of the source statement and is separated from the first field by six blank characters.

The third field is separated from the second by six blank characters and contains optional sequence/identification information (columns 73-80) from the source line.

Diagnostics are recorded immediately following the source line to which they apply. Diagnostics which do not apply to a particular source line appear at the end of the source listing.

Each diagnostic line begins with five asterisks followed by the character W, F, or T to indicate a warning; a fatal error; and premature termination of compilation, respectively. A complete description of the diagnostics generated in the compiler is included in Appendix B.

In Figure 3-1, a warning diagnostic appears after line 5. Correct code is generated and the program runs as expected. To be error free, a specification statement should be added to the program typing EOF as INTEGER.

To-From Transfer Table

This table, page 2 of Figure 3-1, lists the transfers that exist in the source program logic. The report is sorted into descending line number sequence, keying on the originating line number, and will display up to five transfers on one report line. The destination line number field may indicate the word EXIT or RETURN if the transfer statement is a STOP or RETURN statement. For assigned GOTO statements, where no label list is provided, the label variable name is displayed. In Figure 3-1, page 2, lines 28 and 29 contain transfers. Line 29 includes the statement GOTO 7; statement 7 begins on line 10; the first entry in the transfer report indicates this path. Line 28 contains a STOP statement; the second entry in the transfer report indicates this.

Program Preface Summary

The Program Preface summary, page 3 of Figure 3-1, documents the object module preface (card) information in a format similar to that printed by GMAP.

The Program Break and Common Length are displayed in octal followed by the width of the V Count Bits as used in instructions with special (type 3) relocation.

The SYMDEFs entry shows the relative offset of the internal location corresponding to that SYMDEF, in octal.

Next is a list of LABELLED COMMON blocks known or referenced by this module. Associated with each symbol are two octal fields. The first gives the global symbol number associated, for this compilation, with the common name. This is the number that will appear in the V field of any instruction referencing this labelled common region. The number will appear justified according to the V field. Thus if labelled common SPACE is global symbol 2, and the V field is five bits wide, the display will be 020000 (bit zero is the sign bit). If the V field is six bits wide, the display will be 010000. The second field contains the size, in octal, of the labelled common region.

Two labelled common regions, .DATA. and .STAB., receive special treatment by the loader. Although they are not actually labelled common names, they are included in this portion of the Program Preface summary. The first, .DATA., is allocated space to contain all local data required by the program. This includes arrays and scalars not appearing in common or as arguments, constants, encoded FORMAT information, NAMELIST lists, temporary storage for intermediate results, argument pointers, the error linkage pair (.E.L..), etc. The second, .STAB., is generated when the DEBUG option is employed. This block contains a symbol table for all program variables and statement numbers and may be used for symbolic debugging.

A list of SYMREFs is also included with their associated global symbol number, justified as described above, for LABELLED COMMON names.

Storage Map

This report, page 4 of Figure 3-1, provides information on the allocation of storage for identifiable program elements. This report is divided into three parts: variables and arrays, statement numbers, and constants.

The first part of the report lists all program variables and arrays in alphabetical order. It contains four fields as follows:

1. The first field contains the global symbol name relative to which the variable is defined. Local variables and arrays are defined relative to the origin of the .DATA. space. When a variable or array belongs to some labelled common block, the name of its common is shown; when it belongs to blank common, the field is empty. Argument variables and arrays appear as variables of .DATA.; the indicated location is reserved for a pointer to the actual argument and is initialized on entry to the procedure.

2. The two OFFSET fields provide the location relative to the origin of the indicated global name assigned to this variable or array. For arrays, this is the starting location; subsequent elements of the array are allocated higher order locations. The offset is provided in both octal and decimal for the convenience of the programmer.

3. The MODE field provides the type associated with each identifier. Switch variables are indicated by an empty field.

The second part of the report lists all referenced statement numbers in numerical order. The four fields to the right of each entry are the same as defined above. The ORIGIN fields for FORMAT statement numbers will always be .DATA. and the MODE field will indicate FORMAT. For executable statement numbers, the MODE field will always be blank; the ORIGIN field will be either eight dots (.....), if this is a main program, or the first SYMDEF if this is a subprogram. The OFFSET field is as described above.

The third part of this report lists all numeric and character constants requiring unique storage. All constants are allocated storage relative to the .DATA. block. The two OFFSET fields and the MODE field are as described for variables and arrays. Only the first 17 characters are displayed for character constants.

Object Program Listing

This report, pages 5-9 of Figure 3-1, gives a full listing of the generated object program. The original source statement is identified in the object listing by "SOURCE LINE xxx" and the source line. The individual instruction line format is similar to that produced by GMAP. The first field is the location field; next is the compiled machine language instruction, usually divided into address, operation code, and modifier field. The location field and machine language instruction field are in octal. The next three digits are the relocation bits applicable to the instruction.

Following these is the symbolic equivalent of the generated instruction. This consists of a label field, an operation code field, and a variable field for address and modifier symbols. Referenced statement numbers appear in the label field prefixed by the characters ".S". SYMDEF symbols (such as ENTRY names) also appear in the label field. Operation code and modifier mnemonics are the same as the standard GMAP mnemonics except for some of the pseudo-operation codes.

Data initialization, constants, formats, symbol table entries, etc. are displayed at the end of the report following the source END line. No object END instruction is produced.

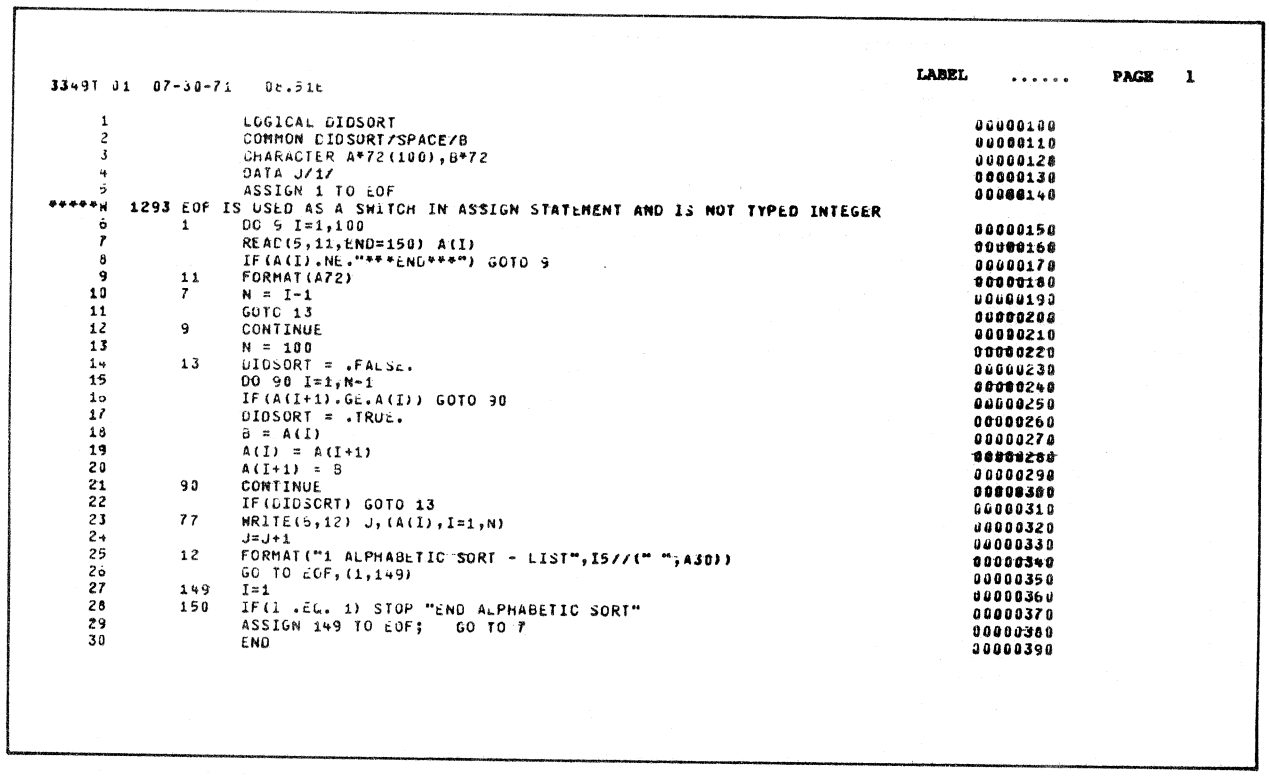
Cross Reference List

This report, page 10 of Figure 3-1, lists in alphabetical order all referenced variables, arrays, statement numbers, SYMREFs and SYMDEFs. Each element results in four or more entries being produced across the line. The first field is the octal location of the item relative to its global symbol. The second field is the item name. Statement numbers are shown with a prefix of ".S". The third field is the applicable global symbol. The fourth field is the line number of the first reference. When there are more references, additional line numbers are displayed across the line. Where required, additional lines are written.

This report is divided into two parts: the second part for statement numbers, the first part for everything else.

Statistics Report

This report, page 11 of Figure 3-1, is produced if any other report is produced or if either the DECK or COMDK option is selected. It contains the edit date and identification of the compiler; processor time and rated lines per minute for each compiler phase; the total compilation time; a count of the number of diagnostics; and the core used for the compilation.



```
3349T 01  07-30-71   08.51E                                   LABEL  .....   PAGE  1

     1              LOGICAL DIDSORT                                        00000100
     2              COMMON DIDSORT/SPACE/B                                 00000110
     3              CHARACTER A*72(100),B*72                               00000120
     4              DATA J/1/                                              00000130
     5              ASSIGN 1 TO EOF                                        00000140
*****W  1293 EOF IS USED AS A SWITCH IN ASSIGN STATEMENT AND IS NOT TYPED INTEGER
     6         1    DO 9 I=1,100                                           00000150
     7              READ(5,11,END=150) A(I)                                00000160
     8              IF(A(I).NE."***END***") GOTO 9                         00000170
     9         11   FORMAT(A72)                                            00000180
    10         7    N = I-1                                                00000190
    11              GOTO 13                                                00000200
    12         9    CONTINUE                                               00000210
    13              N = 100                                                00000220
    14         13   DIDSORT = .FALSE.                                      00000230
    15              DO 90 I=1,N-1                                          00000240
    16              IF(A(I+1).GE.A(I)) GOTO 90                             00000250
    17              DIDSORT = .TRUE.                                       00000260
    18              B = A(I)                                               00000270
    19              A(I) = A(I+1)                                          00000280
    20              A(I+1) = B                                             00000290
    21         90   CONTINUE                                               00000300
    22              IF(DIDSORT) GOTO 13                                    00000310
    23         77   WRITE(6,12) J,(A(I),I=1,N)                             00000320
    24              J=J+1                                                  00000330
    25         12   FORMAT("1 ALPHABETIC SORT - LIST",I5//(" ";A30))       00000340
    26              GO TO EOF,(1,149)                                      00000350
    27         149  I=1                                                    00000360
    28         150  IF(1 .EQ. 1) STOP "END ALPHABETIC SORT"                00000370
    29              ASSIGN 149 TO EOF;   GO TO 7                           00000380
    30              END                                                    00000390
```

Figure 3-1 Compilation Listings and Reports

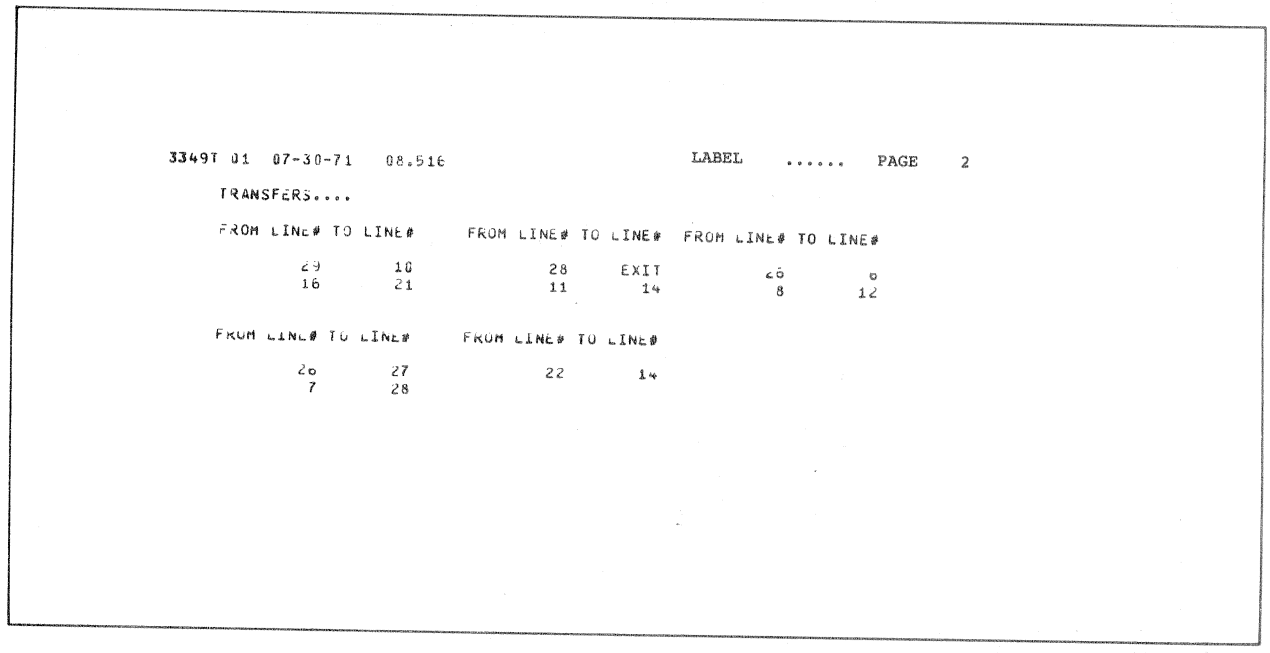


Figure 3-1 (cont) Compilation Listings and Reports

```
                                                LABEL     ......   PAGE    3
3349T 01  07-30-71   08.516

 PROGRAM PREFACE
    PROGRAM BREAK     201
    COMMON LENGTH       1
    V COUNT BITS        5

    SYMDEFS
      ......            0

    LABELLED COMMON              LENGTH
      .DATA.        010000          2314
      .SYMT.        020000            42
      SPACE         030000            14

    SYMREFS
      .FCOM.        040000
      .FCXT.        050000
      .FGERR        060000
      .FFIL.        070000
      .FRTN.        100000
      .FCNVC        110000
      .FCNVI        120000
      .FWRD.        130000
      .FRDD.        140000
```

Figure 3-1 (cont) Compilation Listings and Reports

```
                                                LABEL     ......   PAGE    4
3349T 01  07-30-71   08.516

    STORAGE MAP

    SYMBOLIC  ORIGIN    OFFSET(10)   MODE         OFFSET(8)

    .E.L..    .DATA.       1201      DOUBLE       2261
    A         .DATA.          0      CHARACTER       0
    B         SPACE           0      CHARACTER       0
    DIDSORT                   0      LOGICAL         0
    EOF       .DATA.       1204                   2264
    I         .DATA.       1205      INTEGER      2265
    J         .DATA.       1203      INTEGER      2263
    N         .DATA.       1213      INTEGER      2275

     STATEMENT NUMBERS

          1   ........          2                    2
          7   ........         32                   40
          9   ........         36                   44
         11   .DATA.       1208      FORMAT       2270
         12   .DATA.       1217      FORMAT       2301
         13   ........         42                   52
         90   ........         80                  120
        143   ........        116                  164
        150   ........        118                  166

    CONSTANTS (.DATA.)

      5                     1207      INTEGER      2267
     ***END***              1210      CHARACTER    2272
                            1212      CHARACTER    2274
      6                     1216      INTEGER      2300
     END ALPHABETIC SO      1224      CHARACTER    2310
```

Figure 3-1 (cont) Compilation Listings and Reports

```
                                                                                   LABEL    ......   PAGE    5
3349T 01  07-30-71    08.516

                    000000      ........ NULL
               SOURCE LINE       1                       LOGICAL DIDSORT
               SOURCE LINE       2                       COMMON DIDSORT/SPACE/B
               SOURCE LINE       3                       CHARACTER A*72(100),B*72
               SOURCE LINE       4                       DATA J/1/
               SOURCE LINE       5                       ASSIGN 1 TO EOF
000000   000002 6200 00  010              EAX0   .S1
000001   012264 7400 00  030              STX0   EOF
               SOURCE LINE       6                1      DO 9 I=1,100
                    000002      .S1       NULL
000002   000001 2360 07  000              LDQ    1,DL
000003   012265 7560 00  030              STQ    I
000004   000014 4020 07  000              MPY    12,DL
               SOURCE LINE       7                       READ(5,11,END=150) A(I)
000005   012266 7560 00  030              STQ    .DATA.+1206
000006   140000 7010 00  030              TSX1   .FRDD.
000007   000015 7100 00  010              TRA    *+6
000010   012261 000007   030              ZERO   .E.L..,7
000011   012267 0000 00  030              ARG    .DATA.+1207
000012   012270 0000 00  030              ARG    .S11
000013   000000 0000 00  000              ARG    0
000014   000166 7100 00  010              TRA    .S150
000015   012266 7220 00  030              LXL2   .DATA.+1206
000016   410014 6350 12  030              EAA    A-12,2
000017   110000 7010 00  030              TSX1   .FCNVC
000020   000110 0110 07  000              NOP    72,DL
000021   100000 7010 00  030              TSX1   .FRTN.
               SOURCE LINE       8                       IF(A(I).NE."***END***") GOTO 9
000022   012265 2360 00  030              LDQ    I
000023   000014 4020 07  000              MPY    12,DL
000024   000000 6220 06  000              EAX2   0,QL
000025   012272 6270 00  030              EAX7   .DATA.+1210
000026   410014 6210 12  030              EAX1   A-12,2
000027   005640 5602 01  000              RPD    2,1,TNZ
000030   000000 2350 17  000              LDA    0,7
000031   000000 1150 11  000              CMPA   0,1
000032   000004 6010 04  000              TNZ    4,IC
000033   012274 2350 00  030              LDA    .DATA.+1212
000034   024240 5202 01  000              RPT    10,1,TNZ
000035   000000 1150 11  000              CMPA   0,1
000036   000040 6000 00  010              TZE    *+2
000037   000044 7100 00  010              TRA    .S9
               SOURCE LINE       9                11     FORMAT(A72)
               SOURCE LINE      10                7      N = I-1
                    000040      .S7       NULL
000040   012265 2360 00  030              LDQ    I
000041   000001 1760 07  000              SBQ    1,DL
000042   012275 7560 00  030              STQ    N
               SOURCE LINE      11                       GOTO 13
000043   000052 7100 00  010              TRA    .S13
               SOURCE LINE      12                9      CONTINUE
                    000044      .S9       NULL
```

Figure 3-1 (cont) Compilation Listings and Reports

```
3349T 01  07-30-71   08.516                                                    LABEL  ......  PAGE    6

      000044  012265 2360 00  030          LDQ   I
      000045  000001 0760 07  000          ADQ   1,DL
      000046  000145 1160 07  000          CMPQ  101,DL
      000047  000043 6040 00  010          TMI   *-36
                SOURCE LINE      13                    N = 100
      000050  000144 2360 07  000          LDQ   100,DL
      000051  012275 7560 00  030          STQ   N
                SOURCE LINE      14                    13    DIDSORT = .FALSE.
                               000052     .S13  NULL
      000052  000000 2360 07  000          LDQ   0,DL
      000053  000000 7560 00  020          STQ   DIDSORT
                SOURCE LINE      15                    DO 90 I=1,N-1
      000054  000014 2220 03  000          LDX2  12,DU
      000055  012275 2360 00  030          LDQ   N
      000056  000001 1760 07  000          SBQ   1,DL
      000057  012276 7560 00  030          STQ   .DATA.+1214
      000060  000000 5330 00  000          NEGL  0
      000061  000000 0760 07  000          ADQ   0,DL
      000062  000002 6040 04  000          TMI   2,IC
      000063  000001 3360 07  000          LCQ   1,DL
      000064  012277 7560 00  030          STQ   .DATA.+1215
                SOURCE LINE      16                    IF(A(I+1).GE.A(I)) GOTO 90
      000065  010000 6270 12  030          EAX7  A,2
      000066  410014 6210 12  030          EAX1  A-12,2
      000067  031640 5602 01  000          RPD   12,1,TNZ
      000070  000000 2350 17  000          LDA   0,7
      000071  000000 1150 11  000          CMPA  0,1
      000072  000074 6020 00  010          TNC   *+2
      000073  000120 7100 00  010          TRA   .S90
                SOURCE LINE      17                    DIDSORT = .TRUE.
      000074  000001 2360 07  000          LDQ   1,DL
      000075  000000 7560 00  020          STQ   DIDSORT
                SOURCE LINE      18                    B = A(I)
      000076  410014 6270 12  030          EAX7  A-12,2
      000077  030000 6210 00  030          EAX1  B
      000100  000000 0110 07  000          NOP   0,DL
      000101  031600 5602 01  000          RPD   12,1
      000102  000000 2350 17  000          LDA   0,7
      000103  000000 7550 11  000          STA   0,1
                SOURCE LINE      19                    A(I) = A(I+1)
      000104  010000 6270 12  030          EAX7  A,2
      000105  410014 6210 12  030          EAX1  A-12,2
      000106  000000 0110 07  000          NOP   0,DL
      000107  031600 5602 01  000          RPD   12,1
      000110  000000 2350 17  000          LDA   0,7
      000111  000000 7550 11  000          STA   0,1
                SOURCE LINE      20                    A(I+1) = B
      000112  030000 6270 00  030          EAX7  B
      000113  010000 6210 12  030          EAX1  A,2
      000114  000000 0110 07  000          NOP   0,DL
      000115  031600 5602 01  000          RPD   12,1
      000116  000000 2350 17  000          LDA   0,7
```

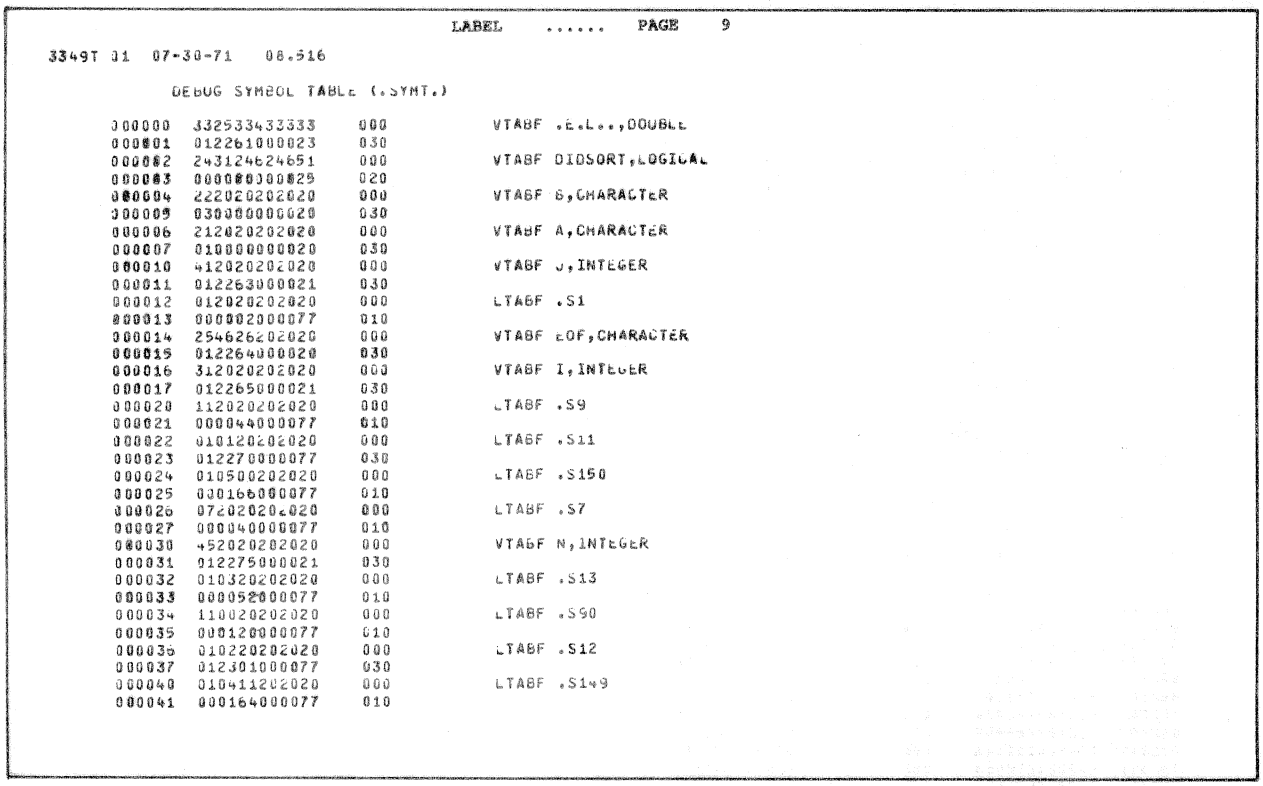
Figure 3-1 (cont) Compilation Listings and Reports

```
3349T 01  07-30-71   08.516                                                LABEL  ......   PAGE    7

      000117  000000 7550 11  000           STA   0,1
                  SOURCE LINE      21              90    CONTINUE
                           000120     .S90     NULL
      000120  000014 0220 03  000           ADLX2 12,DU
      000121  012277 0540 00  030           AOS   .DATA.+1215
      000122  000065 6010 00  010           TNZ   *-29
                  SOURCE LINE      22                    IF(OLDSORT) GOTO 13
      000123  000000 2340 00  020           SZN   OLDSORT
      000124  000052 6010 00  010           TNZ   .S13
                  SOURCE LINE      23              77    WRITE(6,12) J,(A(I),I=1,N)
      000125  130000 7010 00  030           TSX1  .FWRD.
      000126  000132 7100 00  010           TRA   *+4
      000127  012261 0C0027    030           ZERO  .E.L..,23
      000130  012300 0000 00  030           ARG   .DATA.+1216
      000131  012301 0000 00  030           ARG   .S12
      000132  012263 2350 00  030           LDA   J
      000133  120000 7010 00  030           TSX1  .FCNVI
      000134  000014 2220 03  000           LDX2  12,DU
      000135  012275 3360 00  030           LCQ   N
      000136  000002 6040 04  000           TMI   2,IC
      000137  000001 3360 07  000           LCQ   1,DL
      000140  012277 7560 00  030           STQ   .DATA.+1215
      000141  410014 6350 12  030           EAA   A-12,2
      000142  110000 7010 00  030           TSX1  .FCNVC
      000143  000110 0110 07  000           NOP   72,DL
      000144  000014 0220 03  000           ADLX2 12,DU
      000145  012277 0540 00  030           AOS   .DATA.+1215
      000146  000141 6010 00  010           TNZ   *-5
      000147  070000 7010 00  030           TSX1  .FFIL.
                  SOURCE LINE      24                    J=J+1
      000150  012263 0540 00  030           AOS   J
                  SOURCE LINE      25              12    FORMAT("1 ALPHABETIC SORT - LIST",I5///(" ",A30))
                  SOURCE LINE      26                    GO TO EOF,(1,149)
      000151  000006 6210 04  000           EAX1  6,IC
      000152  012264 6350 51  030           EAA   EOF,I
      000153  004300 5202 01  000           RPT   2,1,TZE
      000154  000000 1150 11  000           CMPA  0,1
      000155  777777 6000 31  000           TZE   -1,1*
      000156  000003 7100 04  000           TRA   3,IC
      000157  000002 0000 00  010           ARG   .S1
      000160  000164 0000 00  010           ARG   .S149
      000161  060000 7010 00  030           TSX1  .FGERR
      000162  000164 7100 00  010           TRA   *+2
      000163  012261 000032  030           ZERO  .E.L..,26
                  SOURCE LINE      27              149   I=1
                           000164     .S149    NULL
      000164  000001 2360 07  000           LDQ   1,DL
      000165  012265 7560 00  030           STQ   I
                  SOURCE LINE      28              150   IF(I .EQ. 1) STOP "END ALPHABETIC SORT"
                           000166     .S150    NULL
      000166  000001 2360 07  000           LDQ   1,DL
      000167  012265 1160 00  030           CMPQ  I
```

Figure 3-1 (cont) Compilation Listings and Reports

```
                                                                                LABEL     ......    PAGE     8
3349T 01  07-30-71    08.516

      000170  000176 6010 00  010           TNZ   *+6
      000171  050000 7018 00  030           TSX1  .FCXT.
      000172  000176 7100 00  010           TRA   *+4
      000173  012261 000034   030           ZERO  .E.L..,28
      000174  012310 0000 08  030           ARG   .DATA.+1224
      000175  000023 0004 07  000           ARG   19,DL
                SOURCE LINE      29                    ASSIGN 149 TO EOF;   GO TO 7
      000176  000164 6200 00  010           EAX0  .S149
      000177  012264 7400 00  030           STX0  EOF
      000200  000040 7100 00  010           TRA   .S7
                SOURCE LINE      30                    END



                        002261              ORG   .DATA.+1201
      002261  000000000000     000 .E.L..   OCT
      002262  333333333333     000          ETC
      002263  000000000001     000 J        DEC    1

                        002267              ORG   .DATA.+1207
      002267  000000000005     000          DEC    5
      002270  352107025520     000 .S11     BCI   (A72)

                        002272              ORG   .DATA.+1210
      002272  545454254524     000          BCI   ***END
      002273  545454202020     000          BCI   ***
      002274  202020202020     000          BCI

                        002300              ORG   .DATA.+1216
      002300  000000000006     000          DEC    6
      002301  357001202143     000 .S12     BCI   ("1 AL
      002302  473021222563     000          BCI   PHABET
      002303  312320624651     000          BCI   IC SOR
      002304  632052204331     000          BCI   T - LI
      002305  626376733105     000          BCI   ST",I5
      002306  616135762076     000          BCI   //(" "
      002307  732103065955-    000          BCI   ,A30))
      002310  254524202143     000          BCI   END AL
      002311  473021222563     000          BCI   PHABET
      002312  312320624651     000          BCI   IC SOR
      002313  632020202020     000          BCI   T
```

Figure 3-1 (cont) Compilation Listings and Reports



```
                                        LABEL     ......    PAGE     9

3349T 01  07-30-71   08.516

          DEBUG SYMBOL TABLE (.SYMT.)

      000000  332533433333     000          VTABF .E.L..,DOUBLE
      000001  012261000023     030
      000002  243124624651     000          VTABF DIDSORT,LOGICAL
      000003  000000300025     020
      000004  222020202020     000          VTABF B,CHARACTER
      000005  030000000020     030
      000006  212020202020     000          VTABF A,CHARACTER
      000007  010000000020     030
      000010  412020202020     000          VTABF J,INTEGER
      000011  012263000021     030
      000012  012020202020     000          LTABF .S1
      000013  000002000077     010
      000014  254626202020     000          VTABF EOF,CHARACTER
      000015  012264000020     030
      000016  312020202020     000          VTABF I,INTEGER
      000017  012265000021     030
      000020  112020202020     000          LTABF .S9
      000021  000044000077     010
      000022  010120202020     000          LTABF .S11
      000023  012270000077     030
      000024  010500202020     000          LTABF .S150
      000025  000166000077     010
      000026  072020204020     000          LTABF .S7
      000027  000040000077     010
      000030  452020202020     000          VTABF N,INTEGER
      000031  012275000021     030
      000032  010320202020     000          LTABF .S13
      000033  000052000077     010
      000034  110020202020     000          LTABF .S90
      000035  000120000077     010
      000036  010220202020     000          LTABF .S12
      000037  012301000077     030
      000040  010411202020     000          LTABF .S149
      000041  000164000077     010
```

Figure 3-1 (cont) Compilation Listings and Reports

```
3349T 01  07-30-71   08.51E                                   LABEL ......  PAGE    10

 ORIGIN SYMBOLIC   REFERENCES BY ALTER NUMBER

      0 ........  ........          0
     11 .FCNVC                      7      23
     12 .FCNVI                     23
      4 .FCOM.
      5 .FCXT.                     28
      7 .FFIL.                     23
      6 .FGERR                     26
     14 .FRDC.                      7
     10 .FRIN.                      7
     13 .FWRC.                     23
   2261 .E.___.          .DATA.     0       7      23      26      28
      0 A                .DATA.     7       8      16      18      19      20      23
      0 B                SPACE     18      20
      0 DIDSORT                    14      17      22
   2264 EOF              .DATA.     5      26      29
   2265 I                .DATA.     6       8      10      12      27      28
   2263 J                .DATA.     4      23      24
   2275 N                .DATA.    10      13      15      23

      0 .S0              FORMAT
      2 .S1                         5       6      26
     40 .S7                        10      29
     44 .S9                         8      12
   2270 .S11            FORMAT      7
   2301 .S12            FORMAT     23
     52 .S13                      11      14      22
      0 .S77
    120 .S98                      16      21
    164 .S145                     26      27      29
    166 .S150                      7      28
```

Figure 3-1 (cont) Compilation Listings and Reports

```
                                        LABEL     ......   PAGE   11
33491 01  07-30-71   08.516

   EDIT DATE   06-23-71   *SDL4.0

                ELAPSED          LINES/
              TIME (SEC)        MINUTE

  OVERHEAD         .21
  PHASE 1          .09           19164
  PHASE 2          .00          102278
  PHASE 3          .03           49698
  PHASE 4          .15           11543
  PHASE 5          .54            3293
     TOTAL        1.05            1704

 TOTAL TIME       1.08

  THERE WERE      1 DIAGNOSTICS IN ABOVE COMPILATION
 23552 WORDS WERE USED FOR THIS COMPILATION
```

Figure 3-1 (cont) Compilation Listings and Reports

SECTION IV

FORTRAN STATEMENTS

This section contains a description of FORTRAN statements arranged in alphabetical order.

ASSIGNMENT

ASSIGNMENT

Arithmetic Assignment Statement

The arithmetic assignment statement has the general form V = E, where V is an unsigned variable name or array element name of an arithmetic type (integer, real, double-precision, complex) and E is an arithmetic expression. An arithmetic assignment statement causes FORTRAN to compute the value of the expression on the right and to give that value to the variable on the left of the equal sign. Execution of this statement causes the evaluation of the expression E and the altering of V according to Table 4-1.

Figure 4-1 indicates the legitimate combinations of expressions and variables in arithmetic assignment statements.

C = Complex
D = Double-Precision
I = Integer
R = Real
N = Invalid
T = Typeless
L = Logical
H = Character

		E						
		I	R	D	C	T	L	H
V	I	I	I	I	I	I	N	N
	R	R	R	R	R	R	N	N
	D	D	D	D	D	D	N	N
	C	C	C	C	C	N	N	N
	L	N	N	N	N	L	L	N
	H	N	N	N	N	N	N	H

Figure 4-1. Arithmetic Assignment Statement Combinations

The following examples show various arithmetic assignment statements:

where

R1 and R2	are	real variables
C1 and C2	are	complex variables
D	is	a double-precision variable
I	is	an integer variable

R1 = R2	R2 replaces R1
I = R2	R2 is truncated to an integer, and stored in I.
R1 = I	I is converted to a real variable and stored in R1.

ASSIGNMENT

R1 = 3*R2	The Expression contains a real variable and an integer constant. The statement will be compiled as R1 = 3.*R2.
R1 = R2*D+I	Multiply R2 by D using double precision arithmetic, add the double precision equivalent of I to that result, store the most significant part (rounded) of the result as a real variable R1.
C1 = C2* (3.7,2.0)	Multiply using complex arithmetic and store the result in C1 as a complex number.
C2 = R2	Replace the real part of C2 by the current value of R2. Set the imaginary part of C2 to zero.

Logical Assignment Statement

A logical assignment statement has the form

```
V = E
```

where V is a logical variable name or logical array element and E is a logical expression. Thus if L1,L2, etc. are logical variables, logical assignment statements can be written:

```
L1 = .TRUE.
L2 = .F.
L3 = A.GT.25.0
L4 = I.EQ.0 .OR.A.GT.25.0
L5 = L6
```

The first two are the logical equivalent of statements of the form

```
variable = constant
```

L3 would be set .TRUE. if the value of the real variable A is greater than 25.0, and to .FALSE. if A is equal to or less than 25.0. L4 would be set to .TRUE. if the value of I was zero or A is greater than 25.0 and to .FALSE. otherwise. L5 would be set to the same truth value as L6 currently has.

Character Assignment Statement

A character assignment statement is of the form

```
V = E
```

where V is a character variable name or character array element name and E is a character constant, variable, function or array element. Characters are stored left adjusted in the destination location with trailing blanks if applicable. If the declared length of V is less than E, then E is truncated to the size of V for the assignment, and the left-most characters are assigned. Thus if C1, C2, etc. are character variables, character assignment statements can be written.

```
C1 = "ABCD"          The four characters are assigned to variable C1.
C2 = C1
C3 = 'ABCDEFGHIJKLMNOP'
C4 = CONCAT('ABC',C2)
```

Label Assignment Statement

A label assignment statement is of the form:

```
ASSIGN k to i
```

where k is a statement number and i is a nonsubscripted integer variable. The statement number must refer to an executable statement in the same program unit in which the ASSIGN statement appears. For example:

```
ASSIGN 24 TO M
   .
   .
   .
GO TO M,(1,22,41,24,36)
```

ASSIGNMENT

Table 4-1. Rules for Assignment of E to V

IF V TYPE IS:	AND E TYPE IS:	THE ASSIGNMENT RULE IS:
Integer	Real	Fix and Assign
Integer	Integer	Assign
Integer	Double Precision	Fix and Assign
Integer	Complex	Fix the most significant Real Part and Assign
Integer	Character	Illegal
Integer	Typeless	Assign
Integer	Logical	Illegal
Real	Integer	Float and Assign
Real	Real	Real Assign
Real	Double Precision	Assign the most significant part as Real
Real	Complex	Assign the Real Part
Real	Character	Illegal
Real	Typeless	Assign
Real	Logical	Illegal
Double Precision	Integer	Float and Assign as Double Precision
Double Precision	Real	Real Assign as Double Precision
Double Precision	Double Precision	Assign
Double Precision	Complex	Assign Real Part as Double Precision
Double Precision	Character	Illegal
Double Precision	Typeless	Illegal
Double Precision	Logical	Illegal
Complex	Integer	Float and assign to the Real Part and Assign zero to the Imaginary Part
Complex	Real	Assign to the Real Part, Assign 0 to Imaginary Part
Complex	Double Precision	Assign the Most Significant Part to the Real Part and Assign 0 to the Imaginary Part
Complex	Complex	Assign
Complex	Character	Illegal
Complex	Typeless	Illegal
Complex	Logical	Illegal
Character	Integer	Illegal
Character	Real	Illegal
Character	Double Precision	Illegal
Character	Complex	Illegal
Character	Character	Assign
Character	Typeless	Illegal
Character	Logical	Illegal
Logical	Integer	Illegal
Logical	Real	Illegal
Logical	Double Precision	Illegal
Logical	Complex	Illegal
Logical	Character	Illegal
Logical	Typeless	Assign
Logical	Logical	Assign

Table 4-1 (cont). Rules for Assignment of E to V

NOTES:

1. Assign means transmit the resulting value, without change, to the entity.

2. Real assign means transmit to the entity as much precision of the most significant part of the resulting value as a real datum can contain.

3. Fix means truncate any fractional part of the result and transform that value to the form of an integer datum.

4. Float means transform the value to the form of a real datum.

5. Double precision float means transform the value to the form of a double precision datum, retaining in the process as much of the precision of the value as a double precision datum can contain.

6. Assign with respect to character type implies a move operation. When the receiving variable's size is greater than the size of the sending string the move is performed filling the receiving variable with blanks. When the receiving variable's size is less than that of the sending string truncation takes place.

ABNORMAL

ABNORMAL

The ABNORMAL statement has the following form:

ABNORMAL
or
ABNORMAL $a_1, a_2, \ldots, a_n$

where

a_i is a Function subprogram name whose characteristics are being qualified by this statement.

A function whose name appears in an ABNORMAL statement (or in an EXTERNAL statement with an ABNORMAL modifier) is treated as one whose references, for optimization purposes, cannot be treated the same as a variable or array element reference in an expression. That is, the function has side-effects which may in some way alter its arguments or locations in COMMON, it performs I/O, or it is capable of returning different results given the same actual arguments.

If no functions are typed as ABNORMAL in a given subprogram, then all are treated as ABNORMAL. The appearance of an ABNORMAL statement reverses the default interpretation, and all non-qualified functions are treated as normal.

For best results, the programmer should analyze his functions to make this determination, and wherever applicable include appropriate ABNORMAL statements. If a subprogram has function references, and none of the referenced functions are abnormal, then it may prove beneficial to include an ABNORMAL statement. The first form with no list may be used for this purpose.

Subroutines, as referenced by CALL statements, are always considered ABNORMAL.

This discussion, and the ABNORMAL statement itself, applies only to programs being compiled with the OPTZ option. When optimization is not performed, the presence or absence of ABNORMAL statements is immaterial.

BACKSPACE

BACKSPACE

This statement is operable only for sequential files. Execution of this statement causes the file to be positioned so that the record that had been the preceding record prior to execution becomes the next record. If the file is positioned at its initial point, the statement has no effect. This statement has the form:

```
BACKSPACE f
```

where f is the file designator.

BLOCK DATA

BLOCK DATA

One way to enter data into a labeled COMMON block during compilation is by using a BLOCK DATA subprogram. (Data may not be entered into blank common by the use of a DATA statement in any program or subprogram.) This subprogram may contain only type, EQUIVALENCE,PARAMETER, IMPLICIT, DATA, DIMENSION, and COMMON statements in addition to the BLOCK DATA and END Statements.

A BLOCK DATA statement is of the form:

```
BLOCK DATA
```

The following rules also apply:

1. The BLOCK DATA subprogram may not contain any executable statements.
2. The first statement of this subprogram must be the BLOCK DATA statement. The last must be the END statement.
3. All elements of a COMMON block must be listed in the COMMON statement even though they do not all appear in the DATA statement.
4. Data may be entered into at most 63 COMMON blocks in a single BLOCK DATA subprogram.

If two or more BLOCK DATA subprograms occur for the same application, the data specified by each of them is entered into the appropriate COMMON blocks. The data from the last such subprogram is retained for any area of a COMMON block that is referred to more than once.

Example of BLOCK DATA

```
BLOCK DATA
COMMON/ELN/C,A,B/RMC/Z,Y
DIMENSION B(4), Z(3)
DOUBLE PRECISION Z
COMPLEX C
DATA (B(I),I=1,4)/1.1,1.2,2*1.3/,C/
        (2.4,3.769)/,Z(1)/7.6498085DO/
END
```

This example contains two labeled COMMON blocks, ELN and RMC. All variables in each block must be listed even though not all variables receive values from the DATA statement. (The variable A in the example does not appear in the DATA statement.)

CALL

The CALL statement is used to refer to a SUBROUTINE subprogram (see Subroutines in the Index).

A CALL statement is one of the forms:

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

or

CALL s

where s is the name of a SUBROUTINE and the a_i are actual arguments or alternate returns.

The execution of a CALL statement references the designated subroutine. Execution of the CALL statement is completed upon return from the designated subroutine.

Example: CALL MATMISL(A,B,C,I,J,K)

Execution of the user program continues with the first executable statement of the SUBROUTINE (or SUBROUTINE entry point) MATMISL.

Additional examples:

```
CALL MATMPY(X,5,10,Y,7,2)
CALL QDRTIC(9.732,Q/4.536,R-S**2.0,X1,X2)
CALL OUTPUT
CALL ABC(X,B,C,$5,$200)
```

The CALL statement transfers control to a SUBROUTINE subprogram and presents it with the actual arguments. For purposes of optimization, all subroutine calls are treated as abnormal function references.

The arguments may be any of the following:

1. A constant.
2. A subscripted or nonsubscripted variable or an array name.
3. An arithmetic or logical expression.
4. The name of a FUNCTION or SUBROUTINE subprogram.

5. An omitted argument may be indicated by successive commas in the argument list. A reference to an omitted argument by the called subprogram is undefined.

6. $n where n is a statement number or a switch variable for a nonstandard return.

The arguments presented by the CALL statement must agree in number, order, type, and array size (except as explained under the DIMENSION statement) with the corresponding dummy arguments in the SUBROUTINE or ENTRY statement (see ENTRY in Index) of the called subprogram.

CHARACTER

CHARACTER

The CHARACTER statement is a form of the explicit type statement. It has the form

CHARACTER *b a_1 *s_1 (k_1)/d_1/,...,a_n *s_n (k_n)/d_n/

where

b is a positive integer constant and applies to all variables in the statement unless otherwise specified (see Explicit Type Statement).

a_i is a variable, array, or subprogram name whose characteristics are being qualified by this statement.

s_i is the maximum number of characters (≤511) that may be contained by the CHARACTER element being defined. An adjustable size specification is permitted within a subprogram when both the character variable and its size parameter(s) are included as formal parameters. For example:

```
SUBROUTINE  MOVE (A,I,J,B,K)
CHARACTER  A*I(J,4),B*I
B = A(K,2)
RETURN
END
```

In this example, the number of characters associated with A and B are variable.

Adjustable size specifications are not allowed for the following:

1. As the size specification for a character function.
2. As the size specification in an IMPLICIT statement.
3. For types other than CHARACTER.

k_i supplies the dimension information necessary to allocate storage to arrays.

d_i is the initial data values.

The CHARACTER statement is more fully described under the type statement in this section.

COMMON

A COMMON statement is of the form:

$$COMMON/x_1/a_1/.../x_n/a_n$$

where each a_i is a nonempty list of variable names, array names, or array declarators (no dummy arguments are permitted) and each x_i is a symbolic name or is empty. If x_i is empty, the first two slashes are optional. Each x_i is a block name that bears no relationship to any variable or array having the same name. COMMON assigns two elements in different subprograms or in a main program and a subprogram to the same location(s).

All variables named in a COMMON statement are assigned to storage in the sequence in which the names appear in the COMMON statement. For example if the following statement appeared in the main program:

```
COMMON A,B,C,D
```

the four variables are assigned to storage locations in the order named in a special section of storage called unnamed or blank common. Thus A is a specific storage location followed by B, etc. If in a subprogram we have the statement:

```
COMMON W,X,Y,Z
```

This means W is assigned the first location in blank common, and X the next, etc. Since the storage assigned to blank common is the same for the subprogram as the main program, A and W, B and X, C and Y, and D and Z share the same locations.

Additional blocks of storage can be established by labeling COMMON. Labeled COMMON is established by writing the label between two slashes as follows:

```
COMMON/X/A,B,C
```

Labeled and blank COMMON may be included in the same statement. For example, if the following two statements were to appear in a main program and in a subprogram:

```
COMMON A,B,C/Y1/D,E/Y2/F(50),G(3,10)
COMMON H,I,J/Y1/K,L/Y2/M(50),N(3,10)
```

Blank COMMON would contain A,B,C (in that order) in the program containing the first COMMON statement and H,I,J in the program containing the second. A and H would be assigned the same location as would B and I, and C and J. The common block labelled Y1 would establish D and E in the same locations as K and L. Y2 in the first program contains the 50 locations of F and the 30 locations of G. The same 80 locations will be assigned to M and N in the second program. The following rules apply:

A double precision or complex entity is counted as two logically consecutive storage units. A logical, real, or integer, entity is one storage unit. A character entity is given as many consecutive storage units as are required to contain the specified number of characters.

The following applies to common block with the same number of storage units or to blank common:

1. In all program units which have given the same type to a given position (counted by the number of preceding storage units), references to that position refer to the same quantity.

2. A correct reference is made to a particular position assuming a given type if the most recent value assignment to that position was of the same type.

3. Complex and Double Precision entities will be assigned consecutive storage units (pairs) such that the first unit has an even storage address.

COMPLEX

COMPLEX

The COMPLEX statement is an explicit type statement with the following form:

COMPLEX a_1*s_1(k_1) /d_1/,..., a_n*s_n(k_n)/d_n/

where

- a_i is a variable, array, or subprogram name whose characteristics are being qualified by this statement.
- *s_i is an optional size-in-bytes qualification and is ignored.
- k_i supplies the dimension information necessary to allocate storage to arrays.
- d_i represents the initial data values.

The COMPLEX statement is more fully described under the Type statement in this section.

CONTINUE

The CONTINUE statement is a dummy statement most often used as the last statement in the range of a DO, when the last statement would otherwise have been a GO TO or IF. (See description of the DO statement.) It has the following form:

```
CONTINUE
```

For example:

```
10 DO  12  I = 1,10
   IF (ARG - VAL(I)) 12,13,12
12 CONTINUE
```

Execution of this statement causes a continuation of the normal execution sequence.

DATA

DATA

A data initialization statement is of the following form:

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

where each k_i is a list containing names of variables, arrays, array elements and implied DOs. Each d_i is a list of optionally signed constants of the form:

C or J* C

where C is a constant and J is a repeat modifier which specifies that constant C is to be used J times. J must be an integer constant or PARAMETER symbol.

The DATA statement enables the programmer to enter data into the program at the time of compilation. For example:

```
      DATA A,B,C/14.7,62.1,1.5E-20/
```
or
```
      DATA A/14.7/,B/62.1/,C/1.5E-20/
```

will initially assign the value 14.7 to A, 62.1 to B and 1.5E-20 to C.

The following is an additional example:

```
      DATA ZERO, (A(I), I=1,5),A(9)/
     &  0.0, 5*1.0, 100.5/
```

This will make ZERO the value zero; put 1.0 in the first five elements of A; and 100.5 in A(9).

The following rules apply:

1. Dummy arguments and names in blank common may not appear in the list k_i.

2. Any subscript expression must be an integer expression of the form $C_1 *V \pm C_2$ where C_1 and C_2 are integer constants or parameters and V is an integer variable which appears as the induction variable of some enclosing implied DO.

3. When J* appear ahead of a constant it indicates the constant is to be applied J times, i.e., it will initialize the next J items in the list with C .

4. Any type of constant may appear in the list d. However, type checking is performed to verify that a variable is being initialized with a constant of the same type, subject to the condition in rule 5.

5. Constants of type octal or character may be used to initialize variables of any type.

6. Character variables are initialized with character constants, and truncation or blank-filling may take place if the sizes of the two differ.

7. There must be a one-to-one correspondence between the list items and the data constants. If a non-character type variable is to be initialized with a character constant and the constant is longer than one word of core storage can accommodate, then the variable must appear as an array element reference. The constant will be assigned to consecutive locations in core beginning with referenced location in the array. Thus in the example:

```
INTEGER G(5)
DATA G(1)/15HDATA TO BE READ /
```

there is a one-to-one relationship between the two lists (one variable, one constant) but locations G(1), G(2), G(3) and possibly G(4) (if the mode is ASCII) are affected.

8. DATA defined variables that are redefined during execution assume their new values regardless of the DATA statement.

9. Where data is to be compiled into an entire array, the name of the array (with indexing information omitted) can be placed in the list. The number of data literals must exactly equal the size of the array. For example, the statements

```
DIMENSION B(25)
DATA A,B,C/24*4.0,3.0,2.0,1.0/
```

define the values of A, B(1), ..., B(23) to be 4.0 and the values of B(24), B(25), and C to be 3.0, 2.0, and 1.0 respectively.

DATA

10. DATA statements appearing in a BLOCK DATA subprogram may pre-set data into labeled COMMON storage only. A maximum of 63 such common areas may be pre-set from any one BLOCK DATA subprogram.

11. DATA statements appearing in other than a BLOCK DATA subprogram may pre-set data into program storage local to that subprogram, or labeled COMMON. A maximum of 62 such common areas are permitted.

12. The type statements, described later in this section, may also be used to initialize data values, and are subject to the same rules as given here for the DATA statement.

DECODE

DECODE

A DECODE statement is of the following form:

```
DECODE (a,t) list
```

where t may be a FORMAT statement number, a character scalar, or an array name, giving the format information for decoding; a is a character scalar, array element, or an array of any type, which specifies the starting location of the internal buffer; and list is as specified for the READ statement.

The DECODE statement causes the character string beginning at location a to be converted to data items according to the format specified by t; and stored in the elements of the list.

The format information and list should not require more characters than are in a.

For example:

```
a(1) = "ØØØ1"
DECODE (a,4)i
4 FORMAT (I4)
```

After execution, the array a is not altered but the variable i contains an integer one.

Additional information on the DECODE statement is contained in Section V under Internal Data Conversion.

DIMENSION

The DIMENSION statement provides the information necessary to allocate storage for arrays in the object program, and it defines the maximum size of the arrays. An array may be declared to have from one to seven dimensions by placing it in a DIMENSION statement with the appropriate number of subscripts appended to the variable. The DIMENSION statement has the form:

DIMENSION $v_1(i_1)/d_1/,v_2(i_2)/d_2/,\ldots v_n(i_n)/d_n/$

Each v_i is an array declarator (see Arrays in Section II) with each v being an array name. Each i_i is composed of from one to seven unsigned integer constants, integer parameters, or integer variables separated by commas. Integer variables may be a component of i_i only when the DIMENSION statement appears in a subprogram, and the array may not be in COMMON. Each $/d_i/$ represents optional initial data values. The form for each $/d_i/$ is as specified for the data statement.

1. The DIMENSION statement must precede the first use of the array in any executable statement.

2. A single DIMENSION statement may specify the dimensions of any number of arrays.

3. If a variable is dimensioned in a DIMENSION statement, it must not be dimensioned elsewhere.

4. Dimensions may also be declared in a COMMON or a Type statement. If this is done, these statements are subject to all the rules for the DIMENSION statement.

5. The initial data value are optional, and if specified, apply to the array immediately preceeding their declaration.

In the following examples A, B, and C are declared to be array variables with 4, 1, and 7 dimensions respectively. Note that each element of array B is initialized to contain the value 1.

```
DIMENSION A(1,2,3,4),B(10)/10 *1./
DIMENSION C(2,2,3,3,4,4,5)
```

DO

DO

This statement enables the user to execute a section of a program repeatedly, with automatic changes in the value of a variable between repetitions. The DO statement may be written in either of these forms:

```
DO n i = m1,m2
or
DO n i = m1,m2,m3
```

In these statements n must be a statement number of an executable statement, i must be a nonsubscripted integer variable, and m_1,m_2,m_3 may be any valid arithmetic expression. If m_3 is not stated, it is understood to be 1 (first form). These parameters (m_1,m_2,m_3) are truncated to integers before use.

The statements following the DO up to and including the one with statement number n are executed repeatedly. They are executed first with i = m_1; before each succeeding repetition i is increased by m_3 (when present, otherwise by 1); when i exceeds m_2 execution of the DO is ended.

1. The terminal statement (n) may not be a GO TO (of any form), RETURN, STOP, or DO statement.

2. The terminal statement (n) may be an arithmetic IF statement with at least one null label field. The null path is a simulated CONTINUE statement terminating the DO.

3. The range of a DO statement includes the executable statements from the first executable statement following the DO to and including the terminal statement (n) associated with the DO.

4. Another DO statement is permitted within the range of a DO statement. In this case, the range of the inner DO must be a subset of the range of the outer DO.

5. The values of m_1, m_2 and m_3 must all be positive and m_3 may not be zero. If m_2 is less than or equal to m_1 the loop will be processed once.

6. None of the control parameters, i, m_2, or m_3, may be redefined within the loop or in the extended range of the loop, if such exists.

A completely nested set of DO statements is a set of DO statements and their ranges such that the first occurring terminal statement of any of those DO statements physically follows the last occurring DO statement.

If a statement is the terminal statement of more than one DO statement, the statement number of that terminal statement may not be used in any GO TO or arithmetic IF statement that occurs anywhere but in the range of the innermost DO with that as its terminal statement.

A DO statement is used to define a loop. The action succeeding execution of a DO statement is described in the following steps:

1. The induction variable, i, is assigned the value represented by the initial parameter (m_1).

2. Instructions in the range of the DO are executed.

3. After execution of the terminal statement the induction variable of the most recently executed DO statement associated with the terminal statement is changed by the value represented by the associated step parameter (m_3).

4. If the value of the induction variable after change is less than or equal to the terminal value, then the action described starting at the 2nd step is repeated, with the understanding that the range in question is that of the DO, whose induction variable has been most recently changed. If the value of the induction variable is greater than the terminal value, then the DO is said to have been satisfied.

5. At this point, if there were one or more other DO statements referring to the terminal statement in question, the induction variable of the next most recently executed DO statement is changed by the value represented by its associated step parameter and the action described in the 4th step is repeated until all DO statements referring to the particular termination statement are satisfied, at which time all such nested DO's are said to be satisfied and the first executable statement following the terminal statement is executed.

   (In the remainder of this section a logical IF statement containing a GO TO or an arithmetic IF as its conditional statement is regarded as a GO TO or an arithmetic IF statement, respectively.)

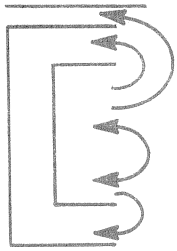
6. Upon exiting from the range of a DO by the execution of a GO TO statement or an arithmetic IF statement, that is, other than by satisfying the DO, the induction variable of the DO is defined and is equal to the most recent value attained as defined in the preceding paragraphs.

DO

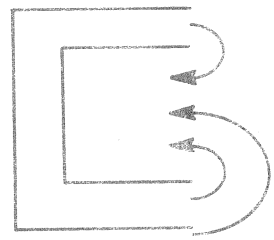
Transfer of Control

The following configurations show permitted and nonpermitted transfers.

Permitted



Not Permitted



An example of the DO statement follows:

```
      K = 0
      DO 10 I = 1,3
      DO 10 J = 1,2
      K = K + I + J

   10 CONTINUE
```

where the K values are computed as:

OLD K	I	J	NEW K
K = 0			
K = 0 +	1 +	1 =	2
K = 2 +	1 +	2 =	5
K = 5 +	2 +	1 =	8
K = 8 +	2 +	2 =	12
K =12 +	3 +	1 =	16
K =16 +	3 +	2 =	21

Extended Range

A DO statement is said to have an extended range if the following two conditions exist:

1. There exists at least one transfer statement inside a DO that can cause control to pass out of this DO, or out of the nest if the DO is nested.

2. There exists at least one transfer statement, not inside any other DO, which can cause control to return into the range of this DO.

DO

If both of these conditions apply, the extended range is defined to be the set of all executable statements that may be executed between all pairs of control statements, the first of which satisfies the condition of (1) and the second of (2). The first of the pair is not included in the extended range; the second is.

A transfer statement may not cause control to pass into the range of a DO unless it is being executed as part of the extended range of that particular DO. Further, the extended range of a DO may not contain a DO that has an extended range or a DO with the same induction variable.

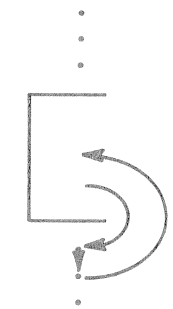
When a procedure reference occurs in the range of a DO the actions of that procedure are considered to be temporarily within that range; i.e., during the execution of that reference.

Note: Use of extended range DO's should be minimized especially when global optimization is desired.

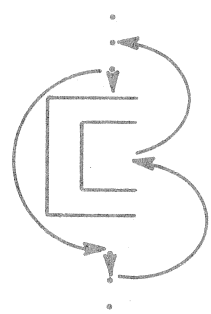
Example:

```
      .
      .
      DO 20 I = 1,K
      DO 20 J = N,M
      .
      .
      IF (J-JJ) 6,80,6
    6 .
      .
   20 CONTINUE
      .                  )
      .                  )
   80 .  .               )
      .                  )  extended range of nested DO
      .                  )
      GO TO 6
      .
      .
```

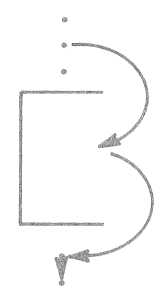
The following illustrate usage of the extended range of a DO. Examples 1 and 2 are permitted; example 3 is not permitted.



Example 1
Permitted



Example 2
Permitted



Example 3
Not permitted

DOUBLE PRECISION

DOUBLE PRECISION

The DOUBLE PRECISION statement is an explicit type statement with the following form:

DOUBLE PRECISION a_1*s_1(k_1) /d_1 /,..., a_n*s_n(k_n)/d_n/

where

a_i is a variable, array, or subprogram name whose characteristics are being qualified by this statement.

*s_i is an optional size-in-bytes qualification and is ignored.

k_i supplies dimension information necessary to allocate storage to arrays.

d_i represents initial data values.

This statement is used to declare real data with extended precision. Such data may also be declared via the REAL statement with a size qualifier of 8 or more.

The DOUBLE PRECISION statement is more fully described under the Type statement in this section.

ENCODE

ENCODE

The ENCODE statement is of the following form:

```
ENCODE(a,t) list
```

where t may be a format statement number, a character scalar, or an array name giving the format information for encoding; a is a character scalar, array element, or an array of any type, which specifies the starting location of the internal buffer; the list is as specified for the WRITE statement.

The ENCODE statement causes the data items specified by list to be converted to the character mode under control of the format specified by t; and placed in storage beginning at location a.

The number of characters caused to be generated by the format information and the list should not be greater than the size of a.

For examples:

```
i = 1
ENCODE (a,3)i
3 FORMAT (I4)
```

After execution array a contains (beginning with the first character position of a(1)):

```
ØØØ1
```

where Ø indicates a blank.

Additional information on the ENCODE statement is given in Section V under Internal Data Conversion.

END

END

END

The END statement specifies the physical end of the source program. It must be the last statement of every program and must be completely contained on that line. END creates no object-program instructions. It has the form:

```
END
```

ENDFILE

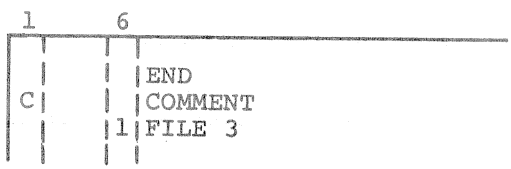
ENDFILE

This statement is operable only for sequential files. Its execution causes the indicated file to be closed with an end-of-file signal. For an output file, the buffer(s) is flushed and a file mark is written. Nothing is done for an input file. (The end-of-file signal is a unique record indicating demarcation of a sequential file). This statement has the form:

```
ENDFILE f
```

Where f is the file designator.

Note: If it is necessary to span two lines/cards for this statement, and if the break is between the letters D and F, then a comment line may not appear between the initial and continuation lines. Specifically, the following is not permitted:



ENTRY

The general form of the ENTRY statement is:

ENTRY name (b_1,b_2,...,b_n)
or
ENTRY name

name is the symbolic name of an entry point

Each b_i is a dummy argument corresponding to an actual argument in a CALL statement or in a function reference. An ENTRY into a FUNCTION subprogram must have at least one argument unless all input arguments required have been specified in a previous entry to the same FUNCTION subprogram.

An ENTRY into a SUBROUTINE subprogram may have arguments of the form * indicating nonstandard returns (dummy statement references).

The following rules apply to the use of multiple entry points:

1. All of the rules regarding adjustable dimensions given with Adjustable Dimensions.
2. In a FUNCTION subprogram, only the FUNCTION name may be used as the variable to return the function value to the using program. The ENTRY name may not be used for this purpose.
3. An ENTRY name may appear in an EXTERNAL statement in the same manner as a FUNCTION or SUBROUTINE name.
4. Entry into a subprogram initializes all references in the entire subprogram from items in the argument list of the CALL or function reference. (For instance, if, in the example that appears in the section Multiple Entry Points into a Subprogram of Section VI, entry is made at SUB2, the variables in statement 10 will refer to the argument list of SUB2.)
5. The appearance of an ENTRY statement does not alter the rules regarding the placement of arithmetic statement functions in subroutines. Arithmetic statement functions may follow an ENTRY statement only if they precede the first executable statement following the SUBROUTINE or FUNCTION statement.
6. None of the dummy arguments of an ENTRY statement may appear in an EQUIVALENCE or COMMON statement in the same subprogram.

EQUIVALENCE

EQUIVALENCE

The EQUIVALENCE statement is of the form:

EQUIVALENCE $(k_1),(k_2),\ldots,(k_n)$

where each k is a list of the form:

$a_1,a_2,\ldots,a_m$

Each a is either a variable name or an array element name (not a dummy argument), the subscript of which contains only integer constants or parameter symbols, and m is greater than or equal to 2. The EQUIVALENCE statement causes two or more variables, or arrays, to be assigned to the same storage location(s). EQUIVALENCE differs from common in that EQUIVALENCE assigns variables within the same program or subprogram to the same storage location; common assigns variables in different subprograms or a main program and a subprogram to the same locations.

One EQUIVALENCE statement can establish equivalence between any number of sets of variables. For example:

```
DIMENSION B(5),C(10,10),D(5,10,15)
EQUIVALENCE (A,B(1),C(5,4)),(D(1,4,3),E)
```

In this example, part of the arrays C and D are to be shared by other variables. Specifically, the variable A is to occupy the same location as the array element C(5,4), and the array B is to begin in this same location; the variable E shares location D(1,4,3) of the D array.

The following rules apply:

1. Each pair of parentheses in the statement list encloses the names of two or more variables that are to be assigned the same location during execution of the object program; any number of equivalences (sets of parentheses) may be given.

2. When using the EQUIVALENCE statement with subscripted variables, two methods may be used to specify a single element in the array. For example, D(1,2,1) or D(p) may be used to specify the same element, where D(p) references the p_i element of the array in storage. (See the discussion on Array Element Successor Function.)

3. Quantities or arrays that are not mentioned in an EQUIVALENCE statement will be assigned unique locations.

4. Locations can be shared only among variables, not among constants.

5. The sharing of locations requires a knowledge of which FORTRAN statements will cause a new value to be stored in a location. There are six such statements:

   a. Execution of an arithmetic statement stores a new value in the location assigned to the variable on the left side of the equal sign.

   b. Execution of a DO statement or an implied DO in an input/output list sometimes stores a new indexing value.

   c. Execution of a READ or DECODE statement stores new values in the locations assigned to the variables mentioned in the input list.

   d. Execution of an ENCODE statement stores new values in the character variable or array locations named as the internal buffer.

   e. Execution of a CALL statement or an abnormal function reference may assign new values to variables in COMMON or to arguments passed to that subprogram.

   f. An initial value may be stored in some location via a DATA statement, or a Data clause in a type statement.

6. Variables brought into a COMMON block through EQUIVALENCE statements may increase the size of the block indicated by the COMMON statements, as in the following example:

```
COMMON /X/A,B,C
DIMENSION D(3)
EQUIVALENCE (B,D(1))
```

   The layout of core storage indicated by this example (extending from the lowest location of the block to the highest location of the block) is:

```
A
B,D(1)
C,D(2)
  D(3)
```

EQUIVALENCE

7. Since arrays must be stored in consecutive forward locations, a variable may not be made equivalent to an element of an array in such a way as to cause the array to extend below the beginning of a COMMON block.

8. The rule for making double-word variables equivalent to single-word variables is:

   a. The effect of the EQUIVALENCE statements must be such that the high-order word of any double-word variable is an even number of locations away from the start of the data space to which it is allocated (COMMON or local).

   b. The effect of the EQUIVALENCE statements must be such that the high-order word of any double-word variable is an even number of words away from the start of any other double-word variable linked to it through EQUIVALENCE statements.

9. Two variables in one COMMON block or in two different COMMON blocks must not be made equivalent.

10. The EQUIVALENCE statement does not make two or more elements mathematically equivalent.

11. Equivalenced variables must not appear as dummy arguments in a FUNCTION, SUBROUTINE, or ENTRY statement.

EXTERNAL

The EXTERNAL statement has the following form:

EXTERNAL $a_1, a_2, \ldots, a_n$

where

a_i is a subprogram name whose characteristics are being qualified by this statement.

each a_i may be of the form:

f or f(ABNORMAL)

where f is the subprogram name. Use of the second form serves to define the subprogram f as both EXTERNAL and ABNORMAL (see the ABNORMAL statement, in this section).

FORTRAN permits the use of a subprogram name as an argument in a subprogram call. When this is done, the name must be included in an EXTERNAL statement in the calling program to distinguish the FUNCTION or SUBROUTINE name from a variable name. The following example illustrates this use in a main calling program and a subroutine subprogram:

Main Program

```
      EXTERNAL SIN, COS
      CALL SUBR (2.0, SIN, RESULT)
      WRITE (6, 10) RESULT
   10 FORMAT ("0 SIN(2.0) = ",F10.6)
      CALL  SUBR (2.0, COS, Result)
      WRITE  (6,20) RESULT
   20 FORMAT ("0 COS(2.0) = ", F10.6)
      STOP
      END
```

SUBROUTINE Subprogram

```
      SUBROUTINE SUBR (X,F,Y)
      Y = F(X)
      RETURN
      END
```

FORMAT

FORMAT

The FORMAT statement is used in conjunction with formatted input/output statements and the ENCODE and DECODE statements to provide conversion and editing information between the internal representation and the external character string.

A FORMAT statement has the form:

m FORMAT ($q_1 t_1 z_1 t_2 z_2 \cdots t_n z_n$)

or m FORMAT (V)

where

m is the statement number
q is a series of slashes or empty
t is a field descriptor or group of field descriptors
z is a field separator

The first form is used for formatted input/output under FORMAT control. The second form is used for formatted input/output under list control, and is generally called list directed input/output in this manual. The syntax of the READ, PRINT, and PUNCH statements make it possible to perform list directed I/O in either of two ways: by omitting a FORMAT reference (e.g., READ,) or by including a reference to a FORMAT statement of the second form. Only the second alternative is permitted when used in conjunction with a WRITE statement, since the syntax of WRITE requires a FORMAT reference.

When the first form is used the following field descriptors are permitted:

pr F w.d pr E w.d pr G w.d pr D w.d r O w r I w r L w	Numeric and Logical Field Descriptors
r A w r R w w H h1h2 ... hw "h1h2 ... hn" 'h1h2 ... hn'	Character Field Descriptors
Tt wX	Field Positioning Descriptors

FORMAT

where

p is an optional scale factor designator

r is an optional repeat count

w is the field width, expressed in number of characters

d is the number of fractional places (characters)

h is a single character

t is a character position, where the positions of a line/card are numbered 1 through the number present.

The F, E, and G descriptors are for REAL values, D is for DOUBLE PRECISION, O is for octal conversion, I is for INTEGER, L is used with LOGICAL values, A, R and H are for CHARACTER values, X and T are for skipping over text. The following briefly describes how these descriptors are formed. Note that the last three, H, T and X, do not require a variable in the input/output list; all others do.

```
Fw.d = Real mode without exponent
Ew.d = Real mode with exponent
Gw.d = F or E editing code is taken dependent on value
       of output item
Dw.d = Double precision mode
Ow   = Field occupies w print positions and is represented
       as an octal number of up to 12 digits.
Iw   = Integer mode and field occupies w print positions
Lw   = Right most position of field w contains T or F
       for logical variable
Aw   = Field occupies w print positions - Left justified data
Rw   = Field occupies w print positions - Right justified data
wH   = Hollerith field to occupy w print positions
Tt   = Next operation begins with position t of record
wX   = Field of width w is blank filled on output, skipped on input
```

See the Input/Output Section of this document for details on the fields of the FORMAT statement.

FUNCTION

FUNCTION

The FUNCTION statement is the first statement of a FUNCTION subprogram. The type of the function may be explicitly stated by preceding the word FUNCTION with the type, by the subsequent appearance of the function name in a type statement, or implicitly by the first letter of the function name. The FUNCTION statement has the forms:

FUNCTION name (a_1, a_2, ..., a_n)
REAL FUNCTION name (a_1, a_2, ... a_n)
INTEGER FUNCTION name (a_1, a_2, ..., a_n)
DOUBLE PRECISION FUNCTION name (a_1, a_2, ..., a_n)
COMPLEX FUNCTION name (a_1, a_2, ..., a_n)
LOGICAL FUNCTION name (a_1, a_2, ..., a_n)
CHARACTER FUNCTION name (a_1, a_2, ..., a_n)

where

name is the symbolic name of a single-valued function

the arguments a_1, a_2, ..., a_n (there must be at least one) are non-subscripted variable names or the dummy name of a SUBROUTINE or FUNCTION subprogram.

Examples:

```
FUNCTION ARCSIN (RADIAN)
REAL FUNCTION ROOT (A,B,C)
INTEGER FUNCTION CONST (ING,SG)
DOUBLE PRECISION FUNCTION DBLPRE (R,S,T)
COMPLEX FUNCTION CCOT (ABI)
LOGICAL FUNCTION IFTRU (D,E,F)
```

1. The FUNCTION statement must be the first statement of a FUNCTION subprogram. At least one dummy variable must be enclosed in parentheses.

2. The name of the function must appear at least once in some definitional context (see EQUIVALENCE statement). This name cannot be used in a NAMELIST or COMMON statement.

Example:

```
FUNCTION CALC (A,B)
.
.
.
CALC=Z+B
.
.
.
RETURN
END
```

By this method the output value of the function is returned to the calling program.

The calling program is the program in which the function is referred to or called.

The called program is the subprogram that is referred to or called by the calling program.

3. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the function reference in the calling program. The actual arguments must correspond in number, order, size and type with the dummy arguments.

4. When a dummy argument is an array name, a statement with dimension information must appear in the FUNCTION subprogram; also, the corresponding actual argument must be a dimensioned array name.

5. None of the dummy arguments may appear in an EQUIVALENCE, NAMELIST, or COMMON statement in the FUNCTION subprogram.

6. The FUNCTION subprogram must be logically terminated by a RETURN statement (see Returns from Subprograms) and physically terminated by an END statement.

7. The FUNCTION subprogram may contain any FORTRAN statements except SUBROUTINE, BLOCK DATA, another FUNCTION statement, or a RETURN statement with an alternate return specified (e.g. RETURN 1).

8. The actual arguments of a FUNCTION subprogram may be any of the following:

   a. A constant.

   b. A subscripted or nonsubscripted variable or an array name.

   c. An arithmetic or a logical expression.

   d. The name of a FUNCTION or SUBROUTINE subprogram.

   e. An omitted or null argument, indicated by successive commas. References to null arguments from within the called Function are undefined.

9. A FUNCTION subprogram is referred to by using its name as an operand in an arithmetic expression and following it with the required actual arguments enclosed in parentheses.

10. A FUNCTION subprogram may not call itself, either directly or indirectly through some other called subprogram.

11. A FUNCTION name must be unique to 6 characters.

See Table 6-2 for the supplied FUNCTION subprograms.

The following example shows the use of a FUNCTION subprogram:

```
Calling Program                 Called Program

        .                       FUNCTION CALC (A,B)
        .                         .
        .                         .
X=Y**2+D*CALC(F,G)              CALC= ...
        .                         .
        .                         .
        .                         .
                                RETURN
                                END
```

GO TO

GO TO, Unconditional

The unconditional GO TO indicates the next statement to be executed. It has the form:

```
GO TO K
```

where K is the statement number of another statement in the program. When this statement is encountered, the next statement to be executed will be the statement having statement number K. This statement can be any executable statement in the program either before or after the GO TO statement subject to the rules for transferring into and out of DO loops. For example:

```
GO TO 5
```

The program continues execution with statement number 5. Control is transferred unconditionally to statement number 5.

GO TO, Assigned

The assigned GO TO statement indicates which statement is the next to be executed. The assigned GO TO has the form:

```
    GO TO I, (K1,K2,...,Kn)
or  GO TO I
```

where

I is an integer switch variable
K_i are statement numbers

The K's are optional. If present, then at the time of execution of an assigned GO TO statement the variable I must have been assigned the value of one of the statement numbers in the parentheses. The next statement to be executed will be the one whose statement number in the parentheses has the same value as the variable I. If statement I is not in the list of K's, a run time diagnostic is generated.

When the K's are not present (form 2 above), no validation of I takes place. Control transfers directly to statement I.

GO TO

For example:

```
ASSIGN 17 TO J
GO TO J, (5,4,17,2)
```

Statement number 17 is executed next.

GO TO, Computed

The computed GO TO indicates the statement that will be executed next. This is determined by using a computed integer value. It has the following form:

GO TO $(K_1,K_2,\ldots,K_n),e$

where the K_i are statement labels or switch variables. The expression e is truncated to an integer at the time of execution. The next statement to be executed will be Ki where i is the integral value of the expression e.

For example:

```
J = 3
GO TO (5,4,17,1),J
```

Statement 17 is executed next.

IF, ARITHMETIC

IF, ARITHMETIC

The arithmetic IF statement causes a change in the execution sequence of statements depending on the value of an arithmetic expression. It has the following form:

IF (e) k_1,k_2,k_3

where e is an arithmetic expression and the k_i are statement numbers, switch variables or are null (not supplied). When k_i is null, the statement referenced is the next executable statement in the program.

The arithmetic IF is a three-way branch. Execution of this statement causes a transfer to one of the statements k_1, k_2, or k_3. The statement identified by k_1, k_2, or k_3 is executed next depending on whether the value of e is less than zero, zero, or greater than zero, respectively. Any two of k_1, k_2, and k_3 are optional, and if null, cause the execution of the program to continue with the next sequential executable statement after the IF statement.

Example:

```
IF   (A(J,K)-B  10,4,30

IF   (A(J,K)-B)  0 control goes to statement 10
IF   (A(J,K)-B) =0 control goes to statement 4
IF   (A(J,K)-B)  0 control goes to statement 30
```

IF, LOGICAL

IF, LOGICAL

The logical IF statement causes conditional execution of a certain statement depending on whether or not a logical expression is true or false. It has the following form:

```
IF(e)s
```

where e is a logical or relational expression and s is any executable statement except a DO statement or another logical IF statement. Upon execution of this statement, the logical or relational expression e is evaluated. If the value of e is true, statement s is executed. If the value of e is false, control is transferred to the next sequential statement.

Example: IF(A.GT.B) GO TO 3

If A is arithmetically greater than B, the execution of the user program continues with the statement labeled with 3. Otherwise execution continues with the next sequential executable statement.

If e is true and s is a CALL statement that does not take a nonstandard return, control is transferred to the next sequential statement upon return from the subprogram.

The following examples illustrate several uses of the logical IF.

```
1.  IF (A.AND.B) F = SIN (R)
2.  IF (16.GT.L) GO TO 24
3.  IF (D.OR.X.LE.Y) GO TO (18,10),I
4.  IF (Q) CALL SUB
```

In example 1, if (A.AND.B) is true, compute F and return to the statement following IF.

In example 2, if (16.GT.L), control transfers to statement 24.

In example 3, if (D.OR.X.LE.Y) is true, control transfers to statement 18 or 20 depending upon whether I is 1 or 2.

In example 4, Q must have been previously typed as LOGICAL. If its current value is true, control goes to the subprogram SUB. Return is to the statement following the IF.

If the operator .NE. or .EQ. is contained in a logical IF expression and both operands are not type integer or character, a warning message will appear at the end of the source listing. The error message indicates that the equality or non-equality relation between the operands may not be meaningful. This is due to the fact that floating point arithmetic is not exact for certain fractions.

IMPLICIT

The IMPLICIT type statement is of the form:

IMPLICIT type*s(l_1, l_2,...),type*s(l_1, l_2,...)

where each ln is a letter or pair of letters (separated by a dash) of the alphabet.

Type is one of the following:

INTEGER, REAL, COMPLEX, DOUBLE PRECISION,
LOGICAL, OR CHARACTER

*s is optional and designates a length specification for its associated data type. Length specifications are ignored if type is INTEGER, DOUBLE PRECISION, COMPLEX, or LOGICAL. When type is REAL, a length specification of 8 or more implies DOUBLE PRECISION: when type is CHARACTER the length specification is as defined for the CHARACTER statement.

The IMPLICIT statement is used to redefine the implicit typing. All variable names beginning with a letter specified in the list or included in the alphabetic interval defined by two letters separated by a dash are typed as specified in the "Type" field. An IMPLICIT statement supersedes previous implicit statements. The IMPLICIT statement must appear before any use of the variable name being typed.

For example:

```
IMPLICIT INTEGER(A-F,X,Y)
```

Any variable name first appearing in the program following this statement, which begins with the letters A through F, X and Y are implicitly typed INTEGER. This also applies to the lower case letters a through f, x, and y.

INTEGER

The INTEGER statement is an explicit type statement with the following form:

INTEGER $a_1*s_1(k_1)/d_1/,a_2*s_2(k_2)/d_2/,\ldots,a_n*s_n(k_n)/d_n/$

where

a_i is a variable, array, or FUNCTION subprogram name whose characteristics are being qualified by this statement.

$*s_i$ is an optional size-in-bytes qualification and is ignored.

k_i supplies the dimension information necessary to allocate storage to arrays.

d_i represents initial data values.

The INTEGER statement is more fully described under the Type statement entry in this section.

LOGICAL

The LOGICAL statement is an explicit type statement with the following form:

LOGICAL a_1*s_1(k_1)/d_1/,...,a_n*s_n(k_n)/d_n/

where

- a_i is a variable, array, or subprogram name whose characteristics are being qualified by this statement
- *s_i is an optional size-in-bytes qualification, and is ignored
- k_i supplies the dimension information necessary to allocate storage to arrays
- d_i contains the initial data values

The LOGICAL statement is more fully described under the Type statement in this section.

NAMELIST

NAMELIST

The NAMELIST statement has the following form:

$$NAMELIST/n_1/k_1/n_2/k_2/.../n_i/k_i$$

where each n_i is a NAMELIST name and each k_i contains lists of variables and/or array names to be associated, for input/output purposes, with the corresponding NAMELIST names.

The following rules apply to the NAMELIST statement:

1. A NAMELIST name consists of one to eight alphanumeric characters; the first character must be alphabetic. The name must be unique within the first six characters.

2. A NAMELIST name is enclosed in slants. The field of entries belonging to a NAMELIST name ends wither with a new NAMELIST name enclosed in slants or with the end of the NAMELIST statement.

3. A variable name or any array name may belong to one or more NAMELIST names. Such variable names may also be of one to eight characters providing they are unique within the first six.

4. A NAMELIST name must not be the same as any other name in the program.

5. A NAMELIST statement defining a NAMELIST name must precede any reference to the name in the program.

6. A dummy argument of a subprogram cannot be used as a variable in a NAMELIST statement.

7. The NAMELIST table can accommodate array variables of no more than seven dimensions.

In the following examples, the arrays A, I, and L and the variables B and J belong to the NAMELIST name, NAM1; the array A and the variables C, J, and K belong to the NAMELIST name, NAM2.

```
DIMENSION A(10), I(5,5), L(10)
NAMELIST /NAM1/A,B,I,J,L/NAM2/A,C,J,K
```

Additional information on Namelist input/output statements is contained in Section V.

PARAMETER

The PARAMETER statement has the form:

PARAMETER $v_1=e_1, v_2=e_2,\ldots,v_n=e_n$

where v_i is a PARAMETER symbol and e_i are arithmetic expressions involving only constants and previously defined PARAMETER symbols.

The PARAMETER statement is used to define program constants with the result of an expression at compilation time. The value of a PARAMETER symbol may not be redefined during the execution of a program. A PARAMETER symbol must not appear where a constant cannot appear.

The appearance of a parameter symbol in some context is interpreted as if its equivalent value had appeared instead.

A PARAMETER symbol v may be of type INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL or CHARACTER depending on the type of its defining expression e. In the following examples I and J are of type INTEGER, k is REAL, L is LOGICAL and M is CHARACTER.

```
PARAMETER I=5/2, J=I*3, K = 3.14159, L=.T., M="060171"
```

The PARAMETER symbol I is initialized to the value 2, the PARAMETER symbol J is initialized to 6, and the parameter symbol K is initialized to the real value 3.14159. L has the value .TRUE., while the parameter symbol M is assigned a CHARACTER equivalence.

The significant difference between a PARAMETER symbol and, say, an ordinary integer variable which may be initialized with a DATA statement is in the usage. For example, a PARAMETER variable may be used to supply dimensionality information.

```
      PARAMETER  I = 20
      PARAMETER  J = I*4
      DIMENSION  A(I,J)
        .
        .
        .
      DO  100 II = 1,I
      DO  100 JJ = 1,J
100   A (II,JJ) = 0.
```

In this example, A is not an adjustably dimensioned array. It has constant dimensions of 20 and 80 respectively. The two DO statements have constant m values of 20 and 80 respectively. I and J are compile time variables, while II and JJ are execute time variables. The program properties change as the value of the parameter symbol I changes. To operate on a 10X40 array, only the first line needs to be changed.

PAUSE

PAUSE

The PAUSE statement causes a temporary halt in the execution of the program until the operator resumes execution. A line is transmitted to the operator console or user terminal (in time sharing mode) consisting of the word "PAUSE", and information derived from the PAUSE statement. When the user transmits a carriage return, execution is continued with the statement following the PAUSE. If the user transmits "STOP", (or anything beginning with the letter S) execution halts. It has the form:

```
PAUSE
  or
PAUSE n
```

where n is an integer or character constant or variable. Integer values are limited to five digits.

Examples:

```
PAUSE
PAUSE  77777
PAUSE  I
PAUSE "TØØ BAD"
```

For PAUSE and PAUSE n, where n is an integer, the message displayed is

```
PAUSE ..... SNUMB snumb - nn
```

The line number field (.....) will contain the line number of the PAUSE statement or the integer n; snumb will be the SNUMB of the job, nn is the activity number.

For PAUSE n where n is character information, the message displayed is

```
PAUSE  .....
```

where the ..... field is the character information.

PAUSE

For example:

```
      SUBROUTINE  PAWS (IDENT,MESSAGE)
      CHARACTER  MESSAGE*8
      IF (IDENT),100,
      PAUSE  IDENT
      RETURN
100   PAUSE MESSAGE
      RETURN
      END
```

A call to the above subroutine of the form:

```
      CALL PAWS  (77777,0)
```

might display:

```
      PAUSE 77777 SNUMB 1234T-02
```

A call of the form:

```
      CALL PAWS (0, "ERROR 27")
```

would display:

```
      PAUSE ERROR 27
```

PRINT

PRINT

```
PRINT, list
```

This form of the PRINT statement is used for list-directed formatted output to the standard system output device. For a complete discussion of list-directed input/output see Section V of this document, List Directed Input/Output Statements.

```
PRINT t, list
      or
PRINT t
```

The formatted PRINT statement causes information (list) to be transmitted to the standard output device and converted according to the format specified in t. The first character of each record supplied is a control character.

To be classified as a formatted PRINT, t must be a FORMAT statement number, a character scalar, or an array name.

```
PRINT x
```

The NAMELIST PRINT statement causes the printout of information at the standard output device in accordance with the NAMELIST group x. For a complete description of NAMELIST input/output see Section V of this document.

To be classified as a NAMELIST PRINT, x must be a NAMELIST name.

PUNCH

PUNCH

```
PUNCH t, list
    or
PUNCH t
```

The formatted PUNCH statement causes information in punchable form to be transmitted to the standard output device, converted according to the format specified in t. (See FORMAT Statement.) To be classified as a formatted PUNCH, t must be a FORMAT statement number, a character scalar or an array name.

```
PUNCH, list
```

This form of the PUNCH statement is used to transmit list directed formatted output in punchable form to the standard output device. See Section V of this document for a complete description of list directed input/output.

```
PUNCH x
```

This NAMELIST PUNCH statement, where x is a NAMELIST name, causes formatted punchable information to be directed to the standard output device. See Section V of this document for a complete description of NAMELIST input/output.

READ

READ

```
READ, list
```

This form of the READ statement is used for list directed formatted input from the standard system input device. For a complete discussion of list directed input/output see Section V of this document.

```
READ t, list
     or
READ t
```

This statement enables the user to read a list referencing format information (t) that describes the type of conversion to be performed. A request is sent to the standard input device. The input is converted according to the format specified in t. The t field may be a FORMAT statement number, a character scalar or an array name.

```
READ x
```

This is a NAMELIST input statement where x is a NAMELIST name. This statement causes a read request to be sent to the standard input device. Input in NAMELIST input format will be accepted. See Section V for a complete description of NAMELIST input/output.

```
READ (f,t,opt1,opt2)list
```

This statement, formatted file READ, includes a reference to format information (t) and a file reference (f). It may include either or both options (opt1 and opt2) and a list specification. The file reference (f) may be an integer constant variable or expression. A file reference of 5 or 41 implies reference to the standard system input device.

The end-of-file transfer (opt1) option is designated as: END=S1. Where S1 is the statement label that is to receive control upon encountering an end-of-file.

The Error Transfer (opt2) option is designated as: ERR=S2. Where S2 is the statement label that is to receive control when an error condition is encountered.

The options may appear in any order and S1 and S2 may be statement numbers or switch variables.

The format reference (t) may be the statement number of a FORMAT statement, a character scalar, or an array name.

```
READ (f,opt1,opt2)list
```

The unformatted file READ statement is the same as the formatted file READ except the FORMAT reference is omitted. This statement applies to word oriented serial access files (binary sequential files).

```
READ (f'n,opt2)list
```

This unformatted file READ is for random binary files. The n is an integer constant, variable or expression that specifies the sequence number of the logical record to be accessed.

```
READ (f,x,opt1,opt2)
```

The NAMELIST file READ statement has a reference to some NAMELIST name (x) and the list is omitted. This statement causes formatted input to be read in accordance with NAMELIST group(x).

REAL

The REAL statement is one of the explicit type statements with the following form:

REAL $a_1*s_1(k_1)/d_1/,\ldots,a_n*s_n(k_n)/d_n/$

where

a_i is a variable, array, or subprogram name whose characteristics are being qualified by this statement

$*s_i$ is the size specification. If it is greater than 7, the type is treated as DOUBLE PRECISION

k_i supplies dimension information necessary to allocate storage to arrays

d_i is the initial data values

The REAL statement is more fully described under the Type statement in this section.

RETURN

RETURN

The logical termination of any subprogram is the RETURN statement, which returns control to the calling program. There may be any number of RETURN statements in the program.

A RETURN statement is of the form:

```
RETURN
RETURN i
```

where i is an integer constant or variable which denotes the ith non-standard return in the argument list, reading from left to right of the CALL statement which invoked this (returning) subroutine. The value of i must be a positive integer no greater than the number of non-standard returns in that argument list. When i has the value zero, a normal return is taken (form 1 of the RETURN statement shown above).

REWIND

This statement refers only to sequential files. It causes the specified file to be positioned at its initial point. The statement has the following form:

```
REWIND f
```

where f is the file reference. f may be an integer constant or variable.

If the file is an output file, an EOF is written before rewinding.

STOP

The STOP statement causes the object-program to halt and control to be returned to the operating system. It has the forms:

```
STOP n
STOP
```

where n is an integer or character constant or variable.

The action taken when a STOP statement is executed varies with batch and TSS execution, and the presence of n. STOP n will print on the standard system output device:

```
STOP AT LINE n
   or
STOP n
```

the former being displayed when n is an integer, the latter when n is character. STOP with no identification (n) goes unrecorded and the program simply terminates when executed in batch; under time sharing the message

```
NORMAL TERMINATION
```

is displayed on the teletypewriter and execution terminates.

SUBROUTINE

The SUBROUTINE statement must be the first statement of a SUBROUTINE subprogram. The SUBROUTINE statement has the following form:

```
SUBROUTINE name (a1, a2, ...,ai)
  or
SUBROUTINE name
```

where

Name is the symbolic name of a subprogram and must be unique within the first six characters.

Each a_i (if present) is a nonsubscripted variable name, the dummy name of a SUBROUTINE or FUNCTION subprogram, an * or $ indicating a non-standard return.

Examples:

```
SUBROUTINE COMP (X,Y,*,$,P)
SUBROUTINE QUADEQ (B,A,C,ROOT1, ROOT2)
SUBROUTINE OUTPUT
```

1. The SUBROUTINE statement must be the first statement of a SUBROUTINE subprogram.

2. The SUBROUTINE subprogram may use one or more of its arguments to return output. The arguments so used must appear in some definitional content within the subprogram other than a DATA statement (which is not allowed). See the EQUIVALENCE statement rule 5 for a definition of the contexts.

3. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the CALL statement which refers to the SUBROUTINE subprogram. The actual arguments must correspond in number, order, size and type with the dummy arguments.

4. When a dummy argument is an array name, a statement containing dimension information must appear in the SUBROUTINE subprogram; the corresponding actual argument in the CALL statement must be a dimensioned array name.

SUBROUTINE

5. No argument in a SUBROUTINE statement may also be included in COMMON, EQUIVALENCE, NAMELIST, or DATA statements in the subprogram.

6. The SUBROUTINE subprogram must be logically terminated by a RETURN statement and physically by an END statement.

7. The SUBROUTINE subprogram may contain any FORTRAN statements except FUNCTION, BLOCK DATA, or another SUBROUTINE statement.

8. The character * appearing in an argument position denotes a non-standard or alternate exit from the subroutine.

TYPE

TYPE

The explicit type statements are of the form:

Type *ba_1*$s_1(k_1)/d_1/,a_2$*$s_2(k_2)/d_2/,\ldots,a_n$*$s_n(k_n)/d_n/$

where:

Type is one of the following: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, or CHARACTER.

b is a positive integer constant and applies to all variables in the statement unless otherwise specified. For example:

```
CHARACTER *10C1,C2,C3*12,C4
```

C1,C2, and C4 have a length of 10 characters. C3 has a length of 12 characters.

a_i is a variable, array, or FUNCTION subprogram name whose characteristics are being qualified by the type statement.

*s_i is an optional size specification. This field is ignored for all types except:

1. for REAL type, a size specification, *s, greater than 7 is treated as DOUBLE PRECISION.

2. for CHARACTER type, the size field, *s, is interpreted as the maximum number of characters that may be contained by the CHARACTER element being defined. When this field is omitted, the size is assumed to be 6 for BCD programs and 8 for ASCII. S may not be greater than 511.

   An adjustable size specification for s is permitted within a subprogram when both the character variable and its size parameter(s) are included as format parameters. Additional information on adjustable size specifications is contained under the CHARACTER statement in this section.

(k_i) if present consists of from one to seven integer constants, PARAMETER symbols, or (for subprograms only) INTEGER variables separated by commas. This field supplies dimension information (information necessary to allocate storage to arrays). If this information does not appear in the type statement, it must appear elsewhere in the program (in a DIMENSION or COMMON statement).

/d_i/ represents initial data values. The form for d_i is as specified for the DATA statement.

Thus, meaningful permutations of the above permit:

INTEGER $a_1(k_1)/d_1/,a_2(k_2)/d_2/,....,a_n(k_n)/d_n/$

REAL *b $a_1*s_1(k_1)/d_1/,a_2*s_2(k_2)/d_2/,...,a_n*s_n(k_n)/d_n/$

DOUBLE PRECISION $a_1(k_1)/d_1/,a_2(k_2)/d_2/,...,a_n(k_n)/d_n/$

COMPLEX $a_1(k_1)/d_1/,a_2(k_2)/d_2/,...,a_n(k_n)/d_n/$

LOGICAL $a_1(k_1)/d_1/,a_2(k_2)/d_2/,...,a_n(k_n)/d_n/$

CHARACTER *b $a_1*s_1(k_1)/d_1/,a_2*s_2(k_2)/d_2/,...,a_n*s_n(k_n)/d_n/$

WRITE

```
WRITE (f,t,opt2) list
```

The formatted file WRITE statement must include a file reference (f) and a FORMAT reference (t). It may include the option opt2 and a list reference.

The file reference may be an integer constant, variable, or expression. A designation of 6 or 42 implies the system standard output printing device. A designation of 43 implies the system standard output punching device.

The FORMAT reference (t) refers to the statement label of a FORMAT statement, a character scalar, or an array name.

The error transfer (opt2) option is designated as: ERR=S2, where S2 is the statement label or switch variable that is to receive control when an error condition is encountered.

```
WRITE (f,opt2) list
```

The unformatted file WRITE statement omits the format reference. It applies to the output of word oriented serial access files (binary sequential files). The f, opt2, and list fields are as specified for the formatted file WRITE.

```
WRITE (f'n,opt2) list
```

The random binary file WRITE statement contains a field (n) that specifies the sequence number of the logical record to be written. n may be a constant, variable, or expression and must be integer. The f,opt2, and list fields are as specified for the formatted file WRITE.

```
WRITE (f,x,opt2)
```

The namelist file WRITE statement includes a reference to the NAMELIST name (x). This statement causes character oriented records to be written on the indicated device. The f and opt2 fields are as specified for the formatted file WRITE; no list field is included in the namelist file WRITE. See Section V for a complete description of NAMELIST input/output.

SECTION V

INPUT AND OUTPUT

GENERAL DESCRIPTION

FORTRAN input/output statements specify the transmission of information between internal storage and input/output devices.

The ENCODE and DECODE statements, while not actually input/output statements, are of the same general form as formatted file WRITE and READ statements, respectively. They differ in that the file reference field of the READ/WRITE statements provides a storage reference for packed character information with DECODE/ENCODE. The information in this section contains general information for all of these statements.

The following notation is used in the description of input/output statements.

```
list = indication of an input/output list
   f = designation of a file reference
   t = reference to the FORMAT information
   x = reference to a NAMELIST name
   a = reference to an internal storage buffer
       for packed character data
 opt = optional transfer condition.
```

Each input/output statement may contain an implicit (NAMELIST) or explicit list of variable names, arrays, and array elements. The named elements are assigned values on input (or DECODE) and have their values transferred on output. With ENCODE, the values are converted to character information and stored in (a).

The list of a WRITE, PRINT, PUNCH, or ENCODE statement may also include constants and expressions of all types.

The file reference (f) may consist of an integer constant, variable, or expression which identifies the input/output unit. The value of the integer will be a 2-digit file code, the value of which must be in the range $1 \leq f \leq 43$. The association of a file with a device is by use of the $ file GCOS control cards or by the fe list of the RUN command described in this manual.

A statement number of a FORMAT statement, a character scalar name, or an array name may be represented by t. If a statement number is represented, the identified FORMAT statement must appear in the same program unit as the input/output statement. If a name is referenced, the variable must contain FORMAT information (See Variable Format Specifications in this section).

NAMELIST input/output is indicated by the presence of a NAMELIST name (x) in the format reference position of the READ, WRITE, and PRINT statements. The NAMELIST statement(s) defining x and its associated list must appear before any input/output statements referencing x.

The internal storage buffer, a, applies only to the ENCODE and DECODE statements. Array names of any type or character variables (scalar or array element) may be used, however, the latter is preferred.

There are two optional transfer conditions: end-of-file and error. These are designated as END= and ERR= respectively. END= may appear in sequential file input statements only; ERR= may appear in any input/output statement. A statement number or switch variable name may follow the equal sign (=). The order of the transfer conditions is not important. Conditions which can give rise to an error return include transmission errors or any of the error conditions described in the File and Record Control manual under Error Procedures - User Supplied Routine.

The information transmitted is collected into records which may be formatted or unformatted. A formatted record consists of a string of permissible characters in the character set. The transfer of such a record requires that FORMAT information be referenced, or implied, to supply the necessary positioning and conversion specifications. The number of records transferred by the execution of a formatted I/O statement is determined by the list and the referenced FORMAT statement. A formatted record may be analogous to a print line or a card image. An unformatted record consists of a string of words.

List Directed formatted input/output may be specified by a FORMAT statement of the form FORMAT(V) or may be implied by the form and content of the input/output statement.

Input/Output statements are grouped as follows:

1. System device input/output statements.

   a. Formatted Input - Permits entering information from the standard input device with reference to a FORMAT statement.

   b. Formatted Output - Permits transfer of information to the standard output device with reference to a FORMAT statement.

   c. Formatted Punch - Permits transfer of information in punchable form to the standard output device with reference to a FORMAT statement.

   d. List Directed Input - Permits entering information from the standard input device without reference to a FORMAT statement.

   e. List Directed Output - Permits transfer of information to the standard output device without reference to a FORMAT statement.

   f. List Directed Punch - Permits transfer of punchable information to the standard output device without reference to a FORMAT statement.

   g. NAMELIST Input - Permits entering information from the standard input device with reference to a NAMELIST name.

   h. NAMELIST output - Permits transfer of information to the standard output device with reference to a NAMELIST name.

2. File Input/Output Statements

   a. Formatted Read/Write Statements - These statements include a FORMAT reference, the file reference, they may include an end-of-file option an error return option, and/or a list specification. List directed I/O is is accomplished via the FORMAT (V). Namelist I/O accomplished with a NAMELIST name as a format reference.

   b. Unformatted Read/Write Statements - These statements refer to binary word oriented sequential and random files.

3. Manipulation Input/Output Statements - These statements are for file operations relating to positioning and file demarcation, and may be used to operate on sequential access files only.

4. FORMAT and NAMELIST statements - These two nonexecutable statements, are used with the formatted input/output statements.

   The FORMAT statement specifies the arrangement of data in the input/output record. If the FORMAT statement is referred to by a READ statement, the input data must meet the specifications described in Data Input Referring to a FORMAT Statement.

   The NAMELIST statement specifies an input/output list of variables and/or arrays. Input/Output of the values associated with the list is effected by reference to the Namelist name in a READ, PRINT or WRITE statement. If the NAMELIST name is referred to by a READ statement, the input data must meet the specifications described in DATA Input Referring to a NAMELIST Statement.

File Designation

In the source program, files may be designated by any integer expression, the value of which must be in the range of 1 to 43. The equation of a numeric file designation with some actual device is accomplished via standard GCOS file allocation control cards, using a two digit file code of the same integer value as the corresponding file designator. Thus WRITE (6,100) will reference file code 06 at run time.

Since the file designator may be any integer expression, the following statements will also reference file code 06.

```
I = 5

WRITE (I+1, 100)
```

Five specific file designators are predefined for all FORTRAN programs (these definitions may be overridden by the programmer): 05 designates the standard input file (file code I*); 06 refers to the standard output file (file code P*); and 41, 42, and 43 are referenced by the READ, PRINT, and PUNCH statements: 41 references the standard input file (I*); 42 the standard output file (P*) with printer destination; and 43 the standard output file (P*) with punch destination.

The default assignment of devices for the standard system file designators 05, 06, 41, 42, and 43 is as described above when the execution is done in a batch environment. In time sharing the default device of all of these file codes is the time sharing terminal. Allocation of actual files (versus the terminal) for one or more file designators is accomplished via the fe file list of the RUN command. In this list, the user supplies a descriptor for each file to be used by the object program. The names of all such files must be a 2-digit file code (nn) in the interval $01 \leq nn \leq 44$. Unless the file has been created with a name which is the 2-digit file code, it will be necessary to specify the file code as an alternate name. Suppose, for example, a program has an input statement of the form: READ (5,100) I,J,K. Normal time sharing execution of this program will access the terminal for input values for I, J, and K. However, if the program is initiated with a RUN command such as

```
RUN PROGRAM # INPUT "05"
```

then the user's catalog is searched for the file named INPUT, that file is accessed, and the AFT name for the file will be 05. Execution of the above READ statement will thus read the file INPUT for its values of I, J, and K.

Conversely, given a statement of the form READ (10,100)... where typical operations would be on a file, the user may specify terminal input by omitting the catalog file descriptor. e.g.,

```
RUN PROGRAM # "10"
```

If any given file descriptor consists only of an unquoted 2-digit logical file code, a temporary file will be created for the user unless a quick-access permanent file with the same name already exists. The PERM command can subsequently be used to make the temporary file permanent. Alternatively, such temporary files can be made permanent at the time the user logs off.

For example:

```
RUN PROGRAM #10
```

If no file exists in the user's catalog of the name 10, a temporary file will be created with that name, and I/O directed to file designator 10 will be routed to the temporary file.

More detail on the fe list and file allocation for time sharing are given in Section III in the discussion of the RUN command.

*

List Specifications

When arrays or variables are to be transmitted, an ordered list of the quantities to be transmitted must be included either in the input/output statements or the referenced NAMELIST statements. The order of the input/output list must be the same as the order in which the information exists or is to exist on the input/output medium.

An input/output list is a string of list items separated by commas. A list item may be:

1. An expression (output only)
2. An implied DO
3. An array name
4. A scalar
5. A constant (output only)
6. An array element

An input/output list is processed from left to right. Parenthesized sublists are permitted only with implied DO's; redundant parentheses will result in a fatal diagnostic.

Examples: A, B, C*D**E, 1.2, SQRT (14.6), F(K,K)

Consider the following input/output list:

```
A,B(3),(C(I),D(I,K),I=1,10),
((E(I,J), I=1,10,2),F(J,3),J=1,K)
```

This list implies that the information in the external input/ouptut medium is arranged as follows:

```
A,B(3),C(1),D(1,K),C(2),D(2,K),....,C(10),D(10,K),
E(1,1),,E(3,1),....,E(9,1),F(1,3),
E(1,2),E(3,2),....,E(9,2),F(2,3),E(1,3),...,F(K,3)
```

The execution of an input/output list is exactly that of a DO loop, as though each left parentheses (except expression and subscripting parentheses) were a DO, with indexing given immediately before the matching right parentheses, and the DO range extending up to that indexing information. The order of the input/output list above may be considered equivalent to the following:

```
A
B(3)
DO 5 I=1,10
C(I);5 D(I,K)
DO 9 J=1,K
DO 8 I=1,10,2;8 E(I,J);9 F(J,3)
```

Any number of quantities may appear in a single list. Essentially, it is the list that controls the quantity of data transmitted. If more quantities are in some input record that in the list, only the number of quantities specified in the list are transmitted, and the remaining quantities are ignored. Conversely, if a list contains more quantities than are given in one input record, more records are read or blanks are supplied or both depending on the FORMAT statement. In this case, blanks are supplied until the FORMAT triggers the record advance. Thus given a list of known length and a well defined FORMAT, it can be accurately predicted how many records will be read, regardless of the record lengths on file.

By specifying an array name in the list of an input/output statement or a NAMELIST, an entire array can be designated for transmission between core storage and an input/output device. Only the name of the array need be given and the indexing information may be omitted. For example:

```
DIMENSION A(5,5)
    .
    .
READ,A
```

In the above example, the READ statement shown will read the entire array A; the array is stored in column order in increasing storage locations, with the first subscript varying most rapidly, and the last varying least rapidly.

LIST DIRECTED FORMATTED INPUT/OUTPUT STATEMENTS

The following input/output statements enable a user to transmit a list of quantities without reference to a NAMELIST name or a detailed FORMAT specification. The type of each variable in the list determines the conversion to be used.

```
READ  t, list
PUNCH t, list
PRINT t, list
READ   , list
PRINT  , list
PUNCH  , list
READ  (f,t,opt1, opt2) list
WRITE (f,t,opt2) list
```

In all cases where a format reference (t) is supplied, the format must be of the form (v). The t may be a FORMAT statement number, a character scalar, or an array name. The table of implied format conversions used for list directed formatted input/output is as follows:

TYPE OF VARIABLE	INPUT	OUTPUT
Real	E (or F) w.d	0PE 16.8
Integer	Iw	I16
Logical	Lw	L2
Double-Precision	D w.d	0PD 26.18
Complex	2Fw.d	0P2E16.8
Character	Am	Am

m = maximum size

With list directed formatted input, record control is determined solely by the list. If some record (terminal input line, for example) is terminated and the list is not satisfied, another record (line) will be read. This process will continue until the list is satisfied.

The input information must satisfy the following rules:

1. Numeric and character input values are separated by commas or blanks.

2. Blanks following exponent indicators E, D, or G are not considered as separators.

3. Quotes (") may be used to bracket a character input value which contains embedded blanks or commas.

4. A given input value must be fully contained on one input line.

5. Consecutive commas, an empty line, or the appearance of a comma as the last character of a line imply null input fields. Conversion of a null field is a function of the corresponding list item type and is shown in the following table:

TYPE	VALUE
Integer	0
Real	0.0
Double-Precision	0.D0
Complex	(0,0)
Logical	F
Character	all blanks

6. When the input device is a time sharing terminal, an end-of-file condition may be signaled by transmitting a file separator character (control, shift, L) as the only character of a line (other than the terminal carriage return).

With list directed formatted output, record control is determined by the list and the standard line lengths. With BCD files, the standard line length is 132 characters; with ASCII files, the standard length is 72 characters. A new line/record is started when the next list item to be transmitted will not fit entirely on the current line. For example; if information has been formatted to character position 60 of some ASCII line and the next item in the list is an integer (implied I16 format), a new line will be started.

Namelist Input/Output Statements

The NAMELIST statement and various forms of the NAMELIST input and output statements provide for the input and output of groups of variables and arrays by referring to a single name. NAMELIST names must conform with the same naming rules as normal variables and arrays except there is no type associated with the name and the name must be unique within 6 characters. A NAMELIST name must not be the same as any other variable procedure or array name in the subprogram defining it.

Each list that is mentioned in the NAMELIST statement is given a NAMELIST name. Therefore, only the NAMELIST name is needed in an input/output statement to refer to that list.

The NAMELIST statement has the general form:

NAMELIST /n_1/k_1/n_2/k_2/..../n_i/k_i/

where each n_i is a NAMELIST name and each k_i is a list of variable and/or array names to be associated, for input/output purposes, with the corresponding NAMELIST names. The NAMELIST statement is fully described in Section IV.

NAMELIST Input

This statement has the following form:

```
READ (f,x,opt1,opt2)
READ x
```

where f is a file reference, and x is a NAMELIST name. The first form causes a read request to be sent to file f, the second issues a read request to the standard input device.

NAMELIST Output

This statement has the following form:

```
WRITE(f,x,opt2)
PRINT x
```

where f is a file referenced and x is a NAMELIST name. This statement causes printout of information on file f in the first case, or in the second on the standard output device, in accordance with the contents of the NAMELIST group x.

Data Input Referring to a NAMELIST Statement

When a READ statement refers to a NAMELIST name, the designated input device is readied and input of data is begun. The first input data record is searched for a $ immediately followed by the NAMELIST name, immediately followed by a comma or one or more blank characters. If the search fails, additional records are examined consecutively until there is a successful match or end-of-file. When a successful match is made of the NAMELIST name on a data record and the NAMELIST name referred to in a READ statement, data items are converted and placed in storage.

Any combination of four types of data items, described in the following text, may be used in a data record. Empty fields (detected as one of the pairs (=,), (ø,), or (,,)) cause an invalid word to be stored. The data items must be separated by commas. If more than one record is needed for input data, the last item of each record must be followed by a comma. The end of a group of data is signaled by a $ following the last item either in the same data record as the NAMELIST name or anywhere in any succeeding records. The $ may replace the comma following the last piece of data. Data is restricted to columns 1 through 72 if the record is card image (media code 2); otherwise, data may appear anywhere in the record.

The form that data items may take is:

1. Variable name = constant

```
CON = 17.5
X(6) = 26.4
```

where the variable name may be an array element name or a simple variable name. Subscripts must be integer constants.

2. Array name = set of constants (separated by commas)

```
X = 1.,2.,3.,5*6.3
```

where k* constant may be included to represent k constants (k must be an unsigned integer). The number of constants must be equal to the number of elements in the array.

3. Subscripted variable = set of constants (separated by commas)

```
Y(4) = 9.,6.,10*1.8
```

where k* constant may be included to represent k constants (k must be an unsigned integer). A data item of this form results in the set of constants being placed in array elements, starting with the element designated by the subscripted variable.

The number of constants given cannot exceed the number of elements in the array that are included between the given element and the last element in the array, inclusive.

4. Variable 1/Variable 2 = constant

where Variable 1 is a counter which is set after the data has been input, indicating the number of constants that have been stored for Variable 2.

Constants used in the data items may take any of the following forms:

a. Integers, e.g., 1,2,3

b. Real numbers, e.g., 1.,2.,3.3

c. Double-precision numbers, -263D15

d. Complex numbers, which must be written in the usual form, (C1,C2), where C1 and C2 are real numbers.

e. Logical constants, which must be written as T or .TRUE., and F or .FALSE.

f. Character data written nH... or '...' where the character string does not exceed the space available on the card. This cannot be used with a repeat count.

Logical or complex constants should be associated only with logical or complex variables, respectively. Character data may be associated with any type of variable. The other types of constants may be associated with integer, real, or double-precision variables and are converted in accordance with the type of variable. With the exception of the character data, blanks must not be embedded in a constant or repeat count field, but they may be used freely elsewhere within a data record.

Any selected set of variable or array names belonging to the NAMELIST name, referred to by the READ statement, may be used as specified in the preceding description of data items. Names that are made equivalent to these names may not be used unless they also belong to the NAMELIST name.

```
                      1  456
                     ----------------------------------
First Data Card     | $NAM1     I(2,3)=5,J=4.2,B=4,
Second Data Card    | A(3)=7,6.4,L=2,3,8*4.3$
```

If the data cards are to be processed by System Input, the $ should not appear in column one. This results in an ambiguity with respect to control cards.

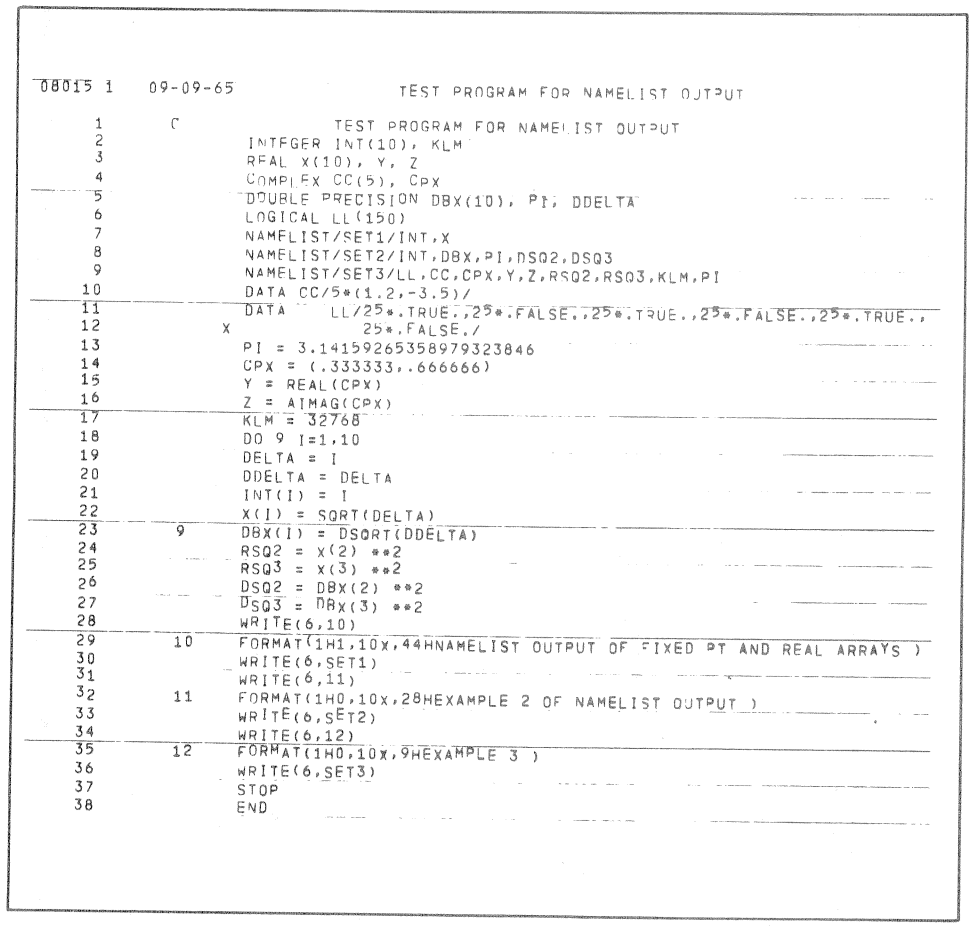
If this data is input to be used with the NAMELIST statement previously illustrated (in Section IV, NAMELIST statement) and with a READ statement, the following actions take place. The input file designated in the READ statement is prepared and the next record is read. The record is searched for a $ immediately followed by the NAMELIST name, NAM1. Since the search is successful, data items are converted and placed in core storage.

The integer constant 5 is placed in I(2,3), the real constant 4.2 is converted to an integer and placed in J and the integer constant 4 is converted to real and placed in B. Since no data items remain in the record, the next input record is read. The integer constant 7 is converted to real and placed in A(3), and the real constant 6.4 is placed in the next consecutive location of the array, A(4). Since L is an array name not followed by a subscript, the entire array is filled with the suceeding constants. Therefore, the integer constants 2 and 3 are placed in L(1) and L(2), respectively, and the real constant 4.3 is converted to an integer and placed in L(3), L(4),..., L(10). The $ signals termination of the input for the READ operation.

Data Output Referring to a NAMELIST Statement

When data is output via NAMELIST, e.g., WRITE(6,NAM1), all variables associated with LIST, as specified in the NAMELIST statement, will be output. The output values are labeled with an appropriate variable name.

The format of the output may appear with or without comma separators. Output directed to file 43 will include commas and will therefore be in agreement with the NAMELIST input format. Output may be directed to file 43 by either the PUNCH statement or a WRITE statement referencing file 43. Output directed to a file other than 43 will not include comma separators. Figures 5-1 and 5-2 contain an example program and sample output from that program in the latter format.



```
08015 1   09-09-65                TEST PROGRAM FOR NAMELIST OUTPUT

      1       C                TEST PROGRAM FOR NAMELIST OUTPUT
      2               INTEGER INT(10), KLM
      3               REAL X(10), Y, Z
      4               COMPLEX CC(5), CPX
      5               DOUBLE PRECISION DBX(10), PI, DDELTA
      6               LOGICAL LL(150)
      7               NAMELIST/SET1/INT,X
      8               NAMELIST/SET2/INT,DBX,PI,DSQ2,DSQ3
      9               NAMELIST/SET3/LL,CC,CPX,Y,Z,RSQ2,RSQ3,KLM,PI
     10               DATA CC/5*(1.2,-3.5)/
     11               DATA    LL/25*.TRUE.,25*.FALSE.,25*.TRUE.,25*.FALSE.,25*.TRUE.,
     12              X           25*.FALSE./
     13               PI = 3.14159265358979323846
     14               CPX = (.333333,.666666)
     15               Y = REAL(CPX)
     16               Z = AIMAG(CPX)
     17               KLM = 32768
     18               DO 9 I=1,10
     19               DELTA = I
     20               DDELTA = DELTA
     21               INT(I) = I
     22               X(I) = SQRT(DELTA)
     23       9       DBX(I) = DSQRT(DDELTA)
     24               RSQ2 = X(2) **2
     25               RSQ3 = X(3) **2
     26               DSQ2 = DBX(2) **2
     27               DSQ3 = DBX(3) **2
     28               WRITE(6,10)
     29       10      FORMAT(1H1,10X,44HNAMELIST OUTPUT OF FIXED PT AND REAL ARRAYS )
     30               WRITE(6,SET1)
     31               WRITE(6,11)
     32       11      FORMAT(1H0,10X,28HEXAMPLE 2 OF NAMELIST OUTPUT )
     33               WRITE(6,SET2)
     34               WRITE(6,12)
     35       12      FORMAT(1H0,10X,9HEXAMPLE 3 )
     36               WRITE(6,SET3)
     37               STOP
     38               END
```

Figure 5-1. Test Program for NAMELIST Output

```
          NAMELIST OUTPUT OF FIXED PT AND REAL ARRAYS

NAMELIST    SET1
INT   (I)
     1              1              2              3              4              5              6              7              8
     9              9             10
X     (I)
     1   0.10000000E 01   0.14142136E 01   0.17320508E 01   0.20000000E 01   0.22360680E 01   0.24494897E 01
     7   0.26457513E 01   0.28284271E 01   0.30000000E 01   0.31622776E 01
END   NAMELIST    SET1


          EXAMPLE 2 OF NAMELIST OUTPUT

NAMELIST    SET2
INT   (I)
     1              1              2              3              4              5              6              7              8
     9              9             10
DBX   (I)
     1   0.10000000000000000D 01   0.14142135623730950D 01   0.17320508075688772D 01
     4   0.20000000000000000D 01   0.22360679774997897D 01   0.24494897427831781D 01
     7   0.26457513110645906D 01   0.28284271247461901D 01   0.30000000000000000D 01
    10   0.31622776601683793D 01
PI       0.31415926535897932D 01
DSQ2     0.20000000E 01      DSQ3     0.30000000E 01
END   NAMELIST    SET2


          EXAMPLE 3

NAMELIST    SET3
LL    (I)
     1  T T T T T   T T T T T   T T T T T   T T T T T   T T T T T   F F F F F   F F F F F   F F F F F
    41  F F F F F   F F F F F   T T T T T   T T T T T   T T T T T   T T T T T   T T T T T   F F F F F
    81  F F F F F   F F F F F   F F F F F   F F F F F   T T T T T   T T T T T   T T T T T   T T T T T
   121  T T T T T   F F F F F   F F F F F   F F F F F   F F F F F   F F F F F
CC    (I)
     1   0.12000000E 01, -0.35000000E 01   0.12000000E 01, -0.35000000E 01   0.12000000E 01, -0.35000000E 01
     4   0.12000000E 01, -0.35000000E 01   0.12000000E 01, -0.35000000E 01
CPX      0.33333300E 00,  0.66666600E 00
Y        0.33333300E 00       7        0.66666600E 00       RSQ2     0.20000000E 01       RSQ3     0.30000000E 01
KLM            32768
PI       0.31415926535897932D 01
END   NAMELIST    SET3
```

Figure 5-2. NAMELIST Output of Fixed Point And Real Arrays

Formatted Input/Output Statements

These statements include a FORMAT reference, may include a file reference, either or both options 1 and 2, and a list specification. These statements pertain to character oriented sequential files. These formatted file statements have the following forms:

```
READ t, list
PRINT t, list
PUNCH t, list
READ (f,t,opt1,opt2)list
WRITE (f,t,opt2)list
```

The file reference, f, may be any integer expression. A designator of 5 or 41 for input or 6, 42 or 43 for output implies reference to the standard input/output devices.

Unformatted Sequential File Input/Output Statements

The unformatted sequential file input/output statements have the following forms:

```
READ (f,opt1,opt2) list
WRITE (f,opt2) list
```

The format designator is omitted and opt1, opt2, and list are optional. These statements apply to word oriented serial access files (binary sequential files).

Unformatted Random File Input/Output Statements

The forms for random binary file references are as follows:

```
READ (f'n,opt2)list
WRITE (f'n,opt2)list
```

where n is an integer constant, variable or expression that specifies the sequence number of the logical record to be accessed.

The principal difference between the unformatted sequential and unformatted random file operations is in the mode of access to the file. To write a file with the random WRITE statement, the file must be accessed as random. Files created as sequential (linked) may be read with the random READ statement. Any attempt to apply a random WRITE statement to a file accessed as sequential will result in the program aborting.

Linked files in time sharing may be accessed in a random mode using the ACCESS subsystem. For example, at the build mode level:

```
*ACCESS AF,/X"02",MODE/RANDOM/, R
*RUN#02
```

This is particularly useful when reading a FORTRAN created, standard system format, unformatted sequential file using random READ statements. Each record in the sequential file must be the same length.

Unformatted random files created by FORTRAN are normally recorded in Standard System Format (see File and Record Control Reference Manual). Thus, files written with sequential I/O statements may subsequently be operated on with random I/O statements and vice versa.

Random files may also be written in a "pure data" format, without block serial numbers or record control words. This can be accomplished by one of the following:

```
$    FFILE    U,NOSRLS,FIXLNG/N
      or
     CALL     RANSIZ(U,N,1)
```

U and N are the file unit number and logical record size parameters.

It is a requirement that FORTRAN random files have a constant record size. Further, before any random I/O can be performed on any given file, its record size must be defined. This is accomplished with either is a $ FFILE control card or with a CALL to the (library) subroutine RANSIZ. Three arguments are required: the first is a file reference, the second provides the record size. Both of these arguments may be any integer expression. The third argument is zero or not supplied when the file is in standard system format. A non-zero value specifies a pure data file. For example:

```
CALL RANSIZ(08,50)
```

This statement specifies that file code 08 has a constant record size of 50 and is in standard system format.

File Properties

Sequential Files - A sequential file may contain zero, one or more records accessed in a sequential manner.

Random Files - A random file consists of records each of which is addressable i.e., each record may be accessed without repositioning the file. Each record in the random file must be of the same length.

File Updating - Input-output routines with Random files permit replacement of individual records in a file. The execution of all random file WRITE statements is considered a record replacment.

Record Sizes - Random files have records, all of the same length.

Sequential files have variable length records. Addition of records on a sequential file requires the determination of the record length by a list and FORMAT statement for formatted output and by the list alone for unformatted (binary) output.

FILE HANDLING STATEMENTS

File handling statements provide for the manipulation of input/output devices for positioning of sequential files and demarcation of sequential files. The following file handling statements are described in Section IV:

```
REWIND
BACKSPACE
ENDFILE
```

INTERNAL DATA CONVERSION

The ENCODE and DECODE statements are similar to the formatted READ and WRITE statements respectively except the ENCODE/DECODE statements do not cause input/output to take place. They cause data conversion and transmission to take place between an internal buffer area and the elements specified by a LIST. The forms of the ENCODE and DECODE statements are:

```
ENCODE (a,t)list
DECODE (a,t)list
```

where a is the internal buffer and t is a format designator.

The buffer area is designated by the first operand within the parentheses. It may be given as:

1. A character scalar
2. A character array element
3. An array

MULTIPLE RECORD PROCESSING

An analogy can be drawn between character array elements and records which can be most useful for some applications. Consider the following example:

```
      CHARACTER TEXT*48(10)
      INTEGER DATA (50)
      DO 100 I=1,50,5
100   DECODE (TEXT(I/5+1),101)(DATA(J),J=I,I+4)
101   FORMAT (5I7)
```

Examination of the format and list reveals that 50 items are to be converted, 5 items per record, hence 10 records are required. The character array TEXT has 10 elements which will be treated as records, each element being 48 characters long. The format requires 35 characters of each element (5x7), thus the first 35 will be processed.

The same can be accomplished by letting the list and format specification cover the full 10 records as follows:

```
      CHARACTER  TEXT *48(10)
      INTEGER DATA (50)
      DECODE (TEXT,10) DATA
10    FORMAT (5I7)
```

In a BCD mode program (six characters per word) the same could also be accomplished with an internal buffer of type INTEGER as follows:

```
      INTEGER TEXT (8,10), DATA(50)
      DECODE (TEXT,10) DATA
10    FORMAT (5I7)
```

If the same program is compiled in the ASCII mode, the format specification describes 35 character records, while the array has provisions for only 32 (8*4) characters per "record". This word size/byte size problem is eliminated by the character data type since

```
      CHARACTER  TEXT *48(10)
```

is valid for both modes. In BCD, the equivalent of an 8x10 array is allocated; in ASCII, the equivalent of a 12x10 array is allocated. The source program is character set independent. For this reason the preferred type of the internal buffer argument of the ENCODE and DECODE statements is CHARACTER. Warning diagnostics will be posted when this is not the case, as in the third example.

EDITING STRINGS WITH ENCODE

With ENCODE, characters not processed are left unchanged. The following example demonstrates this feature.

```
      CHARACTER TEXT*20
      TEXT = "WOW IS THE TIME FOR "
      ENCODE (TEXT,10) "NOW"
10    FORMAT (A3)
20    PRINT, TEXT, "ALL GOOD MEN"
      STOP;END
```

Execution of statement 20 will cause the following to be printed:

```
NOW IS THE TIME FOR ALL GOOD MEN
```

CONDITIONAL FORMAT SELECTION

A problem common in FORTRAN programs arises when the format of the next record cannot be determined without first reading it. This problem can be overcome through the capability of the DECODE statement. As an example, consider that input to a program is in card form, and the cards come in one of three formats. When card column 1 contains a 0 the first format is to be applied; when it contains a 1 the second; and 2 the third. The following subroutine could be used:

```
      SUBROUTINE READ (A,I,Z)
      CHARACTER CARD*79
      READ 101,KOL1,CARD
101   FORMAT(I1,A79)
      GO TO (200,300,400),KOL1+1
200   DECODE (CARD,201) A,I,Z
201   FORMAT (10X,F12.6,3X,I5,E12.6)
      RETURN
300   DECODE (CARD,301) A,Z,I
301   FORMAT (10X,2F12.6,3X,I5)
      RETURN
400   DECODE (CARD,401)I,A,Z
401   FORMAT (50X,I5,2E12.6)
      RETURN ; END
```

CONSTRUCTION OF FORMATS WITH ENCODE

Another similar problem has to do with the building of format specifications at run time for subsequent use in input processing. As an example, consider that some data file is interpersed with control cards which specify the amount and format of ensuing data. The first field of the control card gives the number of data items that will be read; the second gives the number of fields per card (up to 20) or is zero indicating "use the previously developed format"; the remaining fields on the control card come in pairs and provide "w" and "d" sizes for "F" Format specifications needed for correct conversion of each data item; the control card is in free-field format with comma separators. The following subroutine will read and verify control cards, build format specifications, and read a set of data:

```
SUBROUTINE READ (A,I)
DIMENSION A(I)
INTEGER WD(40)
CHARACTER FORM*141/" "/
READ,N,J,(WD(L),L=1, MIN0(2*J,40))
IF (N.GT.I .OR. N.LT. 1)STOP "ITEM COUNT ERROR"
IF (J.GT.20 .OR J.LT.0) STOP "FIELD COUNT ERROR"
IF (J.EQ.0 .AND.FORM.EQ." ")STOP "UNFORMED FORMAT ERROR"
IF (J),200,
NCOL = 0
DO 50 L=1,2*J,2
IF (WD(L+1).LT. 0 .OR. WD(L+1).GT.8)GO TO 300
IF (WD(L).LT. WD(L+1)+2) GO TO 300
```

```
 50   NCOL =NCOL + WD(L)
      IF (NCOL .GT. 80)STOP "COLUMN COUNT ERROR"
      FORM="   "
      ENCODE(FORM,101)("F",WD(L),WD(L+1),",",
     & L=1,2*J-2,2),"F",WD(2*J-1),WD(2*J),")"
101   FORMAT("(",20(A1,I2,".",I2,A1))
200   READ(05,FORM)(A(L),L=1,N)
      RETURN
300   PRINT 301, (L+1)/2, WD(L),WD(L+1)
301   FORMAT ("1 FORMAT SPEC #",I3," IN ERROR. W=",I5," D=",I5)
      STOP" FIELD DESCRIPTOR ERROR"
      END
```

The above examples also illustrate the use of a number of other Series 6000 FORTRAN language features, most notably:

1. Expressions used
   a. As DO parameters
   b. in an output list
   c. as the index of a computed GO TO
2. The CHARACTER data type and A format specifiers for long strings
3. Adjustable dimensions
4. The T (tabulation) format specifier
5. Null label fields on an airthmetic IF
6. STOP with display

Note also that the use of CHARACTER scalars of arbitrary size eliminates program dependency on character set. The above subroutine will run in ASCII or BCD mode, without change.

OUTPUT DEVICE CONTROL

In the absence of the NO SLEW option, the spacing of the printing on the output device is controlled by the first character of the line of output. The first character is not printed but is examined to determine if it is a control character to regulate the spacing of the output device. If the first character is recognized as a control character, the line is printed after the proper spacing has been effected. In any event, it is deleted when the line is printed. This control affects printers, teletypewriters, and displays.

The effects produced by control characters are:

Character	Effect
0	1 blank line prior to print
+	Overprint
1	Slew to top of page before printing
Any other	Space to next line

FORMAT SPECIFICATIONS

Field Separators

The format field separators are the slash and the comma. A series of slashes is also a field separator. The field descriptors or group of field descriptors are separated by a field separator.

The slash is used to separate field descriptors and to specify a demarcation of formatted records. The length of the strings for a given external medium are dependent upon the processor and the external medium.

The processing of the number of characters that can be contained in a record by an external medium does not itself cause the introduction or inception of processing of the next record.

Repeat Specification

It is possible to repeat a field descriptor (except quoted strings, tabulation control, nH and nX) by writing a repetition number in front of it. Thus the field specification 3E12.4 is the same as writing E12.4, E12.4, E12.4.

It is also possible to repeat a group of field descriptors by enclosing the group in parentheses and preceding the left parenthesis with the repeat count. If no count is specified, a repeat count of one is assumed. For example, if four fields on a card are alternately described as F10.6 and E10.2, this can be written as 2(F10.6, E10.2). One additional level of grouping is permitted, using the same rules for representation. If, for example, the fields on a card could be described by (I3, F8.4, E8.2, F8.4, E8.2, I3, F8.4 E8.2, F8.4, E8.2, A10) then a more compact description would be (2(I3,2(F8.4,E8.2)),A10).

Scale Factors

To permit more general use of D-, E-, F-, and G-descriptors, a scale factor followed by the letter P may precede the specification. The magnitude of the scale factor must be between -8 and +8, inclusive. The scale factor is defined for input as follows:

$$10^{-\text{scale factor}} \times \text{external quantity} = \text{internal quantity}$$

For output, the scale factor is defined as follows:

$$\text{external quantity} = \text{internal quantity} \times 10^{\text{scale factor}}$$

For input, scale factors have an effect only on F-conversion. For example, if input data is in the form xx.xxxx and it is desired to use it internally in the form .xxxxxx, then the FORMAT specifications to effect this change is 2PF7.4. For output, scale factors may be used with D-, E-, F-, and G-conversion.

For example, the statement FORMAT (I2,3F11.3) might output the following printed line:

27ØØØØ-93.209ØØØØØ-0.008ØØØØØØ0.554

But the statement FORMAT (I2.1P3F11.3) used with the same data would output the following line:

27ØØØØ-932.094ØØØØØ-0.076ØØØØØØ5.536

whereas, the statement FORMAT (I2,-1P3F11.3) would output the following line:

27ØØØØØ-9.321ØØØØØ-0.001ØØØØØØ0.055

A positive 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-conversions following the scale factor within the same FORMAT statement. This applies to both single-record formats and multiple-record formats. Once the scale factor has been given, a subsequent scale factor of zero in the same FORMAT statement must be specified by 0P. For F-type conversion, output of numbers, whose absolute value is greater than or equal to 2^{35} after scaling, is output in E-conversion. Scale factors have no effect on I- and O-conversion.

Multiple Record Formats

When a list of an input or output statement is used to transmit more than one record (card or line) and with different formats, a slash (/) is used to separate the format specifications for different lines. For example: if two cards are to be read with a single READ statement and the first has a five-digit integer and the second has 5 real numbers, the FORMAT statement is:

```
FORMAT (I5/5E10.3)
```

It is also possible to specify a special format for the first (one or more) records and a different format for subsequent records. This is done by enclosing the last record specifications in parentheses. For example: if the first card of a deck has an integer and a real number and all following cards contain two integers and a real number, the FORMAT statement might be:

```
FORMAT (I6,E10.3/(2I6,E12.3))
```

If a multiple-line format is desired in which the first two lines are to be printed according to a special format, and all remaining lines according to another format, the last line-specification should be enclosed in a second pair of parentheses; for example:

```
FORMAT (I2,3E12.4/2F10.3,3F9.4/(10F12.4))
```

If data items remain to be output after the format specification has been completely "used", the format repeats from the last previous left parenthesis which is at level 0 or 1. The following example illustrates the various levels of parentheses.

```
FORMAT (3E10.3,(I2,2(F12.4,F10.3)),D28.17)
        0       1    2           21       0
```

The parentheses labeled 0 are zero level parentheses; those labeled 1 are first level parentheses; and those labeled 2 are second level parentheses. If more items in the list are to be transmitted after the format statement has been completely used, the FORMAT repeats from the last first-level left parenthesis; that is, the parenthesis preceding I2.

As these examples show, both the slant and the final right parenthesis of the FORMAT statement indicate a termination of a record.

Blank lines may be introduced into a multiline FORMAT statement by inserting consecutive slants. When N+1 consecutive slants appear at the end of the FORMAT, they are treated as follows: for input, n+1 records are skipped; for output, n blank lines are written. When n+1 consecutive slants appear in the middle of the FORMAT, n records are skipped for both input and output.

Carriage Control

The WRITE (f,t), PRINT, and PRINT t, statements prepare formatted fields in edited format for the printer. The first character of each record is examined to see if it is a control character to regulate the spacing of the printer. If the first character is recognized as a control character, it is replaced by a blank in the printed line and the line printed after the proper spacing has been effected. The interpretation of control characters is discussed under Output Device Control. This control is usually obtained by beginning a FORMAT specification with 1H followed by the desired control character.

If a carriage control information is not desired, see $ FFILE/NOSLEW in the Control Card manual.

Data Input Referring to a FORMAT Statement

These specifications must be followed when data input to the object program is under format control:

1. The data must correspond in order, type, and field with the field specifications in the FORMAT statement; or the field may be shortened by using commas as delimiters. For example: for a format specification of 3I6, an input data card containing 1, ØØØØØ2ØØ3, is accepted. The values 1, 2, and 3 are input. Note that the second field is a full six characters wide and no comma appears; however, commas terminate the first and third fields.

2. Plus signs may be omitted or indicated by a +. Minus signs must be indicated.

3. Blanks in numeric fields are regarded as zeros.

4. Numbers for E- and F-conversion may contain any number of digits, but only the high-order 8 digits of precision are retained. For D-conversion, the high-order 18 digits of precision are retained. In both cases, the number is rounded to 8 or 18 digits of accuracy, as applicable.

5. Numeric data must be right justified in the field.

To permit greater ease in input preparation, certain relaxations in input data format are permitted.

1. Numbers for D- and E-conversion need not have four columns devoted to the exponent field. The start of the exponent field must be marked by a D or an E or, if that is omitted, by a plus or minus sign (not a blank). For example, E2, E+2, +2, +02, and D+02 are all permissible exponent fields.

2. Numbers for D-, E-, and F-conversion need not supply a decimal point; the format specification suffices. For example, the number -09321+1 with the specification E12.4 is treated as though the decimal point had been placed between the 0 and the 9. If the decimal point is included in the field, its position overrides the position indicated in the format specification.

Numeric Field Descriptors

Six field descriptors are available for numeric data:

Internal	Conversion Code	External
Floating-point (double-precision)	D	Real with D exponent
Floating-point	E	Real with E exponent
Floating-point	F	Real without exponent
Floating-point	G	Appropriate type
Integer	I	Decimal Integer
Integer or Floating-point	O	Octal Integer

These numeric field descriptors are specified in the forms prDw.d, prEw.d, prFw.d, prGw.d, rIw, rOw, where:

1. D, E, F, G, I, and O represent the type of conversion.

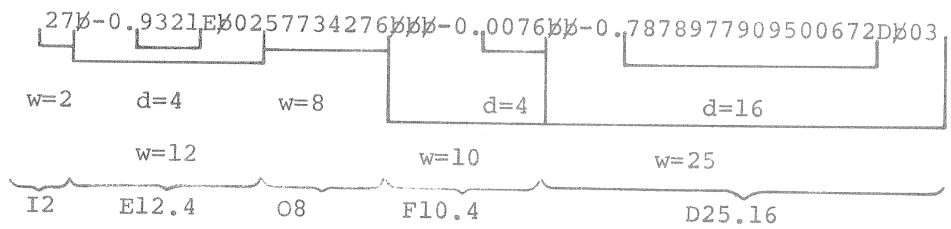
2. The w is an unsigned integer constant representing the field width for converted data; this field width may be greater than required to provide spacing between numbers.

3. The d is an unsigned integer or zero representing the number of digits of the field that appear to the right of the decimal point. For F-conversion, if d is specified ≥ 9, it will be truncated at 8. For D-conversion, if d is specifier ≥ 19, it will be truncated at 18.

4. Each p is optional and represents a scale factor designator.

5. Each r is an optional non-zero integer constant indicating the number of occurrences of the numeric field descriptor that follows.

For example, the statement FORMAT (I2,E12.4,O8,F10.4,D25.16) might cause the following line to be printed.



where b indicates a blank space.

The following are notes on D-, E-, F-, G-, I-, and O-conversion.

1. No format specification should be given that provides for more characters than are permitted for a particular input/output record. Thus a format for a record to be printed should not provide for more characters (including blanks) than the capabilities of the relevant device.

2. Information transmitted with O-conversion may have real or integer names; information transmitted with G-conversion may have real, or complex names; information transmitted with E-, and F-conversions must have real or complex names; information transmitted with I-conversion must have integer names; information transmitted with D-conversion must have double-precision names.

3. The numeric field descriptor Gw.d indicates that the external field occupies w positions with d significant digits. The value of the list item appears or is to appear, internally as a real datum.

   Input processing is the same as for the F-conversion.

   The method of representation in the external output string is a function of the magnitude of the real datum being converted. Let N be the magnitude of the internal datum. The following tabulation exhibits a correspondence between N and the equivalent method of conversion that will be effected:

Magnitude of Datum	Equivalent Conversion Effected
$0.1 \leq N < 1$	F(w-4).d,4X
$1 \leq N < 10$	F(w-4).(d-1),4X
. .	.
. .	.
. .	.
$10^{d-2} \leq N < 10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N < 10^{d}$	F(w-4).0,4X
Otherwise	sEw.d

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

4. The field width w, for D-, E-, F-, and G-conversions, must include a space for a decimal point and a space for the sign. The D-, E-, and G-conversions also require space for the exponent. For example, for D- and E- and G-conversions on output, w d+6, and for F-conversion, w≥d+2.

5. The exponent, which may be used with D- and E-conversions, is the power of 10 to which the number must be raised to obtain its true value. The exponent is written with an E (for E-conversion) or D (for D-conversion) followed by a minus sign if the exponent is negative, or a plus sign or a blank if the exponent is positive, and then followed by two numbers that are the exponent. For example, the number .002 is equivalent to the number .2E-02.

6. For D-conversion input, up to 19 decimal digits are converted and the result is stored in a double word. For D-conversion output, the two core storage words representing the double-precision quantity are considered one piece of data and converted as such.

7. If a number converted by I-conversion requires more spaces than are allowed by the field width w, the most significant part of the number is truncated to fit the field. If the number requires fewer than w spaces, the leftmost spaces are filled with blanks. If the number is negative, the space preceding the leftmost digit contains a minus sign if sufficient spaces have been reserved.

8. If an output number that is converted by D-, E-, F-, G-, or I-conversions requires more spaces than are allowed by the field width w, the most significant part of the number is truncated to fit the field. If the number requires fewer than w spaces, the leftmost spaces are filled with blanks. The output field is filled with blanks if the output number is octal constant +377777777777 (noise word).

9. Specifications for successive fields are separated by commas and/or slants. (See Multiple-Record Formats in this section.)

Complex Number Fields

Since a complex quantity consists of two separate and independent real numbers, a complex number is transmitted either by two successive real number specifications or by one real number specification that is repeated. E.g., 2E10.2=E10.2,E10.2. The first supplies the real part. The second supplies the imaginary part.

The following is an example of a FORMAT statement that transmits an array of six complex numbers.

```
FORMAT (2E10.2, E8.3, E9.4, E10.2, F8.4, 3(E10.2, F8.2))
```

Alphanumeric Fields

Alphanumeric information may be transmitted in two ways. Both ways result in the storing of BCD or ASCII characters (as determined by an option in the $ FORTY or $ FORTRAN card or the YFORTRAN or FORTRAN Time Sharing System RUN commands).

1. The specifications rAw and rRw cause character data to be read into or written from a variable. See the FORMAT statement in Section V for a description of these fields.

2. The specification nH, enclosing the string in quotation marks, or enclosing the string in apostrophes introduces alphanumeric information into a FORMAT statement.

The basic difference is that A and R conversions are given a variable name that can be referenced for processing or modification. The character constant notations do not "use" a list item.

The A and R format specifiers are used for the transfer of character variable data to and from input/output buffers.

If the field width (w) specified for A or R input is equal to or greater than the maximum length (as described in the CHARACTER type statement), the rightmost s characters will be taken from the external field. The I/O pointer will be advanced in accordance with the field width of the format specifier. If the field width is less than the maximum length describing in the CHARACTER type statement, the w characters will be taken from the external field. With A conversion, the data will appear left adjusted with s-w trailing blanks in the internal representation. For R conversion, the internal representation will be right justified with s-w leading zeros.

If the field width (w) specified for A or R output is greater than the maximum length described in the CHARACTER type statement, s characters will be transmitted to the external field preceded by w-s blanks. If the field width is less than or equal to the maximum length as described in the CHARACTER type statement, the external output field will consist of w characters from the internal representation. With A conversion the w leftmost characters will be transmitted; with R conversion, the w rightmost will be used.

The R code is equivalent to A except that the characters are right adjusted with leading zeros in the internal representation.

When the variable associated with an A or R format specifier is not of type CHARACTER then the variable is treated as a character variable with a size of one word of storage (6 characters for BCD, 4 for ASCII).

Logical Field Descriptor

Logical variables may be read or written using the specification Lw, where L represents the logical type of conversion and w is an integer constant that represents the data field width.

1. On input, a value representing either true or false is stored if the first nonblank character in the field of w characters is a T or an F repectively. If all the w characters are blank, a value representing false is stored.

2. On output, a value of .TRUE. or .FALSE. in storage causes w minus 1 blanks, followed by a T or an F, respectively, to be written out.

Character Positioning Field Descriptors

The X and T field descriptors enable a specified number of characters in the record to be skipped. On output, the X descriptor causes a specified number of spaces to be inserted in the external output record.

X Format Code

The field descriptor for space character is nX. On input, n characters of the external input record are skipped. On output, n space characters are inserted in the external output record. If n = 0, a value of one is assumed.

T Format Code

The field descriptor for tabulation purposes is Tt. The position in a FORTRAN record where the transfer of data is to begin is t. The t is an unsigned integer constant. Using this format code permits input or output to begin at any specified position. Tabbing may proceed backward as well as forward.

Variable Format Specifications

Any of the formatted input/output statements (including ENCODE and DECODE) may contain a character scalar or an array name in place of the reference to a format statement label. At the time a variable is referenced in such a manner, the first part of the information must be character data which constitutes a valid format specification. There is no requirement on the information following the right parenthesis that ends the format specification.

The format specification (the value of the variable referenced) must have the same form as that defined for a FORMAT statement, without the word FORMAT. Thus the character text of the specification begins with a left parenthesis and ends with a matching right parenthesis.

The format specification may be defined by a data initialization statement, by a READ statement together with an A format, by use of a character replacement statement, or by ENCODE.

In the following example, A, B, and part of the array C are converted and stored according to the FORMAT specifications read into the array FMT at object time.

```
      DIMENSION FMT (12),C(10)
1     FORMAT (12A6)
      READ (5,1) FMT
      READ (5,FMT) A,B, (C(I), I=1,5)
```

A similar example follows using a character scalar for the variable format.

```
      DIMENSION  C(10)
      CHARACTER  FMT*72

    1 FORMAT (A72)

      READ (5,1)FMT

      READ (5,FMT) A,B,(C(I),I=1,5)
```

SECTION VI

SUBROUTINES, FUNCTIONS, AND SUBPROGRAM STATEMENTS

The three basic elements of scientific programming languages -- arithmetic, control, and input/output -- are given added flexibility through subroutines. Subroutines are program segments executed under the control of another program and are usually tailored to perform some often-repeated set of operations. A subroutine is written only once, but may be used again and again; it avoids a duplication of effort by eliminating the need for rewriting program segments for use in common operations. There are four classes of subroutines in FORTRAN: arithmetic statement functions, built-in functions, FUNCTION subprograms, and SUBROUTINE subprograms. The major differences among the four classes of subroutines are as follows:

1. The first three classes may be grouped as functions; they differ from the SUBROUTINE subprogram in the following respects:

   a. The functions are always single-valued; that is, they return only a single result; the SUBROUTINE subprogram may return more than one value.

   b. A function is referred to by an arithmetic expression containing its name; a SUBROUTINE subprogram is referred to by a CALL statement.

2. The arithmetic statement functions and built-in functions are open subroutines. An open subroutine is a subroutine that is incorporated into the object program each time it is referred to in the source program. The other FORTRAN subroutines are closed; that is, they appear only once in the object program.

NAMING SUBROUTINES

All four classes of subroutines are named in the same manner as a FORTRAN variable (see Variables in the Index). External subroutine names (i.e. FUNCTION and SUBROUTINE subprograms) have the additional requirement that they be unique within the first six characters. The following rules are applicable:

1. A subroutine name consists of one to eight alphanumeric characters, the first of which must be alphabetic.

2. The type of the function, which determines the type of the result, may be defined as follows:

   a. The type of an external function may be indicated by the name of the function or by placing the name in a Type statement.

b. The type of a FUNCTION subprogram may be indicated by the name of the function or by writing the type (REAL, INTEGER, COMPLEX, DOUBLE PRECISION, LOGICAL, CHARACTER) preceding the word FUNCTION. In the latter case, the type implied by its name is overridden. The type of the FUNCTION subprograms in the Subroutine Library (the mathematical subroutines) is defined. Therefore, they need not be typed elsewhere.

c. The type of a built-in function is indicated within the FORTRAN compiler and need not appear in a Type statement.

d. Arithmetic statement functions have no type.

3. The name of a SUBROUTINE subprogram has no type and should not be defined, since the type of results returned is dependent only on the type of the variables returned by that subroutine.

ARITHMETIC STATEMENT FUNCTIONS

An arithmetic function is defined internal to the program unit in which it is referenced. It is defined by a single statement similar in form to the arithmetic assignment statement.

In a given program unit, all statement function definitions must precede the first executable statement of the program unit. The name of a statement function must not appear in EXTERNAL, COMMON, EQUIVALENCE, NAMELIST, or ABNORMAL statements, as a scalar name, or as an array name in the same program unit.

Defining Arithmetic Statement Functions

An arithmetic statement function definition has the form:

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

where f is the function name, the a_i are distinct symbolic names (called dummy arguments of the function), and e is an expression. Since the a_i are dummy arguments, their names, which serves only to indicate number, and order of arguments, may be the same as actual variable names appearing elsewhere in the program unit. The following is a list of exceptions, names of other program symbols which cannot appear as dummy names in the list if they have previously been defined as:

EXTERNAL names
ABNORMAL names
PARAMETER names
NAMELIST names
SUBROUTINE, FUNCTION or ENTRY names
Arithmetic Statement Function names

The expression,e, may contain:

Constants
Scalar references
Intrinsic function references
References to other arithmetic statement functions
External function references
Array element references
Indeterminate references

The last item in the above list, indeterminate references, covers the case where a dummy argument symbol appears in e as a reference of the form:

```
a (list)
```

This syntax may imply a function reference or an array element reference. The decision is made each time the arithmetic statement function is referenced, and is determined by the actual argument.

By way of illustration consider the following:

```
1  DIMENSION  P(10)
2  F(A,B)=A(K)+B(K)
3  X=F(P,SIN)
```

Expansion of line 3 produces an equivalent assignment statement:

```
3  X = P(K)+SIN(K)
```

In this example the first expression term is an array element reference while the second is a function reference.

ASF Left of Equals

An ASF may be referenced on the left hand side of the equal sign in an assignment statement; however, it must expand into a scalar or an array element. For example:

```
AA (I,J) = J(I)
DIMENSION K(10)
     .
     .
AA (3,K) = 4*X (This expands to K(3) = 4*X)
```

Referencing Arithmetic Statement Functions

A statement function is referenced by using its name with a list of actual arguments in standard function notation as a primary in an expression. The actual arguments, which constitute the argument list, must agree in number with the dummy arguments in the function definition. An actual argument in a statement function reference may be any expression providing the corresponding dummy appeared as a scalar reference. If the corresponding dummy argument appears as an indeterminate reference then the actual argument must be an array or function name.

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

At time of reference, the actual arguments are substituted for the dummy argument symbols. Type is introduced at this time and any ambiguities (such as the indeterminate reference described above) are resolved. References to other functions are classified as intrinsic, external, or other ASF, at this time also. Thus to reference another arithmetic statement function, the definition of that function may follow the definition of but must precede any references to, this referencing function.

Arithmetic Statement Function Example

A function can be defined to compute one root of the quadratic equation, $ax^2+bx+c=0$, given values a, b, and c as follows:

```
ROOT (A,B,C)=(-B+SQRT(B**2-4*A*C))/(2*A)
```

This is the definition of the function. This definition can be used by supplying values for a, b, and c. An example of the use of the function using 16.9 for a, 20.5 for b, and T+30 for c follows:

```
ANS = ROOT(16.9,20.5,T+30)
```

SUPPLIED INTRINSIC FUNCTIONS

The functions listed in Table 6-1 are the intrinsic or built-in functions supplied with FORTRAN. The intrinsic functions require only a few machine instructions and are inserted each time the function is used. To use these functions, it is necessary to only write their names where needed and enter the desired expression(s) for argument(s). The names of the functions are established in advance and must be written exactly as specified.

The functions in Table 6-1, except FLD, AND, OR, XOR, BOOL, and COMPL, are the standard FORTRAN intrinsic functions and their use is not described in this document. The use of FLD, AND, OR, XOR, BOOL, and COMPL are described in this section.

Argument Checking and Conversion for Intrinsic Functions

A number of checks on arguments used in intrinsic functions are made by the compiler. Due to the in line code expansion, the number of arguments specified must agree with the number shown in Table 5. Except as noted in Table 5, the argument type must agree with the type of the function. With the exception of the typeless functions (described in this section), argument checking and/or conversion is carried out by the compiler using the following general rules:

1. The hierarchy of argument types considered for conversion is: integer, real, double precision, complex.

2. A generic intrinsic function call will be transformed to the function type that supports the highest level argument type supplied to it.

3. Arguments to a non-generic form of intrinsic function are converted to conform with the function type specified. This is within the constraints of argument types integer through complex.

Automatic Typing of Intrinsic Functions

Use of the generic forms of the mathematical intrinsic functions (see Table 5) allows for the type of the function's value to be determined automatically by the type of the actual arguments supplied. This subset of intrinsic functions contains:

1. Absolute value - ABS
2. Remaindering - MOD
3. Maximum value - MAX
4. Minimum value - MIN
5. Positive difference - DIM
6. Transfer of sign - SIGN

This means that the in line code generated for DABS(D) and ABS(D) would be the same assuming that the type of the variable D is double precision.

When arguments of different types are specified (functions allowing more than one argument) the type of the function itself is determined by the same rules that govern mixed mode expressions. See Table 4-1, Rules for Assignment of E to V.

FLD

FLD is used for bit string manipulation and has the following form:

```
FLD (i,k,e)
```

where:

i and k are INTEGER expressions where $0 \leq i \leq 35$ and $1 \leq k \leq 36$; e is any INTEGER, REAL, or TYPELESS expression, a Hollerith word or one of the typeless functions listed in Table 6-1.

This function extracts a field of k bits from a 36 bit string represented by e starting with bit i (counted from left to right where the 0th bit is the leftmost bit of e). The resulting field is right-justified and the remaining bits are set to zero.

This intrinsic function may appear on the left hand side of the equal sign in an assignment statement. This is defined as follows:

```
FLD (i,j,a) = b
```

where:

i and j are integer expressions equal to or less than 36; a is a scalar or subscripted variable; and b is an expression.

For example:

```
A = "ABCD"
B = "1234"
FLD (9,9,A) = B
PRINT, A
```

This would result in the printing of A4CD.

Typeless Intrinsic Functions

There are five typeless functions as follows:

AND (e1,e2)	Bit by bit logical product of e1 and e2.
OR (e1,e2)	Bit by bit logical sum of e1 and e2.
XOR (e1,e2)	Bit by bit "exclusive or" of e1 and e2.
BOOL (e)	The type of e is disregarded.
COMPL (e)	All bits of e are complemented and the type of e is disregarded.

The expressions of e may be of type INTEGER, REAL, or Typeless; e may also be a Hollerith word, the FLD word, or any of the typeless functions.

Table 6-1. Supplied Intrinsic Functions

INTRINSIC FUNCTION	DEFINITION	NO. OF ARG.	SYMB. NAME	TYPE OF: ARG.	TYPE OF: FUNCT.
Absolute Value	$\|a\|$	1	ABS	Real	Real
			IABS	Integer	Integer
			DABS	Double	Double
			CABS[3]	Complex	Complex
Truncation	Sign of a times largest integer $\leq\|a\|$	1	AINT	Real	Real
			INT	Real	Integer
			IDINT	Double	Integer
Remaindering[1]	a_1(mod a_2)	2	AMOD	Real	Real
			MOD	Integer	Integer
			DMOD[3]	Double	Double
Choosing Largest Value	Max (a_1,a_2,...)	2	AMAX0	Integer	Real
			AMAX1	Real	Real
			MAX0	Integer	Integer
			MAX1	Real	Integer
			DMAX1	Double	Double
			MAX	I,R,D	Note 2
Choosing Smallest Value	Min (a_1,a_2,...)	2	AMIN0	Integer	Real
			AMIN1	Real	Real
			MIN0	Integer	Integer
			MIN1	Real	Integer
			DMIN1	Double	Double
			MIN	I,R,D	Note 2
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a_2 times $\|a\|_1$	2	SIGN	Real	Real
			ISIGN	Integer	Integer
			DSIGN	Double	Double
Positive Difference	a_1-Min (a_1,a_2)	2	DIM	Real	Real
			IDIM	Integer	Integer
			DDIM	Double	Double

[1] Remaindering (AMOD (a_1,a_2)) is defined as $a_1-[a_1/a_2]*a_2$, where the bracketed expression denotes the integral results of the expression a_1/a_2.

[2] Same as argument.

[3] These functions are processed by external library subroutines.

Table 6-1. (Cont.) Supplied Intrinsic Functions

INTRINSIC FUNCTION	DEFINITION	NO OF ARG.	SYMB. NAME	TYPE OF: ARG.	FUNCT.
Obtain Most Significant Part of Double Precision Argument		1	SNGL	Double	Real
Obtain Real Part of Complex Argument		1	REAL	Complex	Real
Obtain Imaginary Part of Complex Argument		1	AIMAG	Complex	Real
Express Single Precision Argument in Double Precision Form		1	DBLE	Real	Double
Express Two Real Arguments in Complex Form	$a_1+a_2\sqrt{-1}$	2	CMPLX	Real	Complex
Obtain Conjugate of a Complex Argument		1	CONJG	Complex	Complex
Logical "and"	$a_1a_2,\ldots$	a	AND	REAL, INTEGER, or TYPELESS	Typeless
Logical "or"	$a_1+a_2+\ldots$	a	OR	REAL, INTEGER, or TYPELESS	Typeless
Logical "exclusive or"	$a_1 \oplus a_2 \oplus \ldots$	2	XOR	REAL, INTEGER, or TYPELESS	Typeless
Ignore Type		1	BOOL	Any except LOGICAL	Typeless
Extract Bit Field	Beginning with bit a_1 of word a_3 extract a_2 bits	3	FLD	(1,2) Integer (3) Any except LOGICAL	Typeless
Logical Complement	-a	1	COMPL	REAL, INTEGER, or TYPELESS	Typeless

[a] An indefinite (≥ 2) number of arguments are permitted.

FUNCTION SUBPROGRAMS

Defining FUNCTION Subprograms

FUNCTION subprograms are defined external to the program unit that references it. The computation desired in a FUNCTION subprogram is defined by writing the necessary statements in a segment, writing the word FUNCTION and the name of the function before the segment, and writing the word END after it. The FUNCTION statement is of the form:

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

where t is either INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER, or empty. The f is the symbolic name of the function to be defined. The a_i (called dummy arguments) are either variable names, array names, or external procedure names.

The symbolic name of the function must appear at least once in the subprogram as a variable name in some defining context (e.g., left of equals). The value of the variable at the time of execution of any RETURN statement in this subprogram is returned as the value of the function.

The symbolic name of the function must not appear in any nonexecutable statement in this program unit, except as the symbolic name of the function in the FUNCTION statement or in a TYPE statement.

An abnormal FUNCTION subprogram may define or redefine one or more of its arguments to effectively return results in addition to the value of the function.

The FUNCTION subprogram may contain any statements except BLOCK DATA, SUBROUTINE, another FUNCTION statement, or any statement that directly or indirectly references the function being defined. The Function Subprogram must contain at least one RETURN statement.

If the function name appears in any of the following contexts, redefinition of the function result is effected.

1. Left of equals in assignment statement
2. In the list of a READ statement
3. In the list of a DECODE statement
4. As the buffer name in an ENCODE statement
5. As the induction variable of a DO loop

Redefinition may also occur if the function name appears in the argument list of a CALL statement or a reference to some abnormal external function, though not necessarily.

Supplied FUNCTION Subprograms

The functions listed in Table 6-2 are the basic external FUNCTION subprograms supplied with the compiler. To use the functions, it is only necessary to write their name where needed and enter the desired expression(s) for argument(s). Except as indicated in Table 6-2, argument types must conform with the type of the function. The compiler does some checking as to type of arguments supplied and will make conversions in accordance with the following rules:

1. The hierarchy of argument types considered for conversion is: integer, real, double precision, complex.

2. A generic function call whose arguments do not conform as to type will be transformed to the function type that supports the highest level argument supplied to it.

3. Integer arguments are converted to the type of the function being called.

4. Arguments to a non-generic form of external function will be converted to conform to the function type specified. This is within the constraints of argument types integer through complex.

A generic name is assigned to each set of functions in Table 6-2. The generic names are:

EXP	ALOG	ALOG10	SIN	COS
TANH	SQRT	ATAN	ATAN2	

When the mathematical library functions are referenced by their generic names, the type of the function is determined by the type of the argument(s) within the constraints of the types described in Table 6-2. The one exception is when an integer argument is specified to a generic function. In this case, the argument is converted and the real form of the function is called. Note that the type of ATAN2 is double precision if at least one of its arguments is double precision.

Table 6-2. Supplied FUNCTION Subprograms

BASIC FUNCTION	DEFINITION	NO. OF ARG.	SYMB. NAME	TYPE OF: ARG.	TYPE OF: FUNCT.
Exponential	e^{a}	1	EXP	Real	Real
		1	DEXP	Double	Double
		1	CEXP	Complex	Complex
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
		1	CLOG	Complex	Complex
Common Logarithm	$\log_{10}(a)$	1	ALOG10	Real	Real
		1	DLOG10	Double	Double
Trigonometric Sine	sin(a)	1	SIN	Real	Real
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex
Trigonometric Cosine	cos(a)	1	COS	Real	Real
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real
Square Root	$(a)^{\frac{1}{2}}$	1	SQRT	Real	Real
		1	DSQRT	Double	Double
		1	CSQRT	Complex	Complex
Arctangent	arctan(a)	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\arctan(a_1/a_2)$	2	ATAN2	Real	Real
		2	DATAN2	Double	Double
Remaindering [a]	$a_1 (\mathrm{mod}\ a_2)$	2	DMOD	Double	Double
Modulus		1	CABS	Complex	Real

[a] Remaindering (DMOD (a_1, a_2)) is defined as $a_1 - [a_1/a_2]a_2$, where the bracketed expression denotes the integral result of the expression a_1/a_2.

Referencing FUNCTION Subprograms

A FUNCTION subprogram is referenced by using its symbolic name with a list of actual arguments in standard function notation as a primary in an expression. The actual arguments, which constitute the argument list, must agree in order, number, and type with the corresponding dummy arguments in the FUNCTION subprogram definition. Actual arguments in the function reference may be one of the following:

1. A variable name
2. An array element name
3. An array name
4. Any other expression
5. Name of an external procedure
6. Constant

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

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

Execution of a FUNCTION reference results in an association of actual arguments with all appearances of dummy arguments in the defining subprogram. If the actual argument is an expression, or constant then this association is by value rather than by name. Following these associations, execution of the first executable statement of the defining subprogram begins. An actual argument which is an array element name containing variables in the subscript could in every case be replaced by the same argument with a constant subscript containing the same values as would be derived by computing the variable subscript just before the association of arguments takes place.

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

If a function reference causes a dummy argument in the referenced function to become associated with another dummy argument in the same function or with an entity in COMMON, a definition of either within the function is prohibited.

Unless it is a dummy argument, a FUNCTION subprogram is also referenced (in that it must be defined) by the appearance of its symbolic name in an EXTERNAL statement.

If a user FUNCTION subprogram is written in a language other than FORTRAN, it is the user's responsibility to insure that the correct indicators, as well as the correct numerical results, are returned to the calling program.

Example of FUNCTION Subprogram

Definition

```
      FUNCTION DIAG (A,N)
      DIMENSION   A (N,N)
      DIAG = A(1,1)
      IF (N .LE. 1) RETURN
      DO 6I = 2, N
6     DIAG = DIAG * A(I,I)
      RETURN
      END
```

Reference

```
      DIMENSION X (8,8)
      DET = DIAG (X,8)
```

SUBROUTINE SUBPROGRAM

A SUBROUTINE subprogram differs from a FUNCTION subprogram in three ways:

1. A SUBROUTINE has no value associated with its name. All results are defined in terms of arguments or common; there may be any number of results.

2. A SUBROUTINE is not called into action simply by writing its name, since no value is associated with the name. A CALL statement brings it into operation. The CALL statement specifies the arguments, and results in storing all output values.

3. There is no type or convention associated with the SUBROUTINE name. The naming is otherwise the same as for the FUNCTION.

It is the user's responsibility to insure that the number and type of arguments in the calling program statement corresponds with the number and type of arguments expected by the called routine. This applies for all subroutines and functions (library or other).

Defining SUBROUTINE Subprograms

A SUBROUTINE statement is of the form:

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

where s is the symbolic name of the SUBROUTINE to be defined.

a_i, called dummy arguments, are each a variable name, an array name, an external procedure name, or an alternate return.

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

The SUBROUTINE subprogram may define or redefine one or more of its arguments so as to effectively return results.

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

The SUBROUTINE subprogram must contain at least one RETURN statement.

Referencing SUBROUTINE Subprograms

A SUBROUTINE is referenced by a CALL statement. The actual arguments which constitute the argument list, must agree in order, number, and type with the corresponding dummy arguments in the defining subprogram. An actual argument in the SUBROUTINE reference may be one of the following:

1. A constant
2. A variable name
3. An array element name
4. An array name
5. Any other expression
6. The name of an external procedure
7. An alternate return.

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

Execution of a subroutine reference results in an association of actual arguments with all appearances of dummy arguments in the defining subprogram. If the actual argument is as specified in (1) or (5) above, this association is by value rather than by name.

Following these associations, execution of the first executable statement of the defining subprogram is undertaken.

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

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

If a SUBROUTINE reference causes a dummy argument in the referenced subroutine to become associated with another dummy argument in the same subroutine, or with an entity in COMMON, a definition of either entity within the subprogram is prohibited.

Unless it is a dummy argument, a SUBROUTINE is also referenced (in that it must be defined) by the appearance of its symbolic name in an EXTERNAL statement.

SUBROUTINE Subprogram Example

Defining

```
      SUBROUTINE LARGE (ARRAY,I,BIG,J)
      DIMENSION ARRAY (50,50)
      BIG=ABS (ARRAY(I,1))
      J=1
      DO 6 K=2,50
      IF (ABS (ARRAY(I,K)) .LE. BIG) GO TO 6
      BIG=ABS (ARRAY(I,K))
      J=K
    6 CONTINUE
      RETURN
      END
```

Referencing

```
      CALL LARGE (ZETA,N,VAL,NCOL)
```

Returns From Function And Subroutine Subprograms

The RETURN statement is used to terminate all subprograms. This statement causes control to be returned to the calling program. There may be any number of RETURN statements in a subprogram. The RETURN statement has the form:

```
RETURN

or RETURN i
```

where i is an integer constant or variable whose value denotes the nth * in the argument list of the SUBROUTINE statement, reading from left to right.

The normal sequence of execution following the RETURN statement of a subprogram is to the next executable statement following the CALL or function name statement in the calling program. It is possible to return to any numbered executable statement in the calling program by using a special return from the called subprogram (for SUBROUTINE subprograms only). This return must not violate the transfer rules for DO loops. FUNCTION subprograms must not have nonstandard returns.

The following text describes the form of the FORTRAN statements that is required to return from the SUBROUTINE to a statement other than the next executable statement following the CALL.

The general form of the CALL statement in the calling program is:

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

where subr is the name of the SUBROUTINE subprogram being called. Each a_i is an argument of the form described with the CALL statement or is of the form:

$n

where n is a statement number, or switch variable used for a nonstandard return.

The general form of the SUBROUTINE statement in the called subprogram is:

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

where subr is the name of the subprogram. Each a_i is a dummy argument of the form described under SUBROUTINE subprograms, or is of the form:

*

where the asterisk (*) denotes a nonstandard return.

The general form of the RETURN statement in the called subprogram is:

RETURN i

where i is an integer constant or variable which denotes the ith nonstandard return in the argument list, reading from left to right. For example:

```
CALLING PROGRAM                    CALLED SUBPROGRAM

        .                          SUBROUTINE SUB(X,Y,Z,*,*)
        .                                  .
        .                                  .
10 CALL SUB(A,B,C,$30,$40)                 .
                                           .
20 - - - - -                  100 IF(R) 200,300,400
        .                     200 RETURN
        .                     300 RETURN  1
30 - - - - -                  400 RETURN  2
        .                         END
        .
40 - - - - -
        .
        .
      END
```

In the preceding example, execution of statement 10 in the calling program causes entry into subprogram SUB. If statement 100, in subprogram SUB, is executed, the return to the calling program will be to statement 20, 30, or 40 if R is less than, equal to, or greater than zero, respectively.

Non-standard returns may best be understood by considering that a CALL statement that uses the non-standard return is equivalent to a CALL and a computed GO TO statement in sequence. For example:

```
CALL NAME (P,$20,Q,$35,R,$22) is equivalent to
```

```
CALL NAME (P,Q,R,I)
GO TO (20,35,22)I
```

where I is set to the value of the integer in the RETURN statement executed in the called subprogram. If the RETURN index is not specified or is zero, a normal (rather than nonstandard) return is made to the statement immediately following the GO TO.

The intermingling of arguments and alternate returns may be done freely in both the CALL and SUBROUTINE statements. The compiler separates the combined list into two separate lists, such that argument n is the nth actual or dummy argument, and alternate return n is the nth statement number or *, reading left to right. Thus, the following are equivalent:

```
CALL NAME (P,$20,Q,$35,R,$22)
CALL NAME (P,Q,R,$20,$35,$22)
CALL NAME ($20,$35,$22,P,Q,R)
```

as are the following:

```
SUBROUTINE NAME (S,*,T,*,U,*)
SUBROUTINE NAME (S,T,U,*,*,*)
SUBROUTINE NAME (*,*,*,S,T,U)
```

Multiple Entry Points Into a Subprogram

The normal entry into a SUBROUTINE subprogram from the calling program is made by a CALL statement that refers to the subprogram name. The normal entry to a FUNCTION subprogram is made by a function reference in an expression. Entry is made at the first executable statement following the FUNCTION or SUBROUTINE statement.

It is also possible to enter a subprogram at an alternate entry point by a CALL statement or a function reference that refers to an ENTRY statement in the subprogram. Entry is made at the first executable statement following the ENTRY statement.

ENTRY statements are nonexecutable and, therefore, do not affect control sequencing during normal execution of a subprogram. The order, type, and number of arguments need not agree between the SUBROUTINE or FUNCTION statement and any ENTRY statement, nor do ENTRY statements have to agree among themselves in these respects. Each CALL or FUNCTION reference, however, must agree in order, type, and number of actual arguments with the dummy arguments of the SUBROUTINE, FUNCTION, or ENTRY statement that it refers to. No subprogram may refer to itself directly or through any of its entry points, nor may it refer to any other subprogram whose RETURN statement has not been satisfied.

Example:

```
Calling Program                 Called Program

   .                            SUBROUTINE SUB1(U,V,W,X,Y,Z)
   .                                .
   .                                .
1 CALL SUB1 (A,B,C,D,E,F)           .
   .                           10 U = V
   .                                .
   .                                .
2 CALL SUB2(G,H,P)                  .
   .                            GO TO 60
   .                                .
   .                                .
                                ENTRY SUB2(T,U,V)
                                GO TO 10
3 CALL SUB3                 60
   .                                .
   .                            GO TO 90
   .                            ENTRY SUB3
END                                 .
                                90 RETURN
                                   END
```

In the preceding example, the execution of statement 1 causes entry into SUB1, starting with the first executable statement of the subroutine. Execution of statements 2 and 3 also cause entry into the called program, starting with the first executable statement following the ENTRY SUB2(T,U,V) and ENTRY SUB3 statements, respectively.

Dummy Argument

A dummy argument is used to make entities in a referencing program available to the referenced subprogram.

A dummy argument of a subprogram may be associated with an actual argument which may be a variable, array, array element, subroutine, external function, constant, expression, or in the case of subroutines, statement number to which a special return may be made from a subroutine program.

The dummy argument may be used in the subprogram as a scalar variable, an array, a subroutine, or function name.

When the use of a statement number is specified (to which a special return may be made from a subroutine subprogram), the use of the * in a dummy argument position is required if a statement number will be associated with that dummy argument.

When the use of an external function name is specified, the use of a dummy argument is permissible if an external function name will be associated with that dummy argument.

When the use of a variable or array element reference is specified, the use of a dummy argument is permissible if a value of the same type will be made available through the argument association.

Unless otherwise specified, when the use of a variable, array, or array element name is specified, the use of a dummy argument is permissible provided that a proper association with an actual argument is made.

Supplied SUBROUTINE Subprograms

Table 6-3 contains a list of FORTRAN supplied SUBROUTINE subprograms.

Table 6-3. Supplied SUBROUTINE Subprograms

SUBPROGRAM	USE	CALL
DUMP (BCD) or DUMPA (ASCII)	Dumps designated area of core in a specified format; terminate execution	CALL DUMP (DUMPA) (A_1,B_1, F_1,...,A_n,B_n,F_n)
PDUMP (BCD) or PDUMPA (ASCII)	Dump designated area of core in a specified format; return	CALL PDUMP (PDUMPA) (A_1,B_1,F_1,...,A_n,B_n,F_n)
EXIT	Purge buffers and terminate current activity	CALL EXIT
FCLOSE	Close File and release buffer	CALL FCLOSE (I)
SLITE	Clear sense lights 1-35	CALL SLITE (ZERO)
SLITE	Turn on sense light i	CALL SLITE (I)
SLITET	Test and turn off sense light i	CALL SLITET (I,J)
SSWTCH	Test sense switch i	CALL SSWTCH (I,J)
FXEM	Execution Error Monitor	
LINK	Restore link and transfer to its entry point	CALL LINK (LINKID)
LLINK	Restore link and return to next statement in calling subroutine	CALL LLINK (LINKID)
SETBUF	Define buffer for FCB	CALL SETBUF (I,A,B)
SETFCB	Define File Control Block	CALL SETFCB (A,I,J...)
SETLGT	Define Logical File Table	CALL SETLGT (A,I)
CNSLIO	Console Communications	CALL CNSLIO (CONSOL, MESSAGE, NWORDS, NREPLY, NREPWS)
OVERFL	Exponent Register Overlow	CALL OVERFL(j)
DVCHK	Divide Check	CALL DVCHK(j)
FLGEOF	End-of-file processing	CALL FLGEOF (U,V)
FLGERR	Data error processing	CALL FLGERR (U,V)
RANSIZ	Specify record size of random file	CALL RANSIZ (U,N,M)
FPARAM	Set or reset I/O parameters	CALL FPARAM (I,J)

Table 6-3 (cont). Supplied SUBROUTINE Subprograms

SUBPROGRAM	USE	CALL
CREATE	Create a Temporary Mass Storage or Teletypewriter File	CALL CREATE (LGU,ISIZE,MODE, ISTAT)
DETACH	Deaccess a Current File	CALL DETACH (LGU, ISTAT,BUFFER)
ATTACH	Access an Existing Permanent File	CALL ATTACH (LGU, CATFIL,IPRMIS, MODE,ISTAT, BUFFER)
FMEDIA	Conforming Output Transliteration	CALL FMEDIA (FC, MEDIA)
ASCB	Obtain Internal Character Set of Executing Program (BCD)	CALL ASCB
ASCBA	Obtain Internal Character Set of Executing Program (ASCII)	CALL ASCBA
TRACE	Invokes time sharing debug and trace package	CALL TRACE

DUMP (DUMPA), PDUMP (PDUMPA)

This SUBROUTINE subprogram dumps all or a designated area of core storage in a specified format. If DUMP is called, execution is terminated by a call to EXIT. If PDUMP is called, control is returned to the calling program.

Calling Sequence:

CALL DUMP or DUMPA (A_1,B_2,F_3,...,A_n,B_n,F_n)

CALL PDUMP or PDUMPA (A_1,B_2,F_3,...,A_n,B_n,F_n)

where A and B indicate the limits of the core storage area to be dumped. A or B may represent the upper or lower limit. F is an integer specifying the dump format. If no arguments are given, all of core is dumped in octal. The values for F are as follows:

F = 0 Octal
F = 1 Integer
F = 2 Real
F = 3 Double Precision
F = 4 Complex
F = 5 Logical
F = 6 Character

DUMPA and PDUMPA are the ASCII versions.

EXIT

This SUBROUTINE purges all buffers and terminates the current activity. Control is returned to the Comprehensive Operating Supervisor.

Calling Sequence:

CALL EXIT

FCLOSE

This SUBROUTINE closes file I and releases the buffer(s) assigned. The buffer is released only if it is the standard size (320 words). Return is to the next executable statement in the calling program.

Calling Sequence:

CALL FCLOSE(I)

where I is the logical file designator.

FLGEOF

This SUBROUTINE provides a signal requesting a return to the calling subprogram if an end-of-file condition occurs. Return is to the next executable statement in the calling program.

Calling Sequence:

```
CALL FLGEOF(I,V)
```

where I is the logical file designator.

V is the address of the variable used to indicate an end-of-file condition (user must test V (for a non-zero) when an end-of-file condition could have occurred). V should not generally be used for any other purpose.

FLGERR

This SUBROUTINE provides a variable used in detecting the occurrence of erroneous data. Return is to the next executable statement in the calling program.

Calling Sequence:

```
CALL FLGERR(I,V)
```

where I is the logical file designator.

V is the variable used to indicate an input data error (user must test V (for a nonzero value) when an error condition could have occurred). V should not generally be used for any other purpose.

SLITE

This SUBROUTINE simulates the setting and testing of sense lights. Normal return is to the next executable statement in the calling program.

Calling Sequences:

CALL SLITE(0) to clear sense lights 1-35

CALL SLITE(I) to turn on sense light i ($1 \geq i \leq 35$)

CALL SLITET(I,J) to test and turn off sense light i.

where I is an integer variable or constant.

J is an integer variable which is set to 1 if sense light i was ON; set to 2 if sense light i was OFF. J may not be the induction variable of a currently active DO loop.

SSWTCH

This SUBROUTINE tests the GCOS switch word for the status of a sense switch. Normal return is to the next executable statement in the calling program.

Calling Sequence:

CALL SSWTCH(I,J) to test sense switch i

where I is an integer variable or constant which must be from 1 to 6.

J is an integer variable that is set to 1 if the switch i is ON and is set to 2 if the switch is OFF. J may not be the induction variable of a currently active DO loop.

Bits 6-11 of the Program Switch Word, described in the General Comprehensive Operating Supervisor reference manual, correspond to sense switches 1-6.

EXECUTION ERROR MONITOR. This SUBROUTINE performs the following functions:

1. Prints a trace of subroutine calls
2. Prints execution error messages
3. Terminates execution with a Q6 abort or does one of the following:
   a. Continues with execution of the program.
   b. Transfers to an alternate error routine.
4. Allows the user to determine if an error has been processed by the Execution Error Monitor.

This is accomplished by the setting/resetting of bits in switch word pairs that control termination, message printing and trace, and alternate error return for the errors in Table 6-4.

Calling Sequences:

CALL ANYERR(V)

where V is a variable into which the Execution Error Monitor will place the error code (see Table 6-4) if an error occurs. V should not generally be used for any other purpose.

CALL FXEM(NCODE,MSG,N)

This call will cause the display on file 06 of an error trace and the message contained in MSG which must be a character constant or variable. The number of words, N, to be printed must be within the limits $0 \geq n \leq 20$. If only the first argument is given, only the trace is printed. NCODE is the error code (see Table 6-4) expressed as an integer in the range $1 \geq n \leq 143$.

```
CALL FXOPT(NCODE,I1,I2,I3)
```

This call to the Execution Error Monitor is used to alter the standard switch word settings listed in Table 6-4. NCODE is the error code. I1, I2, I3 refer to the switch words settings for termination, message printing and trace, and alternate error procedure respectively. If I1=1, continue execution; if=0, abort with a Q6 abort. If I2=1, suppress printing; if=0, print. If I3=1, use alternate error procedure; if=0, use normal return. This option overrides the termination option.

Examples:

1. CALL FXOPT(32,0,1,0)
2. CALL FXOPT(32,1,0,0)
3. CALL FXOPT(32,0,0,1)

Example 1 causes a Q6 abort, for error number 32, when the error occurs. No message or trace is printed. Example 2 causes execution to continue after message and trace are printed. Example 3 indicates that return is to an alternate error routine after trace and message are printed. The alternate return takes precedence over termination.

```
CALL FXALT(SR)
```

The FXALT call is used to set the alternate error procedure location. The statement indicates that SR is the alternate error procedure location. An EXTERNAL SR must be included in the calling routine. If the alternate procedure option for an error code is indicated but no call to FXALT has been made, a Q5 abort occurs when the error condition occurs. A RETURN statement in the alternate routine continues execution at the next executable statement after the statement that caused the error.

The alternate error procedure must not invoke the routine in which the error was found; i.e., the alternate error procedure for a formatted input/output statement must not do formatted input/output.

The statement CALL FXALT($N) designates statement N in the calling program as the alternate error return.

NOTE: If the same error occurs in the alternate error routine, a loop results.

The standard setting of bits in the FXSW1 switchword pairs (termination) are indicated in Table 6-4. The settings in the second and third switch word pairs (trace and alternate return) are initially zero.

Table 6-4. Error Codes and Meanings

ERROR CODE	DEFAULT PROCEDURE ABORT/ CONTINUE	FUNCTION	ERROR	EXCEPTION RETURN	MESSAGE LINE 1	MESSAGE LINE 2
0	A	Not used				
1	C	I**J	I=0,J=0	0→QR	EXPONENTIATION ERROR 0**0	SET RESULT=0
2	C	I**J	I=0,J<0	(2**35)-2→QR	EXPONENTIATION ERROR 0**(-J)	SET RESULT=2**35-2
3	C	DA**J A**J	DA=0,J=0 A=0,J=0	0→EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
4	C	A**J DA**J	A=0,J<0 DA=0,J<0	10**38→EAQ	EXPONENTIATION ERROR 0**(-J)	SET RESULT=10**38
5	C	B**C	B<0,C≠0	0→EAQ	EXPONENTIATION ERROR (-B)**C	SET RESULT=0
6	C	A**B	A=0,B=0	0→EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
7	C	A**C	A=0,C<0	10**38→EAQ	EXPONENTIATION ERROR 0**(-C)	SET RESULT=10**38
8	C	e**B	B>88.028	10**38→EAQ	EXP(B),B GRT THAN 88.028 NOT ALLOWED	SET RESULT=10**38
9	C	LOG(A)	A=0	-(10**38)→EAQ	LOG(0) NOT ALLOWED	SET RESULT-(10**38)
10	C	LOG(B)	B<0	0→EAQ	LOG(-B) NOT ALLOWED	SET RESULT=0.0
11	C	ARCTAN(A/B)	A=0,B=0	0→EAQ	ATAN2(0,0) NOT ALLOWED	SET RESULT=0
12	C	SIN(A) COS(A)	$\|A\|>2^{27}$	0→EAQ	SIN OR COS ARG GRT TH 2**27 NOT ALLOWED	SET RESULT=0
13	C	$\sqrt{B}$	B<0	$\sqrt{B}=\sqrt{\|B\|}$	SQRT(-B) NOT ALLOWED	EVALUATE FOR +B
14	C	CA**K	CA=0,K=0	0→AQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
15	C	CA**J	CA=0,J<0	10**38→AR 0→QR	EXPONENTIATION ERROR 0**(-J)	SET RESULT=(10**38,0.0)

Table 6-4. (cont) Error Codes and Meanings

16	C	DA**DB	DB≠0, DA<0	0⇒EAQ	EXPONENTIATION ERROR (-DA)**DB	SET RESULT=0
17	C	DA**DB	DA=0, DB=0	0⇒EAQ	EXPONENTIATION ERROR 0**0	SET RESULT=0
18	C	DA**DB	DA=0, DB<0	10**38⇒EAQ	EXPONENTIATION ERROR 0**(-DB)	SET RESULT=10**38
19	C	e**DA	DA > 88.028	10**38⇒EAQ	EXP(B), B GRT 88.028, NOT ALLOWED	SET RESULT=10**38
20	C	LOG(DA)	DA=0	-(10**38)⇒EAQ	DLOG(0) NOT ALLOWED	SET RESULT=-(10**38)
21	C	LOG(DA)	DA<0	0⇒EAQ	DLOG(-B) NOT ALLOWED	SET SET RESULT=0
22	C	$\sqrt{DA}$	DA<0	$\sqrt{DA}=\sqrt{\lvert DA\rvert}$	SQRT(-B) NOT ALLOWED	EVALUATE FOR +B
23	C	$\begin{Bmatrix}SIN(DA)\\COS(DA)\end{Bmatrix}$	$\lvert DA\rvert > 2^{54}$	0⇒EAQ	DSIN OR DCOS ARG GRT 2**54 NOT ALLOWED	SET RESULT=0
24	C	ARCTAN(DA/DB)	DA=0, DB=0	0⇒EAQ	DATAN2(0,0) NOT ALLOWED	SET RESULT=0
25	C	CA/CB	CB=(0,0)	10**38⇒AR 10**38⇒QR	COMPLEX Z/0 NOT ALLOWED	SET RESULT=(10**38, 10**38)
26	C	e**CA	REAL CA>88.028	10**38⇒AR 10**38⇒QR	EXP(Z), REAL PART GRT 88.028 NOT ALLOWED	SET RESULT=(10**38, 10**38)
27	C	e**CA	$\lvert$IMAG CA$\rvert > 2^{27}$	0⇒AR 0⇒QR	EXP(Z), IMAG PART GRT 2**27 NOT ALLOWED	SET RESULT=(0,0)
28	C	LOG(CA)	CA=(0,0)	-(10**38)⇒AR 0⇒QR	CLOG(0) NOT ALLOWED	SET RESULT (-(10**38),0.0)
29	C	$\begin{Bmatrix}SIN(CA)\\COS(CA)\end{Bmatrix}$	$\lvert REAL(CA)\rvert > 2^{27}$	0⇒AQ	CSIN OR CCOS ARG WITH REAL PART GRT 2**27 NOT ALLOWED	SET RESULT=0
30	C	$\begin{Bmatrix}COS(CA)\\SIN(CA)\end{Bmatrix}$	|IMAG(CA)|>88.028	10**38⇒AR 10**38⇒QR	CSIN OR CCOS ARG WITH IM PART GRT 88.028 NOT ALLOWED	SET RESULT=(10**38, 10**38)
31	C	BCD I/O	ILLEGAL FORMAT STATEMENT	------	FORMAT AT LLLLLL, FIRST WORD HHHHHH IS ILLEGAL	TREAT AS END OF FORMAT
32	C	BCD I/O	ILLEGAL CHARACTER IN DATA OR BAD FORMAT		ILLEGAL CHAR IN DATA OR BAD FORMAT	TREAT ILLEGAL CHAR AS ZERO

Table 6-4. (cont) Error Codes and Meanings

33	A	BCD I/O	ATTEMPT TO READ OUTPUT FILE -----	READ AFTER WRITE IS ILLEGAL	FC # xx
34	A	BCD I/O	END-OF-FILE -----	END OF FILE READING FILE CODE FC	OPTIONAL RETURN NOT REQUESTED
35	C	REWIND AND END FILE PROCESSOR	ILLEGAL REQUEST	REQUEST TO XXXXXX ON FC WAS IGNORED	-----
36	C	FFFB	BACKSPACE ERROR -----	TAPE POSITIONED AT 1ST FILE	BACKSPACE REQ. LARGER THAN FILE COUNT
37	A	FILE OPENING	FILE NOT DEFINED	LOG. FILE CODE FC DOES NOT EXIST	NO OPTIONAL EXIT EXECUTION TERMINATED
38	A	FILE OPENING	NO SPACE FOR I/O BUFFERS	INSUFFICIENT CORE AVAILABLE FOR BUFFERS	NOT OPTIONAL EXIT EXECUTION TERMINATED
39	A	BINARY I/O	ILLEGAL END-OF-FILE	UNEXPECTED EOF	OR BAD FORMAT; FILE # xx
40	C	BINARY I/O	LIST EXCEEDS LOGICAL RECORD LENGTH	LIST EXCEEDS LOGICAL RECORD LENGTH	STORE ZEROS IN REMAINING LIST ITEMS; FC
41	A	BINARY I/O	SYSOUT/FIXED LENGTH RECORDS	SYSOUT OR FIXED LENGTH RECORDS MUST	BE SMALLER THAN BLOCK SIZE; FILE #xx
42	C	NAMELIST INPUT	ILLEGAL HEADING CARD	ILLEGAL HEADING CARD BELOW	SCAN TERMINATED
43	C	NAMELIST INPUT	ILLEGAL VARIABLE NAME	ILLEGAL VARIABLE NAME BELOW	SKIPPING TO NEXT VARIABLE NAME
44	C	NAMELIST INPUT	ILLEGAL SUBSCRIPT OR ARRAY SIZE EXCEEDED	ILLEGAL SUBSCRIPT BELOW, OR DATA EXCEEDS VARIABLE	SKIPPING TO NEXT VARIABLE NAME
45	C	NAMELIST INPUT	ILLEGAL CHARACTER AFTER RIGHT PARENTHESIS	ILLEGAL CHAR IN DATA BELOW	ASSUME COMMA PRECEDES CHAR
46	C	NAMELIST INPUT	ILLEGAL CHAR IN DATA	ILLEGAL CHAR IN DATA BELOW	TREAT CHAR AS ZERO
47	A	BACKSPACE RECORD	FILE CANNOT BE BACKSPACED	FILE CODE NN, BACKSPACE REFUSED	FILE IS SYSOUT OR IS NOT MAGNETIC TAPE, DISK OR DRUM
48	C	NAMELIST INPUT	ILLEGAL LOGICAL CONSTANT	ILLEGAL LOGICAL CONSTANT APPEARS BELOW (OR AT END OF PRECEDING RECORD)	TREAT ILLEGAL LOGICAL CONSTANT AS F
49	A	BACKSPACE FILE	ERRONEOUS END-OF-FILE	ERRONEOUS END OF FILE ON BACKSPACE	
50	C	BACKSPACE FILE	BLOCK COUNT OF ZERO	BLOCK COUNT IN FCB EQUALS ZERO	

Table 6-4. (cont) Error Codes and Meanings

51	C	SENSE LIGHT SIMULATOR	INDEX NOT $0 \leq n \leq 35$		REFERENCE TO NON-EXISTENT SENSE LIGHT	DECLARED OFF IF TESTING IGNORED IF SETTING
52	C	NAMELIST INPUT	ILLEGAL HOLLERITH FILED		ILLEGAL HOLLERITH FIELD BELOW	SKIPPING TO NEXT VARIABLE NAME
53	C	SENSE SWITCH TEST	INDEX NOT $1 \leq n < 6$		NON-EXISTENT SENSE SWITCH TESTED	SWITCH DECLARED OFF
54	A	FILE OPENING	ATTEMPT TO WRITE I*		ILLEGAL WRITE REQUEST ON SYSIN1	NO OPTIONAL EXIT EXECUTION TERMINATED
55	A	FXEM	NAMELIST INPUT		ILLEGAL COMPUTED GO TO	
56	A	FILE OPENING	ATTEMPT TO READ P*		ILLEGAL READ REQUEST ON SYSOU1 OR SYSPP1	
57	C	BCD I/O	ILLEGAL CHAR FOR L CONVERSION		ILLEGAL CHAR FOR L CONVERSION IN DATA BELOW	TREAT ILLEGAL CHARACTER AS SPACE
58	C	BACKSPACE RECORD			FILE NN IS CLOSED	BACKSPACE REFUSED
59	C	NAMELIST INPUT	EMPTY HOLLERITH FIELD		EMPTY HOLLERITH FIELD	
60	C	I**J	I**J>2**35 |I| >1, J>35 J IS EVEN I<-1, J>35, J IS ODD	(2**35)-2→QR (2**35)-2→QR -((2**35)-2)→QR	EXPONENT OVERFLOW	SET RESULT=+ $\overline{((2**35)-2)}$
61 62 63 64 65 66		RESERVED FOR USERS				
67	C	FAULT	EXPONENT UNDERFLOW		EXPONENT UNDERFLOW	AT LOCATION XXXXXX
68	C	FAULT	INTEGER OVERFLOW		OVERFLOW	AT LOCATION XXXXXX
69	C	FAULT	EXPONENT OVERFLOW		EXPONENT OVERFLOW	AT LOCATION XXXXXX
70	C	FAULT	INTEGER DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX
71	C	FAULT	FLOATING POINT DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX

Table 6-4. (cont) Error Codes and Meanings

72	C	RANDOM I/O	LIST EXCEEDS LOGICAL RECORD LENGTH	LIST EXCEEDS LOGICAL RECORD LENGTH	STORE ZEROS IN REMAINING LIST ITEMS FC # XX
73	A	RANDOM I/O	FILE NOT STANDARD SYSTEM FORMAT. ZERO BLOCK COUNT; BSN ERROR; ZERO RECORD COUNT	FILE NOT STANDARD SYSTEM FORMAT FILE # FC	
74	A	RANDOM I/O	NO FCB FOR FILE	LOGICAL FILE CODE FC DOES NOT EXIST	NO OPTIONAL EXIT EXECUTION TERMINATED
75	A	RANDOM I/O	BAD RECORD REFERENCE	ZERO OR NEGATIVE REC #	FC # XX
76	A	RANDOM I/O	RECORD SIZE NOT SPECIFIED IN FCB. GIVE VIA $ FFILE CARD OR CALL RANSIZ (FC,SIZE)	REC SIZE NOT GIVEN FOR RANDOM FILE	FC # XX
77	A	RANDOM I/O	RANDOM WRITE TO LINKED FILE ILLEGAL. LINKED FILE MAY BE READ RANDOMLY BUT NOT WRITTEN TO.	RANDOM WRITE TO LINKED FILE ILLEGAL	FC # XX
78	A	RANDOM I/O	THE RECORD NO. GIVEN IN THE RANDOM READ OR WRITE STATEMENT IS OUTSIDE THE FILE LIMITS.	REC # OUT-OF-BOUNDS-	FC # XX
79	A	RANDOM I/O (BINARY)..	FILE MEDIA CODE NOT 01	MEDIA NOT BINARY	FC # XX
80	A	RANDOM I/O	FILE IS NOT LARGE ENOUGH TO CONTAIN RECORD	FILE SPACE EXHAUSTED-	FC # XX
81-141		NOT PRESENTLY USED			

NOTATION: I,J,K are integers
A,B,C are real numbers
DA,DB,DC are double-precision numbers
CA,CB,CC where CA=X,Y are complex numbers

OVERFLOW, DIVIDE CHECK

The FORTRAN fault processor processes integer and floating-point divide check, exponent overflow, and exponent underflow faults. A message is output on file 06 stating the type of fault and the core storage location at which the fault occurred. Execution continues with the EAQ register set as follows:

FAULT	EAQ REGISTER
Exponent overflow	Large floating-point value[1]
Divide check (floating-point)	Large floating-point value[1]
Exponent underflow	Floating-point zero
Overflow (Integer)	No change
Divide check (Integer)	No change

To have another value returned in the EAQ after a divide check fault, use the following call prior to the statement that can cause the fault.

```
CALL FXDVCK(R,M)
```

This statement causes the value R to be returned in the EAQ after a floating-point divide check and the value M to be returned in the Q-register after an integer divide check. The second argument may be omitted. The value R must be double-precision.

LINK AND LLINK

The LINK SUBROUTINE enables the programmer to call program overlays. The following call is used to call a link and transfer control to it.

```
CALL LINK(NAME)
```

where NAME designates the variable name of the link as it appears on the $ LINK control card. (See the General Loader Reference Manual, for "Link/Overlay Processing".) Name may be a variable, which currently has a character type value, or it may be a character constant, e.g., "LINK1". The link name must be 1-6 characters if using the BCD option, or must be 5-6 characters if using the ASCII option. Explicit trailing blanks are included in the character count.

To load a link and return to the next sequential statement of the calling routine, the required statement is:

```
CALL LLINK(NAME)
```

---

[1]Allows further computation without another immediate fault. This value is set to approximately 10**36.

SETBUF

This SUBROUTINE allows the user to assign space in core storage for use as an input/output buffer(s). The size of the buffer(s) must be one location greater than the actual record size. The standard buffer size is therefore 321 words. Normal return is to the next executable statement in the calling program.

Calling Sequence:

```
CALL SETBUF(I,A)

CALL SETBUF(I,A,B)
```

where I is the logical file designator.

A is the location of first buffer.

B is the location of second buffer, if required.

SETFCB

This SUBROUTINE allows the user to define a file control block (FCB) for use by the I/O subprograms. Normal return is to the next executable statement of the calling program except for the following possible error conditions:

1. Abort with a Q2 if there is no logical file table.
2. Abort with a Q1 if there is no space available in the logical file table for inserting a specified file control block.

Calling Sequence:

```
CALL SETFCB(A,I,J,...)
```

where A is the location of LOCSYM in the user created file control block.

I,J... are the logical file designators that refer to the file control block.

SETLGT

This SUBROUTINE allows the user to define a logical unit table for use by the I/O library subprograms. Normal return is to the next executable statement of the calling program. This SUBROUTINE must be called before any input/output is requested. It is called when the user wants to suppress the logical file table generated by the General Loader and to place the table in his own portion of core storage. The NOFCB option must be specified in the $ OPTION control card.

Calling Sequence:

```
CALL SETLGT(A,I)
```

where A is the location of the logical unit table to be used.

I is the number of cells in table A.

CNSLIO

This SUBROUTINE permits operator-program communication via the console typewriter. Return is made to the next executable statement in the calling program. This subroutine is restricted to batch execution; it may not be called by a FORTRAN program executing in the time sharing mode.

Calling Sequence:

```
CALL CNSLIO(CONSOLE, MESSGE,NWORDS,NREPLY,NREPWS)
```

```
where CONSOLE = BCI     1,0000T/ for master console
                BCI     1,0000T* for tape console
                BCI     1,0000*T for unit record console
                BCI     1,0000/T for special purposes
```

If none is given, BCI 1,0000/T for special purposes

MESSGE is an array containing the message in BCI to be output.

NWORDS are the number of words to be output. Any value greater than 11 will be set to 11.

NREPLY is optional and is used when a reply is desired. When present, the reply (in BCD) will be stored beginning at location NREPLY.

NREPWS is optional and is used when a reply of more than six characters is desired. When omitted, a one word reply will be stored in NREPLY. When provided, NREPWS words (NREPWS *6 characters) will be stored beginning at location NREPLY.

RANSIZ

This subroutine permits the user to specify the record size for a random binary file. Normal return is to the next executable statement of the calling program. If the record size for a given random file is not provided at load time via the $ FFILE card, a call to this routine before opening (first I/O to) the file is mandatory.

Calling Sequence:

```
CALL RANSIZ (U,N,M)
```

where U is the logical file designator

N is the record size

M is a file format indicator.

U, N and M must be of type integer. They may be any legal arithmetic expression.

Note that a call to RANSIZ may also be used to override a $ FFILE size specification and that this is the preferred method of specification since its function works for both batch and time sharing use of Series 6000 FORTRAN.

The third argument (M) is optional. When not supplied, file U will be processed in standard system format (blocked, variable length records, etc.). When supplied, zero indicates standard system format; non-zero indicates that block and record control words are not to be processed. This latter format provides compatibility with random files generated by Series 600 Time Sharing FORTRAN. The total file space is available for data; records are not blocked, may begin anywhere in a sector and may span device boundaries.

FPARAM

This subroutine permits the user to set or reset some of the I/O parameters of the run-time library. Specifically, it may be used to:

1. Set the line length (modulo 4) for formatted output directed to a teletypewriter. The default setting for this parameter is 72.
2. Set the media code for unformatted file output. The default setting of this parameter is 1.
3. Set the reflexive read characters that are sent to a teletypewriter to request input. The default setting of this parameter is the ASCII CHARACTER constant 'carriage return', 'line feed', 'equal sign', X-ON.

Calling Sequence:

```
CALL FPARAM (I,J)
```

where I is integer, with a value of 1, 2, or 3 corresponding to one of the three functions above

J is integer, providing the line length or media code for I values of 1 and 2, or providing the octal value of four ASCII characters for an I value of 3.

Example of reflexive read signature change to "??":

```
DATA J/0015012077077/
CALL FPARAM (3,J)
```

Example of teletypewriter line length setting to 120 characters:

```
CALL FPARAM (1,120)
```

CREATE

This subroutine is used to create a temporary mass storage or teletypewriter file.

Calling Sequence:

```
CALL CREATE (LGU,ISIZE,MODE,ISTAT)
```

where LGU is the usual FORTRAN file code (integer).

ISIZE is the size, in words, of the file wanted.

MODE = 0 for a linked mas storage file
= 1 for a random mass storage file
= 2 for a teletypewriter file.

ISTAT is the status return word (see the Time Sharing System System Programmers manual for codes). The following codes also apply.

ISTAT = 1 MODE is invalid
= 2 File is currently open

If the CREATE is successful, a FCB is created and the file code, in ASCII, is placed in FCB-10, -9.

DETACH

This subroutine is used to deaccess a current file.

Calling Sequence:

```
CALL DETACH (LGU,ISTAT,BUFFER)
```

where LGU is the usual FORTRAN file code.

ISTAT is the status return word described under CREATE.

BUFFER = null argument: get space for FILSYS
= not null: use this space (at least 380 words)

In time sharing and batch, the file is closed and the buffer released. The file is also deaccessed in time sharing. If more memory is required (to deaccess the file) and the request is denied, the time sharing user is terminated.

```
ISTAT = 0: successful
      = 1: could not get FILSYS buffer (batch only);
           time sharing user is terminated.
```

ATTACH

This subroutine is used to access an existing permanent file.

Calling Sequence:

```
CALL ATTACH (LGU,CATFIL,IPRMIS,MODE,ISTAT,BUFFER)
```

where LGU is the usual FORTRAN file code.

CATFIL is the catalog/file descriptor; CATFIL is a character constant, or variable, containing the catalog/file string. It must be terminated by a semi-colon. Imbedded blanks are ignored. The master catalog password is not needed; subsequent passwords are required if part of the file description. Alternate names are permitted.

IPRMIS is the permission desired. Those are ORed with any permission in the CATFIL.

= 1 READ ONLY
= 2 WRITE ONLY
= 3 READ and WRITE
= Any other This is undefined and subject to change

MODE = 0 Get file as defined
= 1 Get file as random
= 2 Get teletypewriter

ISTAT contains the first status word returned by the File Management Supervisor (see the Time Sharing System System Programmers manual), or will contain:

0 = successful
1 = file is currently open
2 = teletypewriter requested in batch mode (illegal)
3 = additional memory needed, request denied
(time sharing user will be terminated)
4 = CATFIL all blanks

BUFFER = Null ARG: Get a file system buffer.
= Not a null ARG: Use this buffer (at least 380 words).

The following is an example of a null argument:

```
Call ATTACH (LGU, CATFIL, IPRMIS, MODE, ISTAT,  )
```

Upon successful return from ATTACH, an FCB will have been created, and the file name (or alternate name) will be in FCB -10, -9 (in ASCII). If the file was in the AFT with a subset of the desired permissions, it will be deaccessed and reaccessed with the new permissions.

FMEDIA

This subroutine allows the user to cause transliteration to occur on files directed to mass storage or tape.

Calling Sequence:

```
CALL FMEDIA (FC,MEDIA)
```

where MEDIA = 0 for BCD NOSLEW
2 for BCD CARD IMAGES
3 for BCD SLEW
5 for TSS ASCII FORMAT (Time Sharing Only)
6 for STANDARD SYSTEM ASCII FORMAT
others are ignored

FC = Logical File Code

The legal combinations are as follows:

0 to 2	3 to 0
0 to 3	3 to 2
0 to 5	3 to 5
0 to 6	3 to 6
2 to 0	6 to 0
2 to 3	6 to 2
2 to 5	6 to 3
2 to 6	6 to 5

ASCB, ASCBA

These subprograms enable the run time routines of Series 6000 FORTRAN to reference an indicator to obtain the internal character set of executing programs. If Software Release C/5 or earlier object decks are used, the user must include a $ USE .ASCB (for BCD) or $ USE .ASCBA (for ASCII).

TRACE

This subroutine is callable from a FORTRAN object program in the time sharing mode. It is useful in tracing and debugging an object module. See the Debug and Trace Routines manual.

APPENDIX A

CHARACTER SET

ASCII CHAR	ASCII CODE	BCD CHAR	BCD CODE	MODEL 33/35 KEY	HOLLERITH CARD	MEANING
NULL	000	---	---	'CS'P	---	Null or time fill char
SOH	001	---	---	'C'A	---	Start of heading
STX	002	---	---	'C'B	---	Start of text
ETX	003	---	---	'C'C (EOM)	---	End of text
EOT	004	---	---	'C'D (EOT)	---	End of transmission
ENQ	005	---	---	'C'E (WRU)	---	Enquiry (who are you)
ACK	006	---	---	'C'F (RU)	---	Acknowledge
BEL	007	---	---	'C'G (BELL)	---	Bell
BS	010	---	---	'C'H	---	Backspace
HT	011	---	---	'C'I (TAB)	---	Horizontal tabulation
LF	012	---	---	LINE FEED	---	Line Feed (New Line)
VT	013	---	---	'C'K (VT)	---	Vertical Tabulation
FF	014	---	---	'C'L (FORM)	---	Form Feed
CR	015	---	---	RETURN	---	Carriage Return
SO	016	---	---	'C'N	---	Shift Out
SI	017	---	---	'C'Ø	---	Shift In
DLE	020	---	---	'C'P	---	Data Link Escape
DC1	021	---	---	'C'Q (X-ON)	---	Device Control 1
DC2	022	---	---	'C'R (TAPE)	---	Device Control 2
DC3	023	---	---	'C'S (X-OFF)	---	Device Control 3
DC4	024	---	---	'C'T (TAPE)	---	Device Control 4
NAK	025	---	---	'C'U	---	Negative Acknowledge
SYN	026	---	---	'C'V	---	Synchronous Idle
ETB	027	---	---	'C'W	---	End of Transmission Blocks
CAN	030	---	---	'C'X	---	Cancel
EM	031	---	---	'C'Y	---	End of Medium
SS	032	---	---	'C'Z	---	Special Sequence
ESC	033	---	---	'CS'K	---	Escape
FS	034	---	---	'CS'L	---	File Separator
GS	035	---	---	'CS'M	---	Group Separator
RS	036	---	---	'CS'N	---	Record Separator
US	037	---	---	'CS'Ø	---	Unit Separator
SP	040	blank	20	SPACE BAR	blank	Space
!	041	!	77	'S'1	0-7-8	Exclamation Point
"	042	"	76	'S'2	0-6-8	Quotation Mark
#	043	#	13	'S'3	3-8	Number Sign
$	044	$	53	'S'4	11-3-8	Currency Symbol
%	045	%	74	'S'5	0-4-8	Percent

ASCII CHAR	ASCII CODE	BCD CHAR	BCD CODE	MODEL 33/35 KEY	HOLLERITH CARD	MEANING
&	046	&	32	'S'6	12	Ampersand
'	047	'	57	'S'7	11-7-8	Apostrophe
(	050	(	35	'S'8	12-5-8	Opening Parenthesis
)	051	)	55	'S'9	11-5-8	Closing Parenthesis
*	052	*	54	'S':	11-4-8	Asterisk
+	053	+	60	'S';	12-0	Plus
,	054	,	73	,	0-3-8	Comma
-	055	-	52	-	11	Hyphen or Minus
/	057	/	61	/	0-1	Slant
0	060	0	00	0	0	Zero
1	061	1	01	1	1	One
2	062	2	02	2	2	Two
3	063	3	03	3	3	Three
4	064	4	04	4	4	Four
5	065	5	05	5	5	Five
6	066	6	06	6	6	Six
7	067	7	07	7	7	Seven
8	070	8	10	8	8	Eight
9	071	9	11	9	9	Nine
:	072	:	15	:	5-8	Colon
;	073	;	56	;	11-6-8	Semicolon
<	074	<	36	'S',	12-6-8	Less Than
=	075	=	75	'S'-	0-5-8	Equal
>	076	>	16	'S'.	6-8	Greater Than
?	077	?	17	'S'/	7-8	Question Mark
@	100	@	14	'S'P	4-8	Commercial At
A	101	A	21	A	12-1	Uppercase Letter
B	102	B	22	B	12-2	Uppercase Letter
C	103	C	23	C	12-3	Uppercase Letter
D	104	D	24	D	12-4	Uppercase Letter
E	105	E	25	E	12-5	Uppercase Letter
F	106	F	26	F	12-6	Uppercase Letter
G	107	G	27	G	12-7	Uppercase Letter
H	110	H	30	H	12-8	Uppercase Letter
I	111	I	31	I	12-9	Uppercase Letter
J	112	J	41	J	11-1	Uppercase Letter
K	113	K	42	K	11-2	Uppercase Letter
L	114	L	43	L	11-3	Uppercase Letter
M	115	M	44	M	11-4	Uppercase Letter
N	116	N	45	N	11-5	Uppercase Letter
O	117	Ø	46	Ø	11-6	Uppercase Letter
P	120	P	47	P	11-7	Uppercase Letter
Q	121	Q	50	Q	11-8	Uppercase Letter
R	122	R	51	R	11-9	Uppercase Letter
S	123	S	62	S	0-2	Uppercase Letter
T	124	T	63	T	0-3	Uppercase Letter
U	125	U	64	U	0-4	Uppercase Letter
V	126	V	65	V	0-5	Uppercase Letter
W	127	W	66	W	0-6	Uppercase Letter
X	130	X	67	X	0-7	Uppercase Letter
Y	131	Y	70	Y	0-8	Uppercase Letter
Z	132	Z	71	Z	0-9	Uppercase Letter

ASCII CHAR	ASCII CODE	BCD CHAR	BCD CODE	MODEL 33/35 KEY	HOLLERITH CARD	MEANING
[	133	[	12	'S'K	2-8	Opening Bracket
\	134	\	37	'S'L	12-6-8	Reverse Slant
]	135	]	34	'S'M	12-4-8	Closing Bracket
^	136	^	40	'S'N	11-0	Circumflex
_	137	←	72	'S'Ø	0-2-8	Underline
`	140	---	---	---	---	Grave Accent
a	141	---	---	---	---	Lowercase Letter
b	142	---	---	---	---	Lowercase Letter
c	143	---	---	---	---	Lowercase Letter
d	144	---	---	---	---	Lowercase Letter
e	145	---	---	---	---	Lowercase Letter
f	146	---	---	---	---	Lowercase Letter
g	147	---	---	---	---	Lowercase Letter
h	150	---	---	---	---	Lowercase Letter
i	151	---	---	---	---	Lowercase Letter
j	152	---	---	---	---	Lowercase Letter
k	153	---	---	---	---	Lowercase Letter
l	154	---	---	---	---	Lowercase Letter
m	155	---	---	---	---	Lowercase Letter
n	156	---	---	---	---	Lowercase Letter
o	157	---	---	---	---	Lowercase Letter
p	160	---	---	---	---	Lowercase Letter
q	161	---	---	---	---	Lowercase Letter
r	162	---	---	---	---	Lowercase Letter
s	163	---	---	---	---	Lowercase Letter
t	164	---	---	---	---	Lowercase Letter
u	165	---	---	---	---	Lowercase Letter
v	166	---	---	---	---	Lowercase Letter
w	167	---	---	---	---	Lowercase Letter
x	170	---	---	---	---	Lowercase Letter
y	171	---	---	---	---	Lowercase Letter
z	172	---	---	---	---	Lowercase Letter
{	173	---	---	---	---	Opening Brace
|	174	---	---	---	---	Vertical Line
}	175	---	---	---	---	Closing Brace
~	176	---	---	---	---	Tilde
DEL	177	---	---	RUBOUT	12-7-9	Delete

Legend:

'C' = CTRL key
'CS' = CTRL and SHIFT keys
'S' = SHIFT key

APPENDIX B

DIAGNOSTIC ERROR COMMENTS

The error detection capabilities of Series 6000 FORTRAN are extensive, including over 500 unique compiler diagnostics. Each diagnostic has zero, one, or two "plug-in" fields, as appropriate to the message. Diagnostics are generated in-line as part of the source listing report (LSTIN) wherever possible, following the line in error. If this report is being suppressed via the NLSTIN option, lines which have no errors will not be printed, but lines for which a diagnostic is being generated will be displayed.

The general form of a diagnostic line is as follows:

*****S (2) nnnn (1) text

where S is a severity code, nnnn is an errror identification code, and text is the diagnostic message. There are three severity codes as follows:

Code	Meaning
W	This is a warning message only.
F	This is a fatal diagnostic; any subsequent execution activity is deleted.
T	This is a terminal diagnostic; this compilation and any subsequent execution activity are deleted.

If only warning diagnostics are produced for a given compilation, correct object generation is assured, and execution may proceed.

The error identification code may be used to reference the tabulation of error codes given in this appendix, it being a number of from 1 to 4 digits.

A correspondence of error codes with the compiler phase detecting the error is as follows:

Error Number	Compiler Phase
1- 199	Executive
200- 299	Phase 2
300- 399	Phase 3
400- 499	Phase 4
1000-1499	Phase 1

In the subsequent tabulation of diagnostics, phase 1 errors are numbered 1 through 462. These correspond to error identification codes 1001 through 1462 respectively. Diagnostics pertinent to the other phases are shown by actual error identification code.

The text of the diagnostic messages may include "plug-in" information, to more fully detail the nature of the error. There are eleven types of plug-ins. In the subsequent tabulation, a text insertion is indicated by a number, one through eleven, enclosed in slants. The interpretation of that number is as follows:

/No./ in Manual listing	Actual value plugged in for error
1	Variable name, constant, or statement number
2	One of the operator codes from Table B-2
3	One of the statement classification types from Table B-3
4	Character
5	One of the type codes from Table B-4
6	Number
7	One of the statement types from Table B-1
8	One of the Lexical classifications from Table B-6
9	One of the program types from Table B-5
10	Four characters
11	Four or six characters

The following abbreviation is used in the listing:

ASF Arithmetic Statement Function

Table B-1. List of Statement Types for Code /7/

ABNORMAL	RETURN
EXTERNAL	BACKSPACE
CHARACTER	BLOCK DATA
DOUBLE PRECISION	CALL
COMPLEX	COMMON
INTEGER	CONTINUE
LOGICAL	DATA
REAL	DIMENSION
READ	DO
WRITE	END
PRINT	ENTRY
PUNCH	EQUIVALENCE
ENCODE	FUNCTION
DECODE	GO TO
PAUSE	FORMAT
STOP	IF
REWIND	IMPLICIT
ENDFILE	NAMELIST
ASSIGN	SUBROUTINE
PARAMETER	

Table B-2. List of Operator Codes for Code /2/

.OR.	*
.AND.	.EQ.
.NOT.	.NE.
+	.LT.
-	.GE.
/	.LE.
**	.GT.

Table B-3. List of Statement Classification Types Code /3/

block name
intrinsic function
array
scalar
arithmetic statement function
function
abnormal function
subroutine
external
parameter
entry name
namelist name
do loop index
adjustable dimension
bad insert

Table B-4. List of Types Code /5/

INTEGER
REAL
DOUBLE PRECISION
COMPLEX
CHARACTER
LOGICAL
TYPELESS

Table B-5. List of Program Types Code /9/

MAIN
SUBROUTINE
BLOCK DATA

Table B-6. List of Lexical Classifications Code /8/

VARIABLE	,
CONSTANT	END OF STATEMENT
OPERATOR	=
(	'
)	

PHASE1 ERROR MESSAGES

SECTION I - Abnormal Statement

1.	F	VARIABLE NAME IS MISSING BEFORE FIRST , IN ABNORMAL STATEMENT
2.	F	VARIABLE NAME IN /7/ IS MISSING, STARTS WITH DIGIT OR IS 8 CHARACTERS
3.	F	/8/ FOLLOWING , IN ABNORMAL STATEMENT IS ILLEGAL
4.	F	/8/ FOLLOWING /1/ IN ABNORMAL STATEMENT IS ILLEGAL
5.	F	/4/ FOLLOWING 'ABNORMAL' IS ILLEGAL

SECTION II - Assignment Statement

6.	F	UNEXPECTED /8/ ENCOUNTERED IN /7/ STATEMENT
7.	F	/5/ CANNOT BE ASSIGNED TO A LOGICAL VARIABLE
9.	F	LEFT HAND SIDE OF ASSIGNMENT STATEMENT HAS AN ERROR IN TYPING
10.	F	TYPE OF RIGHT HAND SIDE OF ASSIGNMENT IS INCOMPATIBLE WITH LEFT HAND SIDE
273.	W	TYPE OF RIGHT HAND SIDE OF ASSIGNMENT IS INCOMPATIBLE WITH LEFT HAND SIDE

SECTION III - Arithmetic Statement Function

11.	F	/1/ IS AN UNDIMENSIONED ARRAY OR MISPLACED ASF
12.	F	UNEXPECTED /4/ FOLLOWING BALANCING PARENTHESES
13.	F	ILLEGAL STATEMENT OR /1/ USED PREVIOUSLY IN ASF ARGUMENT LIST
14.	F	ILLEGAL STATEMENT OR /8/ USED ILLEGALLY IN ASF
15.	F	ILLEGAL STATEMENT OR RIGHT PAREN AFTER , IN ASF IS ILLEGAL
16.	F	ILLEGAL STATEMENT OR = IS MISSING IN /7/ STATEMENT
224.	F	ILLEGAL STATEMENT OR UNEXPECTED /8/ IN ASF

SECTION IV - Assigned GOTO Statement

17. F 'TO' IS MISSPELLED OR MISSING IN 'ASSIGN' STATEMENT
18. F STATEMENT NUMBER IS MISSING IN 'ASSIGN' STATEMENT
19. F LABEL IN ASSIGN STATEMENT IS NOT BETWEEN 1 AND 99999
20. F 'TO' AND SWITCH NAME ARE MISSING IN 'ASSIGN' STATEMENT
21. F SWITCH NAME IS MISSING IN 'ASSIGN' STATEMENT
22. F SWITCH NAME IN 'ASSIGN' STATEMENT CONTAINS MORE THAN 8 CHARACTERS

SECTION V - CALL Statement

23. F /8/ FOLLOWING ')' IN CALL IS ILLEGAL, EXPECTING END OF STATEMENT
24. F SUBROUTINE NAME IS MISSING IN CALL STATEMENT
25. F MORE THAN 8 CHARACTERS IN SUBROUTINE NAME IN CALL STATEMENT
26. F /1/ USED AS /3/ , INVALID AS A SUBROUTINE NAME

SECTION VI - DATA Statement

27. F FIRST VARIABLE NAME IS MISSING IN /7/ STATEMENT
28. W EXPECTING COMMA AFTER /
29. F /8/ FOLLOWING LEFT PAREN OR COMMA IN /7/ STATEMENT IS ILLEGAL
30. F NON-INTEGER CONSTANT IN ARRAY SUBSCRIPT
31. F IDENTIFIER /1/ ON DATA LIST HAS NOT BEEN DIMENSIONED
32. F /8/ IS ILLEGAL FOLLOWING LEFT PAREN IN DATA STATEMENT
33. F UNEXPECTED /8/ FOLLOWING /1/ IN DATA STATEMENT
34. F /8/ FOLLOWING * IN SUBSCRIPT OF DATA STATEMENT IS ILLEGAL
35. F /8/ FOLLOWING /1/ IN SUBSCRIPT OF DATA STATEMENT IS ILLEGAL
36. F UNEXPECTED /8/ FOLLOWING + IN SUBSCRIPT OF DATA STATEMENT
37. F NUMBER OF SUBSCRIPTS FOR /1/ DOES NOT MATCH NUMBER IN DECLARATION
38. F /8/ FOLLOWING RIGHT PAREN OF /1/ IN DATA STATEMENT IS ILLEGAL
39. F CONTROL INDEX /1/ IS OUTSIDE IMPLIED DO LOOP
40. F /8/ IS ILLEGAL FOLLOWING = IN DATA STATEMENT
41. F /8/ FOLLOWING FIRST , IN DATA STATEMENT IS ILLEGAL
42. F /8/ FOLLOWING SECOND , IN IMPLIED DO LOOP IS ILLEGAL
43. F UNEXPECTED /8/ FOLLOWING STEP IN IMPLIED DO LOOP IN DATA STATEMENT

	44.	F	UNEXPECTED /8/ FOLLOWING RIGHT PARENTHESIS TERMINATING IMPLIED DO LIST
	45.	F	UNEXPECTED /2/ FOLLOWING VARIABLE OR RIGHT PAREN ON DATA LIST
	46.	F	IMPLIED DO SPECIFICATION IS MISSING
*	47.	F	/8/ FOLLOWING / IN DATA LIST IS ILLEGAL STATEMENT
	48.	F	SUBSCRIPT IN DATA STATEMENT IS NOT OF FORM I*J+K OR I*J-K
	49.	F	SUBSCRIPTS AND IMPLIED DO PARAMETERS MUST BE INTEGERS
	445.	F	UNEXPECTED /8/ FOLLOWING /1/ IN DATA STATEMENT
	446.	F	UNEXPECTED /8/ FOLLOWING COMMA IN DATA VARIABLE LIST
	447.	F	UNEXPECTED /8/ FOLLOWING THE IMPLIED DO PARAMETER /1/
	448.	F	VARIABLE /1/ ON IMPLIED DO LIST IS NOT A DO CONTROL
	456.	F	ILLEGAL/8/ IN OR FOLLOWING DATA SUBSCRIPT EXPRESSION

SECTION VII - DO Statement

	50.	F	RECORD NUMBER IN /7/ MUST BE TYPE INTEGER
	51.	F	LABEL FOR DO-END IS NOT BETWEEN 1 AND 99999
	53.	F	EXECUTABLE STATEMENTS ARE ILLEGAL HERE
*	55.	F	INDEX VARIABLE FOR DO STATEMENT IS MISSING, STARTS WITH DIGIT OR > 8 CHARACTERS
	56.	F	UNEXPECTED /8/ FOLLOWING INITIAL PARAMETER OF /7/ STATEMENT
	57.	F	DO END STATEMENT NUMBER /1/ IS MISSING
	253.	F	UNEXPECTED /8/ AFTER INITIAL PARAMETER OF DO STATEMENT
	306.	F	DO LOOP INDEX /1/ MUST BE AN INTEGER
	307.	F	/3/ /1/ ILLEGAL AS CONTROL INDEX
	308.	F	NESTED DO LOOPS ARE USING /1/ AS THE SAME INDEX
	309.	F	ADJUSTABLE DIMENSION /1/ MAY NOT BE USED AS A DO LOOP INDEX

SECTION VIII - Entry, Function, Subroutine, Blockdata Statements

	58.	F	BLOCKDATA STATEMENT IS OUT OF PLACE
	59.	F	ENTRY STATEMENT IS ILLEGAL IN /9/
	60.	F	ENTRY MAY NOT BE DECLARED INSIDE DO LOOP

61. F ILLEGAL CONTINUATION LINE FOLLOWING RIGHT PAREN

62. F THE ARGUMENT /1/ APPEARS TWICE IN ENTRY LIST

63. F /2/ IS ILLEGAL IN ARGUMENT LIST FOR /7/ STATEMENT

64. F OPTIONAL RETURN * OR $ IS ILLEGAL IN /7/ STATEMENT

65. F UNEXPECTED /8/ IN DUMMY ARGUMENT LIST

66. F /8/ AFTER ')' IS ILLEGAL IN ARGUMENT LIST OF /7/ STATEMENT

68. F MISSING NAME IN /7/ STATEMENT

69. F MORE THAN 8 CHARACTERS IN NAME OF /7/

70. F FUNCTION DEFINITION FOR /1/ MUST BE FIRST STATEMENT OF SUBPROGRAM

71. F FUNCTION HAS NO NAME

72. F 'SUBROUTINE' STATEMENT MUST APPEAR FIRST IN SUBPROGRAM

454. F FUNCTION ENTRY MUST HAVE AN ARGUMENT LIST

SECTION IX - Format Statement

8. F UNEXPECTED /4/ FOLLOWING SIGNED NUMBER - EXPECTING A P

79. F FORMAT STATEMENT DOES NOT HAVE A STATEMENT LABEL

85. F ONLY TWO LEVELS OF NESTED PARENTHESES ALLOWED IN FORMAT STATEMENT

111. F X OR H FIELD IN FORMAT STATEMENT MUST HAVE A NUMERIC PREFIX

121. F 'T' SPECIFICATION CANNOT HAVE A NUMERIC PREFIX

161. F THE LETTER /4/ IS ILLEGAL IN A 'P' SPECIFICATION

164. F NO WIDTH FIELD IN FORMAT SPECIFICATION

170. F 'W' FIELD CANNOT BE ZERO IN FORMAT SPECIFICATION

180. F UNEXPECTED END OF STATEMENT IN /7/ SPECIFICATION

181. F TOO MANY CONTINUATION LINES IN FORMAT SPECIFICATION

185. F THE CHARACTER /4/ APPEARS ILLEGALLY IN A FORMAT SPECIFICATION

188. F COMPILER ERROR WHILE PROCESSING FORMAT STATEMENT

222. F THE CHARACTERS PRECEDING ABOVE FORMAT ERROR ARE /11/

257. F 'P' SPECIFICATION IN FORMAT MUST HAVE NUMERIC PREFIX

282. F INCOMPATIBLE W. D FIELD IN /4/ SPECIFICATION

SECTION X - Intrinsic Functions

73. F AN ARGUMENT OF /1/ IS /5/

74. F TOO MANY ARGUMENTS FOR /1/ IN /7/ STATEMENT

75. F TOO FEW ARGUMENTS FOR /1/ IN /7/ STATEMENT

240. F AN ARGUMENT OF /1/ IS NOT TYPELESS OR INTEGER

SECTION XI - CALL or FUNCTION Arguments

76. F /3/ /1/ ILLEGAL AS AN ARGUMENT OF FUNCTION OR CALL

SECTION XII - Equivalence Statement

77. F UNEXPECTED /8/ IN EQUIVALENCE GROUP - EXPECTING VARIABLE NAME

78. F /8/ IS ILLEGAL AFTER A VARIABLE NAME IN EQUIVALENCE STATEMENT

80. F SUBSCRIPTS FOR /1/ MUST BE POSITIVE INTEGERS OR INTEGER PARAMETERS

81. F DIMENSION CONSTANT IS TOO LARGE

82. F MORE THAN 7 DIMENSIONS ARE SPECIFIED FOR /1/

83. F /8/ IS ILLEGAL AFTER PARAMETER OR CONSTANT IN /7/ STATEMENT

84. F /8/ AFTER ')' FOLLOWING DIMENSIONS IN EQUIVALENCE STATEMENT IS ILLEGAL

86. F /8/ IS ILLEGAL AFTER ')' CLOSING GROUP IN EQUIVALENCE STATEMENT

87. F EQUIVALENCE GROUP MUST START WITH LEFT PAREN

310. W COMMA MUST PRECEDE START OF EQUIVALENCE GROUP FOLLOWING A RIGHT PAREN

SECTION XIII - External Statement

88. F FIRST VARIABLE NAME IN EXTERNAL STATEMENT IS MISSING

89. F VARIABLE NAME IN /7/ STATEMENT IS > 8 CHARACTERS

90. F /8/ FOLLOWING , IN EXTERNAL STATEMENT IS ILLEGAL

91. F /8/ FOLLOWING /1/ IN EXTERNAL STATEMENT IS ILLEGAL

92. F /8/ FOLLOWING RIGHT PAREN IN EXTERNAL STATEMENT IS ILLEGAL

SECTION XIV - File Designators

93. F ILLEGAL CHARACTER AFTER FILE REFERENCE CONSTANT IN /7/ STATEMENT

94. F FILE REFERENCE /1/ IN /7/ STATEMENT IS NOT AN INTEGER

95. F FILE REFERENCE IS MISSING IN /7/ STATEMENT

96. F FILE REFERENCE IN /7/ STATEMENT IS 8 CHARACTERS

SECTION XV - GOTO Statement

67.	F	UNEXPECTED /8/ IN GOTO LIST
97.	F	ILLEGAL CHARACTER IN STATEMENT NUMBER IN /7/ STATEMENT
98.	F	STATEMENT NUMBER OR SWITCH IN GOTO STATEMENT IS MISSING OR ILLEGAL
99.	F	ILLEGAL SWITCH VARIABLE OR LABEL IN GO TO STATEMENT
100.	F	INVALID SWITCH OR STATEMENT NUMBER IN COMPUTED GO TO
101.	F	/8/ FOLLOWING /1/ IN COMPUTED GO TO IS ILLEGAL
102.	F	/8/ FOLLOWING RIGHT PAREN IN /7/ STATEMENT IS ILLEGAL
103.	F	/8/ IS ILLEGAL AS TERMINATOR FOR COMPUTED GO TO STATEMENT
104.	F	COMPUTED GO TO EXPRESSION MUST BE TYPE INTEGER
105.	F	COMPILER ERROR IN CONTROL TEST
106.	F	SWITCH /1/ IN /7/ STATEMENT IS NOT TYPE INTEGER
107.	F	/8/ FOLLOWING /1/ IN COMPUTED OR ASSIGNED GO TO IS ILLEGAL
108.	F	/8/ FOLLOWING FIRST ',' IN ASSIGNED GOTO IS ILLEGAL, EXPECTING '('
109.	F	/8/ FOLLOWING STATEMENT NUMBER IN ASSIGNED GO TO IS ILLEGAL
110.	F	STATEMENT NUMBER /1/ IN /7/ STATEMENT IS NOT INTEGER
112.	F	LABEL IN GOTO STATEMENT IS NOT BETWEEN 1 AND 99999
450.	F	GO TO LIST HAS NON INTEGER LABEL

SECTION XVI - Arithmetic IF Statement

113.	F	MISSING RIGHT PAREN IN EXPRESSION SECTION OF /7/ STATEMENT
114.	F	/5/ IS ILLEGAL FOR THE EXPRESSION SECTION OF /7/ STATEMENT
115.	F	ONLY STATEMENT NUMBERS OR SWITCHES ARE LEGAL AFTER EXPRESSION SECTION OF IF
116.	F	ILLEGAL CONSTRUCT OF IF STATEMENT
117.	F	/8/ IS ILLEGAL AFTER A STATEMENT NUMBER OR SWITCH IN ARITHMETIC IF
118.	F	/8/ IS ILLEGAL AFTER THIRD STATEMENT NUMBER OR SWITCH IN ARITHMETIC IF
119.	F	TOO MANY STATEMENT NUMBERS, SWITCHES OR FIELDS IN AN ARITHMETIC IF STATEMENT
120.	F	MISSING A STATEMENT NUMBER, SWITCH OR FIELD IN AN ARITHMETIC IF STATEMENT

SECTION XVII - Logical IF Statement

52. F THE TRUTH CLAUSE OF A LOGICAL IF MAY NOT BE /7/ STATEMENT

122. F MISSING RIGHT PAREN IN THE EXPRESSION SECTION OF /7/ STATEMENT

123. F MISSING VARIABLE NAME BEFORE = IN THE TRUTH CLAUSE OF LOGICAL IF STATEMENT

443. F TRUTH CLAUSE OF A LOGICAL IF CANNOT BE NULL

SECTION XVIII - Implicit Statement

124. F /8/ FOLLOWING TYPE DESIGNATION IN IMPLICIT STATEMENT IS ILLEGAL

125. F /8/ AFTER * IN /7/ STATEMENT IS ILLEGAL

126. F /8/ AFTER SIZE OPTION IN /7/ STATEMENT IS ILLEGAL

127. F UNEXPECTED /8/ ENCOUNTERED IN /7/ STATEMENT

128. F DELIMITER FOLLOWING 'FROM' ENTRY IN IMPLICIT STATEMENT IS ILLEGAL

129. F /8/ FOLLOWING RIGHT PAREN IN /7/ STATEMENT IS ILLEGAL

130. F /8/ IS ILLEGAL, EXPECTING TYPE FOR IMPLICIT STATEMENT

131. F TYPE SPECIFICATION IN IMPLICIT STATEMENT IS MISSING OR MISSPELLED

132. F NON-ALPHABETIC CHARACTER FOUND IN IMPLICIT STATEMENT - EXPECTING LETTER

133. F CHARACTER IN IMPLICIT LIST STARTING WITH /10/ ILLEGAL - EXPECTING LETTER

SECTION XIX - PARAMETER Statement

135. F = IS MISSING IN /7/ STATEMENT

136. F LEFT HAND SIDE OF = IN PARAMETER STATEMENT MUST BE A VARIABLE NAME

137. F FIRST VARIABLE NAME IN PARAMETER STATEMENT IS MISSING

452. F RIGHT OF EQUALS IN PARAMETER STATEMENT IS NOT A CONSTANT

SECTION XX - Return Statement

139. F RETURN STATEMENT IS ILLEGAL IN /9/

140. F ILLEGAL CHARACTER AFTER CONSTANT IN RETURN STATEMENT

142. F VARIABLE NAME IS 8 CHARACTERS OR CONSTANT IS TOO LARGE IN /7/ STATEMENT

143. F ILLEGAL CHARACTER AFTER CONSTANT IN /7/ STATEMENT

SECTION XXI - Specification Statements - Common, Dimension

134. F CAN ONLY HAVE ARRAY /1/ WITH ADJUSTABLE DIMENSIONS IN SUBPROGRAM

138. F THE ADJUSTABLE DIMENSION /1/ MUST BE TYPE INTEGER

141. F CONSTANT IN ARRAY SUBSCRIPT MUST BE INTEGER

145. F CONSTANT FOR ARRAY DIMENSION IS NOT BETWEEN 1 AND 131071

146. F USE OF * FOR SIZE OPTION IS ILLEGAL IN /7/ STATEMENT

147. F /4/ IS ILLEGAL IN /7/ STATEMENT

148. F /8/ IS ILLEGAL AFTER SIZE OPTION IN /7/ STATEMENT

149. F THE /3/ /1/ CANNOT BE USED AS AN ADJUSTABLE DIMENSION

150. F MORE THAN 7 DIMENSIONS ARE SPECIFIED FOR /1/ IN /7/ STATEMENT

151. F /8/ IS ILLEGAL AFTER / IN /7/ STATEMENT

152. F /2/ IS ILLEGAL AFTER SPECIFYING DIMENSIONS FOR /1/

153. F /8/ IS ILLEGAL AFTER '/' IN /7/ STATEMENT

154. F FIRST VARIABLE NAME IN /7/ IS MISSING OR STARTS WITH A DIGIT

155. F /4/ IS ILLEGAL IN /7/ STATEMENT

156. F A DELIMITER IS MISSING BEFORE /1/ IN /7/ STATEMENT

157. F A DELIMITER IS MISSING OR /8/ IS ILLEGALLY USED IN /7/ STATEMENT

158. F /8/ IS ILLEGAL AFTER VARIABLE NAME IN /7/ STATEMENT

159. F /8/ IS ILLEGAL AFTER * IN /7/ STATEMENT

160. F /7/ STATEMENT INCOMPLETE

162. F A DIMENSION IS MISSING IN /7/ STATEMENT *

163. F UNEXPECTED /8/ ON DIMENSION LIST OF /7/ STATEMENT *

165. F TWO DELIMITERS IN A ROW IS ILLEGAL

166. F /8/ IS ILLEGAL AFTER DATA GENERATION OPTION IN /7/

167. F /8/ IS ILLEGAL AFTER A ',' , '/' OR SIZE OPTION

168. F NOT USED

169. F SIZE OPTION /1/ IS NOT A POSITIVE INTEGER CONSTANT

171. F INDEX INTO ARRAY /1/ IS > 131071 *

177. F DIMENSIONS ARE MISSING FOR /1/

192. F /3/ CANNOT BE TYPED

SECTION XXII - Expressions

172. F REAL PART OF COMPLEX EXPRESSION TOO LARGE

173. F REAL PART OF COMPLEX EXPRESSION TOO SMALL

174. F IMAGINARY PART OF COMPLEX EXPRESSION TOO LARGE

175. F IMAGINARY PART OF COMPLEX EXPRESSION TOO SMALL

176. F ILLEGAL COMBINATION OF TYPES IN RELATIONAL EXPRESSION

*

SECTION XXIII - Array Subscripts

178. F SUBSCRIPTS FOR /1/ MUST BE TYPE INTEGER

179. F SUBSCRIPTS FOR /1/ DO NOT MATCH DIMENSION SPECIFICATION

*

SECTION XXIV - Subscripted Identifier Use

*

182. F /1/ IS AN UNDIMENSIONED ARRAY OR AN INVALID FUNCTION

183. F SUBSCRIPTED ARRAY /1/ ILLEGAL AS A SUBSCRIPT IN /7/ STATEMENT

184. F FUNCTION /1/ ILLEGAL AS A SUBSCRIPT IN /7/ STATEMENT

*

186. F THE FUNCTION /1/ IS NOT ALLOWED IN DATA LISTS

187. F NUMBER OR ARGUMENTS IN /1/ ASF DOES NOT MATCH NUMBER IN DEFINITION

312. F ASF CALL HAS MORE THAN 100 ARGUMENTS

460. F ARRAY /1/ CANNOT BE USED AS A SCALAR WHEN A DO PARAMETER

461. F /1/ HAS PREVIOUSLY BEEN DIMENSIONED

SECTION XXV - Scalar Use

*

189. F /1/ MUST BE TYPED INTEGER BECAUSE IT IS USED AS A SUBSCRIPT IN /7/ STATEMENT

190. F THE IDENTIFIER /1/ MUST BE A PREVIOUSLY DEFINED PARAMETER SYMBOL

191. F THE VARIABLE /1/ HAS BEEN PREVIOUSLY USED AS /3/

*

SECTION XXVI - Expression Syntax

193. W ILLEGAL CHARACTER PRECEDING QUOTE IN /7/ STATEMENT

194. F = IS ILLEGAL IN ANY EXPRESSION

195. F /7/ STATEMENT INCOMPLETE

196. F UNEXPECTED /8/ ENCOUNTERED WHILE PROCESSING EXPRESSION

197. F UNEXPECTED /8/ IN EXPRESSION FOLLOWING RELATIONAL OPERATOR

198. F MISSING RIGHT PARENTHESIS OR COMMA

199. F UNEXPECTED /8/ FOLLOWING IMAGINARY COMPONENT OF COMPLEX CONSTANT

200. F UNEXPECTED /8/ IN SUBSCRIPT EXPRESSION

201. F UNEXPECTED /8/ AT BEGINNING OF FIRST SUBSCRIPT EXPRESSION

202. F UNEXPECTED /8/ -EXPECTING RIGHT PARENTHESIS OR COMMA

203. F UNEXPECTED /8/ IN ASF ARGUMENT-EXPECTING RIGHT PAREN OR COMMA

204. F UNEXPECTED /8/ WHILE PROCESSING ARGUMENTS OF CALL STATEMENT

205. F UNEXPECTED /8/ WHILE PROCESSING CALL STATEMENT-EXPECTING ')' OR ','

206. F UNEXPECTED /8/ WHILE PROCESSING DATA CONSTANT LIST

207. F UNEXPECTED /8/ WHILE PROCESSING THE SECOND OR THIRD INDEX PARAMETER

208. F UNEXPECTED /8/ WHILE PROCESSING FIRST INDEX PARAMETER OF DO-EXPECTING ,

209. F UNEXPECTED /8/ WHILE PROCESSING LEFT HAND SIDE OF ASSIGNMENT STATEMENT-EXPECTING =

210. F /8/ ILLEGALLY USED IN /7/ STATEMENT EXPRESSION

211. F THE RELATIONAL OPERATOR /2/ IS ILLEGAL IN A RELATIONAL EXPRESSION

212. F ASF /1/ EXPANDS INCORRECTLY

341. F UNEXPECTED /8/ NEAR START OF I/O LIST

342. F UNEXPECTED /8/ IN ARRAY SUBSCRIPT-EXPECTING RIGHT PAREN OR COMMA

SECTION XXVII - Unary Operators

213. F /5/ 'S MAY NOT BE USED WITH .NOT. OPERATOR

214. F .NOT. MAY NOT BE USED WITH ARITHMETIC EXPRESSIONS

215. F TYPE /5/ MAY NOT BE USED WITH A UNARY + OPERATOR

216. F UNARY + MAY ONLY BE USED IN ARITHMETIC EXPRESSIONS

217. F UNEXPECTED /8/ TERMINATING CHARACTER STRING IN /3/ STATEMENT

218. F /5/ VARIABLE NAMES MAY NOT BE USED WITH UNARY -

SECTION XXVIII - Expression Semantics

219. F ILLEGAL COMBINATION OF LOGICAL, CHARACTER OR TYPELESS ENTITY IN EXPRESSION

220. F LOGICAL OPERATORS MUST BE USED WITH LOGICAL EXPRESSIONS OR VARIABLE NAMES

221. F CHARACTER, LOGICAL OR TYPELESS ILLEGAL IN EXPONENTIAL EXPRESSIONS

*

223. F ILLEGAL COMBINATION OF TYPES IN EXPONENTIAL EXPRESSION

*

SECTION XXIX - Constant Operations

225. F REAL CONSTANT IS TOO SMALL IN /7/ STATEMENT, ZERO ASSUMED

226. F REAL CONSTANT IS TOO LARGE IN /7/ STATEMENT

227. F NEGATIVE INTEGER CONSTANT IS TOO LARGE

228. F NEGATIVE REAL CONSTANT IS TOO LARGE

229. F NEGATIVE REAL CONSTANT IS TOO SMALL, ZERO ASSUMED

230. F NEGATIVE REAL PART OF A COMPLEX CONSTANT IS TOO LARGE

231. F NEGATIVE IMAGINARY PART OF A COMPLEX CONSTANT IS TOO LARGE

232. F NEGATIVE REAL PART OF A COMPLEX CONSTANT IS TOO SMALL, ZERO ASSUMED

233. F NEGATIVE IMAGINARY PART OF A COMPLEX CONSTANT IS TOO SMALL, ZERO ASSUMED

234. F COMPLEX CONSTANTS ARE ILLEGAL IN RELATIONAL EXPRESSIONS

235. F VALUE OF ARITHMETIC EXPRESSION WITH INTEGER CONSTANTS IS TOO LARGE

236. F ARITHMETIC EXPRESSION WITH REAL CONSTANTS IS TOO LARGE

237. F ARITHMETIC EXPRESSION WITH REAL CONSTANTS IS TOO SMALL, ZERO ASSUMED

238. F ARITHMETIC EXPRESSION WITH DOUBLE PRECISION CONSTANTS IS TOO LARGE

239. F ARITHMETIC EXPRESSION WITH DOUBLE PRECISION CONSTANTS IS TOO SMALL, ZERO ASSUMED

*

241. F 0**0 IS ILLEGAL

242. F 0**-J IS ILLEGAL

444. F INTEGER CONSTANT IS TOO LARGE - LARGEST INTEGER VALUE ASSUMED

449. F CHARACTER CONSTANT MAY NOT BE GREATER THAN 511 CHARACTERS

SECTION XXX - Constant List Processor

243. F /2/ IS ILLEGAL FOLLOWING / OR , IN DATA LIST FOR /7/ STATEMENT

244. F /8/ FOLLOWING / OR , IN DATA LIST FOR /7/ STATEMENT IS ILLEGAL

245. F /8/ FOLLOWING VARIABLE NAME OR CONSTANT IN DATA LIST IS ILLEGAL

246. F /8/ FOLLOWING VARIABLE NAME OR CONSTANT IN DATA LIST IS ILLEGAL

247. F REPEAT FACTOR IN /7/ STATEMENT MUST BE A POSITIVE INTEGER

248. F /2/ FOLLOWING * IN DATA LIST FOR /7/ STATEMENT IS ILLEGAL

249. F /8/ AFTER * IN DATA LIST FOR /7/ STATEMENT IS ILLEGAL

*

251. F THE IDENTIFIER /1/ IN DATA CONSTANT LIST IS NOT A CONSTANT

SECTION XXXI - Data List Constant

252. F COMPILER ERROR IN PROCESSING LABEL OR VARIABLE
254. F COMPLILER ERROR IN HANDLING ARGLIST
255. F MAY NOT REDEFINE /3/ /1/
256. F COMPILER ERROR IN PROCESSING ERROR
258. F COMPILER ERROR

SECTION XXXII - Initial Statement Analysis

259. F STATEMENT INCOMPLETE
260. F = FOLLOWING LEFT PARENTHESIS IS ILLEGAL
261. F DELIMITER MISSING
262. F CONSTANT FOLLOWED BY LEFT PAREN IS ILLEGAL
263. F UNEXPECTED /8/ FOLLOWING OPERATOR
264. F PARENTHESES DO NOT BALANCE IN AN ASF DEFINITION
265. F /8/ FOLLOWING RIGHT PAREN IS ILLEGAL
266. F ILLEGAL , IN ASF DEFINITION
267. F END OF STATEMENT OR = FOLLOWING , IS ILLEGAL
268. F PARENTHESES DO NOT BALANCE
269. F = OR ' IS USED ILLEGALLY IN STATEMENT

SECTION XXXIII - Identifier, Constant, and Label Formation

270. F VARIABLE IN STATEMENT HAS >8 CHARACTERS
271. F ILLEGAL LETTER FOLLOWING DIGIT, EXPECTING 'D' OR 'E'
272. F MORE THAT 2 DIGITS IN THE EXPONENT IN /7/ STATEMENT
274. F LOGICAL OR RELATIONAL OPERATOR IS INCOMPLETE IN /7/ STATEMENT
275. F SOMETHING IS MISSING AFTER PERIOD IN /7/ STATEMENT
276. F LOGICAL CONSTANT IS INCOMPLETE IN /7/ STATEMENT
277. F CHARACTER CONSTANT IS INCOMPLETE IN /7/ STATEMENT
278. F EXPONENT INCOMPLETE IN /7/ STATEMENT
279. F NON-FORTRAN CHARACTER ILLEGAL IN STATEMENT

280. F $ IS NOT FOLLOWED BY A STATEMENT NUMBER OR SWITCH IN /7/ STATEMENT

281. F /4/ FOLLOWING $ IS ILLEGAL IN /7/ STATEMENT

*

283. F UNEXPECTED /4/ WHILE PROCESSING IDENTIFIER

284. F ILLEGAL RELATIONAL OR LOGICAL OPERATOR /10/ AFTER PERIOD

285. F /4/ IS ILLEGAL IN RELATIONAL OR LOGICAL OPERATOR OR LOGICAL CONSTANT

*

287. F /4/ IS ILLEGAL AFTER A LOGICAL OR CHARACTER CONSTANT IN /7/ STATEMENT

288. F /4/ IS ILLEGAL IN AN ARITHMETIC CONSTANT IN /7/ STATEMENT

289. F /4/ IS ILLEGAL IN A SWITCH NAME IN /7/ STATEMENT

290. F /4/ IS ILLEGAL IN STATEMENT NUMBER IN /7/ STATEMENT

291. F /2/ IS NOT A RELATIONAL OPERATOR IN /7/ STATEMENT

292. F RECEIVED A /8/ FOLLOWING BLOCKNAME INSTEAD OF A'/'

293. F /1/ IS USED AS A SWITCH IN /7/ STATEMENT AND IS NOT TYPED INTEGER

294. F /1/ MUST BE A SCALAR VARIABLE TO BE A SWITCH

295. F TRUNCATION IS REQUIRED FOR ARITHMETIC CONSTANT IN /7/ STATEMENT

296. F = IS ILLEGAL IN /7/ STATEMENT

297. F STATEMENT NUMBER MUST BE BETWEEN 1 AND 99999

313. F NON-FORTRAN CHARACTER ENCOUNTERED

317. F FIRST LINE NUMBER MAY NOT BE FOLLOWED BY AN &

SECTION XXXIV - General Statement Formation

298. F /4/ IS ILLEGAL AS THE FIRST CHARACTER OF A STATEMENT

299. F NON-FORTRAN CHARACTER IS ILLEGAL AS FIRST LETTER OF STATEMENT

300. F STATEMENT UNRECOGNIZABLE OR FIRST VARIABLE IS > 8 CHARACTERS

301. F NON-FORTRAN CHARACTER ENCOUNTERED NEAR START OF STATEMENT

302. F . OR ' IS USED ILLEGALLY NEAR START OF STATEMENT

303. F FIRST WORD OF THE STATEMENT IS UNRECOGNIZABLE OR THE FIRST DELIMITER IS ILLEGAL

304. F FIRST DELIMITER OF /7/ STATEMENT IS ILLEGAL

305. F FIRST WORD OF STATEMENT IS FOLLOWED BY AN ILLEGAL DELIMITER

*

SECTION XXXV - Statement Legality Checks

314. F /7/ STATEMENT IS ILLEGAL AS A 'DO' END

315. F EXECUTABLE STATEMENTS ARE ILLEGAL IN BLOCKDATA SUBROUTINES

316. F THE TRUTH CLAUSE OF A LOGICAL IF MUST BE EXECUTABLE *

SECTION XXXVI - Statement Labels

319. F STATEMENT NUMBER /1/ HAS PREVIOUSLY BEEN DEFINED

320. F /1/ HAS BEEN REFERENCED AS A FORMAT LABEL

321. F THE STATEMENT WITH STATEMENT LABEL /1/ IS ILLEGAL AS THE END OF A DO LOOP

322. F ILLEGAL REFERENCE TO NON-EXECUTABLE STATEMENT AT STATEMENT NUMBER /1/

323. F STATEMENT NUMBER /1/ PREVIOUSLY REFERENCED AS A NON-EXECUTABLE STATEMENT

336. F /1/ PREVIOUSLY USED AS A REFERENCE TO A FORMAT STATEMENT

SECTION XXXVII - Identifier Semantics

324. F /1/ USED AS A DUMMY ARGUMENT IN AN ENTRY SUBROUTINE OR FUNCTION STATEMENT

325. F /1/ HAS BEEN USED AS AN ABNORMAL FUNCTION OR HAS APPEARED IN A COMMON STATEMENT

326. F /1/ HAS PREVIOUSLY APPEARED IN AN EQUIVALENCE STATEMENT *

327. F /1/ HAS PREVIOUSLY BEEN USED AS A VARIABLE NAME, ASF NAME OR BLOCKNAME

328. F /1/ HAS PREVIOUSLY BEEN REFERENCED AS EXTERNAL TO THIS PROGRAM

329. F /1/ HAS PREVIOUSLY BEEN USED IN A CONFLICTING WAY

330. F /1/ HAS PREVIOUSLY BEEN USED AS A SCALAR VARIABLE

331. F /1/ HAS PREVIOUSLY BEEN USED AS THE NAME OF THIS FUNCTION

332. F /1/ HAS PREVIOUSLY BEEN USED AS THE NAME OF AN ASF

333. F /1/ HAS PREVIOUSLY BEEN DIMENSIONED

334. F /1/ INCOMPATIBLE WITH BEING ADJUSTABLY DIMENSIONED OR USED AS SUBROUTINE, ENTRY OR FUNCTION ARGUMENT

335. F /1/ HAS PREVIOUSLY APPEARED IN A TYPE STATEMENT

337. F /1/ HAS PREVIOUSLY BEEN USED AS A SUBROUTINE NAME IN A CALL STATEMENT *

338. F /1/ PREVIOUSLY IN AN ABNORMAL STATEMENT OR IS USED AS AN ABNORMAL FUNCTION

339. F /1/ HAS PREVIOUSLY BEEN USED AS A NORMAL FUNCTION

457. W /3/ /1/ MAY NOT BE REDEFINED IN CALL OR ABNORMAL FUNCTION

458. F MAY NOT REDEFINE /3/ /1/

459. F MAY NOT REDEFINE /3/ /1/ BY USE AS A BUFFER TERM

SECTION XXXVIII - Miscellaneous

144. F VARIABLE NAME AFTER /7/ MUST BE TYPE INTEGER OR CHARACTER

250. W 'END' STATEMENT MISSING - SIMULATED

286. W 'STOP' STATEMENT MISSING - SIMULATED

318. F FILE REFERENCE IN /7/ STATEMENT IS NOT AN INTEGER

340. F STATEMENT NUMBER /1/ HAS BEEN DEFINED AS A FORMAT LABEL

343. F UNEXPECTED /8/ WHILE PROCESSING I/O LIST-EXPECTING RIGHT PAREN OR COMMA

344. F PROGRAM ILLEGALLY STARTS WITH CONTINUATION CARD OR NFORM NOT SPECIFIED

345. F THE DO-END STATEMENT NUMBER /1/ IS OUT OF PLACE

346. F /1/ USED OR REFERENCED AS THE LABEL OF A FORMAT STATEMENT

347. W STATEMENT NUMBER /1/ REFERENCED AS AN EXECUTABLE STATEMENT

348. F SINGLE OR DOUBLE QUOTE MISPLACED IN 'H' FIELD

349. F CHARACTERS IN 'H' FIELD EXCEED COUNT

350. F TOO MANY DIGITS IN OCTAL CONSTANT

351. F ILLEGAL CHARACTER IN A DATA CONSTANT

352. F ILLEGAL CHARACTER IN OCTAL CONSTANT

353. F UNEXPECTED END OF STATEMENT WHILE PROCESSING /7/ STATEMENT

354. F COMPILER ERROR-UNUSUAL CONSTRUCT ON I/O LIST

355. F I/O LIST ELEMENT HAS REDUNDANT PARENTHESIS OR MISPLACED IMPLIED DO LIST

356. F = FOLLOWING IDENTIFIER IS ILLEGAL

357. F UNEXPECTED /2/ AT START OF I/O LIST ELEMENT

358. F UNEXPECTED /8/ TERMINATING /7/ STATEMENT

359. F TYPE OF COMPONENT OF COMPLEX CONSTANT IS NOT INTEGER, REAL, DOUBLE-PRECISION

360. F THE REAL AND IMAGINARY PARTS OF A COMPLEX CONSTANT MUST BE CONSTANTS

361. F COMPILER ERROR-IMPROPER PROCESSING OF LOGICAL CONSTANT

362. F COMPILER ERROR-IMPROPER PROCESSING OF CONSTANTS

363. F ILLEGAL COMPLEX OPERATION

364. F /1/ HAS PREVIOUSLY BEEN USED IN A CONFLICTING WAY

365. F UNEXPECTED /4/ WHEN EXPECTING OPTION IN /7/ STATEMENT

366. F OPTION BEGINNING WITH /10/ ILLEGAL OR MISSPELLED

367. F ILLEGAL CONSTRUCT IN ASSIGN STATEMENT

368. F ILLEGAL /8/ IN OPTION FIELD OF /7/ STATEMENT

369. F UNEXPECTED /8/ FOLLOWING FIRST PARAMETER OF /7/ STATEMENT

370. F UNEXPECTED /8/ IN /7/ LIST

371. F FORMAT LABEL IS NON INTEGER OR NOT BETWEEN 1 AND 9999

372. F STATEMENT NUMBER IS TOO LARGE IN /7/ STATEMENT

373. F /8/ IS ILLEGAL AS AN OPTION IN /7/ STATEMENT

374. F THERE ARE TWO ERR OPTIONS IN /7/ STATEMENT

375. F THERE ARE TWO END OPTIONS IN /7/ STATEMENT

376. F UNRECOGNIZABLE OPTION IN /7/ STATEMENT

377. F AN I/O LIST SHOULD NOT BE GIVEN WHEN A NAMELIST VARIABLE IS USED

378. F CANNOT PERFORM I/O-ROUTINE NOT AVAILABLE IN LIBRARY

379. F THE PARAMETER SYMBOL /1/ CANNOT BE TYPED

380. F FIRST PARAMETER OF /7/ MUST BE A VARIABLE NAME TYPED CHARACTER

381. F /11/ IS NOT TYPE CHARACTER

382. F /3/ /1/ IS ILLEGAL AS A BUFFER TERM

383. F /1/ IS NOT A LEGAL BUFFER VARIABLE

384. F THE /3/ /1/ IMPROPERLY USED AS A VARIABLE FORMAT LABEL OR NAMELIST NAME

385. F VARIABLE FORMAT LABEL /1/ NOT DIMENSIONED

386. F /1/ MUST BE A NAMELIST NAME OR A VARIABLE FORMAT LABEL

387. F NAMELIST NAME MISSING IN /7/ STATEMENT

388. F FIRST VARIABLE NAME IS NOT A LEGAL VARIABLE NAME

389. F THE FORMAT STATEMENT NUMBER IN /7/ STATEMENT HAS ILLEGAL CHARACTER

390. F INPUT LIST ITEM MAY NOT BE A CONSTANT, EXPRESSION OR UNDIMENSIONED ARRAY

391. F EXPRESSION OR CONSTANT ILLEGAL IN /7/ LIST

392. F THE /3/ /1/ IS ILLEGAL ON I/O LIST

393. F /1/ MUST BE DIMENSIONED IN ORDER TO APPEAR ON I/O LIST

394. F UNEXPECTED /8/ IN /7/ STATEMENT, EXPECTING A '/'

395. F UNEXPECTED /8/ AFTER '/' IN /7/ STATEMENT

396. F UNEXPECTED /8/ AFTER NAMELIST NAME-EXPECTING A '/'

397. F UNEXPECTED /8/ AFTER /1/ IN NAMELIST STATEMENT

398. F UNEXPECTED /8/ AFTER COMMA IN /7/ STATEMENT

399. F UNEXPECTED /2/ FOLLOWING DECLARED VARIABLE NAME IN /1/ STATEMENT

400. F LENGTH OF ARRAY /1/ IS > 131071

401. F = IS USED ILLEGALLY IN /7/ STATEMENT

402. F ' IS USED ILLEGALLY IN /7/ STATEMENT

403. F PREVIOUS TYPING OF PARAMETER SYMBOL /1/ WILL BE IGNORED

404. F DO END STATEMENT NUMBER /1/ REFERENCES A NON-EXECUTABLE STATEMENT

405. F UNEXPECTED /8/ WHILE PROCESSING I/O LIST

406. F /1/ PREVIOUSLY USED AS THE STATEMENT NUMBER OF AN EXECUTABLE STATEMENT

407. F ARGUMENT OF /1/ IS NOT INTEGER, REAL, LOGICAL, TYPELESS OR CHARACTER

408. F MAY NOT REDEFINE INDEX VARIABLE /1/

409. F MAY NOT REDEFINE ADJUSTABLE DIMENSION /1/

410. F /1/ MUST BE TYPE INTEGER

411. F /3/ /1/ CANNOT BE TYPED

412. F ARRAY SUBSCRIPT REPRESENTED BY PARAMETER SYMBOL /1/ IS NOT INTEGER

413. F COMPILER ERROR - TWO CONSECUTIVE IDENTIFIERS IN /7/

414. F /8/ IS ILLEGAL AFTER A BLOCKNAME OR BLOCKNAME IS MISSING

415. F UNEXPECTED /8/ AFTER RIGHT PARENTHESIS IN /7/ STATEMENT

416. F CONSTANT FOR SIZE OPTION IS NOT AN INTEGER

417. W /1/ PREVIOUSLY TYPED IN A CONFLICTING WAY - NEW TYPE IGNORED

418. F ARRAY /1/ HAS LOGICAL OR CHARACTER SUBSCRIPT

419. F A DIVIDE CHECK FAULT OCCURRED WHILE PROCESSING CONSTANT EXPRESSION

420. W ARRAY /1/ IS BEING USED AS A SCALAR

421. F /1/ USED AS AN ADJUSTABLE DIMENSION OR ADJUSTABLY DIMENSIONED ARRAY

422. F /1/ HAS PREVIOUSLY BEEN USED TO REFERENCE AN EXECUTABLE STATEMENT

423. F THE FILE REFERENCE IN /1/ IS NOT AN INTEGER

424. F THE TRUTH CLAUSE OF THE LOGICAL IF ILLEGAL STARTS WITH THE CHARACTER /4/

425. F /1/ PREVIOUSLY USED AS A STATEMENT NUMBER OF A NON-EXECUTABLE STATEMENT

426. F /2/ USED ILLEGALLY IN EXPRESSION

427. F .NOT. OPERATOR USED ILLEGALLY IN EXPRESSION

428. W UNEXPECTED /8/ AFTER RIGHT PARENTHESIS IN /7/ STATEMENT

429. F DO END STATEMENT LABEL /1/ IS MISPLACED

430.	F	/1/ HAS PREVIOUSLY BEEN DEFINED
431.	F	ONLY ONE VARIABLE NAME IN EQUIVALENCE GROUP
432.	W	RETURN STATEMENT MISSING-SIMULATED
433.	F	/1/ HAS PREVIOUSLY BEEN DEFINED, POSSIBLY BY USE IN COMMON OR EQUIVALENCE
434.	F	SWITCH NAME IN GO TO STATEMENT IS ILLEGAL
435.	W	'X' FIELD IN FORMAT MUST HAVE A NUMERIC PREFIX
436.	W	THE LOGICAL IF STATEMENT DOES NOT HAVE AN ASSOCIATED TRUTH CLAUSE
437.	F	ILLEGAL COMBINATION WITH CHARACTER TYPE IN RELATIONAL EXPRESSION
438.	F	UNEXPECTED /8/ ENCOUNTERED IN /7/ STATEMENT
439.	F	THE 'H' FIELD COUNT OF A LITERAL MAY NOT BE ZERO
440.	F	FILE NUMBER MUST BE TYPE INTEGER IN /7/ STATEMENT
441.	F	RECEIVED CHARACTER OR LOGICAL TYPE IN EXPRESSION
442.	F	CONSTANT FOR DO PARAMETER IS NOT BETWEEN 1 AND 262143
451.	F	THE VARIABLE FORMAT /1/ IS NOT TYPE CHARACTER
453.	F	VARIABLE NAME OR PARAMETER CONSTANT MUST BE A POSITIVE INTEGER
455.		UNUSED
462.	F	UNEXPECTED /8/ FOLLOWING RIGHT PARENTHESIS IN I/O LIST ELEMENT
463.	W	THE INITIAL DO PARAMETER /1/ IS ALSO THE CURRENT DO INDEX
464.	F	THE DO TERMINAL PARAMETER OR STEP /1/ IS ALSO THE CURRENT DO INDEX
465.	W	SIZE OPTION FOR THE CHARACTER VARIABLE /1/ EXCEEDS 511, 511 ASSUMED
466.	W	THE CHARACTER '&' APPEARS ILLEGALLY, IT HAS BEEN IGNORED
467.	W	SIZE OPTION IN /7/ STATEMENT IS TOO LARGE - 511 ASSUMED
468.	W	NON-BLANK CHARACTERS IN COLUMNS 1-5 ILLEGAL - CHECK FORM/NFORM OPTION
469.	F	ASF LEFT OF EQUALS MUST EXPAND INTO AN IDENTIFIER OR ARRAY ELEMENT
470.	W	EQUALITY OR NON-EQUALITY COMPARISON MAY NOT BE MEANINGFUL IN LOGICAL IF EXPRESSIONS
471.	W	SIZE OPTION IS MISSING OR IS ZERO - /7/ ASSUMED
472.	F	THE FUNCTION OR SUBROUTINE /1/ MAY NOT HAVE AN ADJUSTABLE SIZE SPECIFICATION
473.	F	THE ADJUSTABLE SIZE OPTION /1/ IS ILLEGAL IN AN IMPLICIT STATEMENT
474.	F	THE VARIABLE MODIFIED BY THE ADJUSTABLE SIZE OPTION /1/ IS NOT TYPE CHARACTER
475.	F	THE ADJUSTABLE SIZE OPTION /1/ IS LEGAL ONLY IN A SUBPROGRAM
476.	F	THE /3/ /1/ CANNOT BE USED AS AN ADJUSTABLE SIZE OPTION

477. W A ZERO SIZE OPTION IS ILLEGAL - STANDARD DEFAULT ASSUMED

478. W PREVIOUS USAGE OF /1/ CONFLICTS WITH BEING TYPED - TYPING IGNORED

479. F THE GENERIC FUNCTION /1/ DOES NOT SUPPORT /5/ ARGUMENTS

480. F TRANSFORMATION OF GENERIC FUNCTION /1/ ILLEGAL DUE TO PRIOR USE OF /1/

481. W TYPING OF ASF /1/ WILL BE IGNORED IN LEFT OF EQUALS APPEARANCE.

482. F 1ST AND 2ND ARGS OF FLD LEFT OF EQUAL MUST BE POSITIVE INTEGERS WHOSE SUM IS < = 36

483. F 3RD ARG OF FLD LEFT OF EQUAL IS NOT SCALAR OR ARRAY OR IS TYPE LOGICAL

484. F TOO MANY ARGS FOR FLD LEFT OF EQUAL

485. F EXPRESSION RIGHT OF EQUAL CANT BE TYPE LOGICAL

PHASE2 ERROR MESSAGES

201. W EQUATING /1/ WITH /1/ IS REDUNDANT

202. F EQUATING /1/ WITH /1/ IS INCONSISTENT

203. F EQUATING /1/ WITH /1/ CAUSES CONTRADICTORY ALIGNMENT

204. F PROGRAM CONTAINS MORE THAN 510 SYMREFS

205. W /1/ AND /1/ IN COMMON HAVE REDUNDANCY IN EQUIVALENCE

206. F /1/ AND /1/ IN COMMON HAVE INCONSISTENCY IN EQUIVALENCE

207. F /1/ IN EQUIVALENCE EXTENDS COMMON BLOCK /1/ BELOW ORIGIN

208. F EQUATING /1/ WITH /1/ CAUSES COMMON EXTENSION

209. W STATEMENT CANNOT BE REACHED

210. W STATEMENT IS NEVER REFERENCED

211. F BRANCH TO NON-EXISTENT LABEL /1/

212. F ILLEGAL TRANSFER TO /1/ RANGE OF DO

213. F ILLEGAL TRANSFER INTO PARALLEL DO AT /1/

214. F ILLEGAL TRANSFER INTO DO AT /1/ FROM NESTED DO

215. F ILLEGAL TRANSFER INTO OF NESTED DO AT /1/ FROM DO

216. F PROGRAM CONTAINS MORE THAN 1023 EQUATED NAMES

217. F /1/ IS USED AS AN ARRAY IN AN EQUIVALENCE STATEMENT BUT IS NOT DIMENSIONED

218. F /1/ AND /1/ IN DIFFERENT COMMON BLOCKS ILLEGALLY EQUIVALENCED

219. W /1/ IS REFERENCED AS AN ARRAY BUT IS NEVER DEFINED

220. F THE SUBPROGRAM DUMMY ARGUMENT /1/ IS EQUIVALENCED

221. F THE ARRAY /1/ HAS ADJUSTABLE DIMENSIONS BUT IT IS NOT AN ARGUMENT TO THIS SUBPROGRAM

222. W THIS ILLEGAL TRANSFER TO /1/ INSIDE A DO CANNOT BE REACHED BY ANY TRANSFER OUT OF THE DO

223. W /1/ IS AN ILLEGALLY DEFINED DO PARAMETER IN THE EXTENDED RANGE OF THE DO AT LINE /6/

224. W THIS DO MAY BE ILLEGALLY EXTENDED. IT IS CONTAINED IN THE EXTENDED RANGE OF THE DO AT LINE /6/

225. W THIS DO IS CONTAINED IN EXTENDED RANGE OF DO AT LINE /6/ AND SOME CONTROL PARAMETERS ARE THE SAME

226. W THERE IS AN ILLEGAL TRANSFER TO /1/ INSIDE A DO FROM THE EXTENDED RANGE OF THE DO AT LINE /6/

227. W COMPILER TABLES EXCEEDED DURING EXTENDED-DO FLOW ANALYSIS. ANALYSIS TERMINATED. COMPILATION CONTINUES

228. F RETURN /6/. THERE ARE ONLY /6/ RETURNS SPECIFIED

PHASE4 ERROR MESSAGES

401. W DATA STATEMENT: /1/ IS INCONSISTENT WITH /1/

402. W DATA STATEMENT: VARIABLE LIST EXCEEDS LITERAL LIST

403. W DATA STATEMENT: LITERAL LIST EXCEEDS VARIABLE LIST

404. F DATA STATEMENT: /1/ IS IN BLANK COMMON

405. F DATA STATEMENT: /1/ IS NOT IN BLOCK COMMON

406. F DATA STATEMENT: /1/ IS IN BLOCK COMMON

407. F DATA STATEMENT: TOO MANY IMPLIED, NESTED DO'S

408. T COMPILER ABORT IN PHASE4

409. F TOO MANY ARGS

410. F LOGICAL TABLE OVERFLOW

411. T COMPILER ABORT IN PHASE4B

412. W /1/ IS NOT DEFINED

413. W THE CHARACTER ASSIGNMENT INVOLVES A TRUNCATION OF THE RIGHT OF EQUALS

414. F THE ADJUSTABLE DIMENSION /1/ IS NOT AN ARGUMENT TO THIS SUBPROGRAM

415. F FILE B* IS NOT BIG ENOUGH

416. F FORMAT NUMBER /1/ DOES NOT EXIST

417. W THE CHARACTER CONSTANT /1/ WAS TRUNCATED TO FIT /1/

418. W FORMAT NUMBER /1/ IS MISSING; FORMAT (V) SIMULATED

419. W SUBSCRIPT FOR ARRAY /1/ IS OUTSIDE RANGE

420. F LABEL /1/ IS NOT DEFINED

421. F COMPILER ERROR. CODE GENERATED MAY BE INCORRECT

422. F THE ADJUSTABLY DIMENSIONED CHARACTER SCALAR /1/ IS NOT AN ARGUMENT TO THIS SUBPROGRAM

EXECUTIVE ERROR MESSAGES

001. W ALTER FILE ERROR FOLLOWING $ ALTER /6/, /6/

002. W ALTER FILE PROCESSING STOPS WITH $ ALTER /6/, /6/

003. W $ UPDATE CARD MUST BE FOLLOWED BY $ ALTER CARD

004. F COMDK SEQUENCE NUMBER IS /6/, IT SHOULD BE /6/

005. F CHECKSUM ERROR ON COMDK CARD NUMBER /6/

006. F PREMATURE EOF WHILE READING COMDK

007. W MEMORY EXPANDED, USE $ LIMITS OR CORE = OPTION FOR NEXT RUN

008. T NO MORE MEMORY EXPANSION ALLOWED

009. F NOT ENOUGH MEMORY AT PRESENT, TRY AGAIN

010. F MEMORY EXPANSION IMPOSSIBLE, USE $ LIMITS CARD OR CORE = OPTION FOR NEXT RUN

011. W OPTZ CAN OPERATE ON NO MORE THAN /6/ SUBEXPRESSIONS

012. T OPTZ CAN OPERATE ON NO MORE THAN /6/ SUBEXPRESSIONS

013. T PREMATURE EOF WHILE READING S*. UNABLE TO COMPILE.

014. T COMPILER ABORT IN PHASE /6/

TIME SHARING BASED SERIES 6000 FORTRAN ERROR MESSAGES

File and Record Control Type Errors

1. GET CODE 5 - File Code

   Record size is zero in record control word

2. PUT CODE 4 - File Code

   Current logical record larger than buffer

3. CLOSE CODE 3 - File Code

   File to be closed is not in chain

4. GET CODE 4 - File Code

   Block serial number error

5. FILE SPACE EXHAUSTED - File Code

   Attempts to "grow" this file have been denied by the Time Sharing System.

6. Back/FORWARDSPACE ERROR - File Code

   Bad Status returned on DRL FILSP

Compiler Abort

1. COMPILER ABORTING

   This message is printed at teletypewriter followed by DRL ABORT. The compiler abort code is stored into slave prefix cell 0.

DIAGNOSTIC MESSAGES ISSUED BY TIME SHARING LOADER

All messages are prefixed by either W, for warning or F for fatal. The majority of errors are diagnosed as warnings because the user has the ability to hit the break key at any time. Thus, the decision is left to the user to continue or stop.

XXXXXX UNDEFINED

Symbol (XXXXXX) is an undefined SYMREF .DRL ABORT is substituted for all references.

XXXXXX LOADED PREVIOUSLY

SYMDEF (XXXXXX) previously defined in load table.

INCONSISTENT PREFACE FIELD (Deck) (Card)

One of two conditions occur on card number (card) in deck number (deck). The conditions are: (1) a SYMREF (type 5) appears with a non-zero size field (bits 0-17) in the preface card; or, (2) a LABELED COMMON (type 6) appears with a zero size field (bits 0-17).

LABELED COMMON XXXXXX - SIZE INCONSISTENT

LABELED COMMON (XXXXXX) defined previously with smaller size. Loading continues using original size.

ILLEGAL CHECKSUM (Deck) (Card)

The checksum on card number (Card) of deck (Deck) does not compare when recalculated. Loading continues.

ILLEGAL BINARY CARD (Deck) (Card)

Card number (Card) of deck (Deck) is not either preface (type 4), binary (type 5), or BCD (type 6). Card is ignored. This message may also appear where a preface or binary card appears out of expected order.

COMMON SIZE INCONSISTENT (Deck) (Card)

Blank common already defined. A subsequent deck is encountered having a larger blank common region specified. The deck is ignored and loading continues.

ILLEGAL LOAD ADDRESS (Deck) (Card)

A calculated storage address falls outside loadable store. The deck is ignored but loading continues.

XXXXXX LOADED PREVIOUSLY, LABELED COMMON ILLEGAL

SYMDEF (XXXXXX) already defined. XXXXXX appearing n current preface record is a Labeled Common. Deck is ignored.

The following diagnostics are preceded by a printout of the record in error and are generally associated with OCTAL correction processing.

NON-OCTAL DIGIT IN LOCATION FIELD

Self explanatory.

FIELD EXCEEDS 12 DIGITS

Twelve octal digits is maximum allowed in word.

ILLEGAL TERMINATOR

Octal field is eliminated incorrectly. Check syntax rules in the General Loader manual.

IC MODIFICATION NOT POSSIBLE

Field requested IC modification ($code). In this case no other modifiers are allowed. Bits 30-35 of the constructed instruction are checked and found to be non-zero.

Fatal Diagnostics

EOF READING BINARY (Deck) (Card)

Unexpected EOF while reading binary, identification of last record read is supplied.

ENTRY NOT FOUND

Primary entry name (..... or first primary SYMDEF) was not found in load table. Diagnostic may also appear when subroutine .SETU. is not found.

H* TOO SMALL

File specified as save file (H*) not large enough to hold program.

LIBRARY SEARCH TABLE EXCEEDED

A "Fatal" error. A fixed size table has been exceeded. The loader must be reassembled increasing the size of this table.

REQUEST FOR MEM TO EXPAND LOAD TABLE - DENIED

A request for 1K to be added at the upper address end of the load table was denied by the system. Loading terminates. Suggest user rerun job.

PROGRAM EXCEEDS STORE SIZE

Not enough store to load program. Rerun with increased "CORE =" option. "CORE=" value is added to the 22K loader requirement.

ILLEGAL STATUS WHILE READING (File)

Only status accepted other than EOF is ready.

BLOCK SERIAL ERROR READING (File)

Block number in file (File) does not agree with expected number.

APPENDIX C

INTRODUCTION TO SERIES 6000 FORTRAN

GENERAL DESCRIPTION

This appendix contains an introduction to the Series 6000 FORTRAN compiler and compares it with other FORTRAN compiler with a primary emphasis on a comparison with the existing Series 600 FORTRAN IV and Time Sharing FORTRAN compilers.

The basic intent of the new compiler is to replace the existing batch and Time Sharing FORTRAN compilers with a single integrated compiler serving both batch and time sharing.

It is actually a system made up of three separate software entities: the FORTRAN compiler; a library of run-time modules to support the execution of object programs; and a new time sharing system interface.

The initial release (in SDL 3.3) included the FORTRAN compiler; extensions and modifications to the FORTRAN library; a modified General Loader; and a new time sharing system, called YFORTRAN, with an extended RUN command. The introduction of Series 6000 FORTRAN has also required changes in other areas such as File and Record Control, System Input, Peripheral Allocator; therefore, this compiler will not operate under SDL's prior to SDL 3.3. The Global Optimizer, an optional phase of the compiler, was included in the SDL 4.0 software release. In the Global Optimizer phase a significant amount of analysis of the program being compiled is done to produce a more efficient object program.

The following are the system objectives:

1. A fully compatible three dimensional system.
2. A wide range of new language features.
3. Full ASCII capability.
4. Subset compatibility with the current batch FORTRAN.
5. A high performance system.
6. An easily maintainable and extendible system.

SYSTEM CHARACTERISTICS

There is only one version of the compiler and it compiles all FORTRAN programs originating from batch or time sharing, local or remote. A collection of source programs may be compiled, some through time sharing, some through batch, and the object modules combined for execution in either environment.

Source Compatibility

The source files processed by FORTRAN may be any combination of the following:

1. A BCD card image file, with or without alters.
2. A COMDK file, with or without alters.
3. A time sharing ASCII file.
4. A formatted BCD line image file, with or without slew controls.
5. A formatted ASCII line image file, with or without slew controls.

File Contents

The source file contents may be in standard source format or in the relaxed "free-form" format especially suitable in time sharing, with or without line numbers. Files in any of the accepted file or source formats may be compiled without conversion, from either batch or time sharing.

Compilation of Subprograms

Many compilations can be done within one activity provided that the options are the same for a collection of subprograms. The batch user stacks his source programs, back to back, behind one compiler call card. The time sharing user lists a series of source files to be compiled or provides multiple subprograms in a source file. To the compiler there is one input file, S*, and source programs are separated by END statements.

For larger programs requiring more core to compile than that allocated to an activity, the compiler will "grow" in an attempt to satisfy this need. Normally a satisfactory compilation will result; however, the operating system may deny more core to the compiler. The user is warned, in any event, that his $LIMITS card should be changed for subsequent recompilations.

Error Detection and Diagnostics

The error detection and diagnostic capabilities have been significantly improved with several hundred unique messages. All diagnostics are classified as either Warning, Fatal, or Terminated. Warning does not cause compilation to be terminated; Fatal causes execution to be deleted; and Terminated causes a termination of compilation.

Compiler Construction

The compiler is written in and generates object modules in "pure procedure". .DATA. space and instruction space are clearly spearated and the instruction space remains constant over the life of the execution process.

Allocation of Storage

Storage allocation for the object program is done in two phases of the compiler. Phase 2 allocates storage for arrays, equivalenced variables, and all data that is in blank or labeled common. Phase 4 allocates storage for local scalars, namelists, switch variables, and compiler generated constants and temporary data. Phase 4 also allocates space and generates code for the procedure.

All variables (except those in blank or labeled common), constants, and temporary data are allocated to the local data storage area .DATA. which is treated by the loader as a local labeled common. Figure C-1 shows the storage layout for two typical Series 6000 FORTRAN object programs.

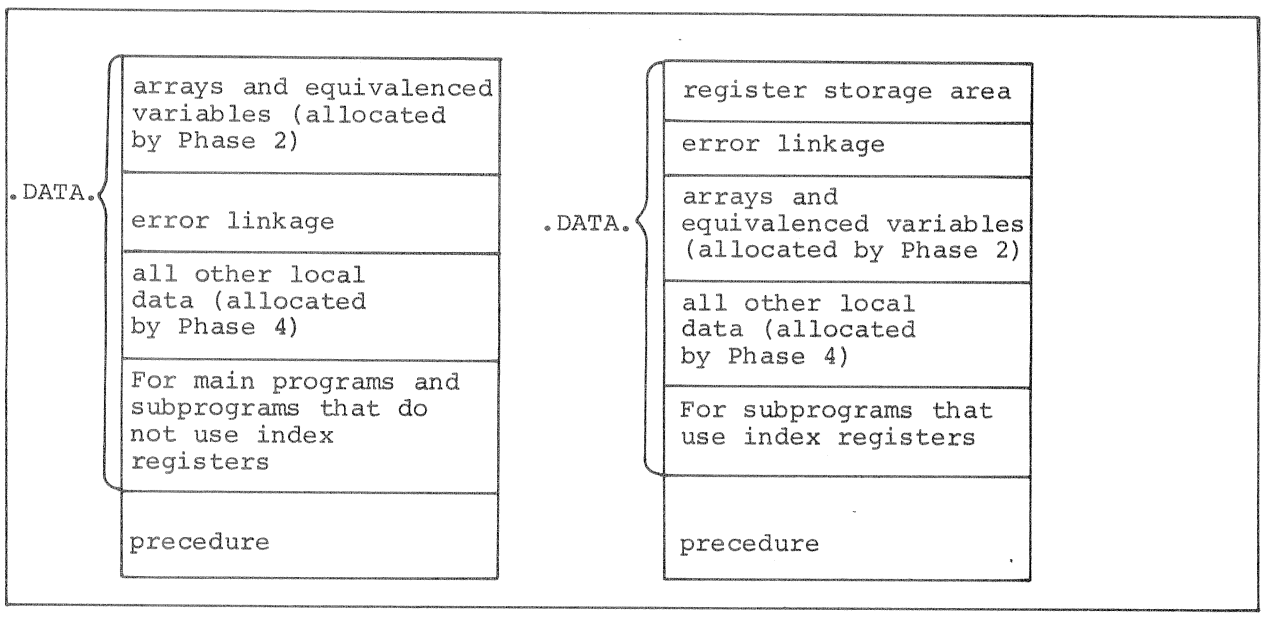


Figure C-1. Storage Allocation for Object Programs

NEW LANGUAGE FEATURES

This section is divided into three parts: Series 6000 FORTRAN features previously available to Series 600 Time Sharing but not available to batch users; Series 6000 FORTRAN features previously available to Series 600 batch users but not available to time sharing users; and completely new features available to the user of Series 6000 FORTRAN.

Series 600 Time Sharing Features Now Available to all Users

1. Core to Core Conversion - ENCODE and DECODE statements enable the user to build or take apart a string of character information under format control going from or to internal binary. This feature relates records to character array elements giving meaning to the slash (/) in referenced FORMATs.

2. List Directed Formatted I/O - This feature permits a user to do formatted I/O without providing a specific format. On input, data is converted as it is read, with comma separators, into the mode appropriate to the type of variable being satisfied from the list. On output, each type has an associated default format conversion that is applied. Line or records are read or written, as required, until the list is exhausted.

3. Random File I/O - This feature permits input/output with random files. Records can be read or written, in binary, at random over the entire file space. It is also possible to randomly access a sequential input file.

4. Mixed-Mode Arithmetic - Arithmetic data types can be combined in an expression with automatic conversion of type.

5. Subscripts May Be Any Expression - Subscripts are not restricted to the classical C*V+C' form. They may be any arithmetic expression including nested subscripts or Real expressions.

6. END = Clause in READ Statements - This feature provides for an end-of-file recovery on input.

Series 600 Batch Features Now Available to All Users

1. Labeled COMMON Storage
2. EQUIVALENCE Statements
3. BLOCK DATA Subprograms
4. COMPLEX and DOUBLE-PRECISIONS Statements
5. PAUSE with display
6. ENTRY statement for multiple entry points in a subprogram.
7. LINK and LLINK capabilities

Features New to All Users

1. IMPLICIT Statement - This statement provides the ability to redefine the default typing rules of the compiler (I to N = Integer, all others Real). It also includes typing for other data types (LOGICAL, DOUBLE-PRECISION, etc).

2. PARAMETER Statement - This statement allows the user to define compile time variables.

3. CHARACTER Statement - Character variables, scalar or arrays, up to a maximum size of $\leq$ 511 can be specified. All character constants and variables, format statements, and formatted I/O statements are controlled, by option, to be either BCD or ASCII. Mapping of size-in-bytes into size-in-words is done automatically.

4. T and R Format Specifiers - The T format specifier provides tabbing within a line, in both directions. The R format specifier is for character data which is carried internally as right justified information with leading zeros.

5. ABNORMAL Statement - This statement qualifies external functions as to side effects.

6. PAUSE and STOP with Display - These statements add the ability to print a message, usually on the standard output device, when the program PAUSEs or STOPs. With PAUSE, there is an option of continuing or terminating the program.

7. Quoted Constants - Character constants may be enclosed in either quotes (") or apostrophes ('). They may be used anywhere a character variable can appear and in FORMAT statements as an alternative to the NH notation.

8. ERR = Clause in READ and WRITE Statements - This feature provides for error recovery from certain I/O error conditions.

9. Switch Variables - Those variables that are defined by the ASSIGN statement can be used in other than the Assigned GO TO statement. For example: in the label list of the computed GO TO statement and the arithmetic IF; in an unconditional GO TO statement; and in the END and ERR clauses of the I/O statements.

10. TYPE Statements with Size-in-Bytes Notation - In the REAL statement, if the size is greater than seven, the type will be taken as DOUBLE-PRECISION. In the character statement the compiler will allocate enough words for an element to accommodate the specified size (varies with BCD/ASCII options).

11. FLD Function - A built-in function that provides bit string and field capabilities. *

12. XOR Function - A built-in 'exclusive or' function.

13. Argument Validation for Built-in Functions - An improvement in the area of diagnostics.

14. Null Label Fields in the Arithmetic IF Statement - This implies a branch to the next statement in the program.

15. Generalized Arithmetic Statement Function - Type association and distinction between functions and arrays is made at call time.

16. Relaxed Rules on Placement of Specification Statements - The specifications for a symbol need only precede the first actual usage. NAMELIST and DATA statements may appear anywhere in the program. A namelist may be extended by additional NAMELIST statements for the same NAMELIST name.

17. Output Reports - A storage map and a cross reference list covering only statement numbers are now part of the output.

18. NAMELIST PUNCH - Output is generated in a format which is acceptable to the NAMELIST input modules.

19. Data Initialization in Type Statements - This provides greater compatability with competing FORTRAN compilers.

ASCII AND BCD CAPABILITIES

The character set, ASCII or BCD, is determined at compile time by options on the $FORTY or $FORTRAN control card and the RUN command of the YFORTRAN or FORTRAN Time Sharing System. The default option for batch users is BCD; for time sharing users the default option is ASCII.

For a BCD compilation the object module is fully BCD oriented; internal character data is carried in BCD; storage is allocated in six characters per word; and library calls for input/output, ENCODE, DECODE, etc are generated to reference the BCD entry names. For ASCII compilations, the object module will have all ASCII properties; character data is carried in ASCII; storage allocation is in four characters per word; and library calls are made to the appropriate ASCII entry names.

Generally, object modules with different character sets cannot be mixed due to the conflicts that arise over which routines are to be loaded from the library, how to index through character arrays, how to interpret FORMAT statements, etc. Transliteration of file media at run time is automatic.

BCD or ASCII internal programs execute in either batch or time sharing, with certain automatic convenience functions for dealing with the variety of file and device types accessible to the program. In terms of specific problems, automatic file transliteration and/or reformatting on a logical record basis is provided for the following:

1. Execution of a BCD program under time sharing.

   a. Input and output may be directed to the teletypewriter.

   b. Input files may be ASCII.

2. Execution of an ASCII program in batch mode.

   a. Input and Output may be directed to the reader, printer, punch, or SYSOUT.

   b. Input files may be BCD (media 0, 2, or 3) or ASCII.

3. Execution of a BCD program in batch mode.

   a. Input files may be ASCII. *

4. Execution of an ASCII program under time sharing.

   a. Input files may be ASCII or BCD (media 0, 2, or 3). *

ASCII Standard System Format Files

This file format is common for batch and time sharing users as are the library routines that read and write them. This common procedure for batch and time sharing guarantees symmetry and compatibility. The file format for ASCII conforms with the File and Record Control rules for "standard system format". Every line is recorded as a logical record. *

SERIES 6000 FORTRAN COMPATIBILITY

Compatibilites With Other FORTRANS

The Series 6000 FORTRAN language is a superset of the ANSI FORTRAN and is fully compatible with this standard. Compatibility with other FORTRAN compilers was based on the following order according to their relative importance to the Series 600/6000:

1. Series 600 Batch FORTRAN
2. 360 FORTRAN Level H
3. 1108 FORTRAN V
4. Series 600 Time Sharing FORTRAN.

The Series 600 Time Sharing FORTRAN is ranked last because of its considerable deviations from ANSI FORTRAN. Many features from the 360 and 1108 FORTRANS are included in Series 6000 FORTRAN. Most of the features in the Series 600 Time Sharing FORTRAN are included.

Batch Incompatibilities

See Appendix D for a description of the DEBUG and FORMAT GENERATOR statements.

Time Sharing Incompatibilities

The FILENAME, OPENFILE, BEGINFILE, and CLOSEFILE statements are not implemented in Series 6000 FORTRAN. The ASCII statement is implemented only to the extent that correct code is generated with a warning diagnostic. ASCII statements should be corrected to CHARACTER statements at the programmer's convenience.

The FILENAME statement is the most significant in the above list. In time sharing, operations on files contained a FILENAME reference (either constant or variable). The execution of such operations used the current User Master Catalog (UMC) and Accessed File Table (AFT) to search for such a filename or to create a temporary file if none existed. This convenient approach had many short comings: the inability to provide a full catalog file-descriptor as a filename reference; the source level dependence on a particular type of file system; and the deviation from ANSI logical file referencing conventions. With Series 6000 FORTRAN standard integer file references are employed and the equation of that file number with some specific file or device is resolved at run time. This implementation is entirely symmetrical with the batch mode of operation where control cards link logical files to actual files.

Another small deviation involves list directed terminal input/output. Under Time Sharing FORTRAN it was represented with a colon. In Series 6000 FORTRAN, a comma or colon is used.

Object Level Compatibility

The preceding paragraphs were concerned with source level compatibility. In general, full object level compatibility has been achieved with the batch FORTRAN as of SDL 3.3; a high degree of object level compatibility has been achieved with pre SDL 3.3 batch FORTRANs; and there is no object level compatibility with Time Sharing FORTRAN. The object module produced by Time Sharing FORTRAN is in a format that is entirely different from batch FORTRAN and is not loadable by the General Loader.

Given the General Loader as a common loader, a common loadable format is defined. Series 6000 FORTRAN produces object modules in this standard format.

Object incompatibilities with pre-SDL 3.3 batch FORTRAN arise from the design decision to generate pure procedure. The problem brought on by this decision is in the area of argument passing across subprograms. Prologue processing by FORTRAN IV includes logic for passing incoming arguments (pointers) down to lower level subprograms. The code generated for this involved fetching the argument pointer from the calling routine and "stuffing" it into the appropriate argument position in the calling sequence to the lower level routine. Similarly, when array elements (e.g. A(I+1)) are passed, the address of an element is computed and stuffed into the calling sequence. These operations altered the procedural text of the current module, making it "impure".

The code generated by Series 6000 FORTRAN is such that pointers of this sort are carried in the local data space of the object module (.DATA.). Calls which pass such arguments (pointers) make use of the indirection capabilities of the machine. These arguments are in the form of pointers using the I modifier, addressing the actual pointer in the data space. Similarly, when an array element's address is computed, its pointer is left in an available index register (if there is one). In the argument position of the calling sequence, a register modified pointer is generated.

Thus, calling sequences generated by FORTRAN IV always contain direct argument pointers. Unfortunately, the code generated for argument pick-up in a called routine capitalized on this. As a consequence, old object modules can CALL Series 6000 FORTRAN object modules, but not all Series 6000 FORTRAN modules can CALL old object modules.

In SDL 3.3, FORTRAN IV was changed to pick up arguments using indirection. On this score, compatibility was achieved by changing the old FORTRAN IV rather than give up compatibility for the sake of pure procedure. There is of course no incompatibility between SDL 3.3 FORTRAN IV produced object routines and previous FORTRAN IV produced object routines introduced by this change. They can continue to be freely mixed.

This incompatibility exists when Series 6000 FORTRAN routines CALL pre SDL 3.3 FORTRAN IV produced routines. Similar problems may exist when Series 6000 FORTRAN routines call a user subprogram coded in GMAP. Here it depends on the technique used in the GMAP routine for fetching arguments. The crux of the difference is brought out by analyzing the following two machine instructions.

```
LDX0    2,1     (1)
EAX0    2,1*    (2)
```

The effect on X0 is the same for both when the argument at 2,1 is a direct pointer. When that argument may involve indirection or register modification, only form (2) does the job.

Another level of incompatibility exists between FORTRAN IV and Series 6000 FORTRAN in the handling of the indicators after a function call. Series 6000 FORTRAN generates code assuming that the indicators reflect the function result; however, FORTRAN IV does not make that assumption. This is only critical in the area of user supplied GMAP function programs since both FORTRAN IV and Series 6000 FORTRAN generate code to set the indicators correctly upon exit from a function subprogram.

File Compatibility

All files, formatted or unformatted, ASCII or BCD, sequential or random, are in standard formats, and are the same for batch and time sharing. Lack of this compatibility has been a major concern for users of the previous FORTRAN product offering. Files written by time sharing FORTRAN programs could not generally be read by batch FORTRAN programs, and vice versa. This does not exist with Series 6000 FORTRAN.

Series 6000 FORTRAN can read/write random binary files in the format generated by Series 600 Time Sharing FORTRAN. This capability is enabled by one of the following:

```
CALL RANSIZ (U,M,1) or
$ FFILE U,NOSRLS,FIXLNG/N
```

Otherwise, random binary files will be recorded in standard system format.

PERFORMANCE

The performance objective of the FORTRAN Series 6000 is simply stated, to provide a fast compiler which can generate fast executing object modules. It is generally realized that the more analysis done to improve the efficiency of the object module, the greater the time spent in compilation. Consequently, this analysis is subdivided into two classes:

1. Local Optimization (LO) - that analysis which can generally be done at the statement level.

2. Global Optimization (GO) - that analysis which is done over many statement, i.e., program blocks as defined by the ANSI FORTRAN standard.

To give the user some control over the balance between compilation and object efficiency it was decided to collect the GO analysis into a unique compiler phase, callable by option. LO analysis is always performed.

The initial release of Series 6000 FORTRAN with SDL 3.3, did not include the global optimization phase. This phase was made available with the release of SDL 4.0.

Local Optimization

Following are some of the object efficiency functions done on a local basis:

1. Logical expressions are sorted so that shorter alternative passages are executed first, and evaluation ceases as soon as the true/false state has been determined.

2. Subscript expressions may be register contained, eliminating multiple computations.

3. Constants may be register contained across statements.

4. Multiplication and division by powers of two are performed using shift or exponent register operations.

5. Constant arithmetic is done at compile time.

6. Many special operator/operand relationships are recognized to capitalize on the 600/6000 machine instruction set. Examples are I*1, I**1, I=0, I=I+1, I=I+J.

7. Where possible operations involving constants make use of the DU, DL modifiers.

8. Where there is no redefinition of a scalar dummy argument within a subprogram, the value of that argument is stored locally. This eliminates an indirect cycle for each reference to that argument.

Global Optimization

The global optimization analyses are:

1. Common subexpression analysis - determination of multiple occurrences of the same subexpression within a program block, aimed at performing the computation only once.

2. Expression compute point analysis - determination of the optimal place/time for the computation of some expression in relation to the loop structure of the program and the redefinition points of the expression's constituent elements.

3. Induction variable expression analysis - determines the optimal computation sequence. Aimed at reducing such expressions to simple operations upon an index register at the loop boundaries.

4. Loop collapsing analysis - aimed at reducing two or more nested loops into a single loop.

5. Register management analysis - determination of how registers and temporary storage are to be allocated. Priorities are assigned according to the number of references to an expression and the loop level of those references. Candidates for global assignment over one or more program loops are selected.

6. Induction variable materialization analysis - determination of the necessity for materializing in core the current value of a DO index.

Many of these functions, some from each group, are new to Series 6000 FORTRAN. Probably GO should be elected for any program which is known to be compute-bound, is large, makes extensive use of DO loops, or was written by an inexperienced programmer.

This leaves a large spectrum of programs which won't significantly benefit from global optimization analysis; some may even take longer if GO is called when compile time and execute time are lumped together. The programmer must determine whether or not GO analysis should be performed on his program. The OPTZ option on the $ FORTY or $ FORTRAN card specifies GO; NOPTZ or nothing (NOPTZ is the default case) specifies no Global Optimization. Time Sharing users use OPTZ in the option list of the RUN command to call the global optimizer.

Compilation Performance

Compile speed is also a function of the properties of the program being compiled and directly related to the options selected on the $ FORTY or $ FORTRAN control card. The GO compiler phase will reduce compile time for most programs by a factor of about twenty percent. For many programs the specification of LSTOU will double the compile time. Measured in statement per minute, the compilation rate improves with larger programs. The smaller the program the greater the effect of the basic overhead to start compilation, step through the phases, and terminate. Binary and compressed decks, source listing, storage maps, cross reference reports, etc. decrease the compilation rates.

APPENDIX D

FORMAT GENERATOR AND DEBUG STATEMENTS

There are two FORTRAN A statements that are not accepted by Series 6000 FORTRAN: DEBUG and FORMAT GENERATOR. The FORMAT GENERATOR-TO-FORMAT conversion program (FGTF) will place an asterisk in column 1 of DEBUG statements (so that they are treated as comments by Series 6000 FORTRAN) and will convert FORMAT GENERATOR statements to equivalent FORMAT statements. The compiler debug capability does not exist in Series 6000 FORTRAN; however, the DEBUG(STAB) option on the $ FORTY or $ FORTRAN card or with the time sharing RUN command and the General Loader debug card processing are available.

FGTF will accept entire decks or FORMAT GENERATOR statements in BCD or COMDK form and will handle the following INCODE's: IBMF, IBMC, IBMEL, and G-225. The two input formats acceptable by FGTF are:

1. Standard System Format - The input file is I* (default data file).
2. Card images (e.g., IMCV tape) - The input file code is IN. If IN is used, the input media must be magnetic tape or cards (via the card reader).

FGTF output options (output is in Standard System Format) are entered on the $ PROGRAM FGTF control card (see the Control Cards manual). These output options are:

LSTIN - A listing of the BCD cards and expanded COMDK cards from the input file are assigned to P*.

NLSTIN - Default option. Only the FORMAT GENERATOR input card images and the resultant FORMAT card images are assigned to P*.

DECK - All card images on the input file are copied to P* for punching. If file code OT is present, the card images will be copied to OT and not P*. If OT is present, the DECK option is forced. An IMCV tape may be copied in this manner.

NDECK - Default option. No BCD cards are produced except as described under DECK.

COMDK - COMDKs of all FORTRAN routines are sent to P* (or to K* if K* is present). If K* is present, the COMDK option is forced.

NCOMDK - Default option. No COMDKs are produced.

ON6 - A $ SELECT control card is treated as a $ FORTRAN card. If no $ cards are read, the input is assumed to be FORTRAN source and the DEBUG and FORMAT GENERATOR changes are made. If $ cards are encountered, only $ FORTRAN, $ FORTY, $ FORTREX and optionally $ SELECT cards will cause subsequent (up to the next $ activity card) data to be treated as FORTRAN source.

LSTOU - A program breakdown by statement type is output after each FORTRAN program. A summary is output at the end in all cases.

All other cards will be copied as directed by the options to the output file and the $ FORTRAN and $ FORTREX cards are changed to $ FORTY cards. The options on the $ FORTRAN and $ FORTREX cards are not changed.

FGTF is invoked as follows:

```
$  PROGRAM  FGTF,options
```

The default memory allocation (16k) is sufficient; however, the output limits should be increased if the input is an IMCV tape or a long program. The minimum memory required is 9k. A suitable set of limits might be:

```
$  LIMITS  20,10k,,10k
```

The output from FGTF will be of questionable value if the source input program contains errors.

Any diagnostics produced will be written as comments in the output deck as well as being listed.

In the following example a FORTRAN A source deck is made available for processing by Series 6000 FORTRAN and is output, as a COMDK, to K*. Series 6000 FORTRAN reads this file (as S*) and subsequent processing is as though a source deck had been presented to Series 6000 FORTRAN in the usual manner.

```
$  PROGRAM  FGTF,LSTIN
$  FILE     K*,X1S,2L   (Forces COMDK option)
$  DATA     I*
    FORTRAN A Source Deck - BCD or COMDK
$  FORTY    NDECK
$  FILE     S*,X1R,2L
$  EXECUTE
```

The following deck setup can be used to copy an IMCV tape. File code IN is used if the input is in the form of cards or card images (14 or 33 words per physical record). File code I* is used if the input is Standard System Format.

```
$  PROGRAM  FGTF     (No printed output)
$  TAPE     IN,X1D,,1234,,FA-IMCV   (See description above)
$  TAPE     OT,X2D,,,,FY-IMCV   (Forces DECK option)
$  ENDJOB
```

INDEX

Honeywell Bull

HONEYWELL INFORMATION SYSTEMS

BJ67, Rev. 1